Alma Mater Studiorum · Università di Bologna

SCUOLA DI SCIENZE
Corso di Laurea in Informatica

TCP a ritrasmissione anticipata tramite Access Point Wi-Fi

Tesi di Laurea in Reti di Calcolatori

Relatore:
Chiar.mo Prof.
Vittorio Ghini

Presentata da:
Claudia Minardi

Sessione I
Anno Accademico 2012/2013

Abstract

La perdita di pacchetti durante una trasmissione su una rete Wireless influisce in maniera fondamentale sulla qualità del collegamento tra due End-System. Lo scopo del progetto è quello di implementare una tecnica di ritrasmissione anticipata dei pacchetti perduti, in modo da minimizzare i tempi di recupero dati e migliorare la qualità della comunicazione.
Partendo da uno studio su determinati tipi di ritrasmissione, in particolare quelli implementati dal progetto *ABPS, Always Best Packet Switching*, si è maturata l'idea che un tipo di ritrasmissione particolarmente utile potrebbe avvenire a livello Access Point: nel caso in cui la perdita di pacchetti avvenga tra l'AP e il nodo mobile che vi è collegato via IEEE802.11, invece che attendere la ritrasmissione TCP effettuata dall'End-System sorgente è lo stesso Access Point che effettua una ritrasmissione anticipata verso il nodo mobile per permettere un veloce recupero dei dati perduti.

Tale funzionalità è stata quindi concettualmente divisa in due parti, la prima riguardante l'applicazione che si occupa della bufferizzazione di pacchetti che attraversano l'AP e della loro copia in memoria per poi ritrasmetterli in caso di perdita, la seconda riguardante la modifica al kernel che permette la segnalazione anticipata dell'errore.

Questo documento tratta della realizzazione della parte applicativa e del metodo di ritrasmissione anticipata.

Indice

Elenco delle figure

Capitolo 1

Panoramica su TCP/IP

Il nome TCP/IP si riferisce ad un insieme di protocolli di comunicazione di dati. Prende il nome dalle sue due componenti principali, TCP e IP, tuttavia è rappresentato nella sua interezza da una moltitudine di tecnologie e protocolli diversi.

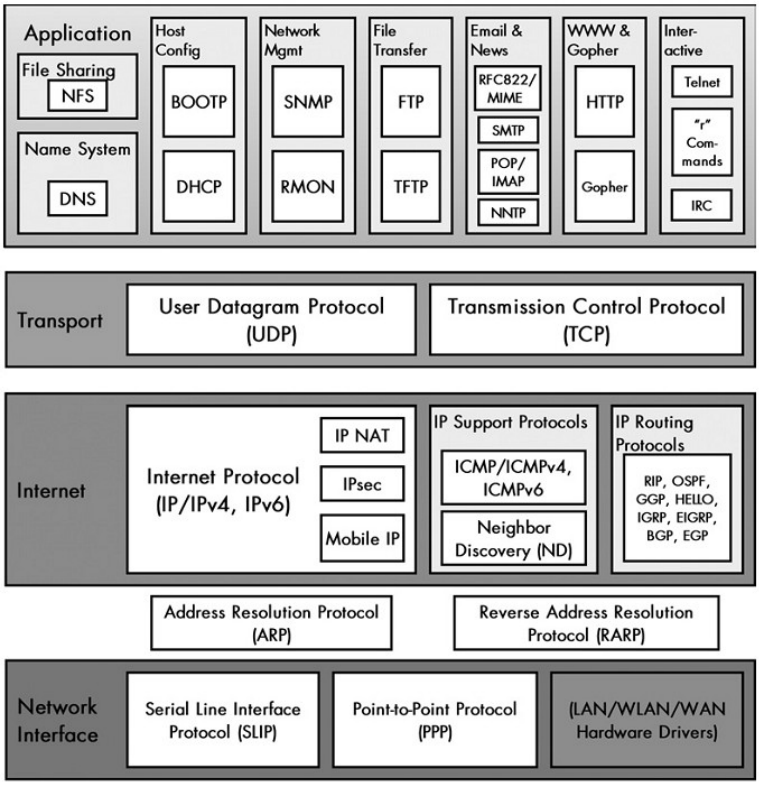

Figura 1.1: Struttura dell'architettura e dei protocolli TCP/IP

TCP/IP utilizza un'architettura basata su quattro livelli (simile al modello OSI) e include numerose applicazioni ad alto livello come HTTP (Hypertext Transfer Protocol) e FTP (File Transfer Protocol).

I protocolli di TCP/IP furono inizialmente creati negli anni settanta come parte della rete di ricerca sviluppata negli Stati Uniti *ARPAnet*, prendendo la forma di un unico protocollo al cuore della rete chiamato *Transmission Control Program* (1973).
Fu nell'agosto del 1977 che si ebbe il primo vero punto di svolta nell'affermazione di TCP/IP: Jon Postel, informatico americano, pubblicò una serie di commenti sullo stato di TCP in cui delineava la costruzione di una struttura modulare costruita a livelli su cui basare la rete, in modo da poter utilizzare TCP strettamente come un protocollo a livello *host-to-host* [1], delegando le operazioni di smistamento delle informazioni a livelli inferiori.

L'osservazione di Postel port quindi alla definizione di due nuovi protocolli, TCP (Transmission Control Protocol) a livello di trasporto e IP (Internet Protocol) a livello di rete, e alla definizione dell'architettura TCP/IP. Quest'ultimo divenne rapidamente il protocollo standard alla base di ARPAnet, e negli anni ottanta a causa di una rapida crescita del numero di computer connessi a questa rete nacque Internet basato su TCP/IP [2].

ARPAnet venne presto sostituita da NSFNet che, nonostante presentasse le stesse prestazioni della rete precedente, mirava ad una diffusione su larga scala dell'utilizzo di Internet e si strutturava in tre componenti: un *core* più veloce, reti regionali e reti locali.

Al giorno d'oggi Internet comprende centinaia di migliaia di reti in tutto il mondo e non dipende più da un *core* gestito dal governo, ma si è evoluto arrivando ad uno stato di indipendenza dalle singole reti e dalle agenzie che hanno contribuito alla sua costruzione; tuttavia attraverso questa molteplicità di trasformazioni una cosa è rimasta costante: Internet è costruito su TCP/IP.

1.1 OSI vs TCP/IP

Un modello architetturale sviluppato dall'International Standards Organization (ISO) è spesso usato per descrivere le relazioni tra le diverse tecnologie e i protocolli che governano la rete. Esso è chiamato OSI (Open System Interconnect) Reference Model, ed è bastato sul concetto di stratificazione della rete, di suddivisione in livelli.

