Sommario

In questo testo verranno illustrate le caratteristiche fondamentali del protocollo di rete **IPv6**. La prima parte dell'elaborato costituisce una raccolta dei concetti che il candidato ha fatto propri, leggendo e consultando testi ed articoli durante il suo lavoro di tesi. I capitoli 1 e 2 riprendono le nozioni fondamentali delle reti di calcolatori per guidare il lettore verso gli argomenti delle sezioni successive. Il capitolo 3 mostra i dettagli del protocollo IPv6 e le basi dei suoi meccanismi di funzionamento. In questo terzo capitolo si fa accenno inoltre ai miglioramenti che la nuova versione di IP potrà apportare al *networking* con una gestione modulare delle funzionalità opzionali ed un agile supporto a qualsiasi futura estensione. Si analizzano alcuni di questi aspetti del protocollo nella seconda parte, costituente l'opera pienamente originale del candidato.

Vengono presentati nel capitolo 4 alcuni test di prestazioni che hanno permesso di misurare la bontà del protocollo ma anche di mostrarne i punti deboli. Nel capitolo 5 si mostrano quindi i risultati dei test effettuati, e si nota che sui *link* a 100Mbps del banco di prova, allestito dal candidato, ciò che penalizza maggiormente IPv6 è l'overhead del preambolo del protocollo. Tuttavia i risultati sono stati considerati incoraggianti: su collegamenti con maggiore banda, sui quali l'uso di cicli di clock del processore può pesare sulla quantità di dati trasmessi, IPv6 potrà esprimere le sue potenzialità. Viene inoltre presentato un interessante capitolo d'appendice sulla trasportabilità del software di rete al nuovo protocollo, viene trattato come caso di studio un semplice server TCP/IP e viene mostrata una proposta di conversione utilizzando tecniche per scrivere applicazioni che siano indipendenti dal protocollo di rete.

Indice

Elenco delle figure **vi**

Elenco delle tabelle **viii**

I Le Reti IPv6 **1**

1 Introduzione **2**
1.1 I modelli utilizzati . . . 3
1.2 Perché IPv6 . . . 5
1.3 Un nuovo schema di indirizzamento . . . 6
1.4 Modalità di indirizzamento . . . 7
1.5 Transizione da IPv4 a IPv6 . . . 8
1.6 Conclusioni . . . 9
Bibliografia . . . 11

2 Una panoramica su IP **12**
2.1 Terminologia . . . 12
2.2 Architettura di una rete . . . 15
2.3 Preambolo IP . . . 17
2.4 Indirizzi e nomi . . . 20
2.5 Router ed internetworking . . . 22
2.6 La tabella di instradamento . . . 24
Bibliografia . . . 25

3 Dettagli del protocollo **26**
3.1 Preamboli IPv6 . . . 26
3.1.1 Preambolo standard . . . 26
3.1.2 Preamboli di estensione . . . 30
3.1.3 Differenze con v4 . . . 34
3.2 Indirizzi IPv6 . . . 35

3.2.1 Sintassi degli indirizzi IPv6 . . . 35
3.2.2 Tipi di indirizzi . . . 37
3.3 ICMPv6 . . . 43
3.3.1 Formato dei messaggi . . . 43
3.3.2 Tipi di messaggi . . . 44
3.3.3 Sicurezza . . . 47
3.4 Neighbor Discovery . . . 47
3.4.1 Strutture dati per il ND . . . 49
3.4.2 Algoritmo di trasmissione di un pacchetto . . . 50
3.4.3 Autoconfigurazione . . . 52
3.5 Il Routing . . . 53
3.5.1 Terminologia . . . 53
3.5.2 Routing statico e dinamico . . . 54
3.5.3 Protocolli di routing per IPv6 . . . 57
3.5.4 Vantaggi nel routing . . . 60
3.6 Sicurezza con IPsec . . . 62
3.6.1 Autenticità e Confidenzialità . . . 62
3.6.2 Security Association e Security Policy . . . 62
3.7 Host mobili . . . 64
3.7.1 Scopi di Mobile IPv6 . . . 65
3.7.2 Differenze con Mobile IP per IPv4 . . . 65
3.7.3 Panoramica sul funzionamento . . . 66
3.8 Conclusioni . . . 69
Bibliografia . . . 70

II Misure di traffico **72**

4 Test di prestazioni **73**
4.1 Introduzione . . . 73
4.2 Banco di prova . . . 74
4.2.1 Topologia . . . 74
4.2.2 Routing . . . 74
4.2.3 Strumenti adottati . . . 79
4.3 Progettazione dei test . . . 80
4.3.1 TCP STREAM test . . . 80
4.3.2 Test applicativi . . . 82
4.3.3 Test con IPsec . . . 82
Bibliografia . . . 84

5 Configurazione e lancio dei test **85**
5.1 Collezione dei risultati . . . 85
5.1.1 NetPerf . . . 85
5.1.2 Wget . . . 87
5.1.3 Gnuplot . . . 88
5.2 Lancio dei test e risultati . . . 90
5.2.1 Senza IPsec . . . 91
5.2.2 Risultati con IPsec . . . 93
5.3 Osservazioni . . . 97
Bibliografia . . . 99

A Script e configurazioni **100**
A.1 Script di shell . . . 100
A.1.1 Lancio dei test con netperf . . . 100
A.1.2 Lancio dei test con wget . . . 103
A.2 Configurazioni software . . . 105
A.2.1 Software di base . . . 105
A.2.2 Software Applicativi . . . 108
A.2.3 Configurazione della Rete . . . 110
A.2.4 Configurazione di IPsec . . . 112
Bibliografia . . . 119

B Porting di applicativi di rete **120**
B.1 Introduzione . . . 120
B.2 Strutture dati . . . 121
B.2.1 Problemi . . . 121
B.2.2 Soluzioni . . . 121
B.3 Funzioni . . . 121
B.4 Un esempio . . . 122
B.4.1 Un server TCP/IPv4 . . . 122
B.4.2 AF independence . . . 133
Bibliografia . . . 137

C Pacchetti IPv6 **138**
C.1 Software di analisi . . . 138
C.2 Esempi di pacchetti ICMPv6 . . . 138
C.2.1 ICMPv6 neigh sol . . . 139
C.2.2 ICMPv6 neigh adv . . . 140
C.2.3 ICMPv6 echo request . . . 141
C.2.4 ICMPv6 echo reply . . . 142
C.3 Esempi di pacchetti TCP/IPv6 . . . 143

C.3.1 Invio di un breve messaggio 143
C.4 Esempi di pacchetti con AH o ESP 155
C.4.1 Authentication Header 155
C.4.2 Encapsulating Security payload 158
Bibliografia 160

Conclusioni **161**

Glossario **162**

Indice analitico **164**

Elenco delle figure

1.1 Confronto fra i modelli ISO/OSI e TCP/IP 5

2.1 Il processo di comunicazione 13
2.2 L'imbustamento 14
2.3 Differenti topologie di reti 15
2.4 Interconnessione fra reti 16
2.5 Preambolo IP 18
2.6 Spazio dei nomi nel DNS 21
2.7 Una rete di medie dimensioni 23
2.8 Esempio sul routing 23
2.9 Una tabella di routing 25

3.1 Preambolo IPv6 standard 27
3.2 Preambolo di estensione Hop-by-Hop 32
3.3 Preambolo per Jumbo Payload 33
3.4 Preambolo di estensione per il routing 33
3.5 Semplice struttura di un indirizzo unicast 39
3.6 Struttura di un indirizzo unicast globale 40
3.7 Struttura di un indirizzo multicast 41
3.8 Formato di un messaggio ICMPv6 43
3.9 Comunicazione sicura in tunnel-mode 63

3.10 Esempio di MobileIP 67

4.1 La nostra infrastruttura di rete 75
4.2 Infrastruttura fisica 76
4.3 Scenario delle prove *host-to-host* 81
4.4 AH in transport mode 83

5.1 Test con netperf 91
5.2 Test con netperf - CPU 92
5.3 Test con wget 93
5.4 Test con netperf con IPsec (ESP) 94
5.5 Test con netperf con IPsec (ESP) - CPU 94
5.6 Test con wget con IPsec (ESP) 95
5.7 Test con netperf con IPsec (AH+ESP) 96
5.8 Test con netperf con IPsec (AH+ESP) - CPU 96
5.9 Test con wget con IPsec (AH+ESP) 97

Elenco delle tabelle

2.1 Opzioni nel preambolo IP 20

3.1 Valori del campo *Next header* 29

3.2 Preamboli di estensione di IPv6 31

Parte I

Le Reti IPv6

Capitolo 1

Introduzione

Riusciremmo ad immaginare il nostro mondo senza le reti di calcolatori? è evidente che sarebbe estremamente diverso dal mondo nel quale viviamo attualmente ed ancora più diverso rispetto a ciò che il futuro ci ha riservato. Le reti rendono oggi possibili le più disparate attività socio-economiche ed una analisi dell'evoluzione del protocollo IP (sul quale si fonda Internet) può mostrarci come cambierà il networking, e forse il nostro modo di vivere, nei prossimi anni.

Una rete di calcolatori si pone come primo obiettivo il trasferimento di informazione fra varie entità autonome, le informazioni viaggiano attraverso un certo mezzo trasmissivo da una sorgente ad una destinazione (o più destinazioni) attraversando eventualmente altre entità e vengono opportunamente codificate, trasmesse e reinterpretate per rendere effettiva la comunicazione.

Da questa estrema sintesi possiamo ricavare alcune indicazioni su quali siano gli elementi costituenti una rete: vi è il mezzo trasmissivo e vi sono i nodi che partecipano alla comunicazione: elementi passivi ed attivi, strumenti hardware e software interagiscono fra loro per dar vita al processo di comunicazione. Le diverse fasi di questo processo possono essere raggruppate in livelli.

Una visione "a livelli" di una rete facilita la progettazione e la gestione poiché rende indipendente il funzionamento di uno strato da un altro, a condizione di avere una interfaccia ben definita fra i vari livelli che permetta di modellare il problema "rete di calcolatori" in maniera conveniente.

1.1 Il modelli utilizzati

Il modello di riferimento per la descrizione di una tale struttura é il modello ISO/OSI a sette livelli che formalizza la distinzione fra i vari strati di una architettura di rete e definisce in maniera chiara le nozioni di servizi, interfacce e protocolli (Per maggiori dettagli si veda [Tanenbaum, 1996]).

Ogni livello incarna un diverso grado di astrazione del processo di comunicazione.

Il modello di riferimento OSI è stato formalizzato negli anni ottanta, quando la rete ARPANET, antenata di Internet, era già operativa e il sistema operativo UNIX, largamente utilizzato nelle università statunitensi, presentava da tempo una implementazione dei protocolli TCP/IP. Ciò costituìì la fortuna del modello TCP/IP che, pur presentando una stratificazione meno rigorosa rispetto ad OSI, si poneva come una tecnologia disponibile e funzionante.

Il modello TCP/IP propone quattro livelli per descrivere l'architettura di rete. Il primo livello, tuttavia, risulta quasi del tutto non specificato:

1. **Host-to-network:** Il livello Host-rete ha il compito di fornire il servizio di trasmissione ai livelli superiori; le funzionalità richieste corrispondono grossomodo a quelle dei primi due livelli del modello OSI.

2. **Internet:** Il livello internet è la base di tutto il modello: esso ha il compito di instradare in modo efficiente ed indipendente pacchetti di dati verso

la destinazione e di definire i percorsi che essi avranno, i pacchetti potranno giungere a destinazione anche in ordine diverso e sarà compito di qualche livello superiore riassemblare il messaggio originale in modo coerente. Possiamo quindi dire che il livello internet è assai simile al livello 3 (rete) del modello OSI.

3. **Transport:** Il livello trasporto permette a due entità di tenere il dialogo e si occupa di far giungere a destinazione i dati in maniera corretta, eventualmente frammentandoli prima di inviarli al livello internet e riassemblandoli poi alla destinazione. Anche il controllo di flusso viene effettuato a questo livello in modo che un mittente veloce non sovraccarichi un destinatario lento. Vengono inoltre definiti due protocolli di trasporto in questo livello: TCP orientato alla connessione e ricco di tutte le funzionalità descritte ed UDP protocollo privo di connessione, veloce e con funzionalità essenziali, che conferisce maggiore libertà al livello superiore nel controllo dell'affidabilità.

4. **Application:** A questo livello vi sono le funzioni per comunicazioni specifiche, non vi è riferimento al livello sessione o presentazione del modello OSI poiché essi non sono necessari in molte applicazioni. Si collocano qui i protocolli di alto livello per il trasferimento di file o per la posta elettronica.

Il modello TCP/IP non si basa quindi sul modello ISO/OSI, tuttavia vi sono molte similitudini fra le due architetture di rete ed in Figura 1.1 si nota come sia possibile individuare una corrispondenza uno-a-uno fra alcuni livelli. Nel seguito si manterrà questo paragone e si classificheranno i protocolli facendo riferimento al livello a cui appartengono nel modello di riferimento

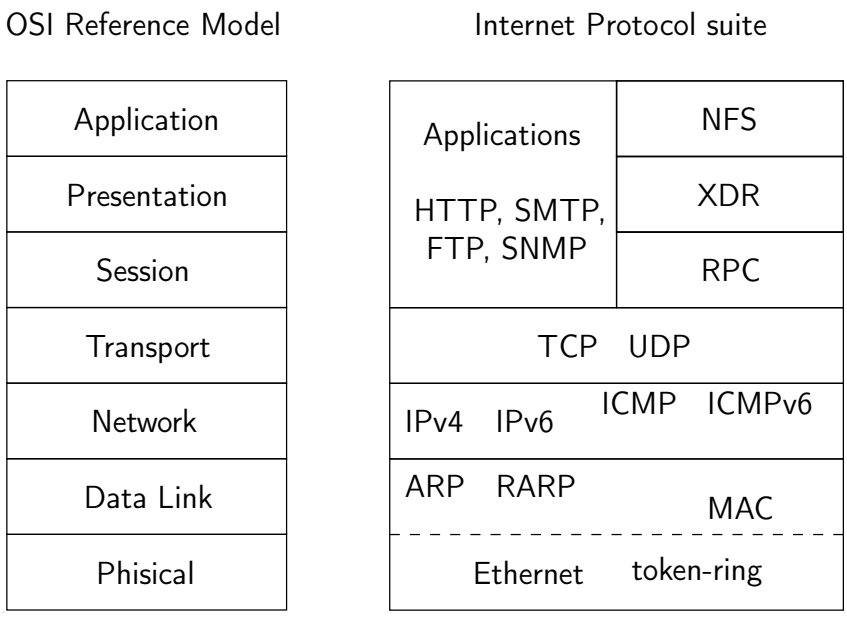

Figura 1.1: Confronto fra i modelli ISO/OSI e TCP/IP

ISO/OSI: TCP, per esempio, verrà visto come un protocollo di livello 4 (OSI) sebbene esso appartenga al livello 3 nel modello TCP/IP.

1.2 Perché IPv6

All'inizio degli anni 90 l'IETF si rese conto di come Internet stesse diventando sempre più un fenomeno sociale e si chiese in quale misura questi cambiamenti avrebbero avuto una ricaduta sugli aspetti tecnologici dell'internetworking. Si ritenne necessaria una evoluzione del protocollo IP che ponesse una soluzione alle problematiche venute alla luce dopo anni di utilizzo.

Vi furono molte proposte per l'estensione dello spazio di indirizzamento di IP e per l'alleggerimento del protocollo e si riconobbe che la nuova versione di IP, inizialmente chiamata *IPng*, avrebbe dovuto avere le seguenti caratteristiche di base (v. [Kramshøj, 2002]):

- Trasparenza negli instradamenti e gestione delle politiche di routing;

- Gestione di particolari flussi di comunicazione;
- Supporto alla qualità del servizio (QoS);
- Sicurezza dei dati.

Dopo lunghe discussioni e considerazioni si giunse alla specifica di un nuovo protocollo di livello 3 a cui venne dato il nome di IPv6 [Deering and Hinden, 1998]. Oggi ci si riferisce alla versione originale di IP con il nome IPv4.

Il motivo principale che ha spinto a tale evoluzione è stato l'inesorabile esaurimento di indirizzi IP ma si è data particolare attenzione anche ad altri aspetti legati alle reti di comunicazione: si pensi all'autenticità e confidenzialità dei dati.

1.3 Un nuovo schema di indirizzamento

Le specifiche originali di IP prevedevano la divisione dello spazio di indirizzamento in tre classi (A, B e C) per reti grandi, medie e piccole, ed una gestione selvaggia di queste classi agli albori di Internet ha contribuito a sollevare il problema dell'esaurimento dello spazio degli indirizzi, rendendo necessaria una soluzione a questo problema.

Una soluzione a breve termine fu trovata con l'introduzione di CIDR [Fuller et al., 1993], una tecnica per gestire lo spazio di indirizzamento in modo più razionale consentendo una aggregazione più efficiente degli indirizzi.

IPv6 utilizza l'idea nata con CIDR per l'aggregazione degli indirizzi, non vi sono infatti classi di indirizzi ne identificatori di rete in senso stretto, un indirizzo IPv6 è costituito unicamente da un prefisso e da un identificatore di interfaccia (e non più di nodo, come in IPv4). Maggiori dettagli sugli indirizzi IPv6 saranno forniti nei prossimi capitoli.

1.4 Modalità di indirizzamento

Abbiamo accennato in precedenza che IPv6 si propone come un protocollo di rete agile, efficiente e completo e fra qualche anno, quando le implementazioni raggiungeranno la dovuta maturità probabilmente lo vedremo sostituire del tutto l'attuale versione di IP, ma quali saranno i vantaggi immediati nell'adottare IPv6? Uno degli obiettivi principali del progetto è di ottenere la trasparenza nell'infrastruttura di routing: fare in modo che un pacchetto resti immutato nei suoi indirizzi sorgente e destinazione permette un maggior controllo negli instradamenti, ma anche una maggiore efficacia delle pratiche di *filtering* e *firewalling*, ciò si ottiene in termini pratici con l'abolizione delle tecniche di NAT le quali, con l'utilizzo di indirizzi privati, rendono oltremodo complessa la gestione di una rete; si noti inoltre che, in generale, una fase di traduzione degli indirizzi comporta anche dei ritardi nel recapito dell'informazione.

Schematizziamo i benefici che il nuovo IP apporterà al networking:

- La dimensione degli indirizzi è di 128 bit: una grandissima quantità di host (o interfacce) può essere indirizzata univocamente ed è inoltre possibile una maggiore gerarchizzazione degli instradamenti.

- Il preambolo è stato semplificato: ora il caso generale é favorito e le comunicazioni ed i pacchetti che fanno eccezione sono trattati con il meccanismo dei preamboli opzionali.

- Miglioramenti nel supporto per le opzioni: le funzionalità aggiuntive sono state spostate in preamboli opzionali, in questo modo si guadagna in flessibilità e ci si prepara a qualsiasi nuova estensione futura.

- Supporto per i "flussi" di traffico: è stata introdotta una etichetta che

identifica un flusso di dati come appartenente ad un certo tipo di dialogo e rende possibile sottoporlo ad una gestione opportuna.

1.5 Transizione da IPv4 a IPv6

Uno dei problemi principali dovuti al passaggio alla nuova generazione di IP è dovuto principalmente al porting del software di rete ed alla disponibilità di implementazioni stabili e mature dello stack IPv6: il software è la chiave di tutto in questa fase.

Il supporto da parte dei sistemi operativi è presente da tempo: i sistemi operativi UNIX della famiglia BSD forniscono una delle migliori implementazioni, Linux sta consolidando il supporto ad IPv6 e così stanno facendo Cisco IOS ed i recenti sistemi Microsoft (che tuttavia non hanno abilitato nativamente la gestione della nuova versione di IP), esistono inoltre implementazioni anche per i sistemi Win 9x. Il vero problema da affrontare è però quello delle applicazioni di rete che necessitano di essere convertite per essere usate con il nuovo protocollo, una soluzione sui sistemi POSIX è quella inizialmente proposta da KAME (`http://www.kame.net`) per la riscrittura di codice indipendente dalla famiglia di protocolli e poi proposta come metodologia standard di conversione [Shin et al., 2004].

In generale le applicazioni per le quali è disponibile il codice sorgente potranno essere convertite con maggiore facilità, ma resta il problema per i sistemi e le applicazioni proprietarie che non verranno convertite dagli sviluppatori originali. In questi casi si possono adottare diverse strategie, come l'uso di *Application Level Gateway* oppure di traduttori di livello 3 che incapsulino pacchetti v4 in pacchetti v6, tuttavia con le ripercussioni negative sulle prestazioni derivate da un tale approccio. Per far comunicare le nuove installazioni v6 con l'universo v4 è stata proposta una soluzione basata

su traduzione chiamata NAT-PT che si ritiene unicamente una soluzione temporanea e da adottare solo se non vi solo altre vie.

Altri metodi per la convivenza di diverse versioni del protocollo IP sono basate sul tunnelling e sull'incapsulamento di pacchetti IPv6 in pacchetti IPv4: vi possono essere tunnel stabiliti in modo automatico, oppure configurati manualmente.

Le principali metodologie di tunneling proposte per la transizione ad IPv6:

Tunnel Broker: L'idea del Tunnel Broker (tunnel 6in4) è quella di implementare una soluzione semplice di connessione IPv6 per i fornitori di servizio che forniscono già IPv4 ai loro clienti [Durand et al., 2001].

6to4: Il meccanismo di tunnelling "6to4" è discusso in [Carpenter and Moore, 2001], questa tecnica permette di avere traffico IPv6 su una rete IPv4 senza la necessità di instaurare un tunnel.

6over4: Il metodo "6over4" utilizza una tecnica per simulare una rete locale virtuale per IPv6 in una internetwork basata su IPv4 utilizzando indirizzi multicast [Carpenter and Jung, 1999].

Questi approcci devono essere adottati nel caso in cui non sia possibile avere una rete che utilizzi nativamente IPv6 o nel caso in cui non vi sia connettività IPv6 a livello globale.

1.6 Conclusioni

Si prevede che il passaggio ad IPv6 sarà incoraggiato maggiormente in quei paesi che sono in forte sviluppo tecnologico quali l'India, la Cina, il Giappone e buona parte dell'Asia, e che la conversione risulterà meno rapida nelle Americhe ed in Europa a causa della forte e consolidata presenza di IPv4 [Deering, 2002].

Tuttavia anche nel nostro continente si dovrebbe incoraggiare l'utilizzo di IPv6 per le nuove installazioni per non creare un ritardo tecnologico nei confronti dei paesi dell'estremo oriente che intendono sfruttare appieno le potenzialità del nuovo protocollo. In molte situazioni il passaggio ad IPv6 potrà essere particolamente delicato, ma qualche scelta coraggiosa potrebbe rivelarsi vincente.

IPv6 è stato inoltre scelto come protocollo di rete per la telefonia mobile di nuova generazione *3GPP* grazie alle semplificazioni che esso introduce anche nella gestione del roaming di host mobili e nel prossimo futuro l'utilizzo di terminali mobili avrà una diffusione senza precedenti.

Insomma, IPv6 è pronto per essere usato,
ma noi siamo pronti ad usarlo ?

Bibliografia

B. Carpenter and C. Jung. *Transmission of IPv6 over IPv4 Domains without Explicit Tunnels*, March 1999. URL `ftp://ftp.isi.edu/in-notes/rfc2529.txt`. RFC 2529.

B. Carpenter and K. Moore. *Connection of IPv6 Domains via IPv4 Clouds*, February 2001. URL `ftp://ftp.isi.edu/in-notes/rfc3056.txt`. RFC 3056.

S. Deering and R. Hinden. *Internet Protocol, Version 6 (IPv6) Specification*, December 1998. URL `ftp://ftp.isi.edu/in-notes/rfc2460.txt`. RFC 2460.

Steve Deering. IPv6 Overview and Status Report. April 2002. URL `http://www.sixxs.net/archive/docs/`.

A. Durand, P. Fasano, I. Guardini, and D. Lento. *IPv6 Tunnel Broker*, January 2001. URL `ftp://ftp.isi.edu/in-notes/rfc3053.txt`. RFC 3053.

V. Fuller, T. Li, J. Yu, and K. Varadhan. *Classless Inter-Domain Routing (CIDR): an Address Assignment and Aggregation Strategy*, September 1993. URL `ftp://ftp.isi.edu/in-notes/rfc1519.txt`. RFC 1519.

Henrik Lund Kramshøj. Designing Internetworks with IPv6, 2002. URL `http://www.inet6.dk/thesis/`.

Myung-Ki Shin, Yong-Guen Hong, Jun ichiro itojun HAGINO, Pekka Savola, and Eva M. Castro. *Application Aspects of IPv6 Transition*, March 2004. URL `http://www.ietf.org/internet-drafts/`.

Andrew S. Tanenbaum. *Computer Networks*. Prentice Hall International, 1996.

Capitolo 2

Una panoramica su IP

In questo capitolo verranno presentati gli aspetti salienti del networking come lo si intende oggi e si forniranno le nozioni necessarie e comprendere lo scopo di IPv6 ed i suoi meccanismi di funzionamento.

2.1 Terminologia

è bene ribadire la nostra definizione di "rete di calcolatori" fornita nel capitolo 1 prima di analizzare le caratteristiche specifiche delle reti basate su IP:

> *Una rete di calcolatori si pone come primo obiettivo il trasferimento di informazione fra varie entità autonome, le informazioni viaggiano attraverso un certo mezzo trasmissivo da una sorgente ad una destinazione (o più destinazioni) attraversando eventualmente altre entità e vengono opportunamente codificate, trasmesse e reinterpretate per rendere effettiva la comunicazione.*

Poniamo l'accento sul fatto che i nodi di una rete sono entità autonome, intendendo con ciò che tali entità hanno un sottosistema dedicato alla gestione della comunicazione: un sistema con un elaboratore centrale e dei terminali "stupidi" che fanno capo ad esso non è una rete di calcolatori secondo questa accezione.

L'informazione viene trasferita in pacchetti, detti anche *datagram* formati da due parti fondamentali: il preambolo (o intestazione, o *header*) ed il corpo del messaggio (in inglese *payload*). Nel preambolo sono presenti delle informazioni "di controllo" che permettono di consegnare il pacchetto di dati alla destinazione, o di effettuare verifiche sulla correttezza di ciò che si sta trasmettendo.

Il messaggio scambiato dai membri della rete attraversa vari livelli del sottosistema di rete prima di giungere a destinazione e, come si vede in Figura 2.1, ogni livello aggiunge il suo preambolo.

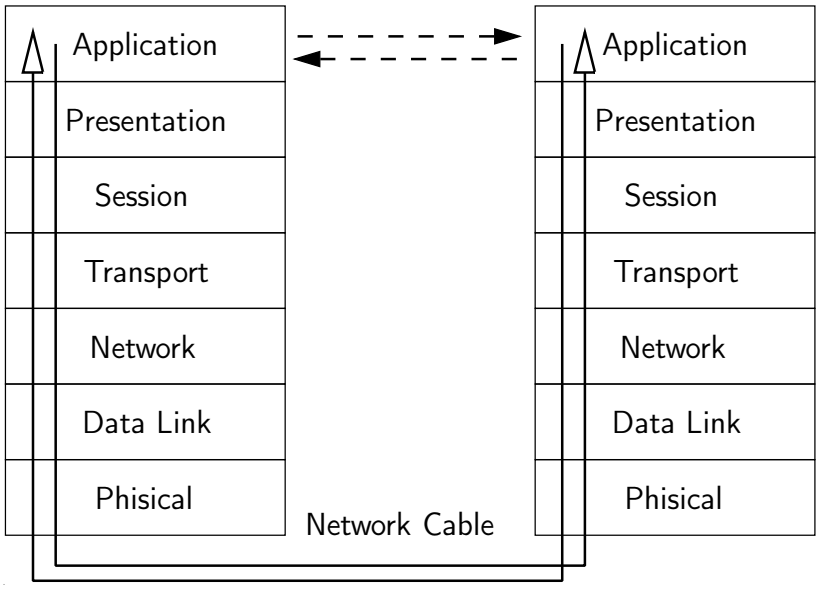

Figura 2.1: Il processo di comunicazione

Questo processo dell'aggiunta di informazioni di controllo viene chiamato anche imbustamento (vedi Figura 2.2) ed é grazie ad esso che la comunicazione può essere vista logicamente come condotta fra moduli di pari livello.

I pacchetti possono essere vincolati a seguire tutti lo stesso percorso fra sorgente e destinazione oppure avere una sorte indipendente l'uno dall'altro,

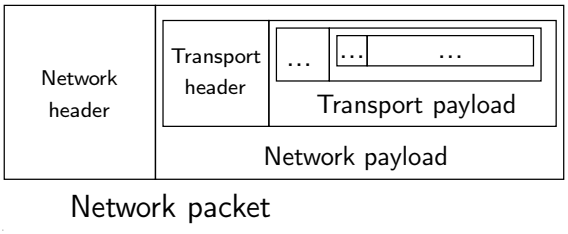

Figura 2.2: L'imbustamento

nel primo caso si potrà parlare di "circuito virtuale", nel secondo caso si avrà una rete a commutazione di datagram. Il primo approccio è utilizzato nei casi in cui si desidera recapitare l'informazione in maniera rapida corretta ed in sequenza, l'idea dei circuiti virtuali si rifà in qualche modo a quella delle reti a commutazione di circuito (si pensi al vecchio sistema telefonico), l'utilizzo della commutazione di datagram fornisce tuttavia una maggiore tolleranza ai guasti lungo le linee di comunicazione ed un utilizzo più efficiente del mezzo trasmissivo da parte dei nodi di una rete.

Si indica con il termine "nodo" una qualsiasi entità appartenente alla rete e si distinguono due tipi fondamentali di nodi:

host: Un *host* è un nodo connesso ad una rete che ha un proprio indirizzo associato ad una interfaccia di rete.

router: Un *router* è una entità con più di una interfaccia di rete, esso si occupa di instradare il traffico fra reti diverse permettendo ciò che noi chiamiamo *internetworking*. Un router è capace di inoltrare i pacchetti di dati da una interfaccia di rete all'altra.

Per capire ancora meglio il ruolo di tali entità vale la pena di addentrarci maggiormente nei dettagli di una architettura di rete.

2.2 Architettura di una rete

Le reti possono essere banalmente raggruppate in categorie a seconda della loro estensione geografica, ma la distinzione logica che possiamo operare per i nostri scopi è basata sulla continuità del mezzo trasmissivo.

Diremo che i nodi sono in rete locale (LAN) se essi sono in grado di condividere il mezzo trasmissivo e se è possibile un collegamento fra essi al livello 2 OSI (link-layer).

Una rete può essere strutturata fisicamente in diversi modi, come si vede in Figura 2.3, una topologia ad anello o a bus può risultare piuttosto economica, una a maglia piena può essere più performante e tollerante ai guasti, ma una disposizione topologica a stella gerarchica riesce solitamente ad essere un buon compromesso fra la semplicità delle prime e la ridondanza della configurazione a maglia.

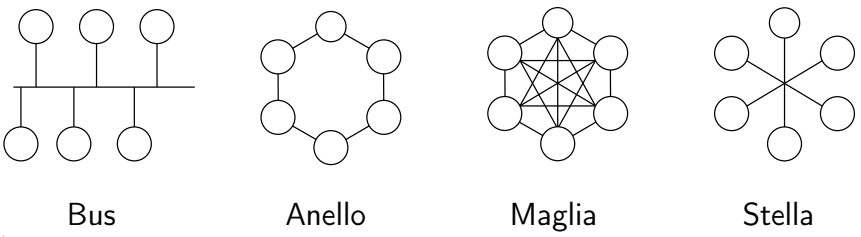

Figura 2.3: Differenti topologie di reti

Nodi di LAN diverse potranno scambiarsi pacchetti di informazione attraverso una internetwork: i pacchetti attraverseranno dei router per raggiungere l'host (e la LAN) di destinazione grazie al suo indirizzo di livello 3 (detto anche *network address*).

In Figura 2.4 si vede in modo schematico quale sia il ruolo di un router in una internetwork.

Operiamo inoltre una distinzione fra una generica internet (con la i mi-

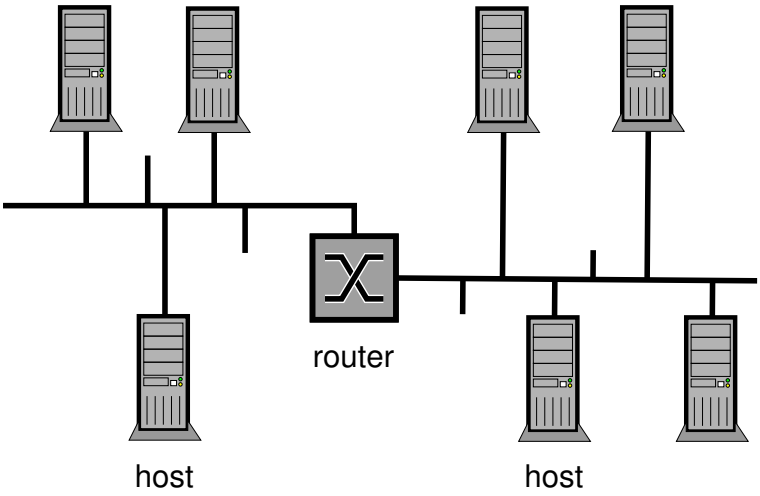

Figura 2.4: Interconnessione fra reti

nuscola) e l'Internet mondiale (I maiuscola), la più estesa rete pubblica di comunicazioni attuale.

Si tenga presente che le comunicazioni fra membri di una rete avvengono effettivamente fra interfacce di rete, e quindi al livello 2, ma affinché il meccanismo dell'interconnessione di reti diverese funzioni, anche gli host appartenenti alla stessa LAN devono necessariamente comunicare utilizzando indirizzi di rete di livello 3 ed è quindi necessario un meccanismo per tradurre gli indirizzi link-layer (solitamente associati alle interfacce hardware) in indirizzi di rete.

Nel caso di IPv4 tale traduzione è compito del protocollo ARP che sfrutta la condivisione del mezzo trasmissivo degli host in una LAN per diffondere messaggi *broadcast*, che cioè siano ricevuti da tutti i membri della LAN. Il protocollo ARP effettua la traduzione di indirizzi di rete in indirizzi link-layer eseguendo un algoritmo di questo tipo:

- Il nodo sorgente invia un messaggio broadcast chiedendo: "Chi ha l'indirizzo di rete x?"(*ARP-request*).

- Il nodo che ha tale indirizzo di rete risponde comunicando il proprio indirizzo link-layer (*ARP-reply*).
- A questo punto la comunicazione fra sorgente e destinazione sulla LAN risulta possibile al livello 2.

L'uso di ARP fa in modo che le applicazioni di rete vedano la comunicazione solo fra nodi di una rete identificati da indirizzi di livello 3.

2.3 Il Preambolo IP

Il livello internet del modello TCP/IP definisce un formato standard per i pacchetti e prevede un protocollo chiamato IP, che analizzeremo in dettaglio. IP ha il compito di indirizzare i membri della rete e di stabilire quale debba essere il percorso che un pacchetto deve compiere per giungere a destinazione, esso si preoccupa quindi del singolo pacchetto ed in generale non ha memoria di un intero dialogo fra nodi.

Un pacchetto IP è composto da un preambolo e da un corpo, ed i bit che lo compongono vengono trasmessi considerando il bit più a sinistra come il più significativo[1], la struttura del preambolo è illustrata in Figura 2.5, si veda anche [Postel, 1981].

Il preambolo presenta una parte fissa di 20 byte ed una parte di lunghezza variabile per le opzioni, descriveremo prima la parte fissa e poi faremo alcune considerazioni sulle opzioni, sul loro utilizzo e sulla loro flessibilità.

Il campo *Version* indica la versione del protocollo che si utilizza nel datagram, questa informazione rende possibile una fase di transizione a nuove

[1]ordine *big-endian* (usato ad esempio sulle architetture RISC); al contrario dell'ordine *little-endian*(usato sulle architetture x86) che considera il bit più a destra come più significativo e per il quale é necessaria una conversione "host-to-network order", prima della trasmissione con IP.

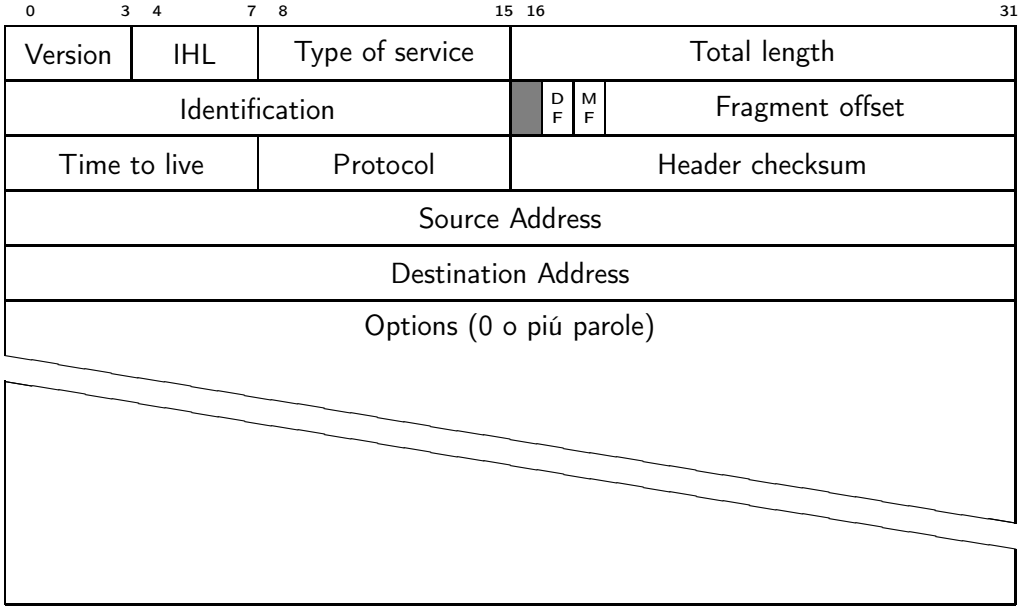

Figura 2.5: Preambolo IP

versioni del protocollo mantenendo temporaneamente la compatibilità con la versione precedente, quella attualmente in uso è la 4.

Il campo *IHL* (IP Header Length) specifica la lunghezza del preambolo in parole da 32 bit, considerando anche le opzioni.

Con il valore di *Type of Service* gli host sono in grado di richiedere alla rete il tipo di servizio del quale necessitano: i primi tre bit da sinistra di questo campo indicano una priorità, i secondi tre specificano alcuni parametri per specificare a cosa l'host sia interessato fra ritardo, capacità di trasmissione, affidabilità, infine gli ultimi due bit sono inutilizzati.

Il campo *Total Length* indica la lunghezza totale del pacchetto includendo sia il preambolo che i dati, con 16 bit possiamo quindi avere datagram di dimensione non superiore 65.536 byte.

Il campo *Identification* serve al destinatario per ricostruire un datagram frammentato, ogni frammento di un dato datagram presenterà lo stesso valore per questo campo.

Vi sono poi un bit inutilizzato e due campi da 1 bit *DF* (Don't Fragment) e *MF* (More Fragment): il primo campo indica ai router che incontrano quel pacchetto di non frammentare il datagram poiché il destinatario potrebbe non essere capace di riassemblarlo, il secondo ha valore 1 per tutti i frammenti di un datagram meno l'ultimo e serve a sapere quando sono arrivati tutti i frammenti di un pacchetto.

Il campo *Fragment Offset* specifica in quale posizione del datagram identificato dal campo *Identification* si trova questo frammento, tutti i frammenti meno l'ultimo devono essere multipli di 8 byte.

Il campo *Time to live* è stato pensato per limitare il tempo di vita (espresso in secondi) di un pacchetto, ma viene comunemente usato come contatore di salti, decrementato ad ogni *hop* che il pacchetto compie nella sua traversata verso l'host destinazione, quando il valore raggiunge lo 0 il pacchetto viene scartato.

Il campo *Protocol* serve per indicare a quale protocollo di trasporto (TCP, UDP o altro) deve essere consegnato un certo datagram dopo essere stato riassemblato.

Il valore di *Header checksum* costituisce una somma di controllo per la verifica dell'integrità del preambolo e deve essere ricalcolato ad ogni variazione di un qualsiasi altro campo.

I campi *Source address* e *Destination address* contengono l'identificatore di rete e di host rispettivamente del mittente e della destinazione: gli indirizzi IP. I 32 bit dell'indirizzo vengono spesso scritti raggruppando i bit in byte (8 bit) ed esprimendo il loro valore in decimale separandoli con un punto per rendere la scrittura di indirizzi più comprensibile agli esseri umani:

$$(\underbrace{11000000}_{8 \text{ bit}})_2.(10101000)_2.(00000000)_2.(00001010)_2 = 192.168.0.10$$

Vi sono inoltre alcuni campi opzionali nel preambolo IP, che rendono quindi la sua lunghezza variabile. Le opzioni vengono gestite in questo modo: un codice di 1 byte identifica l'opzione, per alcune di esse segue un campo di 1 byte che ne comunica la lunghezza e di seguito vi sono i restanti byte di dati.

Il campo *Options* viene completato a multipli di 4 byte e le possibili scelte disponibili sono illustrate in Tabella 2.1.

Opzione	Descrizione
Security	Specifica il livello di segretezza
Strict source routing	Fornisce il percorso completo da seguire
Loose source routing	Fornisce una lista di router che non devono essere saltati
Record route	Forza ogni router ad aggiungere il proprio indirizzo IP
Timestamp	Forza ogni router ad aggiungere il proprio indirizzo IP ed una etichetta temporale

Tabella 2.1: Opzioni nel preambolo IP

Molte di queste opzioni vengono utilizzate solo per motivi di test, altre servono per stabilire o suggerire il percorso che un pacchetto dovrà seguire; il meccanismo per l'utilizzo di queste "funzionalità aggiuntive" di IP non è tuttavia fra i più flessibili: la variabilità della dimensione del preambolo costituisce il principale effetto collaterale di questo approccio alla gestione delle opzioni, si vedrà fra breve come l'evoluzione del protocollo IP (IPv6) ha cercato fra le altre cose di porre una soluzione al problema dell'estendibilità del protocollo di rete.

2.4 Indirizzi e nomi

Poiché risulta scomodo per gli esseri umani (e per i programmi scritti da umani) trattare gli indirizzi numerici IP, una pratica comune é quella di assegnare ad ogni indirizzo un "nome" (o più nomi) ed avere un meccanismo

di risoluzione degli indirizzi IP. Una soluzione semplice per pochi nodi, adottata su ARPANET, è quella di avere un file di testo (chiamato ad esempio *hosts*) nel quale si mantenga una corrispondenza nome-indirizzo; un tale metodo risulta tuttavia impraticabile se si desidera un servizio di traduzione rapido, ma anche tollerante ai cambiamenti di nome o indirizzo di un host e soprattutto che possa tenere conto delle gerarchie di rete per velocizzare la traduzione. Il servizio inventato per soddisfare queste richieste si chiama DNS definito in [Mockapetris, 1987a] e [Mockapetris, 1987b], che usa uno schema di denominazione gerarchico e ripartito in domini implementato mediante una base di dati distribuita; esso può associare indirizzi IP a nomi di host, ma può essere utilizzato anche per effettuare una risoluzione inversa (dall'indirizzo al nome).

Il DNS partiziona concettualmente Internet in domini di nomi ognuno con una propria sotto-gerarchia e questa rappresentazione "ad albero" (Figura 2.6) dello spazio dei nomi garantisce la consistenza delle traduzioni nome-indirizzo minimizzando lo sforzo nell'aggiornare eventuali cambiamenti. Diversi aggiornamenti al servizio DNS sono stati proposti in [Thomson et al.,

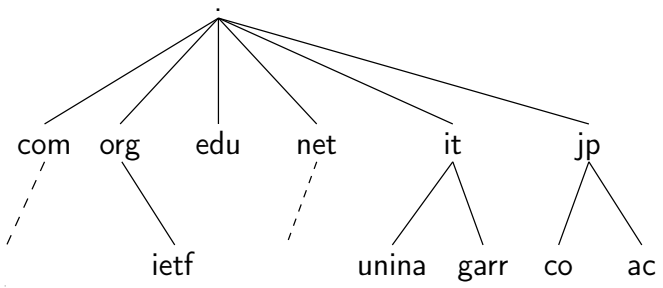

Figura 2.6: Spazio dei nomi nel DNS

2003] e [Bush et al., 2002] che specificano come mantenere dei record di traduzione per indirizzi IPv6 (record AAAA.

Il DNS è un servizio vitale per molte attività in Internet ma lo diventerà

in misura ancora maggiore con la diffusione di IPv6, a causa della maggiore lunghezza degli indirizzi v6, sia per per agevolare l'uso dei meccanismi di autoconfigurazione degli indirizzi IPv6 che per affrontare con meno problemi il *renumbering*[2] di una rete.

2.5 Router ed internetworking

Abbiamo accennato in precedenza quale debba essere il ruolo di un router in una infrastrutture di rete, proseguiamo con una panoramica su come avvengono gli instradamenti e sull'importanza della loro efficienza.

Una grande rete può avere molti router interconnessi tra loro e vi possono essere molti percorsi possibili fra la sorgente di un messaggio e la destinazione, una suddivisione gerarchica degli instradamenti contribuisce dunque a semplificare la struttura della rete e rende possibile operare in maniera più efficiente. In Figura 2.7 si presenta uno schema tipico del sistema di routing in una rete medio-grande, questa organizzazione permette inoltre di affrontare con minori problemi l'espansione di una rete con il conseguente aumento di traffico.

Ogni router conserva delle strutture dati che vengono usate per stabilire il miglior percorso per i pacchetti che vi transitano: un esempio può aiutarci a comprendere meglio come avviene il processo di instradamento. Si ipotizzi la situazione illustrata in Figura 2.8: in questo semplice caso l'host **A** può raggiungere l'host **B** compiendo due percorsi: percorrendo la linea1 a bassa velocità oppure attraversando una non meglio specificata internet ed adoperando le linee veloci linea2 e linea3; sarà compito del router sulla rete dell'host A decidere quale tragitto debbano compiere i pacchetti destinati

[2]Si indica con il termine *renumbering* la procedura di assegnazione di nuovi indirizzi agli host di una certa rete. Una tale rinumerazione può rivelarsi necessaria, ad esempio, nel caso di un cambio di provider per l'accesso ad Internet.

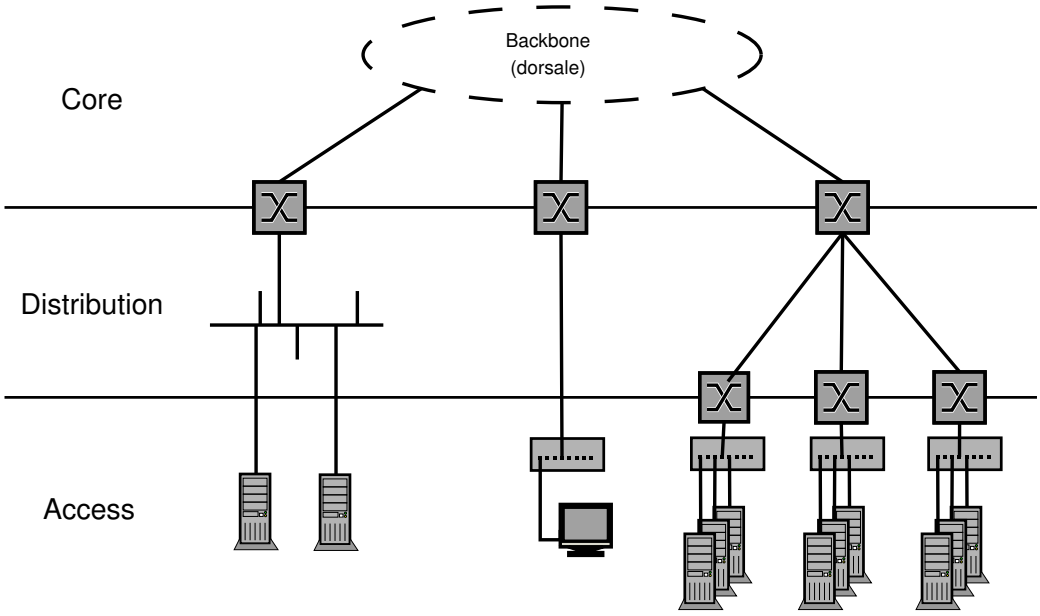

Figura 2.7: Organizzazione tipica di una rete di medie dimensioni

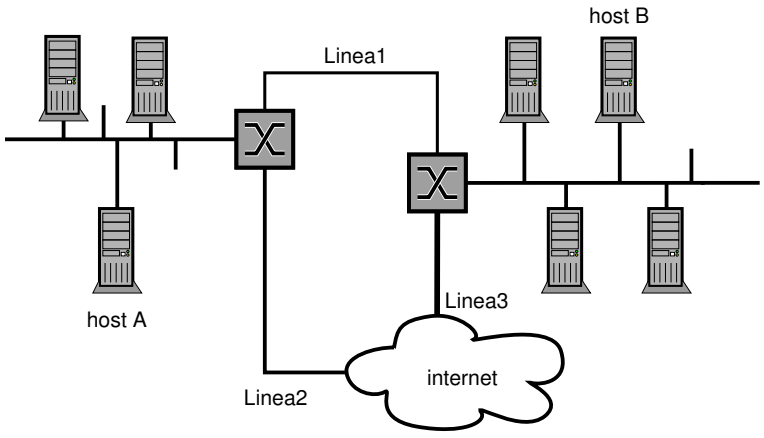

Figura 2.8: Esempio sul routing

all'host B e tale decisione potrà essere presa considerando diversi parametri per la valutazione della bontà del percorso. Le caratteristiche che rendono un percorso preferito possono essere la banda disponibile, i tempi di latenza, il numero di salti, ma anche la confidenzialità dei dati può influire sulle scelte di routing: si potrebbe desiderare che un messaggio molto importante e riservato, ma non estremamente urgente, non passi per la internet ma raggiunga "direttamente" la destinazione, pur compiendo un percorso subottimale dal punto di vista della velocità di consegna.

2.6 La tabella di instradamento

La tabella di instradamento è dunque il cuore del meccanismo di routing, essa può essere schematizzata come una associazione fra una destinazione ed un *link* a questa destinazione, data dalla bontà di tale collegamento espressa secondo una certa misura. A seconda delle tecniche di routing utilizzate, la misura di bontà può coinvolgere diversi parametri, fisici o politici ed è importante notare che il dialogo fra router è fondamentale per gestire gli instradamenti in una rete di dimensioni medie o grandi. Molti algoritmi e protocolli di routing sono stati sviluppati per le reti basate su IP e molti di essi sono stati aggiornati per poter trattare indirizzi IPv6. Si noti che qualsiasi nodo che utilizzi IP ha una tabella di routing, negli host le associazioni fra rete ed interfaccia sono generalmente statiche, ma nei router esse possono cambiare dinamicamente. In Figura 2.9 si vede un esempio di una tabella di un router.

```
Kernel IP routing table
Destination   Gateway       Genmask         Flags Metric Ref  Use Iface
192.168.23.0  *             255.255.255.0   U     0      0    0   eth0
192.168.22.0  192.168.13.1  255.255.255.0   UG    2      0    0   eth0.2
192.168.21.0  192.168.12.1  255.255.255.0   UG    2      0    0   eth0.1
192.168.13.0  *             255.255.255.0   U     0      0    0   eth0.2
192.168.12.0  *             255.255.255.0   U     0      0    0   eth0.1
192.168.11.0  192.168.12.1  255.255.255.0   UG    2      0    0   eth0.1
default       192.168.23.2  0.0.0.0         UG    0      0    0   eth0
```

Figura 2.9: Un tabella di routing mostrata dal comando `route -n` su sistemi GNU/Linux

Bibliografia

R. Bush, A. Durand, B. Fink, O. Gudmundsson, and T. Hain. *Representing Internet Protocol version 6 (IPv6) Addresses in the Domain Name System (DNS)*, August 2002. URL `ftp://ftp.isi.edu/in-notes/rfc3363.txt`. RFC 3363.

P.V. Mockapetris. *Domain names - concepts and facilities*, November 1987a. URL `ftp://ftp.isi.edu/in-notes/rfc1034.txt`. RFC 1034.

P.V. Mockapetris. *Domain names - implementation and specification*, November 1987b. URL `ftp://ftp.isi.edu/in-notes/rfc1035.txt`. RFC 1035.

J. Postel. *Internet Protocol*, September 1981. URL `ftp://ftp.isi.edu/in-notes/rfc791.txt`. RFC 791.

S. Thomson, C. Huitema, V. Ksinant, and M. Souissi. *DNS Extensions to Support IP Version 6*, October 2003. URL `ftp://ftp.isi.edu/in-notes/rfc3596.txt`. RFC 3596.

Capitolo 3

Dettagli del protocollo

Esaminando a fondo le caratteristiche del protocollo IPv6 ed i suoi meccanismi di funzionamento potremo trarre alcune conclusioni sui benefici che esso apporterà alle tecniche di networking e si potranno ipotizzare degli scenari nei quali la transizione risulterà più o meno necessaria ed urgente.

3.1 Preamboli IPv6

3.1.1 Preambolo standard

Alcune delle novità introdotte da IPv6 sono lampanti anche solo osservando la struttura del preambolo in Figura 3.1, si veda anche [Deering and Hinden, 1998]. La prima cosa da notare è sicuramente la dimensione degli indirizzi, il preambolo ha inoltre una lunghezza fissa di 40 byte e si gestiscono quindi le funzionalità opzionali in maniera diversa da IPv4 (cosí chiameremo la versione precedente di IP).

Il campo *Version* è sempre presente ed indica la versione del protocollo, la sua presenza è di cruciale importanza nel periodo di transizione da una versione ad un'altra poiché i router (e gli host) multiprotocollo dovranno analizzare il contenuto di questo campo per stabilire a quale sottosistema di rete passare la gestione di un pacchetto IP.

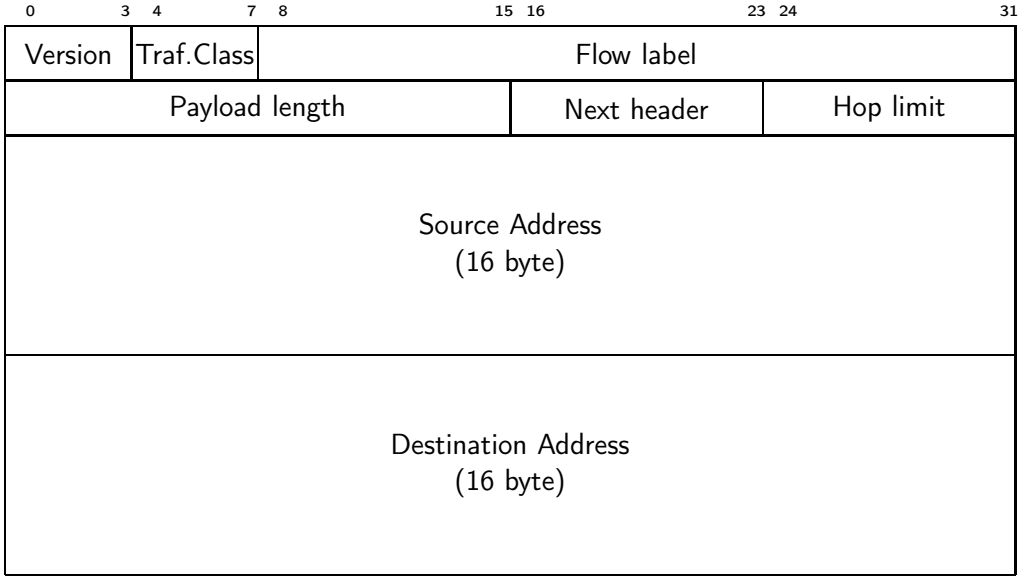

Figura 3.1: Preambolo IPv6 standard

Il campo *Traffic Class* ha lo scopo di distinguere a quale tipo di traffico appartiene un pacchetto senza utilizzare meccanismi espliciti di gestione dei flussi, questa funzionalità può essere usata per la trasmissione di dati multimediali per i quali il tempo di risposta può essere una caratteristica di fondamentale importanza. Il campo è a 4 bit e valori da 0 a 7 sono per i pacchetti che possono essere scartati e ritrasmessi in caso di congestioni, valori fra 8 e 15 vengono usati per i dati per cui la trasmissione deve essere costante anche in caso di perdita di pacchetti.

Il campo *Flow label* assegna al pacchetto una "etichetta di flusso" ovvero un codice che identifica un certo processo di comunicazione che potrebbe avere caratteristiche particolari. Un flusso è infatti una sequenza di pacchetti spediti da una particolare sorgente ad una particolare destinazione (unicast, anycast o multicast, come si descriverà in seguito) che il nodo sorgente desidera etichettare in un certo modo. Un flusso potrebbe consistere di tutti i pacchetti in una specifica connessione di trasporto, ma questa associazione

non è necessariamente sempre uno-a-uno. Tradizionalmente un flusso di dati è individuato dalla quintupla

$$\{Source\ address, Destination\ address, Source\ port, Destination\ port, Transport\ protocol\ type\}$$

ma alcuni di questi elementi potrebbero non essere disponibili per qualche motivo (frammentazione o crittazione) e quindi, per garantire una gestione semplice ed efficiente dei flussi, in IPv6 si è scelto di classificare un flusso mediante la terna

$$\{Source\ address, Destination\ address, Flow\ label\}$$

Il livello minimo di supporto nella gestione dei flussi in IPv6 consiste nella capacità di etichettare i pacchetti con una *label* non nulla: i nodi sorgente capaci di gestire i flussi devono essere almeno capaci di etichettare i flussi conosciuti (connessioni TCP oppure flussi di applicazioni multimediali). Il campo *Flow label* ha una dimensione di 20 bit ed un valore nullo per questo campo indica che il pacchetto non appartiene ad alcun flusso. Una entità classificatrice di pacchetti utilizzerà la tripla suddetta per trattare opportunamente una specifica sequenza di pacchetti come un unico flusso di dati: per fare ciò è necessario che il valore dell'etichetta di flusso resti immutato durante il tragitto fra sorgente e destinazione. Se un nodo IPv6 non ha alcuna capacità di trattare la *Flow label* dovrà: porre il valore a 0 quando origina un nuovo pacchetto, lasciarne inalterato il contenuto quando inoltra un pacchetto e dovrà semplicemente ignorarne il valore senza apportarvi modifiche quando riceve un pacchetto; si ricordi tuttavia che, per ottenere il trattamento desiderato del flusso, tutti i nodi lungo un percorso dalla sorgente e destinazione devono supportare la gestione dello stato del flusso. In

presenza delle estensioni per la sicurezza (IPsec, si veda pag. 62) l'etichetta di flusso non dovrà essere coinvolta nelle pratiche di crittazione, in modo da non cambiare valore e non influire nella fase di verifica dell'intergrità dei dati.

Il campo *Payload length* indica quanti byte di dati "utili" seguono il preambolo di 40 byte.

Il campo *Next header* pone una soluzione alla gestione delle funzionalità aggiuntive: in IPv6 si possono avere uno o più preamboli di estensione a quello standard e questo campo specifica quale sia la successiva estensione (vedi Tabella 3.1); nel caso non vi siano altri preamboli di estensione, il valore

Valore Esadecimale	Valore Decimale	Protocollo/Estensione
00	0	Hop-By-Hop Options ("Reserved" in IPv4)
01	1	ICMPv4
02	2	IGMPv4
04	4	IP in IP Encapsulation
06	6	TCP
08	8	EGP
11	17	UDP
29	41	IPv6
2B	43	Routing Extension Header
2C	44	Fragmentation Extension Header
2E	46	Resource Reservation Protocol (RSVP)
32	50	Encapsulating Security Payload (ESP) Extension Header
33	51	Authentication Header (AH) Extension Header
3A	58	ICMPv6
3B	59	No Next Header
3C	60	Destination Options Extension Header

Tabella 3.1: Valori del campo *Next header*

di *Next header* dà informazioni su quale protocollo di trasporto debba gestire il pacchetto.

Il campo *Hop limit* ha lo scopo di evitare che i pacchetti perdurino nella rete indefinitamente, esso viene decrementato ad ogni salto (solitamente al passaggio per un router) e quando raggiunge il valore 0 il pacchetto viene scartato.

I campi *Source address* e *Destination address* indicano gli indirizzi del mittente e del destinatario di un pacchetto.

Fra gli elementi di novità proposti da IPv6 vi è inoltre un meccanismo semplice ed efficace per l'autoconfigurazione degli host appartenenti ad una rete v6, maggiori dettagli nella sezione 3.4.

3.1.2 Preamboli di estensione

Come accennato in precedenza la gestione delle opzioni nel nuovo protocollo utilizza dei preamboli di estensione la cui presenza é segnalata dal valore del campo *Next Header*. Ogni preambolo di estensione ha inoltre il suo campo *Next Header* in modo da poter utilizzare più opzioni per ogni pacchetto, che andranno fra il preambolo principale ed il preambolo del protocollo di livello superiore. Le opzioni definite al momento sono descritte in Tabella 3.2, alcune di esse prevedono un preambolo di lunghezza fissa, altre un preambolo di lunghezza variabile.

Un preambolo di estensione segue il preambolo precedente nel pacchetto e viene sempre prima della parte di dati del pacchetto (nella quale é incluso il premabolo di trasporto), ogni pacchetto può presentare zero, una o molte estensioni; le estensioni devono essere processate dall'host destinazione nell'ordine in cui sono state inviate e nel caso in cui vi siano più opzioni si raccomanda che esse abbiano questo ordine:

- Preambolo IPv6
- Hop-by-Hop Options

Estensione	Descrizione
IPv6	Preambolo standard
Hop-by-Hop Options	Informazioni opzionali per ogni router lungo il percorso del pacchetto
Routing	Elenco di uno o più nodi intermedi da visitare lungo il percorso verso la destinazione
Fragment	Utilizzo esplicito di frammentazione da parte degli host se il canale trasmissivo lo richiede
Destination Options	Specifica delle informazioni che devono essere esaminate solo dal destinatario
Authentication	Informazioni sull'identità del mittente e sull'autenticità del pacchetto
Encapsulating Security	Opzioni per la riservatezza mediante crittografia

Tabella 3.2: Preamboli di estensione di IPv6

- Destination Options[1]
- Routing
- Fragment
- Authentication [2]
- Encapsulating Security
- Destination Options[3]
- preambolo livello-superiore

I router per i quali passa un pacchetto con preamboli di estensione non devono esaminare questi preamboli tranne nel caso in cui vi sia una estensione *Hop-by-hop*: in questo caso tutti i nodi per cui transita il pacchetto ispezionano

[1]per le opzioni alla prima destinazione contenuta nel campo *Destination Address* e tutte le destinazioni elencate nel preambolo di Routing

[2]Per maggiori informazioni si veda [Kent and Atkinson, 1998b]

[3]Per opzioni riguardanti unicamente l'ultima destinazione del pacchetto

il preambolo in questione ed effettuano le azioni opportune. Questo preambolo *Hop-by-hop*, quando presente, deve sempre seguire immediatamente il preambolo standard.

Ogni preambolo di estensione dovrebbe apparire al massimo una sola volta in un pacchetto, fatta eccezione per *Destination Options* che dovrebbe essere presente al massimo due volte: una prima del preambolo per il Routing ed una prima del preambolo per il livello superiore.

I preamboli *Hop-by-hop* e *Destination Options* possono contenere un numero variabile di opzioni codificate con la terna TLV (Tipo, Lunghezza, Valore) e quindi la lunghezza totale del preambolo deve essere indicata esplicitamente: in figura 3.2 si vede che il campo *Next Header* è il primo campo nei preamboli di estensione seguito dal campo *Hdr Ext Len* (*Header Extension length*: Lunghezza del preambolo di estensione) nel caso di preamboli a lunghezza variabile.

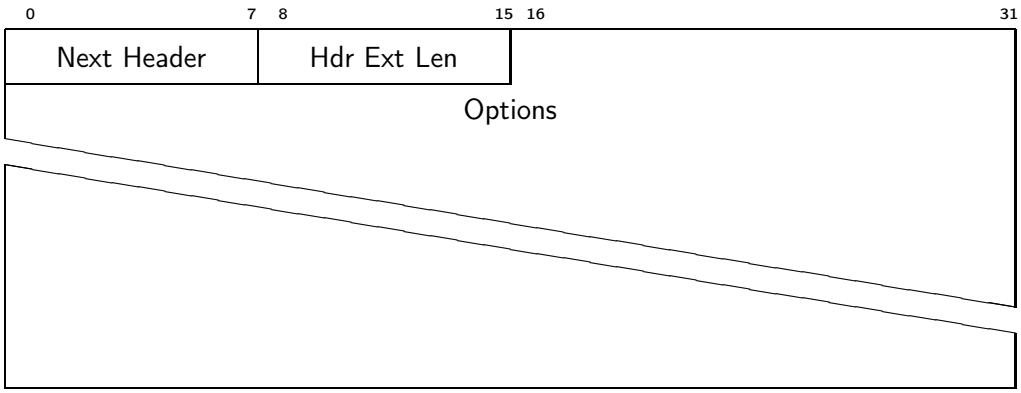

Figura 3.2: Preambolo di estensione Hop-by-Hop

Un utilizzo assai interessante del preambolo *Hop-by-hop* é quello di definire dei *Jumbograms* [Borman et al., 1999]. Il campo *Payload Length* in IPv6 è di soli 16 bit e la dimensione massima di un pacchetto è dunque di 65.536 byte; mediante una opzione è possibile ottenere un campo lunghezza

da 32 bit ed avere quindi pacchetti di dimensioni molto maggiori (fra 65.537 e 4.294.967.296 byte). La struttura di tale opzione è mostrata in Figura 3.3.

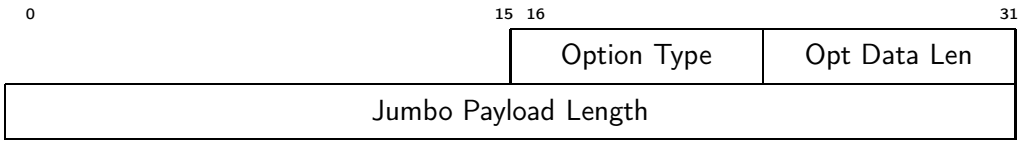

Figura 3.3: Preambolo per Jumbo Payload

Tale opzione risulta utile solo nel caso in cui il collegamento fisico fra sorgente e destinazione abbia un MTU sufficientemente grande, e potrebbe avere la sua applicazione in contesti di supercalcolo con reti dedicate alla distribuzione della computazione fra più processori indipendenti.

La struttura del preambolo per le opzioni di Routing è illustrata in Figura 3.4, questo preambolo è utilizzato per indicare una serie di nodi da visitare lungo il percorso di un pacchetto verso la sua destinazione.

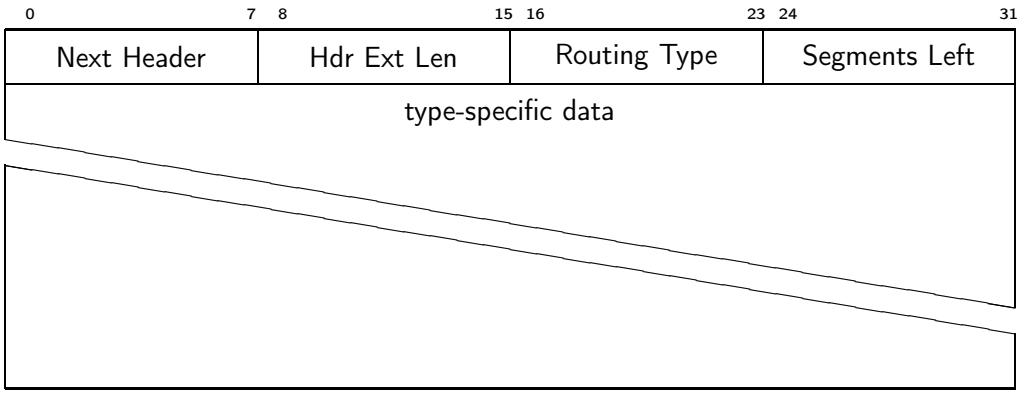

Figura 3.4: Preambolo di estensione per il routing

I primi due campi sono gli stessi del preambolo *Hop-by-Hop*, vi è quindi il campo *Routing Type* che identifica una particolare variante del preambolo per il routing, il campo *Segments Left* che segnala il numero dei nodi intermedi ancora da raggiungere, ed infine vi sono elencati gli indirizzi dei nodi intermedi che interessano.

I preamboli di estensione vanno completati a multipli interi di 8 byte e con opportune tecniche di "padding" (o riempimento) per rispettare l'allineamento in caso di campi con un minore numero di bit.

IPv6 richiede che ogni collegamento nella internet abbia un MTU di almeno 1280 byte, se un link non ammette pacchetti di 1280 byte allora un servizio di frammentazione e riassemblamento specifico per il collegamento deve essere fornito da un livello inferiore ad IPv6.