L'OSI Reference Model contiene sette livelli numerati, ognuno dei quali ha determinate responsabilità e funzionalità specifiche e si occupa di comunicare con il livello immediatamente superiore e con quello immediatamente inferiore.
Procedendo attraverso i livelli dal primo al settimo si risale lo stack, e man mano ci si allontana da funzioni concrete e altamente dipendenti dall'hardware incrementando il livello di astrazione.

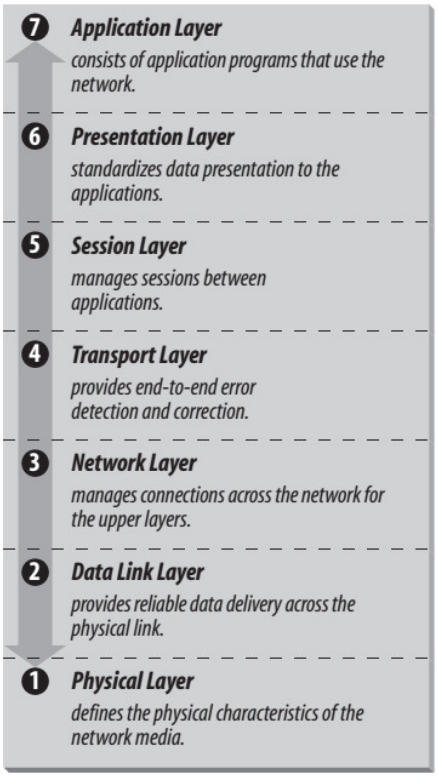

Figura 1.2: Livelli dell'OSI Reference Model

I livelli più bassi (*Physical Layer, Data Link Layer, Network Layer, Transport Layer*) si occupano della formattazione, codificazione e trasmissione dei dati sulla rete, dando minore rilevanza ai dati veri e propri. I livelli superiori (*Session Layer, Presentation Layer, Application Layer*) si occupano invece dell'inerazione con l'utente e della gestione delle informazioni scambiate atraverso la rete, non curandosi di come queste vengono ricevute o inviate.
Descriviamo brevemente i sette livelli ISO/OSI [2].

1 - Physical Layer
Si occupa di definire le specifiche hardware necessarie per effettuare la trasmissione e di trasformare la rappresentazione fisica dei dati in segnali idonei alla trasmissione.

2 - Data Link Layer
Si occupa di consegnare i dati in maniera affidabile, gestendo gli errori e l'eventuale necessità di ritrasmissione; inoltre stabilisce e controlla i link logici tra device sulla

rete, incapsula lo stream di bit in arrivo dal livello inferiore in una struttura dati chiamata *frame* e si occupa dell'indirizzamento delle informazioni.

3 - Network Layer
Gestisce le operazioni di routing tra nodi all'interno della rete, si occupa dell'indirizzamento logico dei nodi, incapsula i frame ricevuti in unità dati chiamate *datagram* o pacchetti e si occupa della frammentazione dei dati nel caso la loro dimensione non sia idonea alla trasmissione.

4 - Transport Layer
E' incaricato di rendere possibile la comunicazione *end-to-end* tra applicazioni, prepara alla trasmissioni i dati inviati dall'applicazione segmentandoli, può fornire servizi di controllo di flusso per garantire una trasmissione affidabile.

5 - Session Layer
Si occupa di genstire e mantenere le sessioni tra applicazioni, grazie alle quali esse possono trasmettersi informazioni nel tempo. Le tecnologie di questo livello sono spesso implementate come strumenti sofware chiamati *application program interfaces*, APIs.

6 - Presentation Layer
Gestisce la rappresentazione dei dati effettuando manipolazioni quali traduzione in diversi formati, compressione e operazioni di crittografia.

7 - Application Layer
Gestisce i processi a livello utente che interagiscono con la rete.

Nonostante il modello appena presentato proponga una visione completa delle interazioni di rete, la suite di protocolli TCP/IP nacque prima che esso venisse sviluppato; essa presenta comunque una struttura a livelli, ma essi furono interpretati in maniera leggermente differente e riconosciuti in numero minore. Come nel modello OSI i dati attraversano lo stack verso il basso quando vengono inviati e ogni livello nello stack aggiunge informazioni di controllo ai dati (*header*) per assicurare la correttezza delle operazioni. Tale azione viene chiamata *incapsulamento*.
Durante la ricezione avviene il fenomeno opposto, ovvero gli header vengono eliminati man mano che i dati risalgono lo stack.

Il modello TCP/IP basato su quattro livelli logicamente corrispondenti ai livelli del modello OSI, come mostrato nella figura (1.3). Procediamo ad un'analisi più attenta di questi quattro livelli.

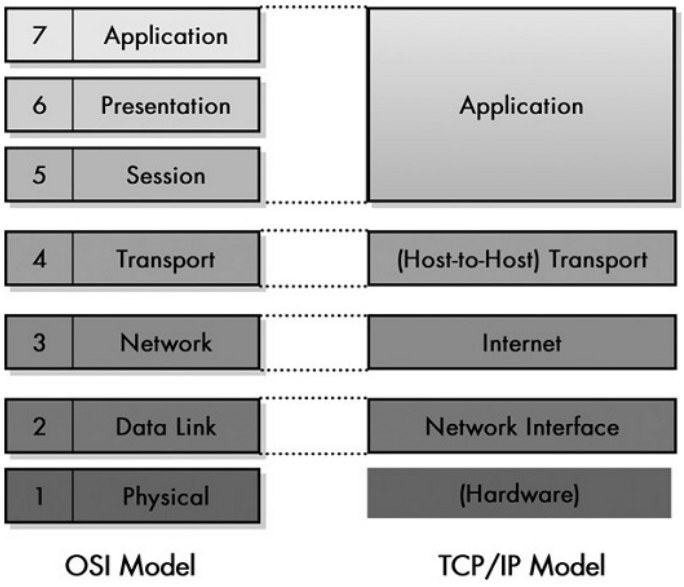

Figura 1.3: Confronto tra OSI Reference Model e TCP/IP

Network Access Layer
Comprende le funzioni dei protocolli inferiori dell'OSI Reference Model. Fornisce i mezzi al sistema per consegnare i dati ad altri device sulla rete, effettuando operazioni quali l'incapsulamento dei datagram IP in frames per la trasmissione e la mappatura degli indirizzi IP su indirizzi fisici [3].

Internet Layer
Corrisponde al livello Network del modello OSI, con le stesse funzionalità, ed il suo protocollo più importante prende il nome di *Internet Protocol*, ovvero IP. Esso è conosciuto come IPv4 o IPv6 a seconda del suo versionamento, ed a causa della sua importanza verrà trattato nel dettaglio più avanti.

Host-to-Host Transport Layer
Controlla e gestisce le connessioni logiche tra device sulla rete che permettono ai dati di essere trasmessi in maniera inaffidabile (senza alcuna garanzia cha raggiungano la destinazione) o affidabile (dove il protocollo mette in atto una serie di operazioni per assicurarsi che la trasmissione vada a buon fine). I protocolli a questo livello sono chiamati TCP, *Transmission Control Protocol*, e UDP, *User Datagram Protocol*.