3.1.3 Differenze con v4

Un confronto fra il preambolo di IPv4 (v. Figura 2.5) e quello di IPv6 (v. Figura 3.1) è a questo punto opportuno. Alcuni campi sono rimasti invariati, altri sono stati rimossi, altri ancora sono divenuti opzionali con l'utilizzo dei preamboli di estensione; si è ottenuto così un protocollo con maggiori proprietà di modularizzazione ed estendibilità. Il campo *IHL* è stato rimosso poiché nella nuova versione il preambolo standard ha lunghezza fissa; il campo *Type of Service* è stato rimpiazzato dal campo *Traffic Class* il cui nome ne esplicita ulteriormente il significato; il campo *Total length* è stato rinominato in *Payload length* poiché ora indica solo l'ammontare di dati escluso il preambolo; il campo *Time to live* si chiama ora *Hop limit* per ribadire l'utilizzo pratico che se ne faceva: si specifica, infatti, inequivocabilmente che esso rappresenta un limite ai "salti" che un pacchetto compie e non più un tempo di vita in secondi; è stato aggiunto il campo *Flow label* che può essere utilizzato per agevolare il routing in particolari contesti ed è stato rimosso il campo *Protocol* poiché la presenza del campo *Next Header* lo ha reso non necessario; infine si noti che il preambolo IPv6 non presenta un campo *Checksum*, questa scelta è stata fatta per velocizzare le operazioni che un router deve compiere nello smistare i pacchetti e con la considerazione che

sia il livello Data Link che quello di Trasporto effettuano le loro somme di controllo sull'informazione da trasmettere.

La gestione delle estensioni di IPv6 ha reso inoltre opzionali molte funzionalità con campi obbligatori in IPv4: la frammentazione, ad esempio, non è più considerata una funzionalità di base di ogni elemento della rete, essa deve essere localizzata solo negli host sorgente e destinazione e secondo l'ottica di IPv6 non deve riguardare i router, se un router v6 riceve un pacchetto di dimensioni eccessive ed ha la necessità di frammentarlo per trasmetterlo su altre linee si limita unicamente e rifiutare il pacchetto e ad informare l'host sorgente del motivo di tale rifiuto, sarà l'host a doversi occupare della frammentazione. Anche le opzioni per il routing sono state spostate in preamboli di estensione come visto precedentemente. Una novità importante e che attirerà l'attenzione su IPv6 è il supporto obbligatorio (l'utilizzo resta opzionale) per IPsec: una architettura di servizi per comunicazioni sicure per IP basata sulla crittografia.

Sebbene i cambiamenti rispetto ad IPv4 siano numerosi si vede come v6 intenda essere una sua evoluzione e non adotta un approccio del tutto rivoluzionario e ciò è dovuto soprattutto al desiderio (ed alla necessità) di mantenere la compatibilità con gli altri protocolli della suite IP e di consentire una transizione quanto più indolore possibile.

3.2 Indirizzi IPv6

3.2.1 Sintassi degli indirizzi IPv6

Gli indirizzi IPv6 hanno una lunghezza di 128 bit (16 byte) e questa scelta è stata fatta per garantire che lo spazio di indirizzamento non fosse mai

esaurito, si pensi che

$$2^{128} \approx 3 \times 10^{38}$$

e, come riportato in [Tanenbaum, 1996] questo numero impressionante di indirizzi[4], permetterebbe di avere una grande quantità di indirizzi IP per ogni metro quadro della superficie terrestre pur ipotizzando un'allocazione poco efficiente dello spazio d'indirizzamento.

Gli indirizzi vengono scritti in notazione esadecimale, a 8 gruppi di 2 byte separati dai due punti come nell'esempio seguente:

$$\overbrace{\underbrace{1111}_{f}\underbrace{1110}_{e}\underbrace{1000}_{8}\underbrace{0000}_{0}}^{\text{16 bit}} : \overbrace{00\cdots00}^{\text{altri 96 bit}} : \underbrace{0000}_{0}\underbrace{1110}_{\texttt{e}}\underbrace{0101}_{5}\underbrace{0010}_{2}$$

si scrive

`fe80 : 0000 : 0000 : 0000 : 0000 : 0000 : 0000 : 0e52`

Sono state inoltre introdotte delle regole per sfruttare alcune proprietà di tali indirizzi ed ottenere una scrittura più compatta:

- Gli zeri più a sinistra di ogni gruppo possono essere omessi, se uno dei gruppi è per esempio `0e52` esso può essere scritto come `e52`.

- Se uno o più gruppi consecutivi hanno tutti i bit a 0 essi possono essere sostituiti da una coppia di caratteri due punti, questa sostituzione può avvenire una sola volta nell'indirizzo.

- Gli indirizzi IPv4 in una rete IPv6 hanno gli 80 bit più significativi uguali a 0 e ne sono previste per ora due tipologie: una coppia di caratteri due punti seguiti dalla usuale notazione decimale presentata nella sezione 2.3 (per esempio `::192.198.0.10`) indica un indirizzo

[4] maggiore di 6.27×10^{23} (Numero di Avogadro)

IPv4-compatibile e viene usato per identificare un host v6 mediante un suo indirizzo v4; un'altra forma è ad esempio `::FFFF:192.198.0.10` ed indica l'indirizzo di un host solo IPv4.

Seguendo le prime due regole l'indirizzo IPv6 presentato come esempio può essere abbreviato in:

`fe80` :: `e52`

semplificando di molto la lettura.

Un insieme di indirizzi IPv6 aggregati può essere rappresentato in modo simile alla notazione CIDR per IPv4: un prefisso v6 è denotato da:

< `indirizzo ipv6` > / < `lughezza prefisso` >

in cui la prima parte è costituita da un indirizzo espresso in una delle suddette notazioni, mentre la seconda parte è un valore decimale che indica quanti dei bit più a sinistra costituiscono il prefisso. Il prefisso `fe80` :: /`10` indica che i 118 bit meno significativi sono dedicati all'identificativo di host.

3.2.2 Tipi di indirizzi

Gli indirizzi IPv6 possono essere di tre tipi:

Unicast: Un identificatore per una singola interfaccia. Un pacchetto inviato ad un indirizzo unicast viene recapitato all'interfaccia identificata da questo indirizzo.

Anycast: Un identificatore per un insieme di interfacce (generalmente appartenenti a nodi distinti). Un pacchetto inviato ad un indirizzo anycast è recapitato ad una delle interfacce identificata da quell'indirizzo

(tipicamente la più "vicina" secondo la misura di distanza del protocollo di routing).

Multicast: Un identificatore per un insieme di interfacce (tipicamente appartenenti a nodi distinti). Un pacchetto inviato ad un indirizzo di anycast viene recapitato a tutte le interfacce identificate da un tale indirizzo.

Non vi sono indirizzi Broadcast in IPv6, la loro funzione viene ricoperta dall'utilizzo del multicasting.

Uno schema tratto da [Hinden and Deering, 2003] mostra i vari tipi di indirizzi IPv6 e la loro allocazione corrente:

```
Address type         Binary prefix        IPv6 notation
------------         -------------        -------------
Unspecified          00...0  (128 bits)   ::/128
Loopback             00...1  (128 bits)   ::1/128
Multicast            11111111             FF00::/8
Link-local unicast   1111111010           FE80::/10
Site-local unicast   1111111011           FEC0::/10
Global unicast       (everything else)
```

Dal primo gruppo di indirizzi sono stati scelti l'indirizzo di *loopback* `::1/128` che viene usato per connessioni locali ad un host e l'indirizzo "non specificato" `::/128` che indica l'assenza di indirizzo e non deve mai essere assegnato ad una interfaccia.

In IPv6 vengono inoltre specificati tre ambiti per gli indirizzi: l'ambito locale (*link-local*), l'ambito di sito[5] (*site-local*) e l'ambito globale (*global scope*), ed un indirizzo di un ambito resta valido solo per gli instradamenti all'interno di quell'ambito. Un indirizzo *link-local* viene usato per la comunicazione su una LAN fra interfacce che possono instaurare una dialogo a

[5]L'utilizzo degli indirizzi *site-local* è stato deprecato da IETF data l'ambiguità nella definizione di "sito".

livello 2, ed un pacchetto destinato ad un tale indirizzo non dovrà mai esser inoltrato da un router; gli indirizzi globali sono usati per la comunicazione in ambito planetario e possono essere opportunamente riconosciuti nel processo di instradamento.

Indirizzi Unicast

Vi sono diverse forme di indirizzi Unicast in IPv6: gli indirizzi globali aggregabili, gli indirizzi per reti NSAP (vedi [Bound et al., 1996]) o IPX, gli indirizzi site-local o link-local e gli indirizzi compatibili con IPv4 e mappati in IPv4.

Un nodo IPv6 potrebbe non avere nessuna conoscenza della struttura interna di un indirizzo (e delle eventuali suddivisioni gerarchiche di un certo prefisso), oppure potrebbe avere nozione del prefisso di rete e dell'identificatore di interfaccia, come illustrato in Figura 3.5.

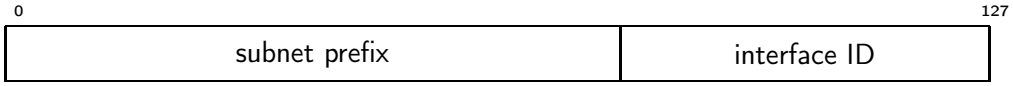

Figura 3.5: Semplice struttura di un indirizzo unicast

In questo caso l'identificatore di interfaccia viene ricavato solitamente dall'indirizzo di livello 2 dell'interfaccia e varia a seconda del tipo collegamento (vedi per esempio [Crawford, 1998]).

Un modo più complesso di vedere l'aggregamento degli indirizzi é illustrato dettagliatamente in [Hinden et al., 1998].

Il formato proposto in [Hinden and Deering, 2003] per un indirizzo globale unicast è quello di Figura 3.6 in cui il *global routing prefix*, prefisso globale di routing (tipicamente strutturato gerarchicamente) è assegnato ad un sito, l'identificatore di sottorete *subnet ID* individua un collegamento all'interno del sito, e l'identificatore di interfaccia è costituito come detto precendentemente.

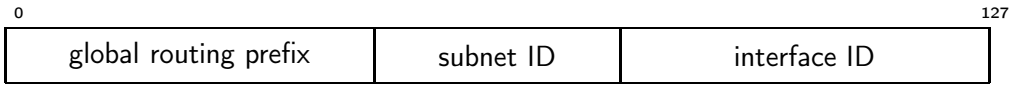

Figura 3.6: Struttura di un indirizzo unicast globale

Livelli multipli di aggregazione degli indirizzi consentono un routing più efficiente; una struttura "multi-provider" per una internetwork può infatti permettere di affrontare procedure di rinumerazione degli host con minori difficoltà rispetto ad una gestione "piatta" degli indirizzamenti.

Indirizzi Anycast

Gli indirizzi Anycast sono allocati dallo spazio di indirizzamento unicast e risultano quindi indistinguibili sintatticamente dagli indirizzi Unicast. Quando un indirizzo unicast è assegnato a più interfacce diventa un indirizzo anycast: il nodo a cui è assegnato tale indirizzo deve essere configurato esplicitamente per gestire quell'indirizzo come anycast.

Per ogni indirizzo anycast assegnato vi sono delle informazioni di carattere topologico che vengono tratte dal suo prefisso. Un uso previsto per gli indirizzi di anycast è quello di identificare l'insieme dei router di una organizzazione che fornisce servizi di rete. Un tale indirizzo può essere usato in un preambolo per il routing per forzare un pacchetto a transitare per un particolare fornitore di servizi di rete.

L'uso spropositato di indirizzi di anycast può tuttavia condurre a qualche problema nel routing per l'aumento della dimensione delle tabelle di instradamento e sono state quindi poste delle regole per impedirne l'utilizzo arbitrario:

- Un indirizzo di anycast non può essere usato come indirizzo sorgente di un pacchetto IPv6.

- Un indirizzo di anycast non può essere assegnato ad *host* IPv6, cioè, può essere assegnato unicamente a *router*.

Indirizzi Multicast

Un indirizzo multicast identifica un gruppo di interfacce, ed una interfaccia può appartenere ad un numero qualsiasi di gruppi multicast.

Un indirizzo multicast ha la struttura illustrata in Figura 3.7:

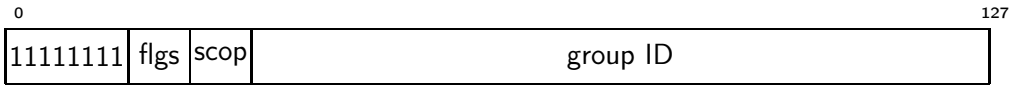

Figura 3.7: Struttura di un indirizzo multicast

gli otto bit più a sinistra posti ad 1 indicano che l'indirizzo é di multicast; *flgs* sono quattro bit di cui: i tre più significativi sono riservati ed il quarto indica se l'indirizzo é permanente (assegnato dalla IANA) o trasitorio; il campo *scop* di 4 bit indica l'ambito al quale è limitato il gruppo di multicast, infine *group ID* identifica univocamente il gruppo, permanente o transitorio, all'interno dell'ambito.

Un esempio di indirizzo multicast è `FF01:0:0:0:0:0:0:101` che indica tutti i server NTP (*group ID* uguale a `0x101`) sulla stessa interfaccia del mittente.

Gli indirizzi di multicast non devono essere usati come indirizzi sorgente in pacchetti IPv6 e non devono comparire in nessun preambolo per il routing.

Alcuni indirizzi di multicast predefiniti sono:

Indirizzi per tutti i nodi: `FF01:0:0:0:0:0:0:1`
`FF02:0:0:0:0:0:0:1`

Gli indirizzi precedenti identificano il gruppo di tutti i nodi IPv6 che sono all'interno dell'ambito 1 (*interface-local*) o 2 (*link-local*).

Indirizzi per tutti i router:	`FF01:0:0:0:0:0:0:2`
	`FF02:0:0:0:0:0:0:2`
	`FF05:0:0:0:0:0:0:2`

Questi ultimi identificano il gruppo di tutti i router IPv6 che appartengono all'ambito 1 (*interface-local*) o 2 (*link-local*) o 5 (*site-local*)

Vi sono inoltre indirizzi di multicast usati nel processo di autoconfigurazione, come vedremo in seguito.

Indirizzi di un nodo IPv6

Un host IPv6 deve riconoscere i seguenti indirizzi:

- Il suo Indirizzo link-local richiesto per ogni interfaccia.
- Ogni altro indirizzo unicast o anycast configurato per una sua interfaccia in modo manuale od automatico.
- L'indirizzo di loopback.
- Gli indirizzi multicast per tutti i nodi.
- Gli indirizzi usati nel *Neighbor Discovery* come il *Solicited-Node Address*, ovvero un indirizzo di multicast calcolato a partire dai bit meno significativi degli indirizzi unicast o anycast del nodo.
- Gli indirizzi di multicast di tutti gli altri gruppi ai quali il nodo appartiene.

Un router deve riconoscere gli indirizzi precedenti ed inoltre i seguenti indirizzi che lo identificano:

- Gli indirizzi di anycast relativi alla sottorete per tutte le interfacce abilitate all'instradamento.

- Tutti gli altri indirizzi configurati sul router.
- Indirizzi multicast per tutti i router.

3.3 ICMPv6

ICMP è un protocollo della famiglia TCP/IP per lo scambio di messaggi fra nodi IP e per la notifica di eventuali problemi o errori nella gestione dei pacchetti. La versione originale del protocollo è definita in [Postel, 1981] ed IPv6 usa tale protocollo con alcune modifiche.

Il nuovo protocollo di messaggi di controllo prende il nome di ICMPv6 e deve essere implementato in modo completo da ogni nodo IPv6. Il preambolo IPv6 precedente un messaggio ICMPv6 deve avere il valore del campo *Next Header* a 58. Le specifiche complete di ICMPv6 sono fornite in [Conta and Deering, 1998].

3.3.1 Formato dei messaggi

Ogni messaggio IPv6 è preceduto da un preambolo IPv6 e da eventuali preamboli IPv6 di estensione secondo lo schema di imbustamento descritto in precedenza. Il formato generale di un messaggio é mostrato in Figura 3.8.

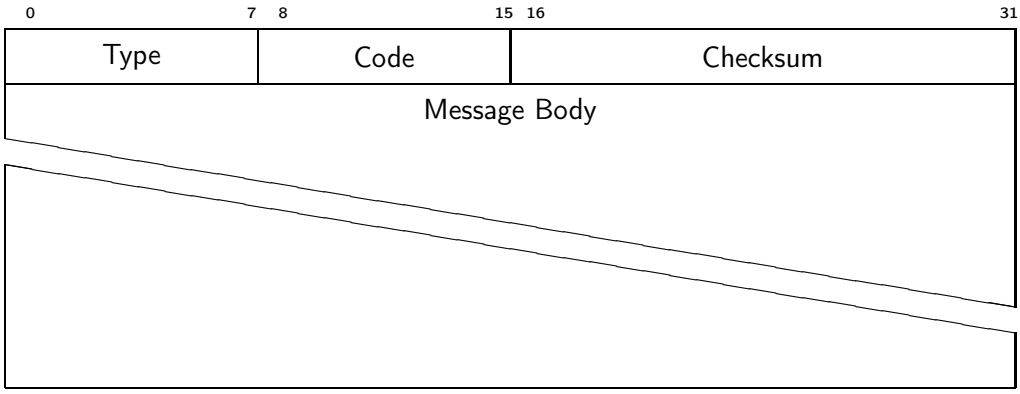

Figura 3.8: Formato di un messaggio ICMPv6

Il campo *Type* identifica il tipo di messaggio, il suo valore determina inoltre il formato dei dati successivi; il campo *Code* specifica con maggiore precisione il tipo di evento che si sta segnalando ed il suo valore dipende dal tipo di messaggio; il campo *Checksum* è utilizzato per controllare eventuali corruzioni nei dati del messaggio ICMPv6 ed in alcune parti del preambolo IPv6.

Un nodo che invia messaggi ICMPv6 deve determinare sia l'indirizzo sorgente che quello destinazione del preambolo IPv6 prima del calcolo della somma di controllo. Se il nodo ha configurati più indirizzi unicast deve scegliere l'indirizzo sorgente in modo da segnalare in modo preciso quale delle sue interfacce è coinvolta nella notifica del messaggio. La scelta dell'indirizzo sorgente può richiedere anche l'esame della tabella di instradamento.

Nel trattamento di messaggi ICMPv6 possono essere usate delle tecniche che limitino la quantità di pacchetti inviati per evitare problemi di sovraccarico della rete.

3.3.2 Tipi di messaggi

I messaggi ICMPv6 sono raggruppati in due classi: messaggi di errore e messaggi di informazione. I messaggi di errore sono identificati come tali dal fatto che hanno il bit più significativo del campo *Type* a zero. I messaggi di errore avranno quindi un campo *Type* da 0 a 127, i messaggi di informazione da 128 a 255.

Vi sono inoltre messaggi che vengono usati specificamete per la determinazione degli altri nodi sulla stessa LAN. Questo processo di *Neighbor Discovery* verrà esaminato nella sezione 3.4.

Messaggi di errore

Il messaggio *Destination Unreachable* dovrebbe essere generato da un router, o dal livello 3 del nodo originario, in risposta ad un pacchetto che non può essere recapitato alla destinazione per ragioni che non dipendano da situazioni di congestione. Il campo *Code* deve essere settato per specificare il tipo di problema che rende impossibile il recapito.

Il messaggio *Packet Too Big* deve essere inviato da un router in risposta ad un pacchetto che non può essere inoltrato a causa della sua dimensione superiore al MTU della linea di uscita. Questo messaggio può essere inviato come risposta anche ad un pacchetto che abbia come indirizzo sorgente un indirizzo multicast. Questo tipo di messaggio può, inoltre, essere usato per invocare la gestione della frammentazione su un host IPv6.

Il messaggio *Time Exceeded* è utilizzato nel caso in cui un router riceva un pacchetto con il campo *Hop Limit* a zero, oppure decrementi tale campo a zero, per segnalare che vi potrebbe essere un ciclo nel percorso di routing oppure che il valore iniziale di *Hop Limit* è troppo piccolo. Il pacchetto che ha generato questa condizione d'errore viene ignorato e non viene quindi inoltrato dal router in questione.

Il messaggio *Parameter Problem* dovrebbe essere inviato se un nodo IPv6 individua un problema in qualche campo del preambolo IPv6 o di qualche preambolo di estensione durante la gestione del pacchetto, per segnalare che non può trattare quel pacchetto. Il pacchetto incriminato viene ignorato e nel messaggio viene indicato il campo che ha generato l'errore usando un campo *Pointer* che indica lo scostamento del campo invalido all'interno del pacchetto d'origine.

Messaggi di informazione

Il messaggio *Echo Request* è utilizzato per diagnosticare eventuali problemi di rete, un tale messaggio contiene anche informazioni per l'identificazione (mediante un contatore di sequenza) della richiesta di eco nel caso in cui se ne invii più d'una.

Ad ogni richiesta di eco verso un indirizzo unicast deve essere generato un messaggio *Echo Reply* e l'indirizzo di destinazione di tale risposta deve essere l'indirizzo del richiedente. Nel caso di *Echo Request* ad indirizzi multicast, il ricevente dovrebbe ugualmente generare una risposta, usando come indirizzo sorgente uno degli indirizzi unicast configurati per l'interfaccia rispondente anche all'indirizzo multicast, ma in questo caso la generazione di un *Echo Reply* non è obbligatoria.

Messaggi per Neighbor Discovery

Il messaggio *Router Solicitation* viene inviato da una interfaccia quando essa diviene attiva, per richiedere ai router di segnalare la loro presenza.

Il messaggio *Router Advertisement* viene usato dai router per segnalare la loro presenza insieme ad altre informazioni e parametri di collegamento, esso può essere spedito con scadenza periodica piuttosto che in risposta ad un messaggio di *Router Solicitation.* I messaggi *Router Advertisement* contengono anche informazioni sui prefissi per quel collegamento e parametri di configurazione come ad esempio un valore suggerito per il campo *Hop Limit* di IPv6

Il messaggio di *Neighbor Solicitation* è inviato da un nodo per determinare l'indirizzo link-layer di un nodo oppure per verificare l'indirizzo link-layer di un altro nodo contattato in precedenza.

Il messaggio *Neighbor Advertisement* costituisce una risposta ad un mes-

saggio di *Neighbor Solicitation*, ma può anche essere utilizzato nel caso in cui un nodo debba comunicare un cambiamento del proprio indirizzo link-layer.

Infine il messaggio *Redirect* é utilizzato dai router per informare gli host che è presente un migliore "primo salto" (ovvero un router) per una certa destinazione.

3.3.3 Sicurezza

Lo scambio di pacchetti ICMPv6 può avvalersi dei meccanismi di autenticazione descritti in [Kent and Atkinson, 1998a] utilizzando un Preambolo di Autenticazione (AH), in questo caso il ricevente dovrà verificare l'informazione di autenticità del mittente prima di processare il pacchetto. Tale meccanismo di autenticazione, unito alla crittazione del pacchetto, può aiutare nella prevenzione di attacchi ICMP che operino mediante manipolazioni del pacchetto contenente il messaggio.

3.4 Neighbor Discovery

Lo scopo principale del protocollo di *Neighbor Discovery* (ND) è quello di determinare gli indirizzi link-layer dei nodi vicini (appartenenti alla stessa LAN) di un certo nodo IPv6 e di tenere aggiornate tali informazioni. Un nodo può inoltre utilizzare il *Neighbor Discovery* per individuare un router che inoltri i pacchetti destinati a nodi non sulla LAN. L'uso di questo protocollo permette di ridurre gli inconvenienti in caso di guasti o aggiornamenti della rete come riportato in [Narten et al., 1998].

Diversi livelli data-link hanno diverse proprietà che influiscono anche sul comportamento della ricerca dei nodi vicini: un collegamento che supporta il multicast è considerato il caso generale, i collegamenti punto-punto possono essere visti come un caso particolare di quest'ultimo, vi sono poi collegamenti

detti *Non-Broadcast Multi-Access* (NBMA) che non supportano nativamente comunicazioni multicast e broadcast, oppure collegamenti che gestiscono il mezzo trasmissivo in maniera particolare o che non hanno un MTU fisso e ben definito o che non rispettino proprietà di riflessività e transitività nella relazione di collegamento; poiché il processo di ND fa un uso intenso della comunicazione multicast nell'ambito link-layer alcuni di questi tipi di collegamento potrebbero necessitare tecniche alternative per la gestione del multicast e dei servizi ad esso correlati.

Oltre agli indirizzi descritti nella sezione 3.2.2 viene usato anche l'indirizzo "non specificato" che non può mai indicare una destinazione, ma può essere usato come sorgente da un nodo che non conosce ancora il suo indirizzo IPv6.

Questo protocollo risolve un insieme di problemi legati all'interazione di nodi sulla stessa LAN come la ricerca di un router predefinito o preferito, l'assegnazione di prefissi per gli indirizzi o di parametri da usare nello scambio di pacchetti, o ancora lo stato di raggiungibilità di una destinazione sulla LAN o la gestione di indirizzi duplicati.

Per esempio, la scoperta di un router avviene grazie all'invio di messaggi di *Router Advertisement* da parte dei router sui collegamenti capaci di multicast, questi messaggi inviati periodicamente segnalano la presenza dei router e gli host costruiscono una lista dei router presenti dalla quale ricavare un router predefinito.

Il Neighbor Discovery è quindi una sintesi delle funzionalità di IPv4 offerte da ARP e dalle tecniche di *Router Discovery* ed *ICMP Redirect* e la vera novità è costituita dal fatto che queste funzionalità siano ritenute parte integrante del nucleo di IPv6.

Molti sono i miglioramenti dell'adozione del *Neighbor Discovery*: efficienza e nuove funzionalità. Mediante questo protocollo si possono autoconfigu-

rare gli host ed aggiornare tali configurazioni anche nel caso in cui sullo stesso link vi siano più prefissi di rete reagendo inoltre in modo più robusto di IPv4 a fenomeni di partizionamento della rete o di parziali malfunzionamenti.

Effettuando la risoluzione degli indirizzi link-layer al livello di ICMP si rende il protocollo meno dipendente dal mezzo trasmissivo rispetto ad ARP e si possono utilizzare meccanismi di autenticazione e sicurezza perfino nel processo di ricerca dei nodi vicini.

3.4.1 Strutture dati per il Neighbor Discovery

Perché il processo di ND abbia luogo tutti gli host devono conservare delle informazioni per ogni interfaccia:

Neighbor Cache: Un insieme di dati sui vicini ai quali il traffico è stato inviato di recente, una struttura corrispondente ad una *arp-table* di IPv4 arricchita con altre informazioni sullo stato e sul tipo dei vicini.

Destination Cache: Un insieme delle destinazioni più recenti, sia sulla LAN che fuori dalla LAN aggiornate anche mediate i messaggi di *Redirect* inviati da un router della LAN.

Prefix List: Una lista di prefissi che specificano quali indirizzi siano appartenenti alla LAN del nodo in questione e che sono notificati mediante messaggi di *Router Advertisements*.

Default Router List: Una lista di router ai quali si può inviare del traffico. Ogni router dovrà necessariamente essere un nodo "vicino" e mediante questa lista si potrà eleggere un router predefinito.

Una implementazione specifica può inoltre presentare ulteriori campi per estendere l'insieme delle informazioni da conservare.

La *Neighbor Cache* è la struttura di riferimento per il Neighbor Discovery ed un host può fornire dei meccanismi per visualizzare la struttura dati. In sistemi **GNU/Linux** si può visualizzare questa struttura con il comando `ip -6 neigh show`,

```
2001:e12:117d::1 dev eth0 lladdr 00:e0:4c:6c:01:54 router nud reachable
2001:e12:117d:1::1 dev ath0 lladdr 00:09:5b:c8:3e:79 router nud reachable
fe80::209:5bff:fec8:3e79 dev ath0 lladdr 00:09:5b:c8:3e:79 router nud stale
fe80::2e0:4cff:fe6c:154 dev eth0 lladdr 00:e0:4c:6c:01:54 router nud stale
```

su sistemi **BSD** con il comando `ndp -a` che fornisce un output del tipo:

```
Neighbor                        Linklayer Address  Netif Expire    S
fe80::290:27ff:fe97:f568%fxp0 0:90:27:97:f5:68    fxp0 permanent R
fe80::2e0:4cff:fe6c:154%fxp0  0:e0:4c:6c:1:54     fxp0 6s        R
fe80::1%lo0                   (incomplete)         lo0 permanent R
```

Come si vede, il campo `S` contiene informazioni sullo stato del vicino, `R` indica che il *Neighbor* si trova nello stato *REACHABLE* ed è quindi raggiungibile da quella interfaccia, altri stati possibili possono essere *INCOMPLETE* quando la risoluzione dell'indirizzo non è giunta al termine, *STALE* se un vicino non è più ritenuto raggiungibile ma si sta tentando di inviargli del traffico, *DELAY* in attesa di verificare nuovamente la raggiungibilità o *PROBE* quando si sta già effettuando la verifica di raggiungibilità.

3.4.2 Algoritmo di trasmissione di un pacchetto

Per recapitare un pacchetto a destinazione, un nodo utilizza una combinazione della *Destination Cache*, della *Prefix List* e della *Default Router List* per conoscere l'indirizzo del prossimo salto verso il destinatario, una volta ricavato tale indirizzo si consulta la *Neighbor Cache* per conoscere l'indirizzo link-layer del vicino a cui sarà spedito il pacchetto.

La determinazione del "prossimo salto" (*next-hop*) avviene in questo modo: mediante la *Prefix List* si controlla se la destinazione si trova sulla LAN

ed in caso di risposta affermativa il next-hop coinciderà con la destinazione del pacchetto; in caso contrario si selezionerà il router appropriato grazie alla *Default Router List* e gli si invierà il traffico da consegnare a destinazione. Se la *Default Router List* è vuota il mittente assumerà che il destinatario si trova nella LAN.

Per il traffico multicast si considera come *next-hop* la destinazione multicast del pacchetto ritenendo che sia sulla LAN e la metodologia per ricavare l'indirizzo link-layer di un indirizzo multicast dipende dal tipo di rete (vedi per esempio [Crawford, 1998]).

La comunicazione fra vicini potrebbe interrompersi in qualsiasi momento e per numerosi fattori come guasti hardware o sostituzione di interfacce; è quindi necessario un meccanismo che tenga traccia della raggiungibilità dei *Neighbor* che ricevono il traffico, siano essi i destinatari ultimi della comunicazione oppure dei router che inoltrano i pacchetti verso la destinazione finale.

La scoperta di tale stato di raggiungibilità (*Neighbor Unreachability Detection*) avviene per tutti i percorsi fra gli host ed i nodi vicini, ma potrebbe avvenire anche fra router nel caso in cui il protocollo di routing utilizzato non disponesse di meccanismi analoghi.

Quando un percorso verso un nodo vicino sembra non essere più valido la procedura di ripristino della comunicazione dipende dal ruolo del vicino. Se quest'ultimo è la destinazione finale del pacchetto, allora si dovrà ricorrere nuovamente alla risoluzione del suo indirizzo, ma se si tratta di un router potrebbe essere sufficiente iniziare ad usare un altro router come next-hop eliminando dalla *Neighbor Cache* i riferimenti al router precedente. Le pratiche di *Neighbor Unreachability Detection* si applicano unicamente nel caso di comunicazioni verso destinazioni unicast.

3.4.3 Autoconfigurazione

Il processo di autoconfigurazione previsto da IPv6 ha l'utilità di predisporre gli host a trasmettere traffico IP senza alcun intervento manuale facendo in modo che gli apparati di rete possano stabilire in maniera autonoma gli indirizzi di rete da utilizzare ed eventualmente anche il periodo di validità degli indirizzi. Esso avviene in diversi passi: inizialmente l'host costruisce un indirizzo link-local e verifica la sua unicità sulla LAN stabilendo, inoltre, quali settaggi dovranno essere autoconfigurati. Si potrà effettuare una configurazione *stateless*, che richiede di configurare manualmente unicamente i router sulla LAN e non prevede la presenza di server dedicati alle configurazioni, oppure *stateful* con la presenza di un server che mantenga un database delle configurazioni degli host.

Vale la pena prestare maggiore attenzione al metodo usato per determinare l'unicità di un indirizzo unicast prima che esso sia assegnato ad una interfaccia di un host IPv6. Tale *Duplicate Address Detection* (Individuazione di Indirizzo Duplicato) deve avvenire per tutti gli indirizzi di unicast, siano essi ottenuti mediante configurazione manuale, *stateless* o *stateful*. La verifica di unicità non deve avvenire per gli indirizzi di anycast e può essere evitata quando un nuovo indirizzo è ricavato da un indirizzo link-local unico lasciandone inalterati i bit dedicati all'identificatore di interfaccia.

La procedura per la verifica di unicità fa uso dei messaggi di *Neighbor Solicitation* e *Neighbor Advertisement* e non è da considerare completamente affidabile, specialmente nel caso in cui un collegamento viene partizionato durante tale verifica. Il primo passo da compiere consiste nello stabilire la validità del nuovo indirizzo secondo le regole di IPv6 (vedere anche [Narten et al., 1998]). Si procede dunque con l'invio di un messaggio di *Neighbor Solicitation* che utilizzi il nuovo indirizzo candidato ad essere unico e si

attende per una risposta (un messaggio *Neighbor Advertisement*) che potrà eventualmente avere come mittente quello che si ritiene l'indirizzo "tentativo", in questo caso l'indirizzo sarà ritenuto duplicato. Perché il meccanismo illustrato sinteticamente funzioni, l'host che effettua la verifica deve entrare a far parte del gruppo multicast relativo a "tutti i nodi" per ricevere messaggi di *Advertisement* ed al gruppo "nodi sollecitati" in modo da riconoscere un eventuale nodo che utilizza il suo stesso indirizzo.

3.5 Il Routing

Come detto in precedenza, nello sviluppo di grandi reti di calcolatori una struttura gerarchica è di fondamentale importanza per una efficiente comunicazione fra nodi; può avvenire che vi siano più vie di comunicazione fra una certa sorgente ed una destinazione, ed il compito dell'infrastruttura di routing è quello di scegliere il percorso migliore.

Molti degli algoritmi e protocolli di routing usati con IPv4 hanno subito degli aggiornamenti per l'utilizzo con IPv6 ed in alcuni casi é stata necessaria una riscrittura del protocollo per il supporto ad IPv6.

3.5.1 Terminologia

Lo sviluppo delle tecniche di routing ha avuto una attenzione sempre maggiore e le problematiche legate al networking sono oggi per lo più relative alla questione degli instradamenti di traffico. Uno degli obiettivi principali di tutto il sistema degli instradamenti è quello di fare in modo che le scelte "locali" effettuate da ogni router, riguardanti il traffico che transita per esso, abbiano una certa valenza di ottimalità "globale" all'interno della rete. Questo obiettivo si raggiunge cercando di far coincidere oppure, come si usa dire, "convergere" la visione che un router ha della rete con la effettiva strut-

tura topologica del sistema di rete e ciò è possibile unicamente mediante lo scambio di informazioni fra router.

L'entità router può essere scomposta logicamente in due componenti, una che opera l'inoltro dei pacchetti sulle opportune interfacce (*Network Processor*), ed una componente adibita a decidere quale traffico debba essere inoltrato attaverso una certa interfaccia (*Routing Processor*). Il primo compito richiede una efficienza dipendente in modo prevalente dalla tecnologia disponibile, mentre il secondo compito, quello di prendere le decisioni sul routing, può dipendere dagli algoritmi e dalle tecniche impiegate; esso deve essere, si, assolto rapidamente ma le scelte devono essere fatte in modo attento e devono spesso tener conto delle informazioni attinte da altri router ed anche delle variazioni delle condizioni della rete. Per la comunicazione fra router si utilizzano quindi dei protocolli specifici, e per il calcolo del percorso ottimale si adoperano degli algoritmi che tengano conto dei dati ricevuti dai router vicini. Ogni algoritmo di routing ha una sua misura della "bontà" di un percorso che può dipendere dalla distanza misurata in salti fra sorgente e destinazione, o essere valutata mediante numerose caratteristiche del percorso quali banda passante, latenza, stato di congestione.

Il risultato delle scelte di routing esprime la "metrica" di un percorso e viene sintetizzato nella cosiddetta tabella di routing, che può essere vista banalmente come una associazione fra una rete di destinazione ed una interfaccia mediante la quale raggiungere tale rete.

3.5.2 Routing statico e dinamico

La forma più semplice di routing è quella in cui i dati sulla bontà di un percorso rispetto ad un altro sono configurati manualmente all'interno della tabella di routing e non vengono modificati dal router se le condizioni della

rete mutano, ad esempio a causa di un guasto, oppure con l'aggiunta di un nuovo link. Questo routing statico non si adatta alla realtà attuale della rete, e risulta applicabile unicamente a reti di dimensioni assai ridotte nelle quali ogni cambiamento può essere riportato nelle tabelle di routing in modo manuale.

Sono chiaramente desiderabili delle tecniche che aggiornino in modo dinamico la rappresentazione che un router ha della rete, e lo sviluppo del routing dinamico ha infatti canalizzato numerosi sforzi. Vi sono state diverse evoluzioni delle tecniche di routing dinamico che saranno mostrate nel seguito.

Altro fattore che accompagna il successo del routing dinamico è la pratica di ridistribuire gli instradamenti calcolati da un processo di routing in un altro processo di routing per integrare le informazioni sulla raggiungibilità ed avere una visione più completa della rete; questa ridistribuzione evidenzia le sue potenzialità nel caso in cui un router possa utilizzare più protocolli di routing diversi, a seconda delle proprietà delle reti a cui è connesso, e godere dei punti forti di ogni protocollo.

Protocolli Distance-Vector

Una prima generazione di routing dinamico ha dato vita ai protocolli *Distance Vector* basati essenzialmente su un algoritmo di Bellman-Ford (vedi [Hedrick, 1988]) per la ricerca del cammino minimo che utilizza dei parametri di peso per i link ed una tecnica di rilassamento progressivo per calcolare le metriche da associare ad ogni percorso. Il nome di questo tipo di protocolli è tratto dalla struttura dati utilizzata nell'algoritmo, si effettua il rilassamento su un vettore di distanze (corrispondenti al numero di salti) per ottenere il percorso di routing ottimale verso una certa destinazione. Lo scambio di

questi vettori di distanza avviene con un periodo fissato oppure quando vi sono cambiamenti nella rete (*Triggered Updates*). Questo tipo di protocollo ha buone proprietà di stabilità ma può dare dei problemi di loop negli instradamenti piochè non reagisce tempestivamente ai cambiamenti della rete, ed inoltre non si adatta bene a grandi internetwork per due motivi fondamentali: al crescere della rete tende a crescere anche il vettore delle distanze, ed inoltre i vettori delle distanze forniscono una visione troppo localizzata della rete creando l'inconveniente del *routing by rumor* (ovvero routing per sentito dire, o per pettegolezzo) nel caso in cui acquisiscano informazioni su reti non raggiungibili direttamente, scatenando un fenomeno di "passa-parola" che non conduce ad instradamenti efficienti. Alcuni di questi problemi possono essere arginati o eliminati adottando degli stratagemmi come *split horizon*, *counting to infinity* e *poison reverse*, ma la natura dei *Distance Vector* li rende comunque inadatti a grandi reti di router per la scarsa attitudine a stabilire gerarchie di instradamento.

Protocolli Link-State

Lo sviluppo dello studio del routing dinamico ha condotto alla progettazione di algoritmi e protocolli più complessi, che risultassero maggiormente flessibili e scalabili. Questa seconda generazione di protocolli prende il nome di *Link-State* per evidenziare il fatto che nel processo decisionale intervengono molteplici fattori di misura e la bontà degli instradamenti non dipende unicamente dal numero di salti verso la rete di destinazione ma da una valutazione sullo "stato" dei collegamenti. I protocolli *Link-State* possono adattarsi meglio a grandi reti ma richiedono un maggiore impegno di risorse computazionali a causa della loro complessità rispetto a protocolli *Distance Vector*. La scalabilità di questo tipo di protocolli di instradamento é garantita dalla

capacità di instaurare delle gerarchie all'interno di un dominio di routing, ed unendo a ciò il supporto per VLMS (*Variable Length subnet-masking* o CIDR) si ottiene un aggregamento delle reti indirizzate e quindi un risparmio di spazio nelle tabelle di routing. Lo scambio di informazioni sullo stato dei link avviene mediante LSA (*Link-state Advertisement*) fra i router vicini mediante un meccanismo di *flooding* nel momento in cui un router diventa attivo. Questa tecnica permette di individuare i router vicini e di stabilire con essi un eventuale rapporto di fiducia aumentando l'accuratezza degli instradamenti; queste relazioni di adiacenza vengono generalmente memorizzate in una base di dati in modo da tener traccia costantemente delle informazioni più aggiornate provenienti dai router vicini. I protocolli *Link-State* risultano quindi uno strumento complesso ma preziosissimo in una infrastruttura di routing.

3.5.3 Protocolli di routing per IPv6

Il primo protocollo di routing riprogettato per IPv6 è stato RIP, un protocollo *Distance Vector* assai diffuso e relativamente semplice. RIP nasce insieme ad IP ed è un protocollo storico di Internet, anche se non viene più usato da tempo in reti medio-grandi conserva un suo ruolo nelle piccole reti periferiche e per questo ha subito diversi aggiornamenti. La sua seconda versione RIPv2 ha introdotto numerose nuove funzionalità e la principale è sicuramente il supporto per la modalità di indirizzamento *classless* CIDR; una più profonda operazione di modifica è stata però necessaria per IPv6, dando vita al protocollo RIPng detto anche RIPv6. RIPng elimina il supporto nativo per l'autenticazione poiché i messaggi fra i router possono godere delle funzionalità di sicurezza offerte da IPv6, non sono supportate inoltre più istanze del processo di routing per ogni rete connessa ad un router. Tutte le altre modi-

fiche sono volte al supporto della sintassi degli indirizzi IPv6 ed alla gestione delle opzioni e, benchè gli indirizzi abbiano una dimensione maggiore degli indirizzi IPv4, un pacchetto RIPng è solo di poco più grande di un pacchetto RIPv2 grazie alla struttura semplificata degli header IPv6.

Un altro protocollo convertito ad IPv6 é IS-IS, anch'esso un IGP (Interior Gateway Protocol come RIP) ma basato su una logica *Link State*. Inizialmente IS-IS fu sviluppato da ISO per instradare CLNS e fu poi modificato per trattare IP. La sua struttura modulare consente di introdurre con pochissime difficoltà nuove funzioni e nuovi servizi, ed infatti IS-IS è stato uno dei primi protocolli ad offrire supporto ad IPv6. Un concetto utilizzato nei protocolli *Link-State* é quello di "area" definita come una porzione del dominio di routing, in IS-IS l'area 0 (la dorsale per la quale tutte le aree devono transitare per raggiungersi vicendevolmente) è denominata area di Livello 2 (*L2 area*). Tutte le altre aree sono classificate come di Livello 1. IS-IS effettua il routing fra le varie aree di un AS (*Autonomous System*) e, utilizzando un *System ID* (identificatore di sistema), opera quindi gli instradamenti fra i dispositivi all'interno di una area (routing L1). Il protocollo determina mediante un indirizzo di area, l'area da raggiungere e dunque identifica la destinazione finale del traffico mediante il *System ID*; sintetizzando, IS-IS effettua gli instradamenti a due livelli di aggregazione differenti: area e stazione.

Il protocollo IGP più usato attualmente é OSPF, esso é stato progettato ispirandosi ad IS-IS e ne conserva alcune idee, ma estende il numero di fattori variabili considerati nella determinazione della stima di un link e dedica un trattamento assi attento alle relazioni di fiducia con i router vicini. Uno dei punti di forza di OSPF risiede nelle sue specifiche pubbliche, che permettono a chiunque di implementarlo su apparati differenti conservando compatibilità e inter-operabilità. Queste caratteristiche lo hanno reso la lingua franca degli

IGP. Come nel caso di IS-IS la struttura modulare del protocollo hanno reso meno ostica una sua riprogettazione per IPv6, tuttavia, molti cambiamenti sono stati introdotti in OSPFv6 e per questo motivo le implementazioni hanno avuto bisogno di tempo per stabilizzarsi e raggiungere un grado di maturità soddisfacente. Il supporto ad IPv6 ha richiesto sia dei cambiamenti che riflettessero la semantica dei nuovi indirizzi v6, sia delle modifiche per trattare le maggiori dimensioni di questi ultimi. Altre innovazioni sono state introdotte per trarre vantaggio da IPv6 e per ottenere anche dei benefici dalla granularità conferita da IPv6 al controllo dell'aggregazione. Si fa ora uso del termine *link* invece di *subnet* e si utilizza il multicast nelle pratiche di *flooding* per ottimizzare lo scambio di *LSA*. Le funzionalità di autenticazione sono state inoltre rimosse dal protocollo, dato il supporto di IPv6 per AH; l'utilizzo di indirizzi link-layer nello stabilire le relazioni di vicinanza è reso possibile direttamente delle funzionalità di *Neighbor Discovery*.

Non verranno qui esposti maggiori dettagli su OSPF, per una visione completa del protocollo si veda [Coltun et al., 1999].

Parlando di IS-IS si è fatto accenno agli AS, possiamo definire una *Autonomous System* come una rete, o un insieme di reti che sono sotto il controllo di una unica autorità amministrativa. Gli instradamenti per quel traffico che transita da un AS ad un altro richiedono una classe di protocolli dedicati che vengono chiamati EGP ed il protocollo più diffuso fra questi prende il nome di BGP.

In BGPv4+ (la versione di BGP con supporto IPv6) le relazioni di adiacenza, dette *neighbor statements*, sono configurate staticamente e sono l'elemento fondamentale nel processo di routing fra gli AS ed i router sono identificati da un *router ID*; in IPv4 questo ID corrisponde ad uno degli indirizzi del router nel caso di IPv6 esso dovrà essere sintatticamente uguale ad

un indirizzo v4, ma potrà essere scelto in maniera arbitraria rispettandone l'unicità sul link. Anche in questo caso l'uso del *Neighbor Discovery* e degli indirizzi link-local fa in modo che gli indirizzi dei router siano unici su un link.

3.5.4 Vantaggi nel routing

Le innovazioni introdotte da IPv6 possono condurre ad un routing più efficiente e stabile, tuttavia la stabilità del routing con IPv6 deve essere ancora dimostrata in maniera incontestabile: lo scenario attuale mostra un uso massiccio di tunnel per l'accesso alla rete v6, e ciò rende difficile affrontare le scelte di routing con la dovuta granularità.

Si può mostrare un esempio semplificativo che ci fa intuire la potenziale agilità del routing con IPv6, la prima sequenza di passi descrive una parte delle attività che un router deve compiere nell'instradare un pacchetto IPv4:

1. Calcolare e controllare l'*Header Checksum.*
2. Controllare i parametri nel preambolo, scartare il pacchetto se qualche parametro non è valido.
3. Decrementare il campo *Time to Live*, se il suo valore è uguale a 0 scartare il pacchetto.
4. Effettuare controlli di sicurezza, scartare il datagram se qualche test fallisce.
5. Selezionare il prossimo salto, occorre deframmentare? Scartare il pacchetto se occorre la deframmentazione ma il bit DF è settato.
6. Esaminare le opzioni, se presenti aggiornare i campi opportuni (per esempio nel caso di *Source Routing*).

7. Aggiornare il preambolo del pacchetto (o dei preamboli, nel caso della frammentazione).

8. Calcolare il nuovo *Header Checksum*.

9. Trasmettere il pacchetto alla prossima destinazione.

Nella seconda sequenza vediamo come le semplificazioni di IPv6, rendano meno oneroso anche l'instradamento:

1. Controllare i parametri nel preambolo, scartare il pacchetto se qualche parametro non è valido.

2. Decrementare il campo *Hop Limit*, se il suo valore è uguale a 0 scartare il pacchetto.

3. Effettuare controlli di sicurezza, scartare il datagram se qualche test fallisce.

4. Selezionare il prossimo salto.

5. Esaminare le opzioni, se presenti aggiornare i campi opportuni (per esempio nel caso di *Source Routing*).

6. Aggiornare il preambolo standard del pacchetto ed i preamboli di estensione.

7. Trasmettere il pacchetto alla prossima destinazione.

Questa agilità si paga solitamente in termini di spazio di memoria occupato, ma si deve tener conto che la latenza ed i tempi di risposta sono risorse ben più preziose dello spazio di memoria, ed è quindi più che ragionevole adottare un protocollo che occupa maggiore spazio se ciò si riflette in maniera positiva sulla latenza di instradamento.

3.6 Sicurezza con IPsec

Altro aspetto di grande importanza in IPv6 è l'adozione di una architettura per comunicazioni sicure basata su crittografia forte. L'architettura IPsec fornisce delle specifiche per garantire l'autenticità e la riservatezza dei dati circolanti su reti IP, tali direttive possono essere applicate anche ad IPv4, il merito di IPv6 risiede principalmente nell'aver contribuito alla diffusione di IPsec ed al processo di maturazione delle sue varie implementazioni.

3.6.1 Autenticità e Confidenzialità

Il supporto per IPsec è obbligatorio in IPv6 ed i servizi di autenticazione e crittazione vedono le loro funzioni dislocate in due preamboli di estensione, il primo detto AH (*Authentication Header*) ed il secondo ESP (*Encapsulating Security Payload*) Le caratteristiche di AH ed ESP sono trattate approfonditamente in [Kent and Atkinson, 1998c]; parafrasandone il contenuto possiamo descrivere AH come fornitore di un servizio di integrità dei dati, di autenticità dell'origine dei dati ma anche di servizi "anti-replay". ESP può fornire servizi di confidenzialità anche per flussi di traffico (mediante tecniche di tunneling), ed offre parte delle funzionalità presenti anche in AH. Entrambi i servizi possono essere utilizzati per il controllo di accesso basato sulla distribuzione di chiavi crittografiche e sulla gestione dei flussi di traffico relativi ai protocolli di sicurezza.

3.6.2 Security Association e Security Policy

Un concetto fondamentale usato in IPsec è quello di "*Security Association*" (SA), una connessione simplex che si occupa dei servizi di sicurezza per il traffico che trasporta. Sia AH che ESP fanno uso delle SA e fra gli scopi dei protocolli di scambio delle chiavi crittografiche vi è anche quello dell'instau-

razione e del mantenimento delle SA. Un servizio di sicurezza è fornito da una *Security Association* mediante AH oppure ESP, ma non entrambi. Se risulta necessario utilizzare sia AH che ESP, allora sarà necessario instaurare due SA, una per ogni tipo di servizio. Si consideri una comunicazione bi-direzionale per la quale siano richiesti servizi di autenticazione ed anche di confidenzialità in entrambe le direzioni, in questo caso vi sarà la necessità di quattro *Security Association*, due per ogni direzione di traffico delle quali una per AH ed una per ESP.

Possono essere definiti due tipi di SA: una SA *tunnel mode* oppure *transport mode*, nel primo caso si ricorre all'incapsulamento dell'intero pacchetto IP in una connessione sicura (come in Figura 3.9), il secondo tipo può essere utilizzato esclusivamente in una comunicazione host-to-host ed utilizza il meccanismo delle estensioni per applicare i servizi di sicurezza ai livelli superiori a quello di rete. Nel caso di comunicazioni host-to-net oppure net-to-net si possono utilizzare esclusivamente SA in *tunnel mode* per fornire i servizi necessari.

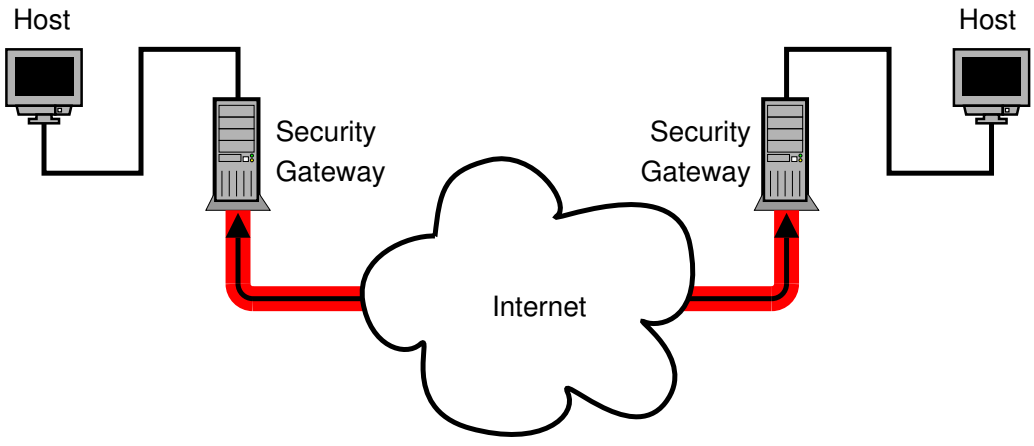

Figura 3.9: Un esempio di una comunicazione sicura in tunnel-mode.

Affinchè il tutto funzioni vi sono due basi di dati che memorizzano le

informazioni necessarie alla fornitura delle funzioni sicurezza: abbiamo il *Security Policy Database* che specifica le regole con cui trattare il traffico in entrata ed in uscita ed il *Security Association Database* contenente i settaggi di ogni SA attiva.

Possiamo quindi ripensare alla *Security Association* come ad un meccanismo mediante il quale applicare una certa *Security Policy*: che specificherà se scartare o lasciar passare un certo traffico.

Le chiavi crittografiche necessarie ai servizi di sicurezza possono essere configurate manualmente (*Pre-Shared Keys*) o scambiate mediante opportuni protocolli utilizzando eventualmente dei certificati di autenticità rilasciati da una *Certification Authority*.

Si può ricorrere ad IPsec nel caso in cui si desideri instaurare una comunicazione sicura attraverso una rete non sicura o non conosciuta (quale Internet, ad esempio) creando in questo modo delle reti private virtuali, ma i servizi di IPsec possono essere preziosi anche nelle pratiche di routing in cui l'attendibilità delle informazioni sulla topologia e sullo stato delle reti vicine è di somma importanza per degli instradamenti efficienti. Molti degli algoritmi di routing su IPv6 fanno uso di questa architettura per garantire l'autenticità e l'integrità di dati, abbandonando delle funzionalità integrate che presentano il rischio di appesantire i protocolli.

3.7 Host mobili

Nei prossimi anni avremo un utilizzo sempre più diffuso di host mobili in Internet, ed i servizi di mobilità acquisiranno sempre maggiore importanza per lo scambio di dati. IPv6 cerca di dare una risposta efficiente e stabile anche al problema posto da IP-Mobile specificando un protocollo che permetta ai

nodi di una rete di essere raggiungibili pur cambiando la loro posizione nella Internet IPv6 (vedi [Johnson et al., 2003]).

3.7.1 Scopi di Mobile IPv6

Senza un supporto esplicito per i servizi di mobilità in IPv6, i pacchetti destinati ad un nodo mobile non potrebbero raggiungerlo nel caso in cui esso si trovasse lontano dalla sua rete originaria (*home link*). Per fare in modo che la comunicazione possa essere possibile indipendentemente dai movimenti dell'host, quest'ultimo potrebbe adottare la pratica di cambiare il suo indirizzo IP in modo opportuno ogni volta che approda in una nuova rete, in questo caso, tuttavia, il nodo mobile non sarà in grado di preservare connessioni di trasporto o di livello superiore nel caso in cui cambi la sua posizione (e cambi la rete che lo ospita).

Il supporto alla mobilità in IPv6 ha dunque lo scopo di rendere raggiungibile un certo nodo ovunque esso si trovi ed utilizzando lo stesso indirizzo di livello 3, ovvero il suo *home address*. I pacchetti saranno quindi instradati al nodo di destinazione indifferentemente dal suo punto di connessione ad Internet.

Il protocollo *Mobile IPv6* fornisce servizi di mobilità in modo indipendente dal mezzo trasmissivo adottato, un nodo potrà passare da un segmento di rete Ethernet ad una cella Wireless continuando ad essere raggiungibile sempre con lo stesso indirizzo di rete.

3.7.2 Differenze con Mobile IP per IPv4

Il progetto di Mobile IPv6 trae beneficio dallo sviluppo dei servizi di mobilità per IPv4 (vedi [Perkins and Ed., 1996; Perkins, 1996a,b]), e dalle opportunità offerte da IPv6.

La differenza che si nota maggiormente risiede nel fatto che scompare la necessità di sviluppare router speciali che fungano da "agenti stranieri", come in *Mobile IPv4* (Figura 3.10(a)). *Mobile IPv6* funziona in ogni situazione, senza alcun trattamento speciale da parte del router locale (Figura 3.10(b)). Inoltre l'ottimizzazione degli instradamenti è stata ritenuta una parte fondamentale del protocollo, e può funzionare in modo sicuro anche senza *Security Association* preesistenti.

I servizi di *Neighbor Unreachability Detection* di IPv6 garantiscono la simmetria nella raggiungibilità fra il nodo mobile ed il router locale predefinito.

I pacchetti spediti ad un host mobile, quando lontano dalla rete originale, utilizzano un preambolo di routing di IPv6 piuttosto che l'incapsulamento IP, riducendo l'utilizzo di banda in favore dei dati.

Mobile IPv6 non dipende da nessun protocollo link-layer particolare poiché utilizza i servizi di *Neighbor Discovery*, e ciò contribuisce alla sua robustezza.

Il meccanismo utilizzato in Mobile IPv6 per la scoperta dinamica di un *Home Agent* ritorna al nodo mobile una unica risposta, eliminando il broadcast utilizzato in Mobile IPv4 che fa pervenire al nodo mobile una risposta distinta da ogni *Home Agent* trovato.

3.7.3 Panoramica sul funzionamento

Un nodo mobile dovrebbe essere sempre raggiungibile al suo *home address*, ovunque sia connesso. Il suo *home address* è un indirizzo IP globale assegnato al nodo mobile all'interno del prefisso utilizzato nella rete originaria (*home link*). Quando il nodo in questione si trova nella sua rete riceve il traffico che gli é destinato mediante le regole di instradamento tradizionali di Internet.

Quando un nodo mobile è connesso a qualche rete estranea alla sua rete

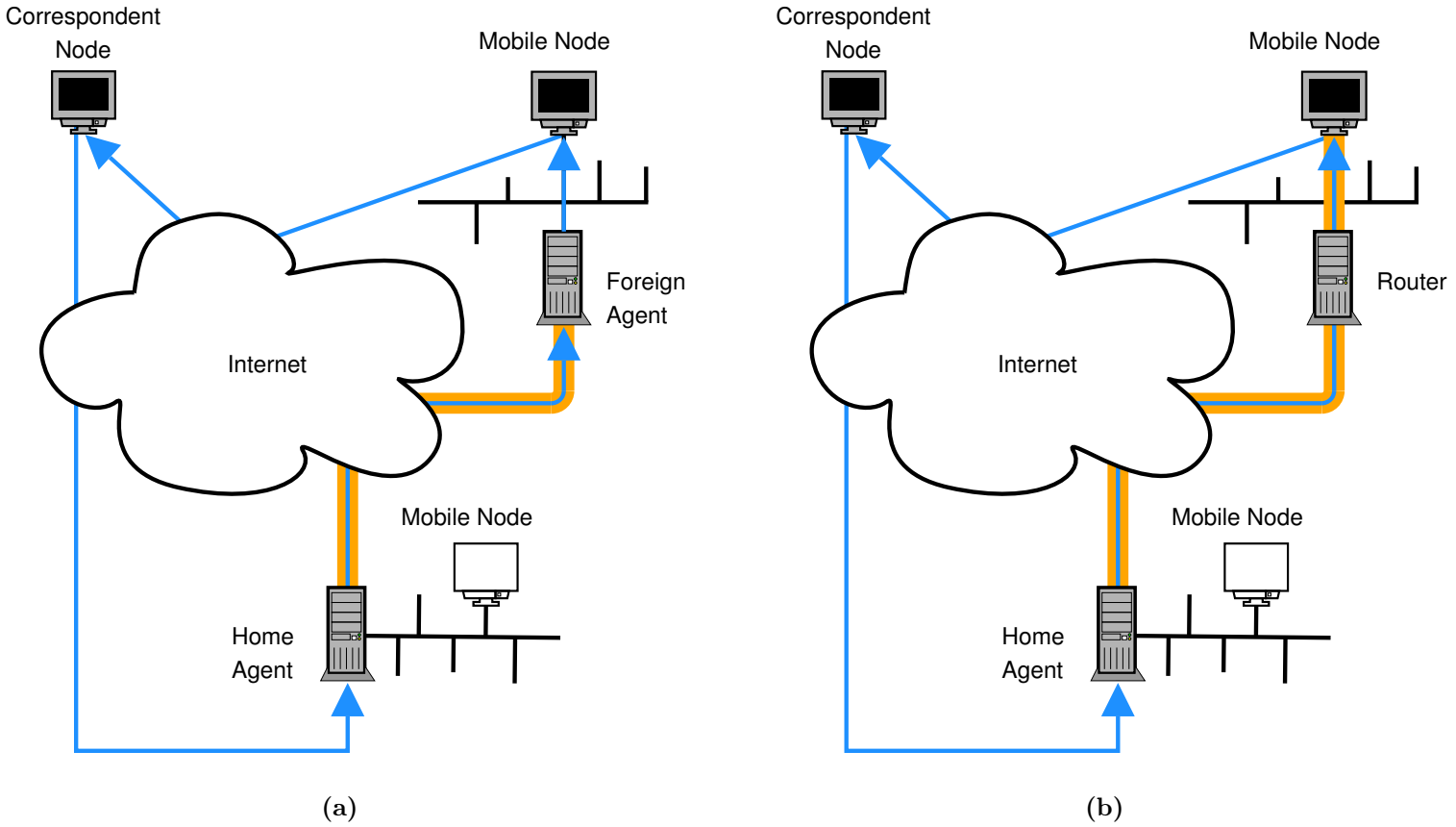

Figura 3.10: **(a)** Mobile IPv4 richiede la presenza di un "Agente Straniero" sulla rete ospite. **(b)** Con Mobile IPv6 non è necessario che il router locale supporti i servizi di mobilità.

originaria esso risulta raggiungibile mediante uno o più indirizzi di *care-of*. Un *care-of address* (CoA) è un indirizzo IP associato al nodo mobile che appartiene al prefisso della rete in cui l'host mobile è ospite. Un nodo mobile può acquisire il suo *care-of address* mediante i meccanismi convenzionali di autoconfigurazione di IPv6 (*stateless* o *stateful*). Fino a quando il nodo mobile non cambia ulteriormente il suo punto d'ingresso ad Internet, i pacchetti destinati al suo indirizzo di *care-of* vengono inoltrati al nodo stesso; quest'ultimo può inoltre accettare traffico da diversi indirizzi di *care-of*, nel caso si trovi in una nuova rete ma la precedente sia ancora raggiungibile.

L'associazione fra l'*home address* di un nodo mobile ed il suo *CoA* viene denominata *binding*. Tale relazione viene registrata presso un router nella rete d'origine detto *Home Agent* (HA). La registrazione è effettuata utilizzando un messaggio di *Binding Update* ed l'*Home Agent* risponde al nodo mobile con un messaggio di *Binding Acknowledgement* dando inizio al servizio di mobilità.

Vi sono due modi possibili per gestire la comunicazione con un host mobile. Il primo non richiede che l'interlocutore del nodo mobile (detto anche *Correspondent node*) supporti esplicitamente le estensioni di Mobile IPv6: si utilizza un tunnel bidirezionale adottando l'incapsulamento di IPv6 in IPv6, in questo modo i pacchetti dal nodo corrispondente sono instradati verso l'*Home Agent* e quindi diretti al nodo mobile mediante un tunnel; i pacchetti di risposta vengono inviati dal nodo mobile ad uno dei suoi *Home Agent* e quindi instradati al *Correspondent Node*. Il secondo modo di procedere viene detto *route optimization*, in questo caso il nodo mobile deve registrare il suo *binding* attuale presso il *Correspondent Node*, cosí facendo qust'ultimo potrà dirigere il traffico direttamente all'indirizzo *care-of* del nodo mobile facendo uso di un nuovo preambolo di estensione per il routing di IPv6.

Instradare il traffico direttamente al nodo mobile permette di intraprendere il percorso più breve e di ridurre, inoltre, la congestione della rete d'origine del nodo mobile, svincolandolo maggiormente da eventuali problemi del suo *home link*.

3.8 Conclusioni

Dopo questa vasta panoramica su IPv6 possiamo concludere che il nuovo IP ha da offrire molte opportunità al networking moderno. Le sue potenzialità non sono ancora emerse e l'adozione diffusa avverrà in modo molto graduale. Le comunicazioni mobili di nuova generazione potranno fare da volano per il lancio di applicazioni peculiari per IPv6, ed una conoscenza dei meccanismi interni al protocollo permetterà di sfruttarne i pregi e di riflettere sui difetti che emergeranno con il tempo. Un confronto con IPv4 anche in termini di prestazioni può essere inoltre utile per valutare un piano di conversione al nuovo protocollo.

Bibliografia

D. Borman, S. Deering, and R. Hinden. *IPv6 Jumbograms*, August 1999. URL `ftp://ftp.isi.edu/in-notes/rfc2675.txt`. RFC 2675.

J. Bound, B. Carpenter, D. Harrington, J. Houldsworth, and A. Lloyd. *OSI NSAPs and IPv6*, August 1996. URL `ftp://ftp.isi.edu/in-notes/rfc1888.txt`. RFC 1888.

R. Coltun, D. Ferguson, and J. Moy. *OSPF for IPv6*, December 1999. URL `ftp://ftp.isi.edu/in-notes/rfc2740.txt`. RFC 2740.

A. Conta and S. Deering. *Internet Control Message Protocol (ICMPv6) for the Internet Protocol Version 6 (IPv6) Specification*, December 1998. URL `ftp://ftp.isi.edu/in-notes/rfc2463.txt`. RFC 2463.

M. Crawford. *Transmission of IPv6 Packets over Ethernet Networks*, December 1998. URL `ftp://ftp.isi.edu/in-notes/rfc2464.txt`. RFC 2464.

S. Deering and R. Hinden. *Internet Protocol, Version 6 (IPv6) Specification*, December 1998. URL `ftp://ftp.isi.edu/in-notes/rfc2460.txt`. RFC 2460.

C.L. Hedrick. *Routing Information Protocol*, June 1988. URL `ftp://ftp.isi.edu/in-notes/rfc1058.txt`. RFC 1058.

R. Hinden and S. Deering. *Internet Protocol Version 6 (IPv6) Addressing Architecture*, April 2003. URL `ftp://ftp.isi.edu/in-notes/rfc3513.txt`. RFC 3513.

R. Hinden, M. O'Dell, and S. Deering. *An IPv6 Aggregatable Global Unicast Address Format*, July 1998. URL `ftp://ftp.isi.edu/in-notes/rfc2374.txt`. RFC 2374.

David B. Johnson, Charles E. Perkins, and Jari Arkko. *Mobility Support in IPv6*, December 2003. URL `http://www.ietf.org/internet-drafts/`.

S. Kent and R. Atkinson. *IP Authentication Header*, November 1998a. URL `ftp://ftp.isi.edu/in-notes/rfc2402.txt`. RFC 2402.

S. Kent and R. Atkinson. *IP Encapsulating Security Payload (ESP)*, November 1998b. URL `ftp://ftp.isi.edu/in-notes/rfc2406.txt`. RFC 2406.

S. Kent and R. Atkinson. *Security Architecture for the Internet Protocol*, November 1998c. URL `ftp://ftp.isi.edu/in-notes/rfc2401.txt`. RFC 2401.

T. Narten, E. Nordmark, and W. Simpson. *Neighbor Discovery for IP Version 6 (IPv6)*, December 1998. URL `ftp://ftp.isi.edu/in-notes/rfc2461.txt`. RFC 2461.

C. Perkins. *IP Encapsulation within IP*, October 1996a. URL `ftp://ftp.isi.edu/in-notes/rfc2003.txt`. RFC 2003.

C. Perkins. *Minimal Encapsulation within IP*, October 1996b. URL `ftp://ftp.isi.edu/in-notes/rfc2004.txt`. RFC 2004.

C. Perkins and Ed. *IP Mobility Support*, October 1996. URL `ftp://ftp.isi.edu/in-notes/rfc2002.txt`. RFC 2002.

J. Postel. *Internet Control Message Protocol*, September 1981. URL `ftp://ftp.isi.edu/in-notes/rfc792.txt`. RFC 792.

Andrew S. Tanenbaum. *Computer Networks*. Prentice Hall International, 1996.

Parte II

Misure di traffico

Capitolo 4

Test di prestazioni

Per misurare l'effettiva utilità del protocollo IPv6 si sono effettuate alcune misure prestazionali atte ad evidenziare i punti di forza del protocollo. Il messaggio che vuole essere trasmesso con questi test è che IPv6 è pronto per essere utilizzato e che farà parte in qualche modo del futuro del networking, che ci piaccia o meno.

4.1 Introduzione

Nel progetto di IPv6 una particolare attenzione è stata rivolta alla gestione delle funzionalità opzionali. La gestione delle opzioni influisce sulla dimensione dei pacchetti e sul lavoro che un router deve operare per stabilire la destinazione di un datagram.

Vale la pena quindi di misurare in qualche modo questa gestione delle opzioni in IPv6 e si è scelto di focalizzare l'attenzione su due estensioni fra le più interessanti di IP: AH ed ESP, che danno vita all'architettura IPsec.

Sia AH che ESP sono dei preamboli di estensione previsti nelle specifiche originali di IPv6; si noti che IPsec è in uso da molto tempo anche nelle reti IPv4 ed un confronto fra le due versioni di IP può mostrare le potenzialità della versione 6.

4.2 Banco di prova

4.2.1 Topologia

La nostra infrastruttura di prova è mostrata in Figura 4.1. Tutti gli apparati coinvolti sono stati configurati in modo da supportare contemporaneamente entrambe le versioni di IP e si è fatto uso di indirizzi privati: nella configurazione IPv6 sono stati usati gli indirizzi appartenenti al prefisso `2001:bd8::/32` riservato per usi di documentazione come riportato in [Huston et al., 2004]. Si ricordi che l'utilizzo di indirizzi *site-local* è stato deprecato da IETF.