Application Layer
Comprende i livelli cinque, sei e sette del modello OSI e si occupa di gestire tutti i processi che utilizzano TCP/IP per trasmettere dati. I protocolli inclusi in questo

ilvello sono numerosi, ma tra i più importanti possiamo citare Telnet, FTP, SMTP e HTTP.

1.2 Internet Protocol

IP è il cuore della suite di protocolli TCP/IP e il protocollo principale all'interno del Network Layer. Il suo scopo principale è la trasmissione di dati a dispositivi che possono trovarsi anche su reti diverse, connessi tra loro in maniera arbitraria.
Prima di definire con precisione cosa ciò implichi, analizziamo le caratteristiche del protocollo.

Innanzitutto IP è definito *senza connessione*, ovvero non avviene una trasmissione di informazioni di controllo per stabilire una connessione prima che i dati vengano trasmessi. Viene inoltre detto *universalmente indirizzato*, ovvero ad ogni host direttamente connesso ad Internet viene assegnato un indirizzo unico e riconoscibile detto 'indirizzo IP'; ogni indirizzo è composto da due parti: un identificatore di rete (netid) e un identificatore di host (hostid)[4].

Una caratteristica fondamentale di IP è la sua indipendenza dai protocolli sottostanti. Esso è infatti costruito per permettere la trasmissione dei dati attraverso ogni tipo di rete supportata dallo stack TCP/IP, adattandosi dunque a vari protocolli di grande diffusione come Ethernet o Wi-Fi 802.11, ma anche speciali protocolli data link come SLIP e PPP. Per fare ciò è rilevante la capacità di IP di frammentare grandi blocchi di dati in unità piú piccole dette 'frames' per assecondare i limiti di dimensione dati imposti dalle reti fisiche.

IP viene definito *inaffidabile*, ovvero non fornisce servizi di riconoscimento errori, di recupero dati o ritrasmissione. Per questo motivo a volte viene definito 'best-effort'.

Il motivo di una simile configurazione senza alcuna garanzia risiede nell'ottimizzazione dei costi di trasmissione: IP è incaricato di trasferire un'immensa quantitá di dati continuamente, e questo tipo di operazione richiede tempo, risorse e determinate dimensioni di banda; appesantire tutto ciò con caratteristiche di affidabilità causerebbe un overhead non sempre necessario.

1.2.1 Datagram IP

Il formato dei dati definito da IP viene chiamato *datagram*. Nella descrizione dei vari campi di questa struttura dati ci concentreremo sulla versione quattro del protocollo IP, detta IPv4.
Le prime cinque o sei (la sesta è opzionale) *word* da 32 bid del datagram rappresentano informazioni di controllo aggiunte dal protocollo stesso e vengono chiamate *header*. Dato che la lunghezza dell'header è variabile, esso contiene un campo fondamentale chiamato Internet Header Length (ILH) che ne descrive le dimensioni.

IP tratta ogni datagram come un'entità indipendente, senza alcuna connessione fisica o logica ad altri dati dello stesso tipo.

All'interno dell'header i campi chiave utilizzati da IP per eseguire correttamente il servizio di trasmissione, a parte logicamente le specifiche di sorgente, destinazione e i payload di dati, sono quattro: *Type of Service, Time to Live, Options* e *Header Checksum* [5].

Type of Service, detto anche *Differentiated Services*, viene usato per indicare la qualitá del servizio desiderato. Questo tipo di indicazione è utilizzata dai gateway per selezionare i corretti parametri di trasmissione per ogni particolare rete, la rete da utilizzare per il prossimo passo da effettuare (*next hop*) o il prossimo gateway verso cui indirizzare l'operazione di routing del datagram. Inoltre gli ultimi due bit del campo vengono utilizzati per comunicare condizioni di congestione.

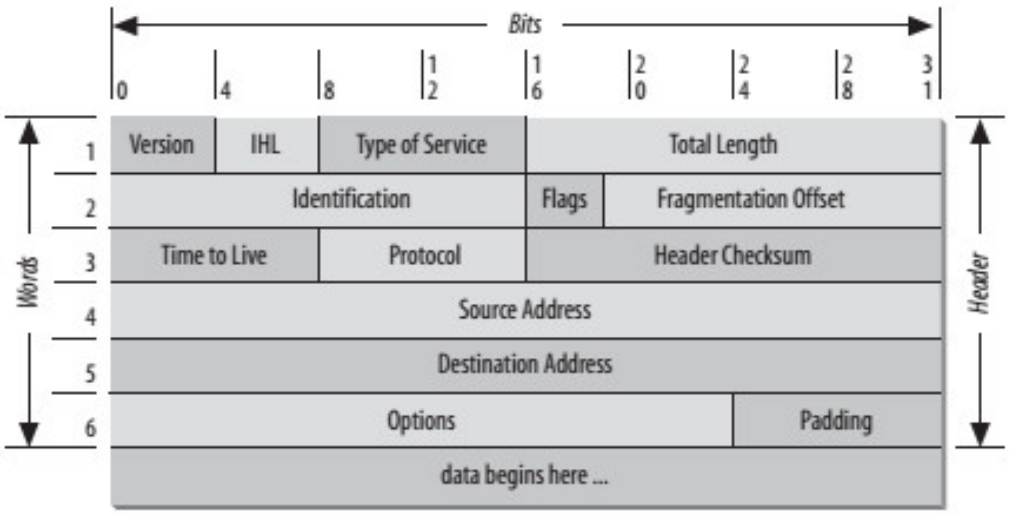

Figura 1.4: Header di un datagram IP

Time to Live rappresenta il limite superiore in tempo al ciclo di vita di un datagram IP. Il campo viene settato da colui che invia il datagram e decrementato man mano che percorre i diversi passi in cui viene processato. Se il tempo raggiunge lo zero prima che il datagram raggiunga la sua destinazione, quest'ultimo viene distrutto.

Options fornisce determinate funzioni di controllo quali timestamp, sicurezza e routing speciale.

Header Checksum fornisce la possibilità di verificare che l'informazione utilizzata nel processare il datagram sia stata trasmessa correttamente. Se il checksum fallisce, il datagram viene scartato immediatamente.

All'interno dell'header possiamo riconosce anche i campi *Version*, che identifica la

versione corrente del protocollo IP, *Total Length* ovvero la dimensione complessiva del datagram (non superiore a 65535 Byte), *Identification*, che permette all'host di assemblare i vari frammenti in un unico pacchetto (tutti i frammenti appartenenti allo stesso pacchetto hanno uguale Identification Value), *Don't Fragment, More Fragments* e *Fragmentation Offset* necessari alla gestione della frammentazione, *Protocol* identifica il protocollo a livello Transport a cui assegnare il pacchetto (TCP, UDP, etc..), *Source Address* e *Destination Address* [6].