Nel seguito vengono mostrati gli indirizzi assegnati ai nodi della suddetta infrastruttura, per maggiore chiarezza si sono partizionati gli indirizzi nei due prefissi `2001:db8:100::/48` e `2001:db8:200::/48` utilizzati rispettivamente per le interfacce verso router o verso degli host, allo stesso modo per IPv4: indirizzi appartenenti alla sottorete `192.168.1x.0/24` appartengono ad interfacce di o verso host, mentre indirizzi `192.168.2x.0/24` appartengono ad interfacce di router verso altri router.

Nella realizzazione della rete illustrata è stato tuttavia utilizzato uno switch per connettere i router e sono state instaurate delle vlan per ottenere dei collegamenti logici distinti. Nella Figura 4.2 si mostra uno schema della struttura fisica della rete confrontata con l'infrastruttura logica vista dai router.

4.2.2 Routing

I router A e B della Figura 4.1 hanno due interfacce verso altri router ed una verso la LAN, gli host interessanti per questi test sono collegati a degli switch che permettono una eventuale espansione delle LAN in questione.

Si riporta per esempio la configurazione del routerA, le configurazioni pre-

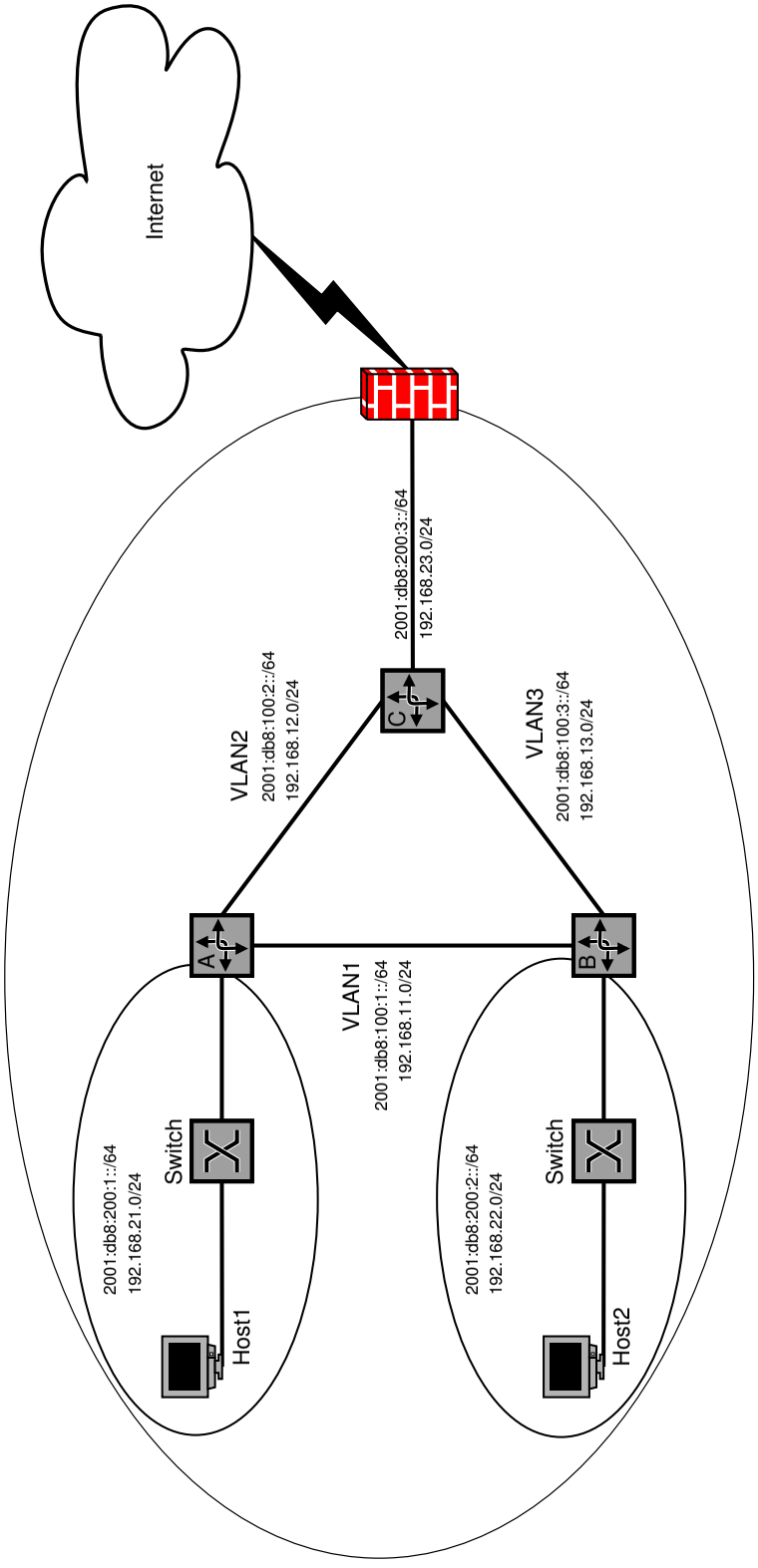

Figura 4.1: La nostra infrastruttura di rete

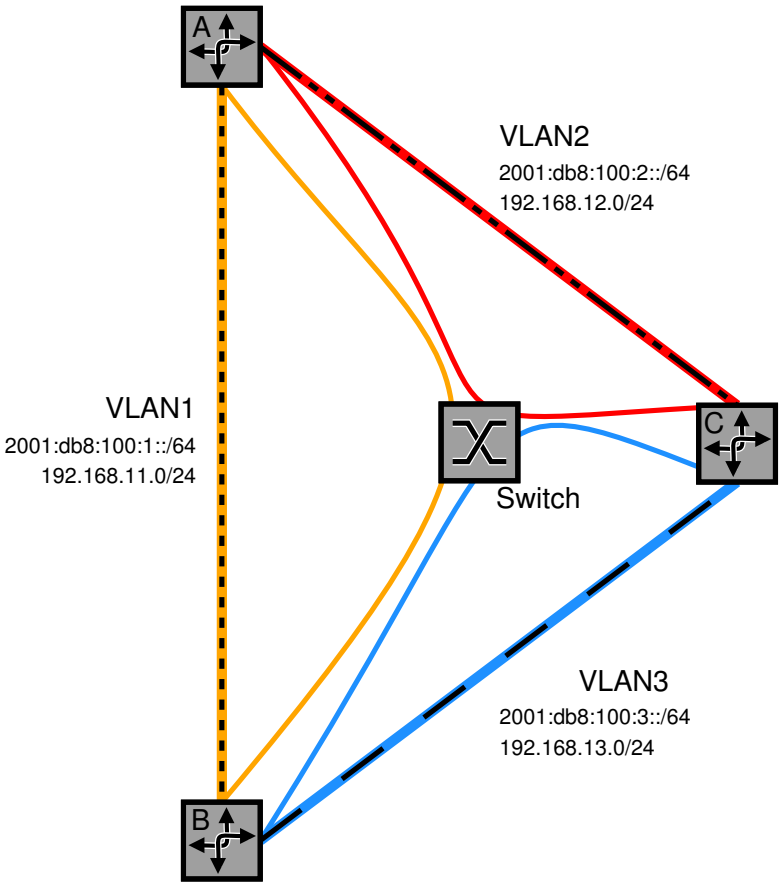

Figura 4.2: La nostra infrastruttura fisica di rete, le linee tratteggiate indicano il link logico realizzato

senti sugli altri router risultano estrememente simili a questa fatte le opportune differenze sugli indirizzi, in accordo allo schema riportato nella sezione 4.2.1.

```
!
! Zebra configuration saved from vty
!   2004/10/24 22:01:41
!
hostname zebra@routerA
password zebra
log syslog
!
interface dummy0
 ipv6 nd suppress-ra
!
interface eth0
 no ipv6 nd suppress-ra
 ipv6 nd prefix 2001:db8:200:1::1/64
!
interface eth0.1
 ipv6 address 2001:db8:100:1::1/64
 ip address 192.168.11.1/24
 ipv6 nd suppress-ra
!
interface eth0.2
 ipv6 address 2001:db8:100:2::1/64
 ip address 192.168.12.1/24
 ipv6 nd suppress-ra
!
interface eth0.3
 ipv6 nd suppress-ra
!
interface eth0.4
 ipv6 nd suppress-ra
!
interface eth0.10
 ipv6 nd suppress-ra
!
interface eth0.15
 ipv6 nd suppress-ra
!
interface lo
!
interface sit0
 ipv6 nd suppress-ra
!
ip forwarding
ipv6 forwarding
```

```
!
line vty
!
```

Le informazioni sulla raggiungibilità e sugli instradamenti sono scambiate fra i router in modo dinamico adottando un protocollo di routing *Distance Vector* (**RIPv2** per IPv4 e **RIPng** per IPv6).

```
!
! Zebra configuration saved from vty
!   2004/10/24 00:22:44
!
hostname ripd@routerA
password zebra
log syslog
!
router rip
 redistribute kernel
 redistribute connected
 redistribute static
 network 192.168.11.0/24
 network 192.168.12.0/24
 neighbor 192.168.11.2
 neighbor 192.168.12.2
!
line vty
!
```

```
!
! Zebra configuration saved from vty
!   2004/10/24 21:54:57
!
hostname ripngd@routerA
password zebra
log syslog
!
router ripng
 network 2001:db8:100:1::1/64
 network 2001:db8:100:2::1/64
 redistribute kernel
 redistribute connected
 redistribute static
!
line vty
!
```

4.2.3 Strumenti adottati

Software di base

I nodi coinvolti nel test adoperano il kernel **Linux** e gli strumenti di base sono quelli inclusi in **Debian GNU/Linux**, la versione del kernel adottata è la **2.6.9-rc2** con una patch del progetto **USAGI** che rende lo stack IPv6 di linux più conforme agli standard e più ricco di funzionalità rispetto all'implementazione base.

Sono state attivate le opzioni necessarie alla gestione della versione 6 di IP, in particolare si sono attivate le seguenti funzionalità:

- Supporto di base al protocollo IPv6
- Funzionalità essenziali ad IPsec e gestione dei preamboli AH ed ESP
- Gestione della *Router Preference List* sugli host, funzionalità utile nel caso su una LAN vi fossero più router
- Supporto ai tunnel ed all'incapsulazione IP, anche se non se ne è fatto uso nei test prestazionali.

I router sono stati inoltre corredati dei servizi di instradamento necessari al corretto funzionamento della nostra *internetwork*.

Software applicativo

Il primo strumento software utilizzato durante lo svolgimento dei test è stato **netperf**, una suite di prove per la misura delle prestazioni di un collegamento di rete.

I test applicativi sono stati svolti mediante l'uso di **wget**: un client http che ci è servito per il trasferimento di un file di grosse dimensioni residente sul web server.

Il web server adottato è stato **Apache2**, scelto per il supporto IPv6. **Apache2** è ricchissimo di funzionalità e permette un accurato controllo sulle connessioni, tuttavia per questo tipo di test la configurazione standard è risultata sufficiente.

I test con IPsec sono stati effettuati adoperando lo stack IPsec incluso nel kernel Linux, sono state configurate le opportune *Security Policies* per entrambe le versioni di IP e si è adoperato il demone **racoon** per l'instauramento delle *Security Association*.

4.3 Progettazione dei test

Si è effettuato il test fra due i host nella nostra internet, come mostrato anche in figura Figura 4.3 per confrontare l'efficienza di IPv4 ed IPv6 in termini di banda passante sia su un collegamento standard, sia abilitando le opzioni per la sicurezza. Si è fatto uso intenso di strumenti *OpenSource* che potessero essere adattati al tipo di test in questione.

Lo scopo dei test è stato quello di mostrare che IPv6 non è un protocollo "pesante" nonostante il suo preambolo ecceda in dimensione quello di IPv4, e che, anzi, il nuovo protocollo mostra una certa agilità nella gestione delle opzioni.

4.3.1 TCP STREAM test

Un primo test ha simulato un flusso TCP fra i due host utilizzando messaggi di dimensione diversa per misurare l'effettivo utilizzo della banda passante dei due protocolli IP anche in relazione alla frammentazione. Questo test è stato effettuato specificando un numero minimo e massimo di iterazioni ed un certo intervallo di confidenza per i risultati grazie al software **netperf** che verrà descritto nella sezione 5.1.1. La simulazione può infatti risentire

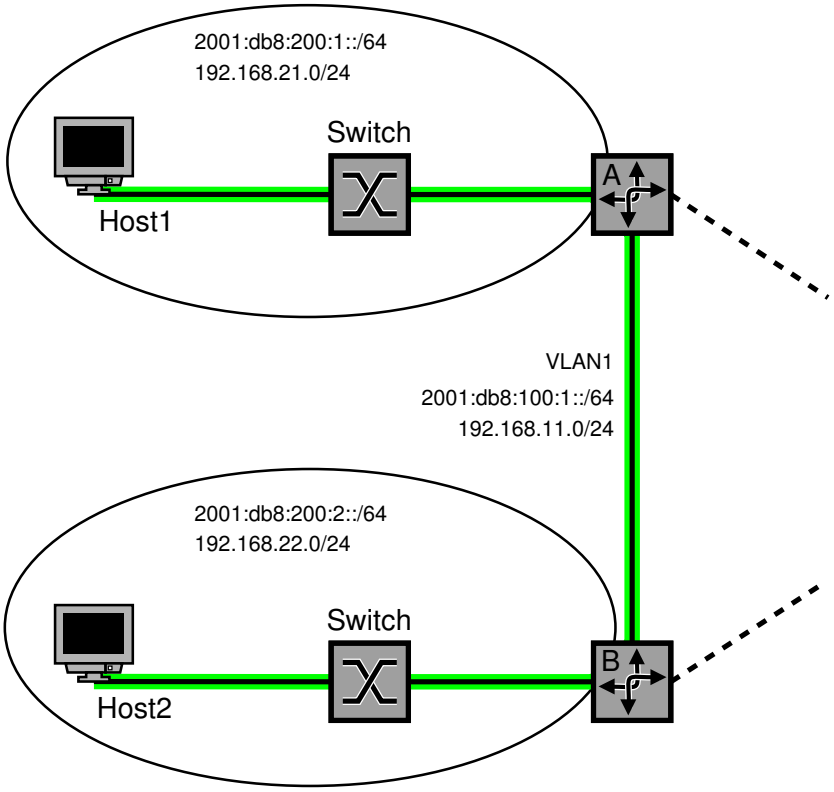

Figura 4.3: Scenario delle prove *host-to-host*

di eventuali fattori variabili introdotti dalle logiche di gestione del sistema operativo e dai programmi applicativi, ripetere più volte i test ed analizzarne i risultati medi aiuta ad avere una visione più valida della reale occupazione della rete.

Si è ripetuto questo ciclo di test una seconda volta, per misurare l'utilizzo del processore da parte del sottosistema di rete del sistema operativo a seconda della versione di IP utilizzata. Avere una idea di come un protocollo risulta computazionalmente più impegnativo dell'altro è utile nella rincorsa ai tempi di latenza sempre più bassi: vogliamo che l'informazione sia recapitata, e possibilmente nel minor tempo possibile.

4.3.2 Test applicativi

Si è scelto inoltre di effettuare anche un test "applicativo" simulando uno scenario reale di utilizzo della rete, ed utilizzando software applicativi generici invece di software dedicati al *benchmarking*. L'uso del test applicativo mette in conto oscillazioni e variazioni episodiche dovute alla variabilità dell'ambiente di calcolo, scopo di questo test è appunto quello di fornire una fotografia istantanea di come i due protocolli si possono confrontare nelle applicazioni vere.

Questo tipo di misura è stato effettuato mediante l'uso di un *web server* e di un client http che prelevasse un file di grandi dimensioni dal server, si è misurata la banda occupata mediante le indicazioni fornite ad intervalli periodici dal client in questione.

4.3.3 Test con IPsec

Questo doppio test è stato nuovamente ripetuto abilitando i servizi di IPsec ESP e quindi ancora in una configurazione AH+ESP per analizzare il com-

portamento del protocollo v6 in presenza di diversi preamboli di estensione.

I servizi di sicurezza sono stati attivati in *transport mode* così da non utilizzare l'incapsulazione IP-in-IP ed adoperare i preamboli di estensione per quanto riguarda IPv6. Si illustra in Figura 4.4 un esempio di come appaia un pacchetto dopo l'uso di AH in transport mode ed in tunnel mode.

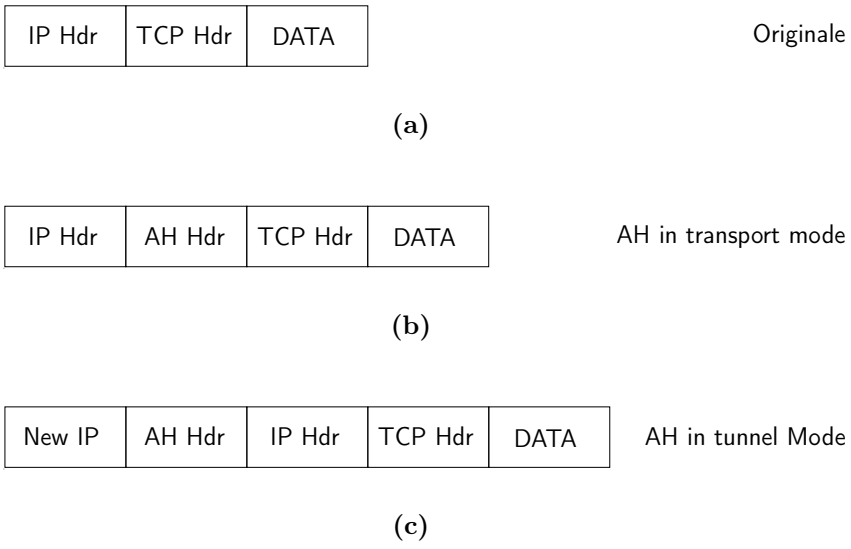

Figura 4.4: 4.4(a) un pacchetto IP, 4.4(b) un pacchetto IP con AH in transport mode, 4.4(c) un pacchetto IP con AH in tunnel mode. in questo caso il pacchetto originale è sottoposto ai servizi di AH in ogni sua parte

Bibliografia

Debian, il sistema operativo universale. `http://www.debian.org`.

Gnu wget. `http://www.gnu.org/software/wget/wget.html`.

The linux kernel archives. `http://www.kernel.org`.

Netperf homepage. `http://www.netperf.org`.

Usagi (universal playground for ipv6). `http://www.linux-ipv6.org`.

André Grüneberg. Aufbau eines Linux-basierten IPv6-Routers zum Einsatz im Umfeld eines Internet-Service-Providers, 2004. URL `http://www.grueneberg.de/andre/diplom/`.

G. Huston, A. Lord, and P. Smith. *IPv6 Address Prefix Reserved for Documentation*, July 2004. URL `ftp://ftp.isi.edu/in-notes/rfc3849.txt`. RFC 3849.

S. Kent and R. Atkinson. *IP Authentication Header*, November 1998a. URL `ftp://ftp.isi.edu/in-notes/rfc2402.txt`. RFC 2402.

S. Kent and R. Atkinson. *IP Encapsulating Security Payload (ESP)*, November 1998b. URL `ftp://ftp.isi.edu/in-notes/rfc2406.txt`. RFC 2406.

S. Kent and R. Atkinson. *Security Architecture for the Internet Protocol*, November 1998c. URL `ftp://ftp.isi.edu/in-notes/rfc2401.txt`. RFC 2401.

Capitolo 5

Configurazione e lancio dei test

5.1 Collezione dei risultati

I software utilizzati nelle misure di traffico producono un output non sempre adatto ad essere visualizzato comodamente sotto forma di grafico, si è scelto quindi di realizzare degli script di shell che trattassero opportunamente questo output per porlo in una forma che risultasse comprensibile al software **gnuplot** che produce i grafici. Altro scopo, non secondario, di questi script è quello di semplificare l'utilizzo degli strumenti a disposizione ed eventualmente di gestire reiterazioni dei test al variare di qualche parametro.

5.1.1 NetPerf

Il software **netperf** offre numerosi tipi di test: misure di prestazioni per TCP su IP e UDP su IP, test di *request and response* e prove relative anche al protocollo ATM; le misure d'interesse in questa sperimentazione sono state quelle riguardanti TCP/IP e TCP/IPv6. Sono state apportate diverse modifiche agli script d'esempio disponibili con **netperf**, il risultato può considerarsi un lavoro originale poiché contiene relativamente pochi riferimenti allo script originale utilizzato.

Il richiamo ai test per IPv6 e la scrittura dei risultati in un file sono stati

aggiunti allo script `netperf_v4_v6_tcp_stream.sh` utilizzando una nuova funzione, come mostrato nel seguito:

```
# Una funzione che esegue i test
function start_test {

  # Argomenti
  # 1. Una descrizione del test
  # 2. L'indirizzo dell'host remoto
  # 3. Tipo di test
  # 4. Un nome di base per il file dei dati
  [ "$#" -ne 4 ] && { echo "function error"; exit 2; }

  DESCRIPTION="$1"
  REM_HOST="$2"
  TEST_TYPE="$3"
  LOGNAME="$4"

  # Lancio dei test
  echo
  echo Start $DESCRIPTION tests...

  for SOCKET_SIZE in ${SOCKET_SIZES}
  do
    LOGFILE=${LOGNAME}_${SOCKET_SIZE}.log

    echo "$DATA_HDR" > $LOGFILE

    for SEND_SIZE in $SEND_SIZES
      do
      echo ------------------------
      echo
      # mostriamo il comando

      cmd="$NETHOME/netperf $NO_HDR $PORT -l $TEST_TIME
           -H $REM_HOST -t $TEST_TYPE $LOC_CPU $REM_CPU
           $STATS_STUFF $UNITS --
           -m $SEND_SIZE -s $SOCKET_SIZE -S $SOCKET_SIZE"

      echo $cmd
      echo

      # Ogni esecuzione del comandi ripetera' il test
```

```
        # in accordo con l'intervallo di confidenza indicato
        eval $cmd | tee -a $LOGFILE

    done
  done
}
```

I test sono stati infine avviati richiamando la funzione `start_test`:

```
# Test per IPv4
start_test "IPv4" $REM_HOSTv4 "TCP_STREAM" $v4_test_name

# Test per IPv6
start_test "IPv6" $REM_HOSTv6 "TCPIPV6_STREAM" $v6_test_name
```

Lo script fa chiaramente uso di variabili per gestire gli indirizzi del server e le dimensioni dei messaggi, verrà presentato nella sua interezza nella sezione A.1.1.

5.1.2 Wget

Lo script per **wget** (`test_wget.sh`) ne manipola l'output per ottenere una tabella che descriva l'uso di banda usata dal protocollo applicativo http, tale script fa uso degli strumenti classici offerti dai sistemi di tipo UNIX per la manipolazione di flussi di testo, lo script nella sua interezza è presentato nella sezione A.1.2, se ne presentano qui solo alcuni stralci per mostrare come si sono trattati i dati prodotti dal programma.

Le informazioni sull'uso di banda vengono riposte in un file temporaneo:

```
# File da un indirizzo IPv4
wget -r --progress=dot:mega \
     --bind-address="$IPV4_CLIENT" \
     "http://${IPV4_SERVER}/${FILENAME}" \
     -O /dev/null 2> ${TEMPFILE}
```

Tale file viene quindi manipolato per produrre una tabella che possa essere graficata facilmente.

```
# Conversione del file in un formato utile per gnuplot
cat ${TEMPFILE} | sed -e "s/\.\.\.\.\.\.\.\.//g" -e "s/^\ *//g" |\
    tr -s ' ' | grep ^[0-9] | grep -v ':' | cut -d ' ' -f 1,3 |\
    tr -d 'K' > ${IPv4_DATAFILE}
```

Per evitare fenomeni di compressione dei dati il file da trasferire, residente sul web server, è stato creato con contenuto casuale in questo modo:

```
#!/bin/sh
# 2^30 bytes
dd bs=1K count=1M if=/dev/urandom of=test_file
```

5.1.3 Graficare i risultati: gnuplot

Per la produzione dei grafici si è adoperato *gnuplot* in modalità non interattiva, utilizzando le funzionalità di scripting offerte dal software.

Il file di dati prodotto dal primo test con **netperf** è strutturato in colonne in questo modo:

```
# Test:
# Begin Test for socket size
#
# Recv   Send    Send
# Socket Socket  Message  Elapsed
# Size   Size    Size     Time     Throughput
# bytes  bytes   bytes    secs.    10^6bits/sec
131072  65536    512    120.00     93.78
131072  65536   1024    120.00     93.90
131072  65536   2048    120.00     93.94
131072  65536   4096    120.00     93.91
131072  65536   8192    120.00     93.90
131072  65536  16384    120.00     94.00
131072  65536  32768    120.00     93.94
131072  65536  65536    120.00     93.93
```

I primi tre campi fanno riferimento alla dimensione dei dati trasmessi, vi sono poi le indicazioni sul tempo di test e sulla banda utilizzata (il *throughput*), per la produzione del grafico si sono utilizzati il terzo ed il quinto

campo, ovvero la dimensione del messaggio trasmesso e l'utilizzo di banda ottenuto:

```
# Plot
set yrange [0:*]
plot "ipv4_tcp_32768.log" using 3:5 w lp title "IPv4 ", \
     "ipv6_tcp_32768.log" using 3:5 w lp title "IPv6 "
```

Il test relativo alla misura dell'utilizzo del processore produce un output lievemente diverso da quello del test precedente:

```
# Test:
# Begin Test for socket size
#
# Recv   Send    Send                          Utilization
# Socket Socket  Message  Elapsed              Send     Recv
# Size   Size    Size     Time     Throughput  local    remote
# bytes  bytes   bytes    secs.    10^6bits/s  % S      % S
217088  65536    512    120.00       93.86   18.25    -1.00
131072  65536   1024    120.00       93.90   16.39    -1.00
217088  65536   2048    120.00       93.94   15.70    -1.00
217088  65536   4096    120.00       93.90   14.64    -1.00
217088  65536   8192    120.00       93.88   13.34    -1.00
217088  65536  16384    120.00       94.00   13.35    -1.00
217088  65536  32768    120.00       93.86   13.43    -1.00
217088  65536  65536    120.00       93.93   13.27    -1.00
```

In questo caso i dati di nostro interesse sono la dimensione del messaggio inviato, l'utilizzo di banda e la percentuale di utilizzazione del processore sul client, non si è misurato in questo caso l'uso di processore sul server. La formula utilizzata per i dati sulle ordinate del grafico è la seguente:

$$y = \frac{banda}{\%CPU}$$

come si vede dal frammento dello script per **gnuplot**:

```
# Plot
set yrange [0:*]
plot "ipv4_tcp_cpu_32768.log" using 3:( ($5) / ($6) ) w lp title "IPv4 ", \
     "ipv6_tcp_cpu_32768.log" using 3:( ($5) / ($6) ) w lp title "IPv6 "
```

Il caso del test applicativo si presenta analogo a quelli descritti finora, il file di dati presenta in prima colonna la quantità di dati trasmessa ed in seconda colonna la banda espressa in **MiB/s** (2^{20} bytes al secondo) al secondo:

```
0 9.67
3072 11.23
6144 11.22
9216 11.23
12288 11.22
...
```

è opportuna una conversione del secondo valore in Mbps (10^6 bit al secondo) per confrontarlo con i risultati precedenti, si utilizza la formula

$$Mb = MiB \times 8 \times \frac{2^{20}}{10^6}$$

e ciò viene fatto durante la creazione del grafico:

```
# Plot
set yrange [0:*]
plot "ipv4.data" using 1:(($2) * 8. * (2.**20)/(10.**6)) w l lt 1 title "IPv4
    Transfer ", \
     "ipv6.data" using 1:(($2) * 8. * (2.**20)/(10.**6)) w l lt 2 title "IPv6
    Transfer "
```

5.2 Lancio dei test e risultati

I test con **netperf** descritti precedentemente sono stati lanciati dal client con la seguente riga di comando:

```
./netperf_v4_v6_tcp_stream.sh $SERVERv4 $SERVERv6
```

ed i test per la valutazione dell'utilizzo del processore:

```
./netperf_v4_v6_tcp_stream.sh $SERVERv4 $SERVERv6 "CPU"
```

Il test con **wget** è stato avviato con lo script `test_wget.sh`:

```
./test_wget.sh $FILENAME $SERVERv4 $CLIENTv4 \
                        $SERVERv6 $CLIENTv6
```

5.2.1 Senza IPsec

Come si vede in Figura 5.1 le prestazioni di IPv6 sono pittosto simili a quelle di IPv4, di poco inferiori a causa della maggiore dimensione del preambolo.

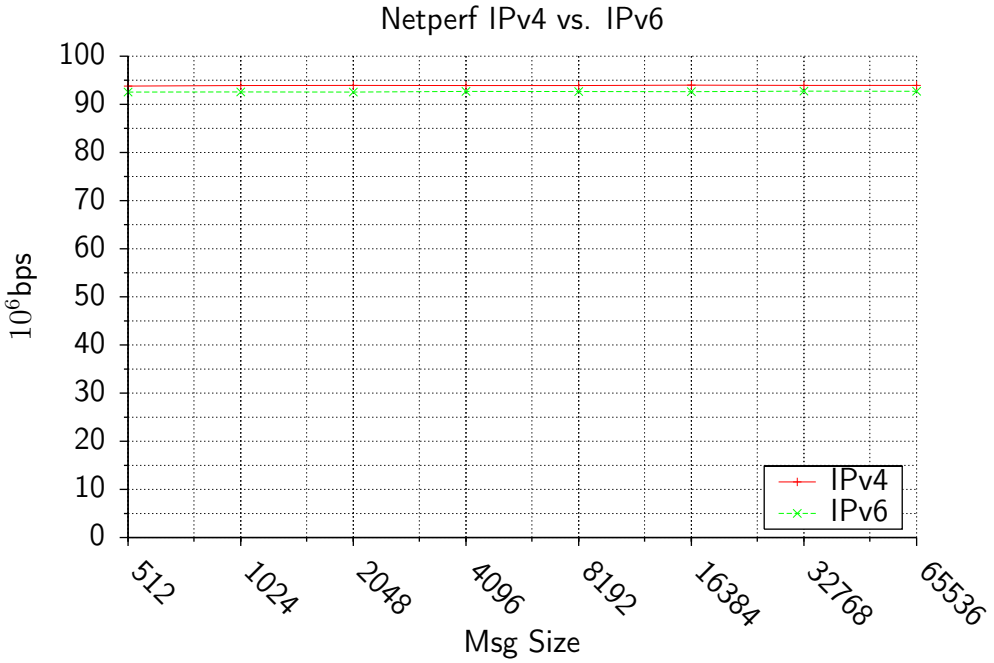

Figura 5.1: Test con netperf

I risultati in relazione all'utilizzo della CPU non sono molto differenti, come si vede in Figura 5.2, a parte qualche caso in cui l'utilizzo del processore risulta particolarmente basso.

I test con wget ribadiscono le misure effettuate con netperf: IPv6, in condizioni normali e su reti locali a 100Mbps, ha prestazioni lievemente inferiori ad IPv4.

Si tenga presente che nel caso di *link* con capacità maggiori (nell'oridine

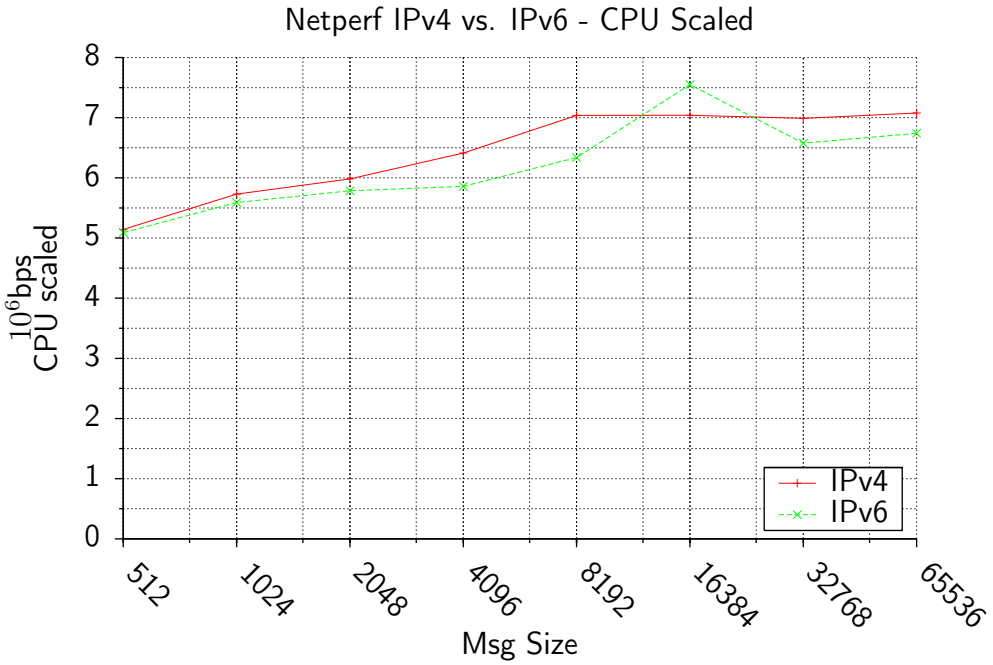

Figura 5.2: Test con netperf in relazione all'utilizzo di CPU

dei Gbps) il risparmio in termini computazionali permesso da IPv6 nelle pratiche di routing potrebbe incrementare significativamente la performance nel nuovo protocollo.

Il grafico in Figura 5.3 mostra che il test applicativo conferma sostanzialmente i risultati precedenti, non vi sono grandi scarti prestazionali ma IPv4 è ancora in leggero vantaggio.

Questo test su http non simula tuttavia il caso di connessioni multiple al server, tipico di applicazioni web, ma per i nostri scopi non si é ritenuto significativo questo tipo di misura influenzato pesantemente dal protocollo di trasporto.

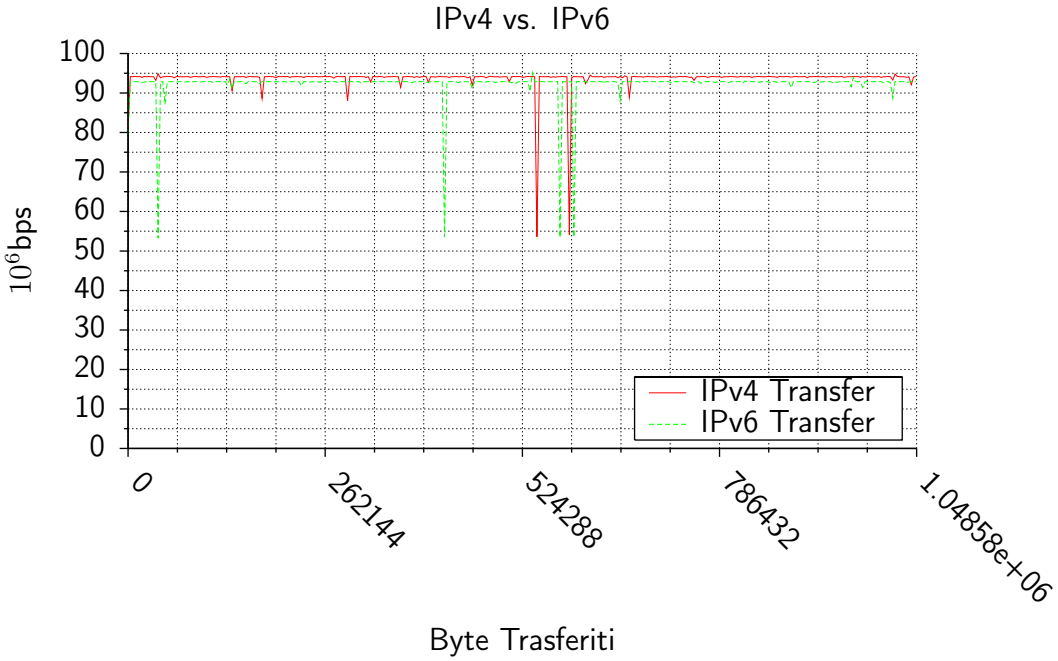

Figura 5.3: Test con wget

5.2.2 Risultati con IPsec

Quando entra in scena l'utilizzo delle funzionalità per la sicurezza la situazione risulta lievemente diversa, IPsec su IPv6 non va tanto male, le differenze di prestazioni con IPv4 si assottigliano.

ESP

Con solo ESP attivato abbiamo una situazione illustrata in Figura 5.4

La differenza dovuta al preambolo di dimensioni maggiori diventa ora meno determinanate al crescere della dimensione totale del pacchetto con ESP.

Osservando la Figura 5.5 si intuisce inoltre che anche l'uso del processore sia minore in IPv6, meno campi del preambolo da sottoporre alle pratiche di crittazione.

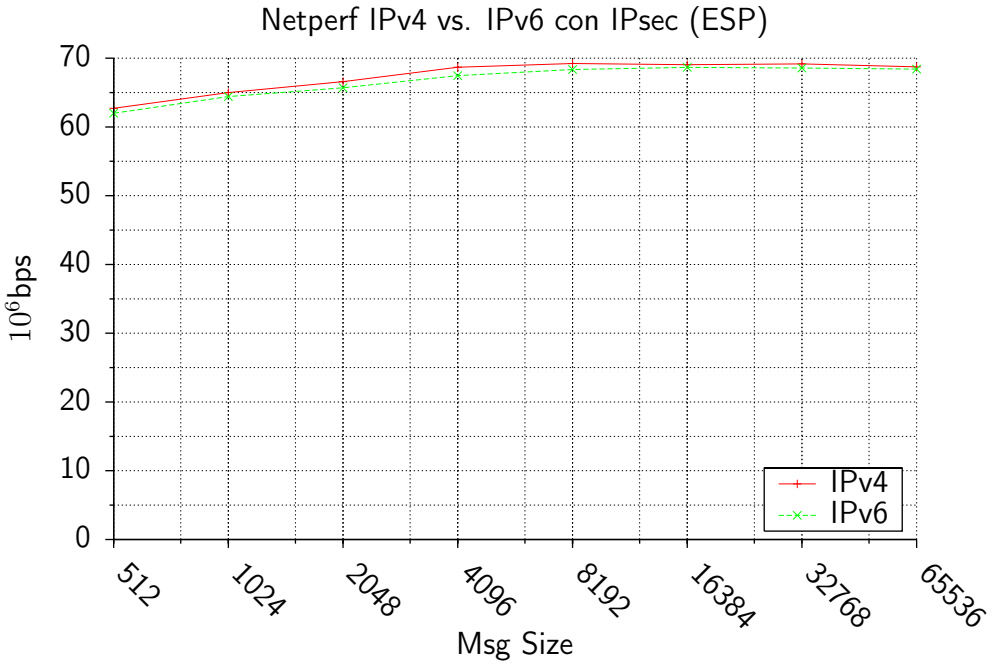

Figura 5.4: Test con netperf con IPsec (ESP)

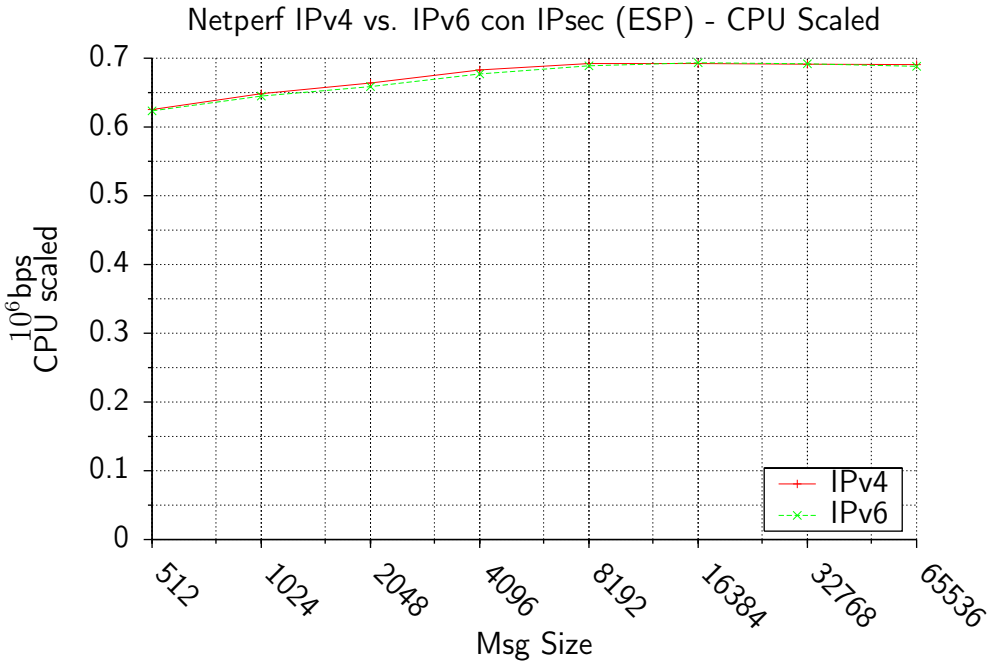

Figura 5.5: Test con netperf con IPsec (ESP) in relazione all'utilizzo di CPU

All'atto pratico tuttavia non vi sono differenze apprezzabili usando http, si veda la Figura 5.6, il divario resta comunque accetabilmente contenuto.

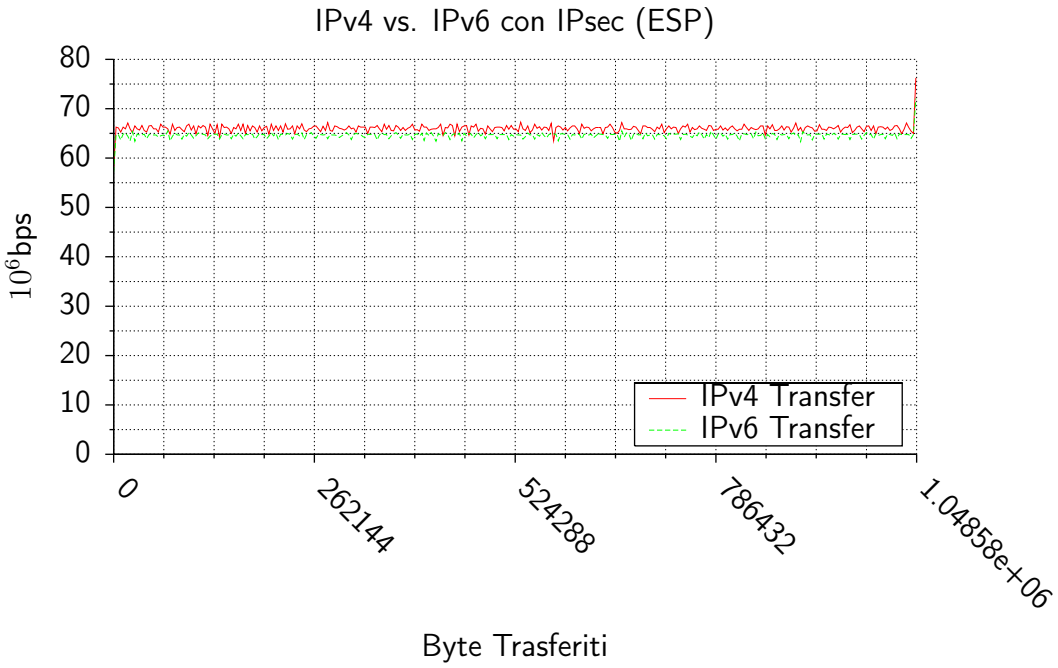

Figura 5.6: Test con wget con IPsec (ESP)

AH+ESP

Nel caso in cui sia AH che ESP siano attivi, IPv6 si rivela più inefficiente, questo risualtato potrebbe essere attribuito ad una implementazione software ancora inadeguata, o semplicemente all'uso del preambolo di autenticazione sottoposto ad un carico maggiore data la maggiore dimensione del preambolo.

In Figura 5.7 si vede infatti un calo quando le dimensioni dei messaggi sono di media dimensione.

Risultati confermati (v. Figura 5.8) anche se relazionati all'utilizzo del processore.

Il test applicativo di Figura 5.9 ci comunica che all'atto pratico l'uso di AH ed ESP non costrituiscono una grave penalizzazione in termini di differenza

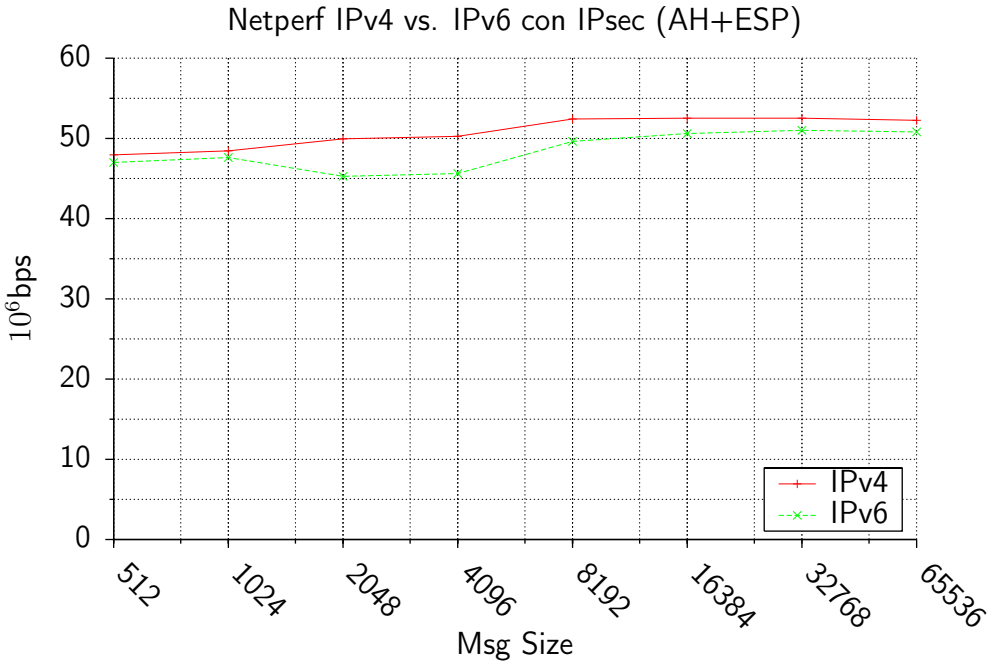

Figura 5.7: Test con netperf con IPsec (AH+ESP)

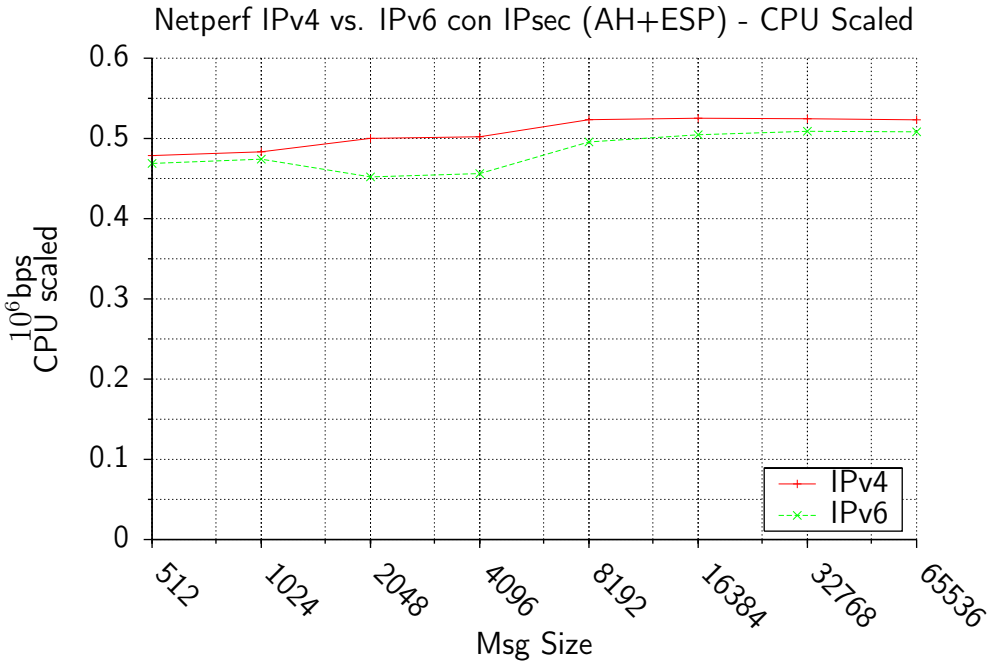

Figura 5.8: Test con netperf con IPsec (AH+ESP) in relazione all'utilizzo di CPU

di prestazioni fra IPv4 ed IPv6, l'*overhead* introdotto dai due servizi di IPsec risulta pesante indipendentemente dal protocollo utilizzato.

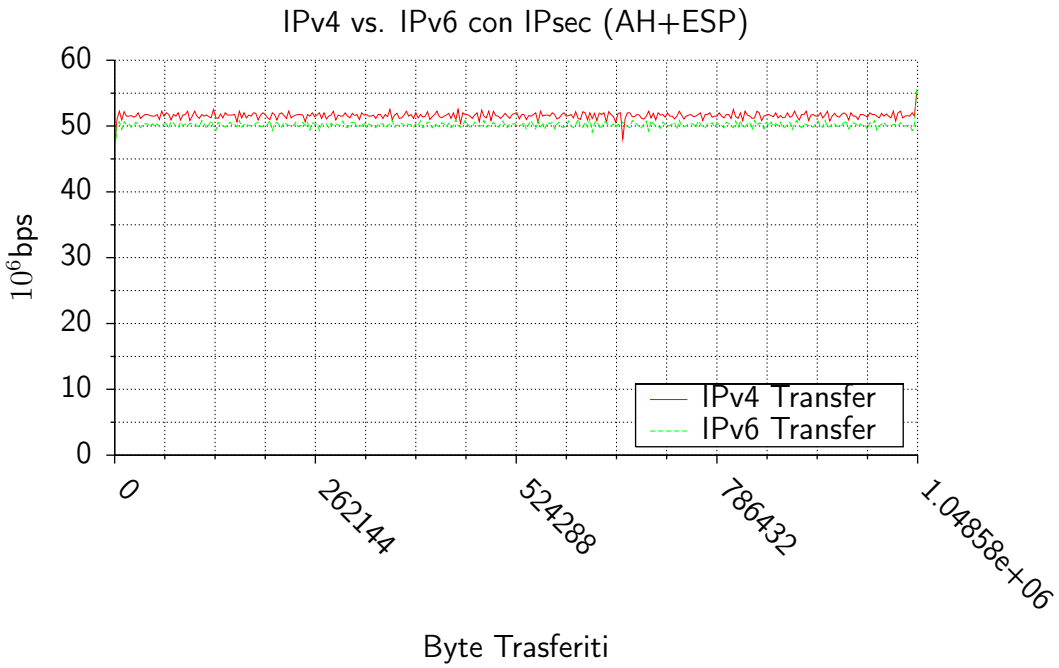

Figura 5.9: Test con wget con IPsec (AH+ESP)

5.3 Osservazioni

Come mostrato dai grafici, il nuovo protocollo non risulta significativamente più pesante di IPv4, in questo caso l'utilizzo del processore è stato determinante unicamente nei test con IPsec poiché, generalmente, il limite di banda è stato dettato dalle interfacce di rete. Ulteriori test interessanti da svolgere potrebbero essere quelli convolgenti infrastrutture di routing ben più complesse, magari con link da 1Gbps o 10Gbps, in questi casi il processore é sottoposto a carichi maggiori nel trattamento del flusso di pacchetti.

Altri test proponibili sono quelli relativi alla misura dei tempi di latenza: in contesti di reti per il supercalcolo, ad esempio, in cui sono adoperate

infrastrutture di routing fra i nodi del supercalcolatore, avere delle latenze minori negli instradamenti può ripercuotersi positivamente sulle prestazioni dell'intero sistema distribuito, in questi scenari anche l'utilizzo di pacchetti di dimensione maggiore (*Jumbograms*) potrebbe contribuire all'eliminazione dei colli di bottiglia.

Bibliografia

Gnu wget. `http://www.gnu.org/software/wget/wget.html`.

Gnuplot homepage. `http://www.gnuplot.info/`.

Netperf homepage. `http://www.netperf.org`.

Appendice A

Script di shell e configurazioni degli host

Si riportano nel seguito gli script usati per il lancio dei test e per la configurazione degli host, se ne sono illustrati i dettagli più significativi nelle sezioni precedenti, questo capitolo si pone lo scopo di testimoniare il lavoro svolto durante la preparazione dei test.

A.1 Script di shell

A.1.1 Lancio dei test con netperf

Lo script relativo al lancio dei test con *netperf* viene qui presentato interamente, si guardi la documentazione interna allo script per i dettagli sulle operazioni svolte passo dopo passo.

```
#!/bin/sh
#
# usage:
#    ./netperf_v4_v6_tcp_stream.sh <ipv4-addr> <ipv6-addr> [CPU]
#

# Controllo degli argomenti
( [ "$#" -lt 2 ] || [ "$#" -gt 3 ] ) && \
{ echo "usage: $0 <ipv4-addr> <ipv6-addr> [CPU]"; exit 1; }
```

```
# path dei programmi
NETHOME=/usr/bin

# Porta su cui ascolta netserver
#PORT="-p some_other_portnum"
PORT=""

# Tempo massimo di ogni test
TEST_TIME=120

# Numero massimo e minimo di iterazioni (-i)
# Livello di confidenza (99 o 95) ed intervallo (in percentuale)
STATS_STUFF="-i 15,2 -I 95,5"

# Unita' di misura usata nell'output
UNITS="-f m"

# Dimensione della socket
SOCKET_SIZES="32768"

# Dimensioni dei messaggi
SEND_SIZES="00512 01024 02048 04096 08192 16384 32768 65536"

# Se vi sono tre parametri settiamo alcune variabili per il test di
# misurazione della CPU

if [ "$#" -eq 3 ]; then
  REM_HOSTv4=$1
  REM_HOSTv6=$2
  LOC_CPU="-c"
  REM_CPU="-C"
  # Nomi dei test
  v4_test_name="ipv4_tcp_cpu"
  v6_test_name="ipv6_tcp_cpu"
  # Descrizione dei dati del test
  DATA_HDR="# Test: $DESCRIPTION
# Begin Test for socket size $SOCKET_SIZE
#
# Recv   Send    Send                          Utilization
    Service Demand
# Socket Socket  Message  Elapsed              Send     Recv
    Send    Recv
# Size   Size    Size     Time     Throughput  local    remote
```

```
    local   remote
# bytes  bytes   bytes    secs.    10^6bits/s  % S      % S      us/
    KB   us/KB"
fi

if [ "$#" -eq 2 ]; then
  REM_HOSTv4=$1
  REM_HOSTv6=$2
  # Nomi dei test
  v4_test_name="ipv4_tcp"
  v6_test_name="ipv6_tcp"
  # Descrizione dei dati del test
  DATA_HDR="# Test: $DESCRIPTION
# Begin Test for socket size $SOCKET_SIZE
#
# Recv   Send    Send
# Socket Socket  Message  Elapsed
# Size   Size    Size     Time     Throughput
# bytes  bytes   bytes    secs.    10^6bits/sec"
fi

# Disabilitiamo le intestazioni
NO_HDR="-P 0"

# Una funzione che esegue i test
function start_test {

  # Argomenti
  # 1. Una descrizione del test
  # 2. L'indirizzo dell'host remoto
  # 3. Tipo di test
  # 4. Un nome di base per il file dei dati
  [ "$#" -ne 4 ] && { echo "function error"; exit 2; }

  DESCRIPTION="$1"
  REM_HOST="$2"
  TEST_TYPE="$3"
  LOGNAME="$4"

  # Lancio dei test
  echo
  echo Start $DESCRIPTION tests...
```

```
  for SOCKET_SIZE in ${SOCKET_SIZES}
  do
    LOGFILE=${LOGNAME}_${SOCKET_SIZE}.log

    echo "$DATA_HDR" > $LOGFILE

    for SEND_SIZE in $SEND_SIZES
      do
      echo ------------------------------------
      echo
      # mostriamo il comando

      cmd="$NETHOME/netperf $NO_HDR $PORT -l $TEST_TIME
           -H $REM_HOST -t $TEST_TYPE $LOC_CPU $LOC_RATE
           $STATS_STUFF $UNITS --
           -m $SEND_SIZE -s $SOCKET_SIZE -S $SOCKET_SIZE"

      echo $cmd
      echo

      # Ogni esecuzione del comandi ripetera' il test
      # in accordo con l'intervallo di confidenza indicato
      eval $cmd | tee -a $LOGFILE

    done
  done
}


# Test per IPv4
start_test "IPv4" $REM_HOSTv4 "TCP_STREAM" $v4_test_name

# Test per IPv6
start_test "IPv6" $REM_HOSTv6 "TCPIPV6_STREAM" $v6_test_name
```

A.1.2 Lancio dei test con wget

I test applicativi sono risultati più rapidi da preparare ed eseguire, è bastato trattare opportunamente l'output di **gnuplot**:

```
#!/bin/sh
# Test per wget:
```

```
#   ./test_wget.sh <file> \
#         <server-v4> <client-v4> <server-v6> <client-v6>

set -x

[ "$#" -eq 5 ] || \
{ echo "usage: $0 <file> <server-v4> <client-v4> <server-v6> <client-
    v6>"; exit 1; }

# Argomenti

FILENAME="$1"

IPV4_SERVER="$2"
IPV4_CLIENT="$3"

IPV6_SERVER="$4"
IPV6_CLIENT="$5"

# File temporaneo per i  risultati
TEMPFILE=‘tempfile‘

IPv4_DATAFILE=ipv4.data
IPv6_DATAFILE=ipv6.data

# File da un indirizzo IPv4
wget -r --progress=dot:mega \
         --bind-address="$IPV4_CLIENT" \
         "http://${IPV4_SERVER}/${FILENAME}" \
         -O /dev/null 2> ${TEMPFILE}

# Conversione del file in un formato utile per gnuplot
cat ${TEMPFILE} | sed -e "s/\.\.\.\.\.\.\.\.//g" -e "s/^\ *//g" |\
        tr -s ' ' | grep ^[0-9] | grep -v ':'  | cut -d ' ' -f 1,3 |\
        tr -d 'K' > ${IPv4_DATAFILE}


# File da un indirizzo IPv6
wget -r --progress=dot:mega \
         --bind-address="$IPV6_CLIENT" \
         "http://[${IPV6_SERVER}]/${FILENAME}" \
         -O /dev/null 2> ${TEMPFILE}
```

```
# Conversione del file in un formato utile per gnuplot
cat ${TEMPFILE} | sed -e "s/\.\.\.\.\.\.\.\.//g" -e "s/^\ */g" |\
        tr -s ' ' | grep ^[0-9] | grep -v ':' | cut -d ' ' -f 1,3 |\
        tr -d 'K' > ${IPv6_DATAFILE}

# Eliminiamo il file temporaneo
rm ${TEMPFILE}
```

A.2 Configurazioni software

Per una maggiore comodità nella gestione delle configurazione di base degli host sono stati sviluppati alcuni altri script: in questo modo i due sistemi risultano sufficientemente omogenei ed i risultati dei test corrono un rischio minore di essere inficiati da eventuali incompatibilità software.

A.2.1 Software di base

Il primo script `setup.sh` si occupa della configurazione di base degli host: dal prelievo del kernel (e delle patch per IPv6) alla configurazione, compilazione ed installazione del nucleo del sistema operativo.

L'uso di alcune variabili permette di facilitare un eventuale modifica delle versioni del software da utilizzare nelle prove:

```
#!/bin/sh
#set -x

# Script per la configurazione dei test

# Inizializzazione di alcune variabili
# che renderanno lo script piu' leggibile
BASE_KENEL_VERSION="2.6.8"
PATCH_KERNEL_VERSION_BASE="2.6.9"
PATCH_KERNEL_VERSION="${PATCH_KERNEL_VERSION_BASE}-rc3"
USAGI_SNAPSHOT="20041011"

KERNEL_ORG="http://kernel.org/pub/linux/kernel/v2.6"
USAGI_FTP="ftp://ftp.linux-ipv6.org/pub/usagi/snap/split/"
```

```
16 BASE_KERNEL="linux-${BASE_KENEL_VERSION}"
   BASE_KERNEL_ARCHIVE="${BASE_KERNEL}.tar.bz2"
18
   PATCH="patch-${PATCH_KERNEL_VERSION}.bz2"
20 PATCHED_KERNEL="linux-${PATCH_KERNEL_VERSION}-usagi"

22 USAGI_PATCH="usagi-linux26-s${USAGI_SNAPSHOT}-${
       PATCH_KERNEL_VERSION}.diff.bz2"
   USAGI_TOOLS="usagi-tool-s${USAGI_SNAPSHOT}.tar.bz2"
24
```

Le configurazioni sono state effettuate nella directory `/usr/src`, qui risiedono inoltre gli archivi prelevati dal web:

```
   # Spostiamoci in una directory opportuna
26 cd /usr/src

28 # Preleviamo i sorgenti del kernel
   wget -c ${KERNEL_ORG}/${BASE_KERNEL_ARCHIVE}
30
   # Preleviamo la patch -rc2
32 wget -c ${KERNEL_ORG}/testing/${PATCH}

34 # Preleviamo la patch USAGI
   wget -c ${USAGI_FTP}/${USAGI_PATCH}
36
   # Prendiamo gli usagi-tools
38 #wget -c ${USAGI_FTP}/${USAGI_TOOLS}
```

Il kernel è stato modificato e preparato per la compilazione:

```
40 # Scompattiamo il kernel
   tar xjvf ${BASE_KERNEL_ARCHIVE}
42
   # Rinominiamo la directory del kernel
44 mv ${BASE_KERNEL} ${PATCHED_KERNEL}

46 # Entriamo nella directory del kernel
   cd ${PATCHED_KERNEL}
48
   # Applichiamo la prima patch
```

```
50 bzcat ../${PATCH} | patch -p1 | tee out.log

52 # Controlliamo se vi sono errori
   less out.log
54
   # Cancelliamo il file di log
56 rm out.log

58 # Applichiamo la seconda patch
   bzcat ../${USAGI_PATCH} | patch -p1 | tee out.log
60
   # Controlliamo se vi sono errori
62 less out.log

64 # Cancelliamm il file di log
   rm out.log
66
```

Particolare attenzione è stata posta alla configurazione del kernel, sono state abilitate le funzionalità interessanti ai fini della sperimentazione:

```
   # Modifichiamo il Makefile per aggiungere -usagi
68 # al nome del kernel nella variabile EXTRAVERSION
   vi Makefile
70
   # Configuriamo il kernel facendo attenzione che ALMENO
72 # le seguenti voci siano attivate in
   # DeviceDrivers->NetworkingSupport->NetworkingOptions:
74 #
   # CONFIG_IPV6=y
76 # CONFIG_IPV6_PRIVACY=y
   # CONFIG_IPV6_ROUTER_PREF=y
78 # CONFIG_INET6_AH=m
   # CONFIG_INET6_ESP=m
80 # CONFIG_INET6_IPCOMP=m
   # CONFIG_INET6_TUNNEL=m
82 # CONFIG_IPV6_TUNNEL=m

84 make menuconfig

86 # Se abbiamo una configurazione di un kernel precedente possiamo riutilizzarla
   # e non eseguire "make menuconfig".
88 #
```

```
# Per esempio:
# $ cp /boot/config.old .config
# $ make oldconfig

```

Il kernel è stato quindi compilato:

```
# Compiliamo il kernel generando dei pacchetti debian per una comoda
# installazione
make-kpkg clean
make-kpkg --bzimage \
          --arch-in-name --revision "${PATCH_KERNEL_VERSION_BASE}"
    kernel_image modules_image

```

E successivamente installato:

```
# Installiamo i pacchetti del kernel
cd ..
dpkg -i kernel-image-${PATCH_KERNEL_VERSION}-usagi_${
    PATCH_KERNEL_VERSION_BASE}_i386.deb

# Configuriamo il bootloader per utilizzare il nuovo kernel
vi /etc/lilo.conf
lilo -v

# Reboot
shutdown -r now
```

A.2.2 Software Applicativi

Uno script chiamato `apps.sh` si occupa dell'installazione e configurazione delle applicazioni.

Il software **netperf** è stato ricompilato ed installato in maniera opportuna; la ricompilazione si è resa necessaria per l'abilitazione dei test inerenti IPv6 e per l'attivazione del codice specifico per la misura dell'utilizzo del processore sui sistemi *GNU/Linux*:

```
# Dopo il reboot scarichiamo il sorgente del pacchetto Debian di "netperf",
# dovremo ricompilare il software per abilitare il supporto per i test su IPv6
apt-get source netperf

# entriamo nella directory crata da apt-get
cd netperf-2.3/

# Aggiungiamo una opzione al file debian/rules per abilitare il supporto per i
# test su IPv6:
#
# la riga 8: CFLAGS = -O2 -Wall
# diventa : CFLAGS = -O2 -Wall -DDO_IPV6 -DPROC_STAT
vi debian/rules

# Costruiamo il nuovo pacchetto Debian
./debian/rules binary

# Installiamo netperf
cd ..
dpkg -i netperf_2.3-1_i386.deb

# Blocchiamo il pacchetto in modo che non vanga aggiornato con versioni che
# non supportino i test per IPv6
echo "netperf hold" | dpkg --set-selections
```

Anche il software **wget** ha richiesto una ricompilazione per attivare il supporto alla versione 6 di IP:

```
# Scarichiamo il sorgente del pacchetto Debian di "wget",
# dovremo ricompilare il software per abilitare il supporto per i test su IPv6
apt-get source wget

# entriamo nella directory crata da apt-get
cd wget-1.9.1/

# Aggiungiamo una opzione al file debian/rules per abilitare il supporto per i
# test su IPv6:
#
# la riga 7: --with-ssh
# e' sostituita da: --enable-ipv6
vi debian/rules
```

```
40 # Costruiamo il nuovo pacchetto Debian
   ./debian/rules binary
42
   # Installiamo netperf
44 cd ..
   dpkg -i wget_1.9.1-7_i386.deb
46
   # Blocchiamo il pacchetto in modo che non vanga aggiornato con versioni che
48 # non supportino i test per IPv6
   echo "wget hold" | dpkg --set-selections
```

Abbiamo infine installato il webserver **Apache2** per il test applicativo e predisposto il sistema per il lancio dei test:

```
50
   # Installazione di apache2
52 apt-get install apache2

54
```

A.2.3 Configurazione della Rete

Le impostazioni di rete sono settate a seconda del ruolo dell'host, nel nostro caso uno dei due elaboratori avrà ruolo di server e l'altro opererà da client. Alcuni parametri del kernel vengono settati utilizzando l'utility `sysctl`.

Le impostazioni del kernel per IPv4:

```
net.ipv4.conf.all.disable_policy = 0
net.ipv4.conf.all.arp_ignore = 0
net.ipv4.conf.all.arp_announce = 0
net.ipv4.conf.all.arp_filter = 0
net.ipv4.conf.all.proxy_arp = 0
net.ipv4.conf.all.accept_source_route = 0
net.ipv4.conf.all.rp_filter = 1
net.ipv4.conf.all.forwarding = 0
net.ipv4.icmp_echo_ignore_all = 0
```

e per IPv6

```
net.ipv6.conf.all.max_addresses = 16
net.ipv6.conf.all.use_tempaddr = 0
```

```
net.ipv6.conf.all.router_solicitation_delay = 1
net.ipv6.conf.all.router_solicitation_interval = 4
net.ipv6.conf.all.router_solicitations = 3
net.ipv6.conf.all.dad_transmits = 1
net.ipv6.conf.all.hop_limit = 64
net.ipv6.conf.all.forwarding = 0
```

La configurazione del client:

```
#!/bin/sh
#set -x

SERVER_ADDRESS_v4="192.168.21.2"
CLIENT_ADDRESS_v4="192.168.22.2"

SERVER_NET_v4="192.168.21.0/24"
CLIENT_NET_v4="192.168.22.0/24"

SERVER_ADDRESS_v6="2001:db8:200:1:2e0:4cff:fe6c:154"
CLIENT_ADDRESS_v6="2001:db8:200:2:290:27ff:fe97:f568"

DEVICE=eth0
ip link set dev $DEVICE down
ip link set dev $DEVICE up

# Flush old addresses
ip addr flush dev $DEVICE
ip -6 addr flush dev $DEVICE

# Configure client IPv4 networking
ip addr add $CLIENT_ADDRESS_v4/24 dev $DEVICE
ip route add $SERVER_ADDRESS_v4 dev $DEVICE
sysctl -p ipv4.sysctl

# Configure client IPv6 networking
ip -6 addr add $CLIENT_ADDRESS_v6/64 dev $DEVICE
ip -6 route add $SERVER_ADDRESS_v6/64 dev $DEVICE
sysctl -p ipv6.sysctl
```

e quella del server:

```
#!/bin/sh
#set -x
```

```
SERVER_ADDRESS_v4="192.168.21.2"
CLIENT_ADDRESS_v4="192.168.22.2"

SERVER_NET_v4="192.168.21.0/24"
CLIENT_NET_v4="192.168.22.0/24"

SERVER_ADDRESS_v6="2001:db8:200:1:2e0:4cff:fe6c:154"
CLIENT_ADDRESS_v6="2001:db8:200:2:290:27ff:fe97:f568"

DEVICE=eth0
ip link set dev $DEVICE down
ip link set dev $DEVICE up

# Flush old addresses
ip addr flush dev $DEVICE
ip -6 addr flush dev $DEVICE

# Configure server IPv4 networking
ip addr add $SERVER_ADDRESS_v4/24 dev $DEVICE
ip route add $CLIENT_NET_v4 dev $DEVICE
sysctl -p ipv4.sysctl

# Configure server IPv6 networking
ip -6 addr add $SERVER_ADDRESS_v6/64 dev $DEVICE
ip -6 route add $CLIENT_ADDRESS_v6/64 dev $DEVICE
sysctl -p ipv6.sysctl
```

A.2.4 Configurazione di IPsec

La configurazione di IPsec è stata effettuata con l'utilizzo di **racoon-tool** un software di gestione per il server di scambio delle chiavi utilizzato recentemente su GNU/Linux.