Option	Description
Security	Specifies how secret the datagram is
Strict source routing	Gives the complete path to be followed
Loose source routing	Gives a list of routers not to be missed
Record route	Makes each router append its IP address
Timestamp	Makes each router append its address and timestamp

Figura 1.5: Alcune possibilitá del campo Options nell'header IP

Per meglio comprendere i concetti illustrati, analizziamo con attenzione lo scenario di trasmissione di un datagram IP.
L'applicazione che invia i dati, dopo averli correttamente preparati, chiama il modulo di rete locale per inviare i dati come datagram passandogli le informazioni necessarie come destinazione e opzioni di invio. Il modulo di rete prepara un header IP per il datagram e vi collega i dati ricevuti; dopodiché determina l'indirizzo di rete locale relativo all'indirizzo di destinazione del datagram. Assumiamo per comoditá che ci sia un solo gateway tra la sorgente e la destinazione, quindi che la ricerca di un indirizzo locale riporti l'indirizzo di tale gateway.

I dati vengono quindi inviati all'interfaccia di rete locale, che crea un nuovo header a cui allegare il datagram ricevuto e che trasmette il risultato sulla rete. Il datagram arriva al gateway host, la cui interfaccia di rete locale spoglia i dati dell'header più esterno e consegna il datagram al modulo internet; esso, leggendo le informazioni contenute nell'header, stabilisce la necessità di inoltrare il pacchetto verso la sua destinazione reale, determinandone l'indirizzo. Il datagram viene quindi nuovamente incapsulato dal modulo di rete locale e inviato a destinazione.

Una volta giunto a destinazione, l'header più esterno viene nuovamente rimosso e il controllo passato al protocollo internet; esso stabilisce che la destinazione del datagram sia a tutti gli effetti un'applicazione su quell'host e trasmette il datagram a livello più alto grazie alle funzionalità fornite dalle chiamate di sistema [5].

1.2.2 Funzionalità di IP

I compiti principali assegnati a IP si possono dividere in quattro categorie.

Incapsulamento

Come già illustrato precedentemente, il protocollo riceve i dati dal livello superiore e li incapsula in un datagram con appropriato header per poi farli proseguire nella discesa dello stack verso la rete.

Indirizzamento

Prima di spedire un datagram, IP deve conoscerne la destinazione. È quindi molto importante che il protocollo venga fornito di un meccanismo di indirizzamento, tramite cui riconoscere univocamente ogni host sulla rete. Gli indirizzi IP premettono la corretta identificazione delle interfacce di rete a cui sono associati e la realizzazione del processo di routing. A questo punto è necessaria una distinzione tra nomi, indirizzi e le cosiddette *routes*, strade: il nome identifica cosa dtiamo cercando, l'indirizzo contiene informazioni riguardo a dove trovarlo, la *route* indica come raggiungerlo. Il compito di IP è quello di mappare correttamente l'indirizzo di destinazione su un indirizzo locale, mentre sta a i protocolli di livello superiore ricondurre tale indirizzo ad un nome e a quelli di livello inferiore di trovare la *route* corretta. Gli indirizzi sono stringhe di lunghezza fissata (32 bit nel caso di IPv4, 128 bit nel caso di IPv6) composti come già visto da un *netid* e da un *hostid*.

Frammentazione

Frammentare e deframmentare i pacchetti é uno dei compiti piú importanti del protocollo ed è la feature più interessante agli scopi del progetto trattato, quindi verrà analizzato con particolare attenzione.

IP definisce la dimensione massima di un datagram (MTU) a 64KB (65535 Byte), per il fatto che il campo che stabilisce la lunghezza massima all'interno dell'header é lungo 16 bit. Tuttavia non sono molte le interfacce che inviano normalmente pacchetti di tale lunghezza; ciò implica che quando IP deve trasmettere un pacchetto la cui dimensione é superiore a MTU necessita di dividerlo in parti più piccole. Il processo di frammentazione crea una serie di frammenti di uguale dimensione e se MTU non divide esattamente la dimensione del pacchetto originale, l'ultimo frame risulterà essere più piccolo degli altri. Questo tipo di frammentazione può essere fatto non solo dall'host che invia i pacchetti, ma anche da router e altri dispositivi lungo la strada; per questo motivo ogni frammento a sua volta può essere ulteriormente spezzato, e deve quindi esistere un modo per permettere al destinatario di ricomporre il pacchetto originale. A questo scopo vengono utilizzati i campi dell'-header IP mostrati precedentemente, in particolare i campi relativi alla sorgente e alla destinazione del datagram e l'ID del pacchetto IP generato dal kernel.

Durante la ricezione di un datagram, esso non può essere inoltrato da IP ai livelli

superiori dello stack finchè tutti i frammenti non vengono correttamente riassemblati (processo di deframmentazione). Tuttavia ciò non implica che i frammenti vengano tenuti per tempo illimitato all'interno della memoria dell'host: ogni router e host è provvisto di un timer utilizzato per ripulire le risorse usate dai frammenti IP nel caso tutti i frames del pacchetto non riescano ad essere recuperati [7].

Routing
Quando un datagram dev'essere inviato all'interno di una rete locale, ciò può essere tranquillamente eseguito utilizzando protocolli come LAN, WLAN o WAN; tuttavia nella maggior parte dei casi la destinazione è su una rete lontana e non direttamente connessa alla rete di partenza. In questo tipo di situazione il datagram dev'essere inviato indirettamente, attraverso un'operazione di routing del datagram attraverso device chiamati appunto *router*. IP svolge questo compito con la collaborazione di altri protocolli come ICMP e i protocolli di gateway/routing propri di TCP/IP come RIP e BGP.

1.2.3 ICMP

Una parte integrante di IP è l'*Internet Control Message Protocol* (ICMP), costruito per effettuare controlli e riportare eventuali errori o informazioni funzionali a Livello Internet. Come già illustrato IP non è un protocollo affidabile, per cui lo scopo di questi messaggi di controllo è di fornire feedback riguardo a problemi nell'ambiente di comunicazione; non c'è ancora alcuna garanzia che la consegna dei datagram avvenga in modo affidabile, ma grazie a ICMP si riescono ad individuare errori durante la lavorazione dei dati [8].
ICMP fornisce supporto a IP nella forma di *messaggi* che permettono l'occorrere di diversi tipi di comunicazione tra device IP, e ognuno di questi messaggi è incapsulato in un datagram IP per la trasmissione. Differisce dagli altri protocolli di TCP/IP in quanto non esegue un particolare tipo di azioni per uno scopo ma definisce un standard di scambio di informazioni.

Una delle funzionalità di ICMP è fornire informazioni riguardo al controllo di flusso: quando i datagram arrivano troppo velocemente, l'host di destinazione o un gateway intermedio inviano un messaggio ICMP del tipo *Source Quench Message* per rallentare il flusso di informazioni.

Quando una destinazione è irraggiungibile, il sistema che si accorge del problema invia un *Destination Ureachable Message* alla sorgente, e nel momento in cui un gateway ritiene che la strada migliore sia attraverso un altro gateway invia un messaggio di tipo *Redirect*. Inoltre i messaggi di tipo ICMP possono anche essere utilizzati per controllare l'operativitá di IP su host remoti tramite la funzionalità *Echo*.

1.3 Transmission Control Protocol

Il protocollo primario per quanto riguarda il Transport Layer nella suite di protocolli TCP/IP è il Transmission Control Protocol (TCP). Esso fornisce un meccanismo di indirizzamento ai processi sotto forma di porte, attraverso le quali é possibile stabilire una connessione; una volta connessi i dispositivi possono effettuare uno scambio di tati bidirezionale.
TCP comprende una vasta serie di meccanismi; essi garantiscono che i dati transitino dalla sorgente alla destinazione in maniera affidabile, coerente e con tempistiche accettabili. La chiave per il suo funzionamento in questo senso è il cosiddetto *Sliding Window Acknowledgement System*, che permette ad ogni dispositivo di tenere traccia della quantità di dati che ha inviato e di confermare la corretta ricezione dei dati da parte dell'altro dispositivo. Dati non confermati vengono eventualmente ritrasmessi in modo automatico e i parametri del sistema possono essere riadattati ai bisogni dei device e delle connessioni.

Per comprendere il concetto di 'Sliding Window' dobbiamo partire da un modello di base di ciò che riteniamo 'affidabilità'.
Supponiamo che, in una comunicazione tra due dispositivi, ad ogni pacchetto inviato corrisponda una risposta di tipo *acknowledgement*, riconoscimento, che ne certifichi l'avvenuta ricezione. Ci troviamo di conseguenza in un modello in cui il dispositivo sorgente deve attendere un segnale dalla destinazione prima di poter inviare un nuovo pacchetto e nel caso ritrasmettere quello eprduto nel caso non venga riconosciuto.

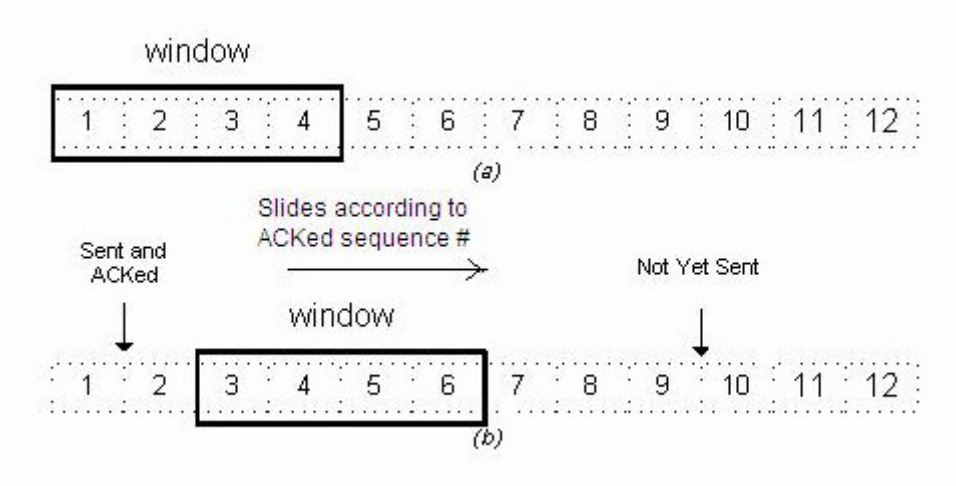

Figura 1.6: Rappresentazione della Sliding Window

La tecnica della Sliding Window è una forma pù complessa del sistema di riconoscimento e ritrasmissione appena visto, che fa un migliore uso della larghezza di banda

disponibile in quanto permette alla sorgente di trasmettere più pacchetti prima di inviare un acknowledgement.
Si consideri l'invio di una sequenza di pacchetti: il protocollo assegna una piccola finestra di dimensione fissata alla sequenza e trasmette tutti i pacchetti che risiedono all'interno dell'intervallo considerato. Un pacchetto viene detto 'non riconosciuto' se in seguito alla sua trasmissione non viene ricevuto il relativo acknowledgement; tecnicamente in questo caso il numero di pacchetti non riconosciuti ad un determinato istante è limitato dalla dimensione della finestra.

Nel momento in cui la sorgente riceve l'acknowledgement per il primo pacchetto della finestra, quest'ultima 'scorre' verso i pacchetti non ancora inviati causandone la trasmissione; lo spostamento continua man mano che arrivano gli acknowledgement.

Concettualmente il protocollo che implementa tale finestra tiene in memoria i pacchetti che sono stati inviati correttamente, mantenendo un timer separato per ciascuno dei pacchetti non ancora riconosciuto. Nel momento in cui un pacchetto viene perso, la ritrasmissione avviene allo scadere di tale timer.

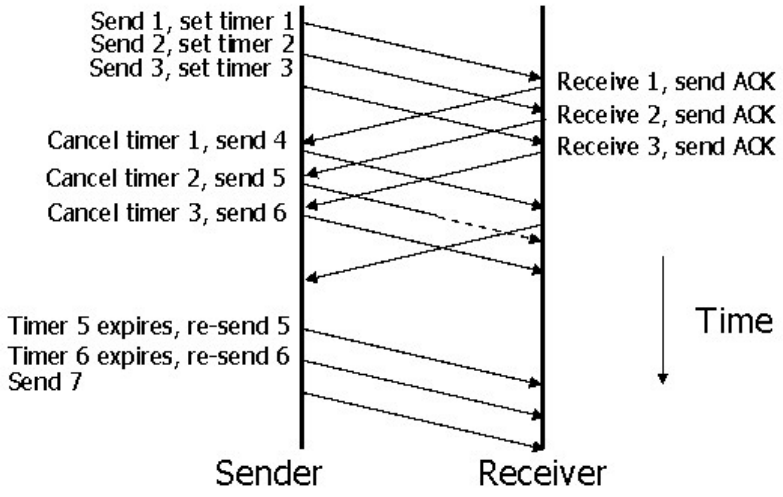

Figura 1.7: Invio dei pacchetti con meccanismo di Sliding Window

In sintesi, la finestra crea tre partizioni di pacchetti all'interno dello stream di dati: i pacchetti a sinistra della finestra sono stati trasmessi con successo, ricevuti e riconosciuti, i pacchetti a destra devono ancora essere trasmessi e i pacchetti contenuti all'interno della finestra sono in trasmissione. Il pacchetto con indice inferiore nella finestra è il primo in attesa di riconoscimento [9].
La caratteristica fonamentale del meccanismo di Sliding Window implementato su TCP è la variabilità della dimensione della finestra nel tempo: a seguito di ogni trasmissione riuscita il protocollo riconosce nel messaggio di acknowledgement la quantità di dati che il

destinatario è disposto a ricevere e aggiusta le dimensioni della finestra di conseguenza. Questa funzionalità viene implementata non solo per una questione di affidabilità del protocollo, ma anche perchè permette un meccanismo di *controllo di flusso*, ovvero di gestione della quantità di dati inviata in modo da non sovraccaricare il dispositivo che li riceve.

1.3.1 Segmento TCP

TCP viene definito un protocollo *byte-oriented*, ovvero il cui scambio di dati attraverso la connessione avviene tramite la trasmissione diretta di byte. Il servizio offerto da TCP alle applicazioni può essere quindi definito uno 'stream di byte', che vengono organizzati in strutture dati ben precise durante la trasmissione.

Lo stream proveniente dall'applicazione viene organizzato in un buffer da TCP, e una volta raggiunta una quantità sufficiente per l'invio i dati all'interno del buffer vengono incapsulati in un pacchetto e trasmessi. Tali pacchetti prendono il nome di 'segmenti' in quanto ciascuno di essi trasporta un segmento dello stream di byte.

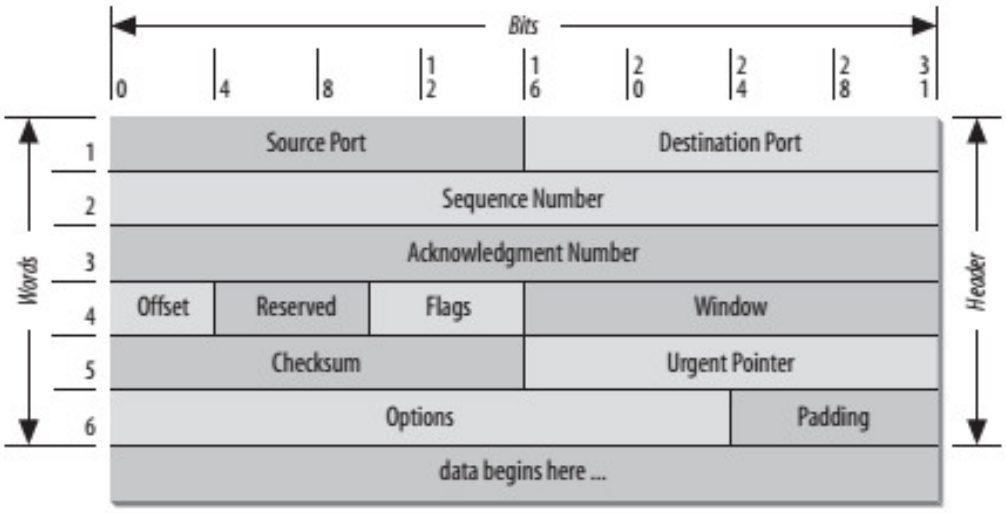

Figura 1.8: Header di un segmento TCP

Ciascun segmento è dotato di un header che contiene le informazioni necessarie alla trasmissione. *Source Port* e *Destination Port* sono i campi che, combinati con l'indirizzo IP, identificano univocamente la connessione e stabiliscono il tragitto del segmento. *Sequence Number, Acknowledgement Number* e *Window* concorrono a realizzare il meccanismo di Sliding Window: il primo contiene il numero di sequenza (di byte nello stream) del primo byte trasportato in quel segmento, il secondo contiene il numero di sequenza

che la destinazione si aspetta di ricevere al prossimo invio, il terzo contiene la quantità di dati che la destinazione è ancora disposta a ricevere.

Il campo *Flags* (6 bit) viene utilizzato per trasmettere informazioni di controllo sulla comunicazione; tra i possibili flags troviamo SYN, FIN, RST (o RESET), PSH (o PUSH), URG e ACK. SYN e FIN vengono utilizzati rispettivamente per stabilire e terminare la connessione, ACK viene settato per notificare la validità del campo Acknowledgement, URG che il segmento contiene dati urgenti, PUSH richiede l'immediata trasmissione dei dati a livello applicazione e RESET indica la volontà del destinatario di terminare la connessione. Nel caso il bit che notifica dati urgenti sia settato, essi si troveranno a partire dall'inizio del segmento fino alla locazione puntata da *Urgent Pointer*.

Checksum protegge l'integrità del pacchetto e *Offset* indica la quantità di parole da 32 bit che sono presenti all'interno dell'header [10].

1.3.2 Connessione

TCP è costruito sul concetto di 'astrazione di connessione', ovvero prima che i dati possano essere trasmessi tra due device essi devono essere in accordo. Ogni connessione viene identificata da una coppia di *endpoint* o *socket*, ciascuno di essi è a sua volta una coppia <*host, port*>dove *host* rappresenta l'indirizzo IP della macchina e *port* è il numero della porta TCP. Ad esempio, l'endpoint <128.10.2.3, 25>identifica la porta 25 sulla macchina con indirizzo IP 128.10.2.3.

Questo tipo di definizione di connessione a coppie implica che ogni porta può essere condivisa da molteplici connessioni sulla stessa macchina.
Siccome ogni connessione è a sé stante è necessario mantenere distinti i dati che la riguardano. TCP utilizza uina struttura particolare a questo scopo detta *Transmission Control Block*, TCB; questo contiene tutte le informazioni rilevanti riguardo alla connessione come l'identificativo dei socket, i puntatori ai buffer che contengono i dati in uscita o in entrata e le variabili che permettono la realizzazione del meccanismo di Sliding Window.

TCP/IP è basato su un modello operazionale di tipo client/server che si riflette sulla modalità di setup della connessione; sia il client che il server si preparano alla trasmissione effettuando un'operazione di tipo *Open*, che può esser di due tipi:

- Attiva: il client prende il controllo e inizia attivamente la connessione inviando un messaggio di tipo SYN;
- Passiva: il server si rende disponibile per la connessione su una determinata porta, mettendosi in uno stato di attesa;

Sia il client che il server creano il TCPB necessario nel momento in cui effettuano la *Open*. Mentre il client conosce sia il proprio indirizzo che quello del server e riesce quindi a settare correttamente tutti i campi del TCB, il server, che si trova in attesa passiva,

non conosce l'indirizzo del client che farà richiesta di connessione e lascia non specificato il campo identificatore del sockjet fino ad avvenuta connessione.

Il TCB è mantenuto lungo tutta la durata della connessione e viene distrutto quando essa viene terminata e i dispositivi tornano ad uno stato *CLOSED*.

Handshake a tre vie

Per stabilire al connessione, ogni dispositivo deve inviare un messaggio di tipo SYN e riceverne uno di tipo ACK; questo tipo di scambio genera quattro messaggi. È tuttavia possibile fare di meglio, in quanto si possono settare contemporaneamente i flag SYN e ACK all'interno dell'header TCP, per raggiungere un totale di tre messaggi: il client invia il messaggio SYN al server, che risponde con un messaggio SYN+ACK per dichiarare la propria intenzione a connettersi; alchè il client riconoscerà l'intenzione del server con un ulteriore ACK.
Questo meccanismo prende il nome di 'Handshake a tre vie'.

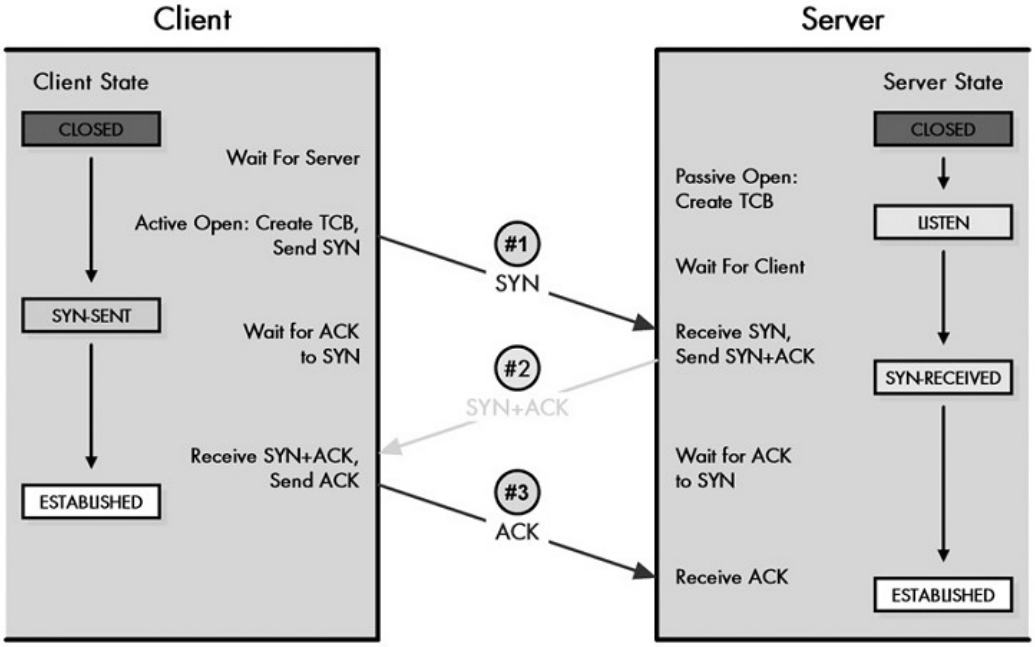

Figura 1.9: Handshake a tre vie

Questo tipo di meccanismo stabilisce anche altri parametri della connessione, come ad esempio l'*Initial Sequence Number* (ISN) che viene inviato dal client assieme al messaggio di SYN per stabilire l'identificativo della sequenza di byte da cui partirà la trasmissione di dati, lo spazio disponibile per la ricezione dei dati (window) usato per stabilire la

dimensione della finestra, e la dimensione massima del segmento (MSS) che l'host può accettare.

Costruiamo un esempio per comprendere meglio i concetti appena illustrati. Un server si mette in attesa passiva e aspetta che il client trasmetta il pacchetto SYN, in cui stabiliremo un ISN di 2000, un MSS di 1460 (abbastanza comune in quanto molti host si trovano su Ethernet LAN) e un'ampiezza di finestra pari a 5840 (4x1460).
Il server a sua volta risponde con SYN+ACK e dichiara la connessione aperta, settando il proprio ISN a 4000 e riconoscendo l'ISN del client a 2001 (ovvero il server si aspetta che il prossimo pacchetto inviato dal client abbia ISN pari a 2001); quindi stabilisce il proprio MSS uguale a 1460 e l'ampiezza della propria finestra a 8760 (6x1460).
Infine il client dichiara la connessione aperta e ritorna un ACK con l'ISN atteso, ovvero 2001, e il campo di riconoscimento settato a 4001 [11].

Il numero di sequenza iniziale è di fondamentale importanza in quanto non solo se non settato correttamente può portare al danneggiamento della trasmissione da parte di entitá esterne malintenzionate, ma viene utilizzato anche per rilevare eventuali perdite di pacchetti, come vedremo in seguito.

1.3.3 Gestione del timer

Il metodo utilizzato da TCP per il riconoscimento e la ritrasmissione è concettualmente semplice: ogni volta che un segmento viene inviato, si fa partire un timer di ritrasmissione. Se il timer scade prima della ricezione di un acknowledgement per quel segmento, esso viene ritrasmesso.

TCP necessita di un'attenta gestione del timer, in quanto il precorso del segmento attraverso una rete che può essere composta da numerose sottoreti diverse rende impossibile la conoscenza a priori del tempo che il segmento di acknowledgement impiegherà per arrivare alla sorgente. Inoltre il *delay* ad ogni router dipende dal traffico sulla rete, quindi il tempo impiegato per il viaggio di un segmento può cambiare in maniera sostanziale ad ogni istante.

TCP asseconda questa repentina possibilità di cambiamento facendo uso di un algoritmo di ritrasmissione adattivo. In sostanza TCP controlla la performance di ogni connessione e deduce valori approssimativi per il timeout, e nel momento in cui queste subiscono modifiche adatta il valore del timeout di conseguenza. Per raccogliere i valori necessari a questo tipo di calcolo TCP tiene in memoria l'istante di arrivo del pacchetto e l'istante di arrivo del segmento di acknowledgement, computando il tempo trascorso tra i due istanti per costruire un *Round Trip Sample*. ogni nuovo RTS ottenuto contribuisce al calcolo del cosiddetto *Round Trip Time*, RTT.

Tcp calcola il RTT come una media pesata, usando il valore del RTS per effettuare cambiamenti sulla stima.
Una delle prime tecniche pesate per calcolare il RTT fa uso di una variabile α detta peso,

$0<\alpha<1$, e la seguente formula [12]:

$$RTT = (\alpha \cdot oldRTT) + ((1 - \alpha) \cdot RTS)$$

Scegliendo un valore di α vicino ad 1 si rende la media calcolata immune a cambiamenti poco duraturi, scegliendo invece un valore vicino a 0 la si rende estremamente suscettibile.
Durante l'invio di un pacchetto, TCP computa il valore del timeout come una funzione della corrente stima del RTT, definendo una variabile $\beta>1$ e la seguente formula:

$$Timeout = (\beta \cdot RTT)$$

Scegliere β in maniera arbitraria da luogo ad imprecisioni: valori troppo vicini ad 1 causano una buona velocità di ritrasmissione contrapposta a scarsa elasticità del protocollo, valori troppo grandi ottengono l'effetto opposto. La specifica originaria consiglia β uguale a 2.

Il tipo di calcolo appena illustrato rappresenta tuttavia numerose imprecisioni, che con l'evolversi delle reti sono emerse sempre di pù. Un algoritmo valido per la risoluzione di questo tipo di problemi è chiamato Karn/Partridge, dal nome dei suoi creatori.

Il problema è rappresentato dal fatto che un ACK non è relativo ad una singola trasmissione, ma riconosce una generica ricezione di dati. Nel caso un segmento venga trasmesso più di una volta in seguito ad una iniziale perdita, è impossibile stabilire se l'ACK sia relativo alla prima trasmissione o alle successive, e ciò può falsare in maniera significativa il calcolo del RTT. La soluzione proposta dall'algoritmo è quella di non considerare i pacchetti ritrasmessi nel calcolo del RTS.
Karn e Partridge proposero anche una seconda modifica al meccanismo di timeout: ogni volta che TCP ritrasmette, il timer viene settato al doppio della sua dimensione precedente, piuttosto che basato sulla stima del RTT. Ciò da luogo ad un meccanismo di *backoff esponenziale*, motivato dal fatto che la maggior parte delle volte che un segmento viene perso è a causa di congestione e quindi un eccesso di cautela da parte di TCP non può che portare giovamento.

1.3.4 Error detection e ritrasmissione

Nel momento in cui viene trasmesso un segmento TCP, ne viene salvata una copia in una struttura chiamata 'Coda di Ritrasmissione' e viene stabilito un timeout, come visto in precedenza, che ne notifichi la necessità di trasmissione. Le modalità con cui TCP rileva la perdita di pacchetti e decide di ritrasmetterli sono diverse.

Ritrasmissione basata sul timer

Se il segmento inviato riceve il relativo acknowledgement entro lo scadere del timer viene rimosso dalla coda di trasmissione, altrimenti avviene un *retrasmission timeout* e il segmento viene automaticamente ritrasmesso. Naturalmente non avremo nessuna certezza che il segmento ritrasmesso venga effettivamente ricevuto, non più di quanta ne avevamo per il pacchetto originale, quindi esso dev'essere rimesso all'interno della coda di ritrasmissione e il suo timer azzerato. Dato che un meccanismo del genere può causare la creazione di un ciclo infinito, un segmento può essere ritrasmesso un numero limitato di volte prima di notificare un errore e terminare la connessione.

TCP utilizza un meccanismo di acknowledgement cumulativo, ovvero il numero di sequenza riportato nell'acknowledgement indica il prossimo pacchetto che il destinatario si aspetta di ricevere. TCP sa che un segmento è stato completamente ricevuto e riconosciuto quando il valore indicato in quel campo é maggiore dell'ultimo byte inviato.

Ritrasmissione veloce

Innanzitutto è necessario sapere che TCP genera un immediato acknowlegement (ACK duplicato) nel momento in cui i segmenti vengono ricevuti senza rispettare l'ordine atteso in base al numero di sequenza. Ciò viene eseguito per notificare la sorgente che c'è un 'buco' tra i pacchetti ricevuti, comunicandole il numero di sequenza che il destinatario si aspettava di ricevere.

Questo fenomeno ha due possibili cause: il pacchetto è stato perso oppure la sequenza dei pacchetti è stata disordinata durante il loro viaggio da sorgente a destinazione. Per eliminare quest'ultima opzione e sfruttare l'informazione per rilevare errori, il protocollo attende un certo numero stabilito di ACK duplicati prima di concludere che il pacchetto è stato perso e iniziare la procedura di ritrasmissione. Tale procedura quindi non necessita di aspettare lo scadere del timeout, per questo motivo viene chiamata 'veloce' [13].

Ritrasmissione con Riconoscimento Selettivo

Introduciamo il concetto di SACK, *Selective ACK*, ovvero di un ACK duplicato particolare a cui è stata aggiunta l'informazione riguardo a pacchetti non contigui ricevuti. In parole semplici, utilizzando il meccanismo degli ACK duplicati è possibile comunicare alla sorgente non solo quali sono i pacchetti ricevuti prima del 'buco' causato dalla non contiguità, ma anche quali pacchetti sono stati ricevuti in maniera non consecutiva e quindi dei quali non è stato inviato alcun ACK (es. il destinatario può aver ricevuto i pacchetti in ordine fino al numero 3, poi aver ricevuto i pacchetti 5 e 6). Questo meccanismo previene la ritrasmissione di segmenti già ricevuti, permettendo comunque alla sorgente di rimediare alla perdita di pacchetti.

Capitolo 2

Standard 802.11

IEEE 802.11 è un insieme di standard riguardanti il Livello Fisico del modello OSI utilizzati per regolare la comunicazione tra dispositivi all'interno di una Wireless LAN. Questo capitolo si propone di illustrare brevemente le caratteristiche fondamentali del protocollo e di mostrarne la rilevanza con il progetto trattato.

Dal 1997 ad oggi sono nati sette diverse tipologie di protocolli 802.11, per rispondere al veloce sviluppo dei mezzi fisici e tecnologici nell'implementazione delle reti wireless. Ognuno di questi protocolli si basa su un tipo di comunicazione che avviene attraverso frequenze radio, limitato da una determinata larghezza di banda, o *bandwidth*. Un concetto chiave del protocollo il Media Access Control, o MAC, ovvero un sottolivello del Data Link Layer che fornisce funzionalità di indirizzamento e meccanismi di controllo di accesso ai canali di comunicazione.

2.1 Componenti

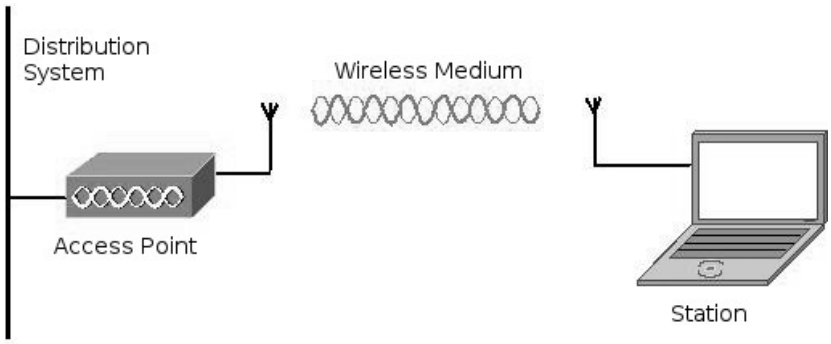

Figura 2.1: Componenti di 802.11

La rete 802.11 è composta da quattro componenti fisiche principali [14] :

Distribution System
Quando numerosi Access Point sono connessi tra loro in modo da formare un'ampia area di copertura, necessitano di comunicare l'uno con l'altro per tenere traccia dei movimenti delle stazioni mobili. Il Disrtibution System è la componente logica di 802.11 utilizzata per inoltrare i frame alla