Sono stati generati dei certificati x509 per l'autenticazione dei due host sulla VPN.

La configurazione del server è la seguente:

```
#
```

```
# Configuration file for racoon-tool
#
# See racoon-tool.conf(5) for details
#

# How to control the syslog level
global:
  log: notify

# Generic Template
spdadd(encap_template): spdadd ___src_range___ ___dst_range___
    ___upperspec___ -P out ipsec ___encap___/___mode___//
    require; spdadd ___dst_range___ ___src_range___
    ___upperspec___ -P in ipsec ___encap___/___mode___//require;

# AH
spdadd(ah): spdadd ___src_range___ ___dst_range___
    ___upperspec___ -P out ipsec ah/___mode___//require; spdadd
    ___dst_range___ ___src_range___ ___upperspec___ -P in ipsec
    ah/___mode___//require;

# ESP
spdadd(esp): spdadd ___src_range___ ___dst_range___
    ___upperspec___ -P out ipsec esp/___mode___//require; spdadd
    ___dst_range___ ___src_range___ ___upperspec___ -P in ipsec
    esp/___mode___//require;

# AH + ESP
spdadd(ah_esp): spdadd ___src_range___ ___dst_range___
    ___upperspec___ -P out ipsec esp/___mode___//require ah/
    ___mode___//require; spdadd ___dst_range___ ___src_range___
     ___upperspec___ -P in ipsec esp/___mode___//require ah/
    ___mode___//require;

#
# Host to Host IPv4
#
connection(host-to-host-IPv4):
  mode: transport
  upperspec: any
  encap: esp
  src_range: 192.168.21.2/32
  dst_range: 192.168.22.2/32
```

```
  src_ip: 192.168.21.2
  dst_ip: 192.168.22.2
  admin_status: enabled
   lifetime : time 500 min
  authentication_algorithm: hmac_sha1
  encryption_algorithm: aes
  compression: enabled
  spdadd_template: ah_esp

peer(192.168.22.2):
   certificate_type : x509 vpngateway_cert.pem vpngateway_key.pem
  verify_cert: on
  my_identifier: asn1dn
   peers_identifier : asn1dn
    verify_identifier : on
  encryption_algorithm[0]: aes
  hash_algorithm[0]: sha1
  authentication_method[0]: rsasig
  passive: off
   lifetime : time 500 min

#
# Host to Host IPv6
#
connection(host-to-host-IPv6):
  mode: transport
  upperspec: any
  encap: esp
  src_range: 2001:db8:200:1:2e0:4cff:fe6c:154/128
  dst_range: 2001:db8:200:2:290:27ff:fe97:f568/128
  src_ip: 2001:db8:200:1:2e0:4cff:fe6c:154
  dst_ip: 2001:db8:200:2:290:27ff:fe97:f568
  admin_status: enabled
   lifetime : time 500 min
  authentication_algorithm: hmac_sha1
  encryption_algorithm: aes
  compression: enabled
  spdadd_template: ah_esp

peer(2001:db8:200:2:290:27ff:fe97:f568):
   certificate_type : x509 vpngateway_cert.pem vpngateway_key.pem
  verify_cert: on
  my_identifier: asn1dn
```

```
  peers_identifier : asn1dn
  verify_identifier : on
 encryption_algorithm[0]: aes
 hash_algorithm[0]: sha1
 authentication_method[0]: rsasig
 passive: off
 lifetime : time 500 min
```

Le *Security Policies* sono stampate dal comando:

```
racoon-tool spddump
```

Sul lato server abbiamo come risultato:

```
2001:db8:200:2:290:27ff:fe97:f568[any] 2001:db8:200:1:2e0:4cff:fe6c:154[any] any
  in ipsec
  esp/transport//require
  ah/transport//require
  created: Oct 11 16:01:43 2004  lastused:
  lifetime: 0(s) validtime: 0(s)
  spid=120 seq=13 pid=24459
  refcnt=1
192.168.22.2[any] 192.168.21.2[any] any
  in ipsec
  esp/transport//require
  ah/transport//require
  created: Oct 11 16:01:43 2004  lastused:
  lifetime: 0(s) validtime: 0(s)
  spid=136 seq=12 pid=24459
  refcnt=1
2001:db8:200:1:2e0:4cff:fe6c:154[any] 2001:db8:200:2:290:27ff:fe97:f568[any] any
  out ipsec
  esp/transport//require
  ah/transport//require
  created: Oct 11 16:01:43 2004  lastused:
  lifetime: 0(s) validtime: 0(s)
  spid=113 seq=11 pid=24459
  refcnt=1
192.168.21.2[any] 192.168.22.2[any] any
  out ipsec
  esp/transport//require
  ah/transport//require
  created: Oct 11 16:01:43 2004  lastused:
  lifetime: 0(s) validtime: 0(s)
  spid=129 seq=10 pid=24459
  refcnt=1
0.0.0.0/0[any] 0.0.0.0/0[any] any
```

```
  in none
  created: Oct 11 16:01:43 2004  lastused:
  lifetime: 0(s) validtime: 0(s)
  spid=99 seq=9 pid=24459
  refcnt=1
0.0.0.0/0[any] 0.0.0.0/0[any] any
  in none
  created: Oct 11 16:01:43 2004  lastused:
  lifetime: 0(s) validtime: 0(s)
  spid=83 seq=8 pid=24459
  refcnt=1
::/0[any] ::/0[any] any
  in none
  created: Oct 11 16:01:43 2004  lastused:
  lifetime: 0(s) validtime: 0(s)
  spid=67 seq=7 pid=24459
  refcnt=1
::/0[any] ::/0[any] any
  in none
  created: Oct 11 16:01:43 2004  lastused:
  lifetime: 0(s) validtime: 0(s)
  spid=51 seq=6 pid=24459
  refcnt=1
::/0[any] ::/0[any] any
  in none
  created: Oct 11 16:01:43 2004  lastused:
  lifetime: 0(s) validtime: 0(s)
  spid=19 seq=5 pid=24459
  refcnt=1
0.0.0.0/0[any] 0.0.0.0/0[any] any
  out none
  created: Oct 11 16:01:43 2004  lastused:
  lifetime: 0(s) validtime: 0(s)
  spid=108 seq=4 pid=24459
  refcnt=1
0.0.0.0/0[any] 0.0.0.0/0[any] any
  out none
  created: Oct 11 16:01:43 2004  lastused:
  lifetime: 0(s) validtime: 0(s)
  spid=92 seq=3 pid=24459
  refcnt=1
::/0[any] ::/0[any] any
  out none
  created: Oct 11 16:01:43 2004  lastused:
  lifetime: 0(s) validtime: 0(s)
  spid=76 seq=2 pid=24459
  refcnt=1
::/0[any] ::/0[any] any
  out none
```

```
  created: Oct 11 16:01:43 2004  lastused:
  lifetime: 0(s) validtime: 0(s)
  spid=60 seq=1 pid=24459
  refcnt=1
::/0[any] ::/0[any] any
  out none
  created: Oct 11 16:01:43 2004  lastused:
  lifetime: 0(s) validtime: 0(s)
  spid=28 seq=0 pid=24459
  refcnt=1
```

La configurazione del client è molto simile e non viene qui riportata: la destinazione e la sorgente risultano invertite rispetto a quella del server. Visualizziamo le *Security Association* che il demone `racoon` ha provveduto a creare, usiamo il comando:

```
racoon−tool saddump
```

Per esempio, le *Security Association* instaurate nel nostro caso, con solo ESP attivo, sono le seguenti:

```
2001:db8:200:2:290:27ff:fe97:f568 2001:db8:200:1:2e0:4cff:fe6c:154
  esp mode=transport spi=1167752(0x0011d188) reqid=0(0x00000000)
  E: aes-cbc  b4db137b dc7434ad 9c23acf0 759e1cdf
  A: hmac-sha1  f42574bd 7af76930 a6f3f4ae 87dc8bb6 ae8c647f
  seq=0x00000000 replay=4 flags=0x00000000 state=mature
  created: Sep 26 11:55:17 2004 current: Sep 26 11:59:41 2004
  diff: 264(s)  hard: 30000(s)  soft: 24000(s)
  last: Sep 26 11:55:20 2004  hard: 0(s)  soft: 0(s)
  current: 576(bytes) hard: 0(bytes)  soft: 0(bytes)
  allocated: 9  hard: 0 soft: 0
  sadb_seq=3 pid=4473 refcnt=0
2001:db8:200:1:2e0:4cff:fe6c:154 2001:db8:200:2:290:27ff:fe97:f568
  esp mode=transport spi=13918402(0x00d460c2) reqid=0(0x00000000)
  E: aes-cbc  d088d77d 428962f6 3de98e60 224d1b2c
  A: hmac-sha1  409de360 6ead5c78 512b3e89 9ec6247a bc79fdb6
  seq=0x00000000 replay=4 flags=0x00000000 state=mature
  created: Sep 26 11:55:17 2004 current: Sep 26 11:59:41 2004
  diff: 264(s)  hard: 30000(s)  soft: 24000(s)
  last: Sep 26 11:55:20 2004  hard: 0(s)  soft: 0(s)
  current: 1404(bytes)  hard: 0(bytes)  soft: 0(bytes)
  allocated: 9  hard: 0 soft: 0
  sadb_seq=2 pid=4473 refcnt=0
192.168.22.2 192.168.21.2
  esp mode=transport spi=166885604(0x09f278e4) reqid=0(0x00000000)
```

```
  E: aes-cbc  e6fe9e10 11661bc8 11173a52 f1f13269
  A: hmac-sha1  a75ac162 25809276 c4c74954 8bd1a724 9a470b91
  seq=0x00000000 replay=4 flags=0x00000000 state=mature
  created: Sep 26 11:56:04 2004 current: Sep 26 11:59:41 2004
  diff: 217(s)  hard: 30000(s)  soft: 24000(s)
  last: Sep 26 11:56:05 2004  hard: 0(s)  soft: 0(s)
  current: 192(bytes) hard: 0(bytes)  soft: 0(bytes)
  allocated: 3  hard: 0 soft: 0
  sadb_seq=1 pid=4473 refcnt=0
192.168.21.2 192.168.22.2
  esp mode=transport spi=187445126(0x0b2c2f86) reqid=0(0x00000000)
  E: aes-cbc  25f4119a 19a867d5 2794a21a cb316e0d
  A: hmac-sha1  597295d8 86665463 cddfcea4 7b54396f 2b67f646
  seq=0x00000000 replay=4 flags=0x00000000 state=mature
  created: Sep 26 11:56:04 2004 current: Sep 26 11:59:41 2004
  diff: 217(s)  hard: 30000(s)  soft: 24000(s)
  last: Sep 26 11:56:05 2004  hard: 0(s)  soft: 0(s)
  current: 408(bytes) hard: 0(bytes)  soft: 0(bytes)
  allocated: 3  hard: 0 soft: 0
  sadb_seq=0 pid=4473 refcnt=0
```

Bibliografia

The official ipsec howto for linux. `http://www.ipsec-howto.org/`.

Appendice B

Porting di applicativi di rete

B.1 Introduzione

Un sistema operativo offre diversi servizi agli utenti ed ai programmatori, il servizio di astrazione del sottosistema di rete consente, per esempio, di scrivere applicazioni indipendentemente dalla tecnologia di rete adottata e nel migliore dei casi in maniera indipendente anche dal protocollo di rete utilizzato. Una delle API per la programmazione di rete più diffusa fra i sistemi operativi moderni è quella ispirata alle *socket BSD*, sviluppata inizialmente sui sistemi UNIX di Berkeley e poi integrata per buona parte nello standard POSIX. Tuttavia, durante lo sviluppo del protocollo IPv6, è stato notato che questa interfaccia di astrazione, nella sua formulazione originale, non risultava sufficientemente adatta alla programmazione multiprotocollo, risultava scomodo e faticoso scrivere applicazioni che supportassero in modo trasparente sia IPv4 che IPv6. Sono state proposti quindi degli aggiornamenti che contribuissero al porting degli applicativi alla nuova versione di IP, e si è cercato di fornire una interfaccia che non soffrisse degli stessi problemi di quella originale qualora fosse richiesto il supporto per un qualunque nuovo protocollo di rete.

B.2 Strutture dati

B.2.1 Problemi

Il primo aspetto della scarsa adattabilità della interfaccia originale risiede nel fatto che essa richiede all'interno del codice dell'applicazione (*hardcoded*) riferimenti a strutture dati e costanti dipendenti dal protocollo di rete. La famiglia di indirizzi, ad esempio, deve essere spesso denotata esplicitamente, è frequentissimo vedere nel codice degli applicativi di rete riferimenti alla famiglia degli indirizzi IPv4 (`AF_INET`). Per scrivere applicativi pronti ad IPv6 si deve trattare la nuova famiglia di indirizzi (`AF_INET6`) come un caso particolare rendendo il codice più complesso e difficile da revisionare. Allo stesso modo le strutture dati utilizzate per memorizzare gli indirizzi variano a seconda della versione del protocollo e delle dimensione degli indirizzi, la struttura `sockaddr_in` è fatta per gli indirizzi Internet v4 e risulta necessaria una struttura diversa per gli indirizzi IPv6 (`sockaddr_in6`)

B.2.2 Soluzioni

Per scrivere applicazioni portabili ed indipendenti dal protocollo di rete è stata proposta una struttura dati che mascherasse i dettagli sulla dimensione degli indirizzi, la struttura `sockaddr_storage` è definita in modo da contenere un indirizzo di dimensione arbitraria. Non dovranno quindi esserci riferimenti presenti nel codice alle diverse famiglie di indirizzi che l'applicazione dovrà supportare.

B.3 Funzioni

Un discorso analogo lo si può fare per le funzioni che manipolano gli indirizzi di rete all'interno delle applicazioni e che gestiscono eventuali tradu-

zioni in nomi di dominio: le funzioni `gethostbyname()`, `gethostbyaddr()`, `inet_ntop()`, `inet_pton()`, `getservbyname()` e `getservbyport()` possono essere tutte sostituite da opportune chiamate a `getaddrinfo()` e `getnameinfo` demandando in questo modo al sistema operativo la gestione dei diversi protocolli di rete supportati.

Usando tali accorgimenti una applicazione di rete può essere adattata ad IPv6 essendo già pronta ad ulteriori successive evoluzioni dei protocolli di rete, ed in qualche caso il codice dell'applicazione potrà addirittura essere semplificato e reso più elegante.

B.4 Un esempio

B.4.1 Un server TCP/IPv4

Nel seguito si mostra il codice di un semplicissimo server che funziona con IPv4 e che stampa sullo standard output il testo inviato da un qualsia client capace di instaurare una sessione TCP (telnet, netcat, etc.)

Makefile

```
# A simple Makefile to simplify your life ;)
# Antonio Ospite
# ospite@studenti.unina.it

TMPDIR=/tmp
ARCHIVE=$(shell basename `pwd`)

ifdef DEBUG
DEFINES=-DDEBUG
endif

#CC=cc
CC=gcc
DEPEND = $(CC) -MM -MG
```

```
#CFLAGS=-ansi -pedantic -pedantic-errors

CFLAGS=-pedantic -pedantic-errors -D_ANSI_SOURCE_ \
        -Wall -O2

EXECUTABLE  := smsg
SOURCES     := $(wildcard *.c)
DEPFILES    := $(SOURCES:.c=.d)
OBJS        := $(SOURCES:.c=.o)

all: $(EXECUTABLE)

$(EXECUTABLE): $(OBJS)
    $(CC) $(CFLAGS) $(DEFINES) $(OBJS) -o $(EXECUTABLE)

clean:
    -rm -f $(DEPFILES) $(OBJS) $(EXECUTABLE) core *~

dist: clean
    @-rm -rf $(TMPDIR)/$(ARCHIVE)
    @mkdir $(TMPDIR)/$(ARCHIVE)
    @cp -a * $(TMPDIR)/$(ARCHIVE)
    @tar czf $(ARCHIVE).tar.gz -C $(TMPDIR) $(ARCHIVE)
    @-rm -rf $(TMPDIR)/$(ARCHIVE)
    @echo "Distribution archive created as $(ARCHIVE).tar.gz"

beauty:
    -astyle --style=ansi -bs2 *.c
    -astyle --style=ansi -bs2 *.h
    -rm -f *.orig

%o: %.c
    $(CC) $(CFLAGS) $(DEFINES) -c $*.c

%.d : %.c
    @-$(DEPEND) $(CCFLAGS) $< > $@

-include $(DEPFILES)
```

tools.h

```
/*
 *  Author: Antonio Ospite
 *          <ospite@studenti.unina.it>
 */

#ifndef __TOOLS_H
#define __TOOLS_H

#ifdef DEBUG
#define d(x)  {x}
#else
#define d(x)
#endif

/*
 * Some functions to make the code more readable
 */
static void err_sys (char *msg)
{
  perror(msg);
  exit  (1);
}

#endif /* __TOOLS_H */
```

readline.h

```
/*
 * A simple readline() function.
 * Taken from: Advanced Unix Programming (R. W. Stevens)
 *
 */

#ifndef __READLINE_H
#define __READLINE_H

#include <sys/types.h>

#define MAXLINE 1024

ssize_t  readline(int  fd,  void *vptr, size_t  maxlen);
```

```
/*
ssize_t Readline(int fd, void *ptr, size_t maxlen)
{
    ssize_t      n;

    if ( (n = readline(fd, ptr, maxlen)) < 0)
        err_sys("readline error");
    return(n);
}
*/

#endif /* __READLINE_H */
```

readline.c

```
/*
 * A simple readline() function.
 * Taken from: Advanced Unix Programming (R. W. Stevens)
 *
 */

#include <stdio.h>
#include <unistd.h>
#include <errno.h>
#include <sys/types.h>

#include "readline.h"

static ssize_t my_read(int fd, char *ptr)
{
  static int     read_cnt = 0;
  static char   *read_ptr;
  static char    read_buf[MAXLINE];

  if (read_cnt <= 0)
  {
again:
    if ( (read_cnt = read(fd, read_buf, sizeof(read_buf))) < 0)
    {
      if (errno == EINTR)
        goto again;
      return(-1);
```

```
    }
    else if (read_cnt == 0)
      return(0);
    read_ptr = read_buf;
  }

  read_cnt--;
  *ptr = *read_ptr++;
  return(1);
}

ssize_t readline(int fd, void *vptr, size_t maxlen)
{
  int        n, rc;
  char  c, *ptr;

  ptr = vptr;
  for (n = 1; n < maxlen; n++)
  {
    if ( (rc = my_read(fd, &c)) == 1)
    {
      *ptr++ = c;
      if (c == '\n')
        break;      /* newline is stored, like fgets() */
    }
    else if (rc == 0)
    {
      if (n == 1)
        return(0);  /* EOF, no data read */
      else
        break;          /* EOF, some data was read */
    }
    else
      return(-1);       /* error, errno set by read() */
  }

  *ptr = 0;           /* null terminate like fgets() */
  return(n);
}
/* end readline */

/*
ssize_t Readline(int fd, void *ptr, size_t maxlen)
```

```
{
  ssize_t         n;

  if ( (n = readline(fd, ptr, maxlen)) < 0)
    err_sys("readline error");
  return(n);
}
*/
```

server.h

```
/*
 *  Author: Antonio Ospite
 *          <ospite@studenti.unina.it>
 */

#ifndef __SERVER_H
#define __SERVER_H

void server_loop(int connfd, int clientid );

#endif /* __SERVER_H */
```

server.c

```
/*
 *  Author: Antonio Ospite
 *          <ospite@studenti.unina.it>
 */

#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <unistd.h>
#include <netdb.h>

#include "server.h"
#include "tools.h"
#include "readline.h"

void server_loop(int connfd, int clientid )
{
  int n;
  char buffer[MAXLINE];

  struct sockaddr_storage clientname;
  socklen_t addrlen = sizeof(clientname);

  char clienthost[NI_MAXHOST] = "\0";
  char clientservice[NI_MAXSERV] = "\0";

  if( getsockname(connfd, (struct sockaddr *)&clientname, &addrlen) == 0 )
```

```
  {
    memset(clienthost, 0, sizeof( clienthost ));
    memset(clientservice, 0, sizeof( clientservice ));

    getnameinfo((struct sockaddr *)&clientname, addrlen,
                clienthost , sizeof( clienthost ),
                clientservice , sizeof( clientservice ),
                NI_NUMERICHOST);
  }

  for (;;)
  {
    if ( (n = readline(connfd, buffer , MAXLINE)) < 0)
      err_sys("readline error");
    if (n == 0 || strncmp (buffer, "quit\n", strlen( buffer )) == 0)
    {
      fprintf (stderr , "client:%2d ([%s]:%s) disconnected\n",
               clientid , clienthost , clientservice );
      close (connfd);
      exit (0);
    }
    buffer [ strlen ( buffer )-1] = '\0';
    fprintf (stderr , "[%s]:%s client:%2d -> %s\n",
             clienthost , clientservice , clientid , buffer );
  }
}
```

smsg.c

```
/*
 *  Author: Antonio Ospite
 *          <ospite@studenti.unina.it>
 *
 * Thanks to the KAME project and
 * thanks to Richard W. Stevens :)
 *
 */

#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <getopt.h>
#include <unistd.h>
#include <fcntl.h>
#include <signal.h>
#include <sys/types.h>
#include <sys/socket.h>
#include <arpa/inet.h>
#include <netinet/in.h>
#include <errno.h>

#include "tools.h"
#include "server.h"

#define MAXLINE 1024
#define MAXPENDING 10

#define DEFAULT_SERVICE 9999

#include <sys/wait.h>
static void sig_chld(int signo)
{
  pid_t pid;
  int stat;

  while ( (pid = waitpid( -1, &stat, WNOHANG)) > 0)
    d( fprintf (stderr, "child %d terminated\n", pid));
}

#ifndef usage
```

```
#define usage(x) fprintf(stderr,"usage: %s [-p <port>]\n", x)
#endif

static void help()
{
  printf("\n"
         "The Simple MeSsaGe server.\n"
         "\n"
         "Options:\n"
         "\t-p <port> \t specify the port (default 9999).\n"
         "\t-h \t\t too see the help text.\n"
         "\n"
         "Bye,\n"
         "  Antonio Ospite\n"
         "  <ospite@studenti.unina.it>\n"
         "\n");
}

int main(int argc, char *argv[])
{

  pid_t pid;
  int counter = 0;
  int  listenfd =-1, connfd;

  unsigned short service;
  struct sockaddr_in clientname, servername;
  socklen_t addrlen = sizeof(servername);

  char buffer[MAXLINE];

  char c;

  service=htons(DEFAULT_SERVICE);

  /* get command line options */
  while( (c=getopt(argc, argv, "p:h")) != -1 )
    switch(c)
    {
    case 'p':
      service=htons(atoi(optarg));
      break;
```

```
  case 'h':
    help();
    exit(0);
    break;

  case '?':
    usage(argv[0]);
    exit(1);
  }

/* create socket */
listenfd = socket(AF_INET, SOCK_STREAM, 0);
if (listenfd < 0)
  err_sys("socket");

/* bind */
servername.sin_family = AF_INET;
servername.sin_port = service;
servername.sin_addr.s_addr = htonl(INADDR_ANY);
if (bind(listenfd, (struct sockaddr *)&servername, addrlen) < 0)
  err_sys ("bind");

/* listen */
if (listen (listenfd, MAXPENDING) < 0)
  err_sys ("listen");

printf ("\nSimple MeSsaGe server started\n\n");

if ( signal(SIGCHLD, sig_chld) == SIG_ERR )
  err_sys("signal");

/* take care: accept is a "slow" system call */
for (;;)
{
  if ((connfd=accept(listenfd, (struct sockaddr *)&clientname, &addrlen)) < 0)
  {
    if (errno == EINTR)
      continue;
    else
      err_sys ("accept");
  }
```

```
    printf ("\nConnection %d from %s port %d \n",
            ++counter,
            inet_ntop(AF_INET, &(clientname.sin_addr), NULL, addrlen),
            ntohs(clientname.sin_port));

    if ( (pid = fork()) == 0)
    {
      close( listenfd );
      printf ("Logging started...\n");
      sprintf( buffer, "\nWelcome to my server.\nType \"quit\" to... quit ;)\n\n");
      write(connfd, buffer, strlen( buffer ));
      server_loop(connfd, counter);
      exit (0);
    }
    close(connfd);
  }
  exit (0);
}
```

B.4.2 Indipendenza dalla famiglia degli indirizzi

Le modifiche necessarie a rendere il server indipendente dal protocollo di rete:

```
diff -pruN smsg-0.1_old/smsg.c smsg-0.1/smsg.c
--- smsg-0.1_old/smsg.c 2004-10-17 20:19:14.000000000 +0200
+++ smsg-0.1/smsg.c 2004-10-17 20:16:00.000000000 +0200
@@ -16,6 +16,7 @@
 #include <signal.h>
 #include <sys/types.h>
 #include <sys/socket.h>
+#include <netdb.h>
 #include <arpa/inet.h>
 #include <netinet/in.h>
 #include <errno.h>
@@ -59,27 +60,34 @@ static void help()

 int main(int argc, char *argv[])
 {
+ struct addrinfo hints, *res, *ressave;
+ int error;

   pid_t pid;
   int counter = 0;
   int listenfd =-1, connfd;
```

```
- unsigned short service;
- struct sockaddr_in clientname, servername;
- socklen_t addrlen = sizeof(servername);
+ char service[NI_MAXSERV];
+ int ReUseAddr = 1;
+ struct sockaddr_storage clientname;
+ socklen_t addrlen = sizeof(clientname);
+
+ char *serverhost=NULL;
+ char clienthost[NI_MAXHOST];
+ char clientservice[NI_MAXSERV];

  char buffer[MAXLINE];

  char c;

- service=htons(DEFAULT_SERVICE);
+ snprintf(service, NI_MAXSERV, "%d\0", DEFAULT_SERVICE );

  /* get command line options */
  while( (c=getopt(argc, argv, "p:h")) != -1 )
    switch(c)
    {
    case 'p':
-     service=htons(atoi(optarg));
+     strncpy(service, optarg, NI_MAXSERV);
      break;

    case 'h':
@@ -92,17 +100,54 @@ int main(int argc, char *argv[])
      exit(1);
    }

- /* create socket */
- listenfd = socket(AF_INET, SOCK_STREAM, 0);
- if (listenfd < 0)
-   err_sys("socket");
-
- /* bind */
- servername.sin_family = AF_INET;
- servername.sin_port = service;
- servername.sin_addr.s_addr = htonl(INADDR_ANY);
```

```
- if (bind(listenfd, (struct sockaddr *)&servername, addrlen) < 0)
-   err_sys ("bind");
+ /* Clear the hints variable */
+ memset(&hints, 0, sizeof(hints));
+
+ /*
+   AI_PASSIVE flag: the resulting address is used to bind
+   to a socket for accepting incoming connections.
+   So, when the hostname==NULL, getaddrinfo function will
+   return one entry per allowed protocol family containing
+   the unspecified address for that family.
+ */
+ hints.ai_flags = AI_PASSIVE;
+ hints.ai_family = AF_UNSPEC;
+ hints.ai_socktype = SOCK_STREAM;
+
+ error = getaddrinfo(serverhost, service, &hints, &res);
+ if (error != 0)
+ {
+   /* handle getaddrinfo error */
+   d(fprintf(stderr,"Error in getaddrinfo."));
+   err_sys("getaddrinfo");
+ }
+
+ ressave=res;
+
+ /*
+   Try open socket with each address getaddrinfo returned,
+   until getting a valid listening socket.
+ */
+ while (res)
+ {
+   /* create socket */
+   listenfd = socket(res->ai_family, res->ai_socktype, res->ai_protocol);
+   if (listenfd >= 0)
+   {
+     /* allow it to always reuse the same port */
+     ReUseAddr = 1;
+     setsockopt(listenfd, SOL_SOCKET,
+                SO_REUSEADDR, &ReUseAddr, sizeof(ReUseAddr));
+
+     if (bind(listenfd, res->ai_addr, res->ai_addrlen) == 0)
+       break;
```

```
+
+       close( listenfd );
+       listenfd =-1;
+     }
+     res  = res->ai_next;
+  }
+  freeaddrinfo(ressave);

    /* listen */
    if (listen  (listenfd, MAXPENDING) < 0)
@@ -124,11 +169,18 @@ int main(int argc, char *argv[])
          err_sys ("accept");
      }

+     memset(clienthost, 0, sizeof(clienthost));
+     memset(clientservice, 0, sizeof(clientservice));
+
+     getnameinfo((struct sockaddr *)&clientname, addrlen,
+                 clienthost, sizeof(clienthost),
+                 clientservice, sizeof(clientservice),
+                 NI_NUMERICHOST);

-     printf ("\nConnection %d from %s port %d \n",
+     printf ("\nConnection %d from %s port %s \n",
              ++counter,
-             inet_ntop(AF_INET, &(clientname.sin_addr), NULL, addrlen),
-             ntohs(clientname.sin_port));
+             clienthost,
+             clientservice);

      if ( (pid = fork()) == 0)
      {
```

Con queste modifiche il server si porrà in ascolto sulla porta scelta per ogni famiglia di indirizzi supportata dal sistema su cui verrà eseguito, in modo trasparente senza la necessità di considerare ogni protocollo di rete come un caso particolare.

Bibliografia

Myung-Ki Shin, Yong-Guen Hong, Jun ichiro itojun HAGINO, Pekka Savola, and Eva M. Castro. *Application Aspects of IPv6 Transition*, March 2004. URL `http://www.ietf.org/internet-drafts/`.

W. Richard Stevens. *Advanced programming in the UNIX environment.* Addison Wesley Longman Publishing Co., Inc., 1992. ISBN 0-201-56317-7.

W. Richard Stevens. *UNIX Network Programming: Networking APIs: Sockets and XTI.* Prentice Hall PTR, 1997. ISBN 013490012X.

Appendice C

Pacchetti IPv6

In questo capitolo verranno mostrati alcuni pacchetti IPv6 catturati con un analizzatore di rete. Analizzare il traffico può mostrare degli esempi reali di come siano strutturati i dati trasmessi sulle reti.

C.1 Software di analisi

Il software utilizzato per la cattura dei pacchetti è **tethereal**, la versione a linea di comando dell'ottimo *ethereal*, che ci permette di specificare dei filtri di cattura, in modo da effettuare l'analisi di un determinato tipo di traffico in transito sulla rete.

C.2 Esempi di pacchetti ICMPv6

Nel caso del traffico ICMPv6 la linea di comando utilizzata è la seguente:

```
$ tethereal -c 4 -R "icmpv6" -i eth0 -l -n -p -T text -V
```

l'opzione `-c 4` specifica il numero di pacchetti da catturare, i 4 pacchetti seguenti mostrano gli effetti di una richiesta di ping da un host ad un altro, operazione che si completa in 4 passi, nel caso in cui debba essere effettuata la ricerca dell'indirizzo link-layer del vicino.

C.2.1 ICMPv6 neigh sol

Il primo passo consiste nella ricerca dell'indirizzo *Layer 2* dell'host di destinazione, un pacchetto di *Neighbor Solicitation* viene inviato ad una destinazione multicast.

```
Frame 1 (86 bytes on wire, 86 bytes captured)
    Arrival Time: Sep 26, 2004 23:01:15.282239000
    Time delta from previous packet: 0.000000000 seconds
    Time since reference or first frame: 0.000000000 seconds
    Frame Number: 1
    Packet Length: 86 bytes
    Capture Length: 86 bytes
Ethernet II, Src: 00:e0:4c:6c:01:54, Dst: 33:33:ff:00:00:02
    Destination: 33:33:ff:00:00:02 (33:33:ff:00:00:02)
    Source: 00:e0:4c:6c:01:54 (00:e0:4c:6c:01:54)
    Type: IPv6 (0x86dd)
Internet Protocol Version 6
    Version: 6
    Traffic class: 0x00
    Flowlabel: 0x00000
    Payload length: 32
    Next header: ICMPv6 (0x3a)
    Hop limit: 255
    Source address: fec0::1 (fec0::1)
    Destination address: ff02::1:ff00:2 (ff02::1:ff00:2)
Internet Control Message Protocol v6
    Type: 135 (Neighbor solicitation)
    Code: 0
    Checksum: 0x2d78 (correct)
    Target: fec0::2 (fec0::2)
    ICMPv6 options
        Type: 1 (Source link-layer address)
        Length: 8 bytes (1)
        Link-layer address: 00:e0:4c:6c:01:54
```

C.2.2 ICMPv6 neigh adv

Secondo passo: l'host di destinazione risponde comunicando il proprio indirizzo L2, a questo punto i due host in questione aggiornano la loro *Neighbor Cache*.

```
Frame 2 (86 bytes on wire, 86 bytes captured)
    Arrival Time: Sep 26, 2004 23:01:15.282365000
    Time delta from previous packet: 0.000126000 seconds
    Time since reference or first frame: 0.000126000 seconds
    Frame Number: 2
    Packet Length: 86 bytes
    Capture Length: 86 bytes
Ethernet II, Src: 00:90:27:97:f5:68, Dst: 00:e0:4c:6c:01:54
    Destination: 00:e0:4c:6c:01:54 (00:e0:4c:6c:01:54)
    Source: 00:90:27:97:f5:68 (00:90:27:97:f5:68)
    Type: IPv6 (0x86dd)
Internet Protocol Version 6
    Version: 6
    Traffic class: 0x00
    Flowlabel: 0x00000
    Payload length: 32
    Next header: ICMPv6 (0x3a)
    Hop limit: 255
    Source address: fec0::2 (fec0::2)
    Destination address: fec0::1 (fec0::1)
Internet Control Message Protocol v6
    Type: 136 (Neighbor advertisement)
    Code: 0
    Checksum: 0xfbcb (correct)
    Flags: 0x60000000
        0... .... .... .... .... .... .... .... = Not router
        .1.. .... .... .... .... .... .... .... = Solicited
        ..1. .... .... .... .... .... .... .... = Override
    Target: fec0::2 (fec0::2)
    ICMPv6 options
        Type: 2 (Target link-layer address)
        Length: 8 bytes (1)
        Link-layer address: 00:90:27:97:f5:68
```

C.2.3 ICMPv6 echo request

Ora che è possibile una comunicazione a livello di link, viene inviata una richiesta di eco: il terzo passo dà inizio al "ping" vero e proprio.

```
Frame 3 (118 bytes on wire, 118 bytes captured)
    Arrival Time: Sep 26, 2004 23:01:15.282380000
    Time delta from previous packet: 0.000015000 seconds
    Time since reference or first frame: 0.000141000 seconds
    Frame Number: 3
    Packet Length: 118 bytes
    Capture Length: 118 bytes
Ethernet II, Src: 00:e0:4c:6c:01:54, Dst: 00:90:27:97:f5:68
    Destination: 00:90:27:97:f5:68 (00:90:27:97:f5:68)
    Source: 00:e0:4c:6c:01:54 (00:e0:4c:6c:01:54)
    Type: IPv6 (0x86dd)
Internet Protocol Version 6
    Version: 6
    Traffic class: 0x00
    Flowlabel: 0x00000
    Payload length: 64
    Next header: ICMPv6 (0x3a)
    Hop limit: 64
    Source address: fec0::1 (fec0::1)
    Destination address: fec0::2 (fec0::2)
Internet Control Message Protocol v6
    Type: 128 (Echo request)
    Code: 0
    Checksum: 0xfc0e (correct)
    ID: 0x9230
    Sequence: 0x0001
    Data (56 bytes)

0000  1b 2e 57 41 92 4d 04 00 08 09 0a 0b 0c 0d 0e 0f   ..WA.M..........
0010  10 11 12 13 14 15 16 17 18 19 1a 1b 1c 1d 1e 1f   ................
0020  20 21 22 23 24 25 26 27 28 29 2a 2b 2c 2d 2e 2f    !"#$%&'()*+,-./
0030  30 31 32 33 34 35 36 37                          01234567
```

C.2.4 ICMPv6 echo reply

Una risposta di eco costituisce il quarto passo, l'host di destinazione ha risposto.

```
Frame 4 (118 bytes on wire, 118 bytes captured)
    Arrival Time: Sep 26, 2004 23:01:15.282488000
    Time delta from previous packet: 0.000108000 seconds
    Time since reference or first frame: 0.000249000 seconds
    Frame Number: 4
    Packet Length: 118 bytes
    Capture Length: 118 bytes
Ethernet II, Src: 00:90:27:97:f5:68, Dst: 00:e0:4c:6c:01:54
    Destination: 00:e0:4c:6c:01:54 (00:e0:4c:6c:01:54)
    Source: 00:90:27:97:f5:68 (00:90:27:97:f5:68)
    Type: IPv6 (0x86dd)
Internet Protocol Version 6
    Version: 6
    Traffic class: 0x00
    Flowlabel: 0x00000
    Payload length: 64
    Next header: ICMPv6 (0x3a)
    Hop limit: 64
    Source address: fec0::2 (fec0::2)
    Destination address: fec0::1 (fec0::1)
Internet Control Message Protocol v6
    Type: 129 (Echo reply)
    Code: 0
    Checksum: 0xfb0e (correct)
    ID: 0x9230
    Sequence: 0x0001
    Data (56 bytes)

0000  1b 2e 57 41 92 4d 04 00 08 09 0a 0b 0c 0d 0e 0f   ..WA.M..........
0010  10 11 12 13 14 15 16 17 18 19 1a 1b 1c 1d 1e 1f   ................
0020  20 21 22 23 24 25 26 27 28 29 2a 2b 2c 2d 2e 2f    !"#$%&'()*+,-./
0030  30 31 32 33 34 35 36 37                          01234567
```

C.3 Esempi di pacchetti TCP/IPv6

Un esempio di traffico generato dal server TCP **smsg** presentato in appendice B e catturato con la riga di comando seguente:

```
$ tethereal -c 10 -R "tcp.port==9999" -i eth0 -l -n -p -T text -V
```

C.3.1 Invio di un breve messaggio

Il client instaura una sessione TCP, invia il messaggio "Pacchetto catturato" e chiude la comunicazione.

```
Frame 189 (94 bytes on wire, 94 bytes captured)
    Arrival Time: Oct 19, 2004 19:58:30.187597000
    Time delta from previous packet: 0.099193000 seconds
    Time since reference or first frame: 5.815777000 seconds
    Frame Number: 189
    Packet Length: 94 bytes
    Capture Length: 94 bytes
Ethernet II, Src: 00:90:27:97:f5:68, Dst: 00:e0:4c:6c:01:54
    Destination: 00:e0:4c:6c:01:54 (00:e0:4c:6c:01:54)
    Source: 00:90:27:97:f5:68 (00:90:27:97:f5:68)
    Type: IPv6 (0x86dd)
Internet Protocol Version 6
    Version: 6
    Traffic class: 0x00
    Flowlabel: 0x00000
    Payload length: 40
    Next header: TCP (0x06)
    Hop limit: 64
    Source address: fec0::2 (fec0::2)
    Destination address: fec0::1 (fec0::1)
Transmission Control Protocol, Src Port: 32777 (32777),
                               Dst Port: 9999 (9999), Seq: 0, Ack: 0, Len: 0
    Source port: 32777 (32777)
    Destination port: 9999 (9999)
    Sequence number: 0    (relative sequence number)
    Header length: 40 bytes
    Flags: 0x0002 (SYN)
        0... .... = Congestion Window Reduced (CWR): Not set
        .0.. .... = ECN-Echo: Not set
```

```
        ..0. .... = Urgent: Not set
        ...0 .... = Acknowledgment: Not set
        .... 0... = Push: Not set
        .... .0.. = Reset: Not set
        .... ..1. = Syn: Set
        .... ...0 = Fin: Not set
    Window size: 5760
    Checksum: 0xa0f1 (correct)
    Options: (20 bytes)
        Maximum segment size: 1440 bytes
        SACK permitted
        Time stamp: tsval 1444304, tsecr 0
        NOP
        Window scale: 2 (multiply by 4)

Frame 192 (94 bytes on wire, 94 bytes captured)
    Arrival Time: Oct 19, 2004 19:58:30.188025000
    Time delta from previous packet: 0.000014000 seconds
    Time since reference or first frame: 5.816205000 seconds
    Frame Number: 192
    Packet Length: 94 bytes
    Capture Length: 94 bytes
Ethernet II, Src: 00:e0:4c:6c:01:54, Dst: 00:90:27:97:f5:68
    Destination: 00:90:27:97:f5:68 (00:90:27:97:f5:68)
    Source: 00:e0:4c:6c:01:54 (00:e0:4c:6c:01:54)
    Type: IPv6 (0x86dd)
Internet Protocol Version 6
    Version: 6
    Traffic class: 0x00
    Flowlabel: 0x00000
    Payload length: 40
    Next header: TCP (0x06)
    Hop limit: 64
    Source address: fec0::1 (fec0::1)
    Destination address: fec0::2 (fec0::2)
Transmission Control Protocol, Src Port: 9999 (9999),
                               Dst Port: 32777 (32777), Seq: 0, Ack: 1, Len: 0
    Source port: 9999 (9999)
    Destination port: 32777 (32777)
    Sequence number: 0    (relative sequence number)
    Acknowledgement number: 1    (relative ack number)
    Header length: 40 bytes
    Flags: 0x0012 (SYN, ACK)
```

```
        0... .... = Congestion Window Reduced (CWR): Not set
        .0.. .... = ECN-Echo: Not set
        ..0. .... = Urgent: Not set
        ...1 .... = Acknowledgment: Set
        .... 0... = Push: Not set
        .... .0.. = Reset: Not set
        .... ..1. = Syn: Set
        .... ...0 = Fin: Not set
    Window size: 5712
    Checksum: 0xa6fb (correct)
    Options: (20 bytes)
        Maximum segment size: 1440 bytes
        SACK permitted
        Time stamp: tsval 33256046, tsecr 1444304
        NOP
        Window scale: 2 (multiply by 4)
    SEQ/ACK analysis
        This is an ACK to the segment in frame: 189
        The RTT to ACK the segment was: 0.000428000 seconds

Frame 193 (86 bytes on wire, 86 bytes captured)
    Arrival Time: Oct 19, 2004 19:58:30.188170000
    Time delta from previous packet: 0.000145000 seconds
    Time since reference or first frame: 5.816350000 seconds
    Frame Number: 193
    Packet Length: 86 bytes
    Capture Length: 86 bytes
Ethernet II, Src: 00:90:27:97:f5:68, Dst: 00:e0:4c:6c:01:54
    Destination: 00:e0:4c:6c:01:54 (00:e0:4c:6c:01:54)
    Source: 00:90:27:97:f5:68 (00:90:27:97:f5:68)
    Type: IPv6 (0x86dd)
Internet Protocol Version 6
    Version: 6
    Traffic class: 0x00
    Flowlabel: 0x00000
    Payload length: 32
    Next header: TCP (0x06)
    Hop limit: 64
    Source address: fec0::2 (fec0::2)
    Destination address: fec0::1 (fec0::1)
Transmission Control Protocol, Src Port: 32777 (32777),
                               Dst Port: 9999 (9999), Seq: 1, Ack: 1, Len: 0
    Source port: 32777 (32777)
```

```
    Destination port: 9999 (9999)
    Sequence number: 1    (relative sequence number)
    Acknowledgement number: 1    (relative ack number)
    Header length: 32 bytes
    Flags: 0x0010 (ACK)
        0... .... = Congestion Window Reduced (CWR): Not set
        .0.. .... = ECN-Echo: Not set
        ..0. .... = Urgent: Not set
        ...1 .... = Acknowledgment: Set
        .... 0... = Push: Not set
        .... .0.. = Reset: Not set
        .... ..0. = Syn: Not set
        .... ...0 = Fin: Not set
    Window size: 5760
    Checksum: 0xe65d (correct)
    Options: (12 bytes)
        NOP
        NOP
        Time stamp: tsval 1444305, tsecr 33256046
    SEQ/ACK analysis
        This is an ACK to the segment in frame: 192
        The RTT to ACK the segment was: 0.000145000 seconds

Frame 194 (136 bytes on wire, 136 bytes captured)
    Arrival Time: Oct 19, 2004 19:58:30.190335000
    Time delta from previous packet: 0.002165000 seconds
    Time since reference or first frame: 5.818515000 seconds
    Frame Number: 194
    Packet Length: 136 bytes
    Capture Length: 136 bytes
Ethernet II, Src: 00:e0:4c:6c:01:54, Dst: 00:90:27:97:f5:68
    Destination: 00:90:27:97:f5:68 (00:90:27:97:f5:68)
    Source: 00:e0:4c:6c:01:54 (00:e0:4c:6c:01:54)
    Type: IPv6 (0x86dd)
Internet Protocol Version 6
    Version: 6
    Traffic class: 0x00
    Flowlabel: 0x00000
    Payload length: 82
    Next header: TCP (0x06)
    Hop limit: 64
    Source address: fec0::1 (fec0::1)
    Destination address: fec0::2 (fec0::2)
```

```
Transmission Control Protocol, Src Port: 9999 (9999),
                               Dst Port: 32777 (32777), Seq: 1, Ack: 1, Len: 50
    Source port: 9999 (9999)
    Destination port: 32777 (32777)
    Sequence number: 1    (relative sequence number)
    Next sequence number: 51    (relative sequence number)
    Acknowledgement number: 1    (relative ack number)
    Header length: 32 bytes
    Flags: 0x0018 (PSH, ACK)
        0... .... = Congestion Window Reduced (CWR): Not set
        .0.. .... = ECN-Echo: Not set
        ..0. .... = Urgent: Not set
        ...1 .... = Acknowledgment: Set
        .... 1... = Push: Set
        .... .0.. = Reset: Not set
        .... ..0. = Syn: Not set
        .... ...0 = Fin: Not set
    Window size: 5712
    Checksum: 0xfddc (incorrect, should be 0x382c)
    Options: (12 bytes)
        NOP
        NOP
        Time stamp: tsval 33256049, tsecr 1444305
Data (50 bytes)

0000  0a 57 65 6c 63 6f 6d 65 20 74 6f 20 6d 79 20 73   .Welcome to my s
0010  65 72 76 65 72 2e 0a 54 79 70 65 20 22 71 75 69   erver..Type "qui
0020  74 22 20 74 6f 2e 2e 2e 20 71 75 69 74 20 3b 29   t" to... quit ;)
0030  0a 0a                                             ..

Frame 195 (86 bytes on wire, 86 bytes captured)
    Arrival Time: Oct 19, 2004 19:58:30.190489000
    Time delta from previous packet: 0.000154000 seconds
    Time since reference or first frame: 5.818669000 seconds
    Frame Number: 195
    Packet Length: 86 bytes
    Capture Length: 86 bytes
Ethernet II, Src: 00:90:27:97:f5:68, Dst: 00:e0:4c:6c:01:54
    Destination: 00:e0:4c:6c:01:54 (00:e0:4c:6c:01:54)
    Source: 00:90:27:97:f5:68 (00:90:27:97:f5:68)
    Type: IPv6 (0x86dd)
Internet Protocol Version 6
    Version: 6
```

```
    Traffic class: 0x00
    Flowlabel: 0x00000
    Payload length: 32
    Next header: TCP (0x06)
    Hop limit: 64
    Source address: fec0::2 (fec0::2)
    Destination address: fec0::1 (fec0::1)
Transmission Control Protocol, Src Port: 32777 (32777),
                               Dst Port: 9999 (9999), Seq: 1, Ack: 51, Len: 0
    Source port: 32777 (32777)
    Destination port: 9999 (9999)
    Sequence number: 1    (relative sequence number)
    Acknowledgement number: 51    (relative ack number)
    Header length: 32 bytes
    Flags: 0x0010 (ACK)
        0... .... = Congestion Window Reduced (CWR): Not set
        .0.. .... = ECN-Echo: Not set
        ..0. .... = Urgent: Not set
        ...1 .... = Acknowledgment: Set
        .... 0... = Push: Not set
        .... .0.. = Reset: Not set
        .... ..0. = Syn: Not set
        .... ...0 = Fin: Not set
    Window size: 5760
    Checksum: 0xe626 (correct)
    Options: (12 bytes)
        NOP
        NOP
        Time stamp: tsval 1444307, tsecr 33256049
    SEQ/ACK analysis
        This is an ACK to the segment in frame: 194
        The RTT to ACK the segment was: 0.000154000 seconds

Frame 620 (106 bytes on wire, 106 bytes captured)
    Arrival Time: Oct 19, 2004 19:58:44.364030000
    Time delta from previous packet: 0.026218000 seconds
    Time since reference or first frame: 19.992210000 seconds
    Frame Number: 620
    Packet Length: 106 bytes
    Capture Length: 106 bytes
Ethernet II, Src: 00:90:27:97:f5:68, Dst: 00:e0:4c:6c:01:54
    Destination: 00:e0:4c:6c:01:54 (00:e0:4c:6c:01:54)
    Source: 00:90:27:97:f5:68 (00:90:27:97:f5:68)
```

```
    Type: IPv6 (0x86dd)
Internet Protocol Version 6
    Version: 6
    Traffic class: 0x00
    Flowlabel: 0x00000
    Payload length: 52
    Next header: TCP (0x06)
    Hop limit: 64
    Source address: fec0::2 (fec0::2)
    Destination address: fec0::1 (fec0::1)
Transmission Control Protocol, Src Port: 32777 (32777),
                               Dst Port: 9999 (9999), Seq: 1, Ack: 51, Len: 20
    Source port: 32777 (32777)
    Destination port: 9999 (9999)
    Sequence number: 1    (relative sequence number)
    Next sequence number: 21    (relative sequence number)
    Acknowledgement number: 51    (relative ack number)
    Header length: 32 bytes
    Flags: 0x0018 (PSH, ACK)
        0... .... = Congestion Window Reduced (CWR): Not set
        .0.. .... = ECN-Echo: Not set
        ..0. .... = Urgent: Not set
        ...1 .... = Acknowledgment: Set
        .... 1... = Push: Set
        .... .0.. = Reset: Not set
        .... ..0. = Syn: Not set
        .... ...0 = Fin: Not set
    Window size: 5760
    Checksum: 0x9126 (correct)
    Options: (12 bytes)
        NOP
        NOP
        Time stamp: tsval 1458481, tsecr 33256049
Data (20 bytes)

0000  50 61 63 63 68 65 74 74 6f 20 63 61 74 74 75 72   Pacchetto cattur
0010  61 74 6f 0a                                       ato.

Frame 621 (86 bytes on wire, 86 bytes captured)
    Arrival Time: Oct 19, 2004 19:58:44.364536000
    Time delta from previous packet: 0.000506000 seconds
    Time since reference or first frame: 19.992716000 seconds
    Frame Number: 621
```

```
    Packet Length: 86 bytes
    Capture Length: 86 bytes
Ethernet II, Src: 00:e0:4c:6c:01:54, Dst: 00:90:27:97:f5:68
    Destination: 00:90:27:97:f5:68 (00:90:27:97:f5:68)
    Source: 00:e0:4c:6c:01:54 (00:e0:4c:6c:01:54)
    Type: IPv6 (0x86dd)
Internet Protocol Version 6
    Version: 6
    Traffic class: 0x00
    Flowlabel: 0x00000
    Payload length: 32
    Next header: TCP (0x06)
    Hop limit: 64
    Source address: fec0::1 (fec0::1)
    Destination address: fec0::2 (fec0::2)
Transmission Control Protocol, Src Port: 9999 (9999),
                               Dst Port: 32777 (32777), Seq: 51, Ack: 21, Len: 0
    Source port: 9999 (9999)
    Destination port: 32777 (32777)
    Sequence number: 51    (relative sequence number)
    Acknowledgement number: 21    (relative ack number)
    Header length: 32 bytes
    Flags: 0x0010 (ACK)
        0... .... = Congestion Window Reduced (CWR): Not set
        .0.. .... = ECN-Echo: Not set
        ..0. .... = Urgent: Not set
        ...1 .... = Acknowledgment: Set
        .... 0... = Push: Not set
        .... .0.. = Reset: Not set
        .... ..0. = Syn: Not set
        .... ...0 = Fin: Not set
    Window size: 5712
    Checksum: 0x7760 (correct)
    Options: (12 bytes)
        NOP
        NOP
        Time stamp: tsval 33270225, tsecr 1458481
    SEQ/ACK analysis
        This is an ACK to the segment in frame: 620
        The RTT to ACK the segment was: 0.000506000 seconds

Frame 671 (91 bytes on wire, 91 bytes captured)
    Arrival Time: Oct 19, 2004 19:58:46.354342000
```

```
    Time delta from previous packet: 0.115783000 seconds
    Time since reference or first frame: 21.982522000 seconds
    Frame Number: 671
    Packet Length: 91 bytes
    Capture Length: 91 bytes
Ethernet II, Src: 00:90:27:97:f5:68, Dst: 00:e0:4c:6c:01:54
    Destination: 00:e0:4c:6c:01:54 (00:e0:4c:6c:01:54)
    Source: 00:90:27:97:f5:68 (00:90:27:97:f5:68)
    Type: IPv6 (0x86dd)
Internet Protocol Version 6
    Version: 6
    Traffic class: 0x00
    Flowlabel: 0x00000
    Payload length: 37
    Next header: TCP (0x06)
    Hop limit: 64
    Source address: fec0::2 (fec0::2)
    Destination address: fec0::1 (fec0::1)
Transmission Control Protocol, Src Port: 32777 (32777),
                               Dst Port: 9999 (9999), Seq: 21, Ack: 51, Len: 5
    Source port: 32777 (32777)
    Destination port: 9999 (9999)
    Sequence number: 21    (relative sequence number)
    Next sequence number: 26    (relative sequence number)
    Acknowledgement number: 51    (relative ack number)
    Header length: 32 bytes
    Flags: 0x0018 (PSH, ACK)
        0... .... = Congestion Window Reduced (CWR): Not set
        .0.. .... = ECN-Echo: Not set
        ..0. .... = Urgent: Not set
        ...1 .... = Acknowledgment: Set
        .... 1... = Push: Set
        .... .0.. = Reset: Not set
        .... ..0. = Syn: Not set
        .... ...0 = Fin: Not set
    Window size: 5760
    Checksum: 0x8a97 (correct)
    Options: (12 bytes)
        NOP
        NOP
        Time stamp: tsval 1460471, tsecr 33270225
Data (5 bytes)
```

```
0000  71 75 69 74 0a                                        quit.

Frame 672 (86 bytes on wire, 86 bytes captured)
    Arrival Time: Oct 19, 2004 19:58:46.354997000
    Time delta from previous packet: 0.000655000 seconds
    Time since reference or first frame: 21.983177000 seconds
    Frame Number: 672
    Packet Length: 86 bytes
    Capture Length: 86 bytes
Ethernet II, Src: 00:e0:4c:6c:01:54, Dst: 00:90:27:97:f5:68
    Destination: 00:90:27:97:f5:68 (00:90:27:97:f5:68)
    Source: 00:e0:4c:6c:01:54 (00:e0:4c:6c:01:54)
    Type: IPv6 (0x86dd)
Internet Protocol Version 6
    Version: 6
    Traffic class: 0x00
    Flowlabel: 0x00000
    Payload length: 32
    Next header: TCP (0x06)
    Hop limit: 64
    Source address: fec0::1 (fec0::1)
    Destination address: fec0::2 (fec0::2)
Transmission Control Protocol, Src Port: 9999 (9999),
                               Dst Port: 32777 (32777), Seq: 51, Ack: 26, Len: 0
    Source port: 9999 (9999)
    Destination port: 32777 (32777)
    Sequence number: 51    (relative sequence number)
    Acknowledgement number: 26    (relative ack number)
    Header length: 32 bytes
    Flags: 0x0010 (ACK)
        0... .... = Congestion Window Reduced (CWR): Not set
        .0.. .... = ECN-Echo: Not set
        ..0. .... = Urgent: Not set
        ...1 .... = Acknowledgment: Set
        .... 0... = Push: Not set
        .... .0.. = Reset: Not set
        .... ..0. = Syn: Not set
        .... ...0 = Fin: Not set
    Window size: 5712
    Checksum: 0x67ce (correct)
    Options: (12 bytes)
        NOP
        NOP
```

```
        Time stamp: tsval 33272216, tsecr 1460471
    SEQ/ACK analysis
        This is an ACK to the segment in frame: 671
        The RTT to ACK the segment was: 0.000655000 seconds

Frame 673 (86 bytes on wire, 86 bytes captured)
    Arrival Time: Oct 19, 2004 19:58:46.355059000
    Time delta from previous packet: 0.000062000 seconds
    Time since reference or first frame: 21.983239000 seconds
    Frame Number: 673
    Packet Length: 86 bytes
    Capture Length: 86 bytes
Ethernet II, Src: 00:e0:4c:6c:01:54, Dst: 00:90:27:97:f5:68
    Destination: 00:90:27:97:f5:68 (00:90:27:97:f5:68)
    Source: 00:e0:4c:6c:01:54 (00:e0:4c:6c:01:54)
    Type: IPv6 (0x86dd)
Internet Protocol Version 6
    Version: 6
    Traffic class: 0x00
    Flowlabel: 0x00000
    Payload length: 32
    Next header: TCP (0x06)
    Hop limit: 64
    Source address: fec0::1 (fec0::1)
    Destination address: fec0::2 (fec0::2)
Transmission Control Protocol, Src Port: 9999 (9999),
                               Dst Port: 32777 (32777), Seq: 51, Ack: 26, Len: 0
    Source port: 9999 (9999)
    Destination port: 32777 (32777)
    Sequence number: 51    (relative sequence number)
    Acknowledgement number: 26    (relative ack number)
    Header length: 32 bytes
    Flags: 0x0011 (FIN, ACK)
        0... .... = Congestion Window Reduced (CWR): Not set
        .0.. .... = ECN-Echo: Not set
        ..0. .... = Urgent: Not set
        ...1 .... = Acknowledgment: Set
        .... 0... = Push: Not set
        .... .0.. = Reset: Not set
        .... ..0. = Syn: Not set
        .... ...1 = Fin: Set
    Window size: 5712
    Checksum: 0x67cd (correct)
```

```
Options: (12 bytes)
    NOP
    NOP
    Time stamp: tsval 33272216, tsecr 1460471
```

C.4 Esempi di pacchetti con AH o ESP

Alcuni esempi di pacchetti sottoposti ai servizi di sicurezza di IPsec e catturati con il comando:

```
$ tethereal -c 2 -i eth0 -l -n -p -T text -V
```

C.4.1 Authentication Header

Eseguiamo un ping con AH attivo:

```
Frame 1 (142 bytes on wire, 142 bytes captured)
    Arrival Time: Oct 19, 2004 23:20:00.010798000
    Time delta from previous packet: 0.000000000 seconds
    Time since reference or first frame: 0.000000000 seconds
    Frame Number: 1
    Packet Length: 142 bytes
    Capture Length: 142 bytes
Ethernet II, Src: 00:90:27:97:f5:68, Dst: 00:e0:4c:6c:01:54
    Destination: 00:e0:4c:6c:01:54 (00:e0:4c:6c:01:54)
    Source: 00:90:27:97:f5:68 (00:90:27:97:f5:68)
    Type: IPv6 (0x86dd)
Internet Protocol Version 6
    Version: 6
    Traffic class: 0x00
    Flowlabel: 0x00000
    Payload length: 88
    Next header: AH (0x33)
    Hop limit: 64
    Source address: fec0::2 (fec0::2)
    Destination address: fec0::1 (fec0::1)
Authentication Header
    Next Header: ICMPv6 (0x3a)
    Length: 24
    SPI: 0x0f1200f5
    Sequence: 39
    ICV
Internet Control Message Protocol v6
    Type: 128 (Echo request)
    Code: 0
    Checksum: 0x3ec3 (correct)
    ID: 0xef1c
```

```
    Sequence: 0x0001
    Data (56 bytes)

0000  ff 84 75 41 e8 55 0c 00 08 09 0a 0b 0c 0d 0e 0f   ..uA.U..........
0010  10 11 12 13 14 15 16 17 18 19 1a 1b 1c 1d 1e 1f   ................
0020  20 21 22 23 24 25 26 27 28 29 2a 2b 2c 2d 2e 2f    !"#$%&'()*+,-./
0030  30 31 32 33 34 35 36 37                          01234567

Frame 2 (142 bytes on wire, 142 bytes captured)
    Arrival Time: Oct 19, 2004 23:20:00.011556000
    Time delta from previous packet: 0.000758000 seconds
    Time since reference or first frame: 0.000758000 seconds
    Frame Number: 2
    Packet Length: 142 bytes
    Capture Length: 142 bytes
Ethernet II, Src: 00:e0:4c:6c:01:54, Dst: 00:90:27:97:f5:68
    Destination: 00:90:27:97:f5:68 (00:90:27:97:f5:68)
    Source: 00:e0:4c:6c:01:54 (00:e0:4c:6c:01:54)
    Type: IPv6 (0x86dd)
Internet Protocol Version 6
    Version: 6
    Traffic class: 0x00
    Flowlabel: 0x00000
    Payload length: 88
    Next header: AH (0x33)
    Hop limit: 64
    Source address: fec0::1 (fec0::1)
    Destination address: fec0::2 (fec0::2)
Authentication Header
    Next Header: ICMPv6 (0x3a)
    Length: 24
    SPI: 0x0d9c98b5
    Sequence: 39
    ICV
Internet Control Message Protocol v6
    Type: 129 (Echo reply)
    Code: 0
    Checksum: 0x3dc3 (correct)
    ID: 0xef1c
    Sequence: 0x0001
    Data (56 bytes)

0000  ff 84 75 41 e8 55 0c 00 08 09 0a 0b 0c 0d 0e 0f   ..uA.U..........
```

```
0010  10 11 12 13 14 15 16 17 18 19 1a 1b 1c 1d 1e 1f   ................
0020  20 21 22 23 24 25 26 27 28 29 2a 2b 2c 2d 2e 2f    !"#$%&'()*+,-./
0030  30 31 32 33 34 35 36 37                          01234567
```

C.4.2 Encapsulating Security payload

Eseguiamo un ping con ESP attivo:

```
Frame 1 (170 bytes on wire, 170 bytes captured)
    Arrival Time: Oct 19, 2004 23:22:38.799765000
    Time delta from previous packet: 0.000000000 seconds
    Time since reference or first frame: 0.000000000 seconds
    Frame Number: 1
    Packet Length: 170 bytes
    Capture Length: 170 bytes
Ethernet II, Src: 00:90:27:97:f5:68, Dst: 00:e0:4c:6c:01:54
    Destination: 00:e0:4c:6c:01:54 (00:e0:4c:6c:01:54)
    Source: 00:90:27:97:f5:68 (00:90:27:97:f5:68)
    Type: IPv6 (0x86dd)
Internet Protocol Version 6
    Version: 6
    Traffic class: 0x00
    Flowlabel: 0x00000
    Payload length: 116
    Next header: ESP (0x32)
    Hop limit: 64
    Source address: fec0::2 (fec0::2)
    Destination address: fec0::1 (fec0::1)
Encapsulating Security Payload
    SPI: 0x0af45127
    Sequence: 15
    Data (108 bytes)

0000  84 f2 13 dd 26 d9 fe 35 19 79 a3 5c 3e ed f0 7a   ....&..5.y.\>..z
0010  77 97 a3 2e 21 e5 7a cb 49 09 94 ae 3b 11 1d d8   w...!.z.I...;...
0020  32 95 f7 fe 11 50 42 c7 d3 32 75 e6 ba c8 6f 70   2....PB..2u...op
0030  c1 22 58 e1 c4 f3 c5 b5 8f 79 5e 0f ea d5 a4 40   ."X......y^....@
0040  b9 1d 30 67 ba 2c ea e3 6e c5 9b 14 17 a9 89 7d   ..0g.,..n......}
0050  13 63 97 9c f2 8a 57 a7 78 eb 0f 7b 16 04 44 b7   .c....W.x..{..D.
0060  06 38 6f ad 7b b0 87 e1 5c eb a1 14               .8o.{...\...

Frame 2 (170 bytes on wire, 170 bytes captured)
    Arrival Time: Oct 19, 2004 23:22:38.800449000
    Time delta from previous packet: 0.000684000 seconds
    Time since reference or first frame: 0.000684000 seconds
    Frame Number: 2
    Packet Length: 170 bytes
```

```
    Capture Length: 170 bytes
Ethernet II, Src: 00:e0:4c:6c:01:54, Dst: 00:90:27:97:f5:68
    Destination: 00:90:27:97:f5:68 (00:90:27:97:f5:68)
    Source: 00:e0:4c:6c:01:54 (00:e0:4c:6c:01:54)
    Type: IPv6 (0x86dd)
Internet Protocol Version 6
    Version: 6
    Traffic class: 0x00
    Flowlabel: 0x00000
    Payload length: 116
    Next header: ESP (0x32)
    Hop limit: 64
    Source address: fec0::1 (fec0::1)
    Destination address: fec0::2 (fec0::2)
Encapsulating Security Payload
    SPI: 0x0070369a
    Sequence: 15
    Data (108 bytes)

0000  2a 5f f1 2d df 25 3b 5a 57 46 6b f0 67 5f 6c c9   *_.-.%;ZWFk.g_l.
0010  04 8f 62 8c b2 16 34 aa 88 1e 3e c3 4c 21 de cb   ..b...4...>.L!..
0020  5c 67 ff f7 05 d9 7c f5 71 75 df 0f d4 02 76 70   \g....|.qu....vp
0030  ac 67 9f ce 47 ab 95 49 3e a3 15 9e ce 31 09 a1   .g..G..I>....1..
0040  fa 7e 71 80 23 29 13 b7 eb 8f e8 d1 d2 2e 73 08   .~q.#).........s.
0050  79 18 1a cb 2b 07 31 58 f3 c5 7b 7c e6 1c 0b 69   y...+.1X..{|...i
0060  19 c5 be 50 2c c8 88 33 6b e8 7f 5f               ...P,..3k.._
```

Bibliografia

Ethereal: A network protocol analyzer. `http://www.ethereal.com/`.

Conclusioni

Da ciò che è stato mostrato in precedenza possiamo notare le indubbie grandi potenzialità di IPv6, esso offre soluzioni pulite ed efficaci per i servizi di mobilità e non dimentica tutti i progressi ottenuti nello studio del routing nei lunghi anni di utilizzo di IPv4.

Alcune delle caratteristiche peculiari alle pratiche di routing saranno usate in maniera sempre più intensa nel prossimo futuro, si pensi al routing multicast nelle applicazioni multimediali, ed alle ottimizzazioni per gli instradamenti verso terminali mobili.

L'esperienza accumulata durante la stesura di questo testo fa emergere una considerazione semplice ma che merita di essere esposta: il *networking* non cambia i suoi principi di base con l'introduzione di IPv6, non vi è alcun motivo di temere la sua larga diffusione, il nuovo protocollo costituisce piuttosto una evoluzione di IP verso una maggiore modularità e verso una più attenta gestione delle risorse.

Glossario

3GPP: 3rd Generation Partnership Project.

AH: Authentication Header.

ARP: Address Resolution Protocol.

BGP: Border Gateway Protocol.

CIDR: Classless Inter-Domain Routing.

CLNS: ConnectionLess Network Service.

CoA: Care-of Address.

DNS: Domanin Name System.

EGP: Exterior Border Protocol.

ESP: Encapsulating Security Payload.

HA: Home Agent.

IANA: Internet Assigned Number Authority.

ICMP: Internet Control Message Protocol.

ICMPv6: ICMP for IPv6.

IETF: Internet Engineer Task Force.

IP: Internet Protocol.

IPng: IP next generation.

IPsec: IP security.

IPv6: Internet Protocol version 6.

IS-IS: Intermediate System - Intermediate System.

ISO: International Standards Organization.

LAN: Local Area Network.

LSA: Link-state Advertisement.

MTU: Maximum Transmission Unit.

NAT: Network Address Translation.

NAT-PT: Network Address Translation - Protocol Traslation.

ND: Neighbor Discovery.

NTP: Network Time Protocol.

NUD: Neighbor Unreachability Detection.

OSI: Open Standard Interconnection.

OSPF: Open Short Path First.

QoS: Quality of Service.

RA: Router Advertisement.

RIP: Routing Internet Protocol.

RIPng: RIP new generation.

RS: Router Solicitation.

SA: Security Association.

SAD: Security Association Database.

SPD: Security Policy Database.

TCP: Transmission Control Protocol.

UDP: User Datagram Protocol.

VLMS: Variable Length subnet-masking.

Indice analitico

[0-9]

3GPP 10

A

Apache2 77

ARP 15, 47

B

BGP 58

broadcast 15

C

Care-of Address 65

CIDR 6

CLNS 57

D

DNS 20

record AAAA 20

E

EGP 58

H

Home Agent 67

I

ICMP 42

ICMPv6 42

IETF 5

IP 3, 16

IPng 5

IPsec 34, 60

AH 61

ESP 61

SAD 62, 112

Security Association 61

Security Policy 61

SPD 62, 110

IPv6 6

IS-IS 57

ISO 3

L

LAN 14

linux 76

livello 3

Application 4

Host-to-network 3
Internet 3
Transport 4
loopback 37
LSA 56

M

MTU 32

N

NAT 7
NAT-PT 9
Neighbor Discovery 46
Router Advertisement 47
Router Solicitation 45
netperf 76, 95
NUD 50

O

OSI 3
OSPF 57

Q

QoS 6

R

RIP 56
RIPng 56

T

TCP 3, 4

U

UDP 4
USAGI 76

V

VLMS 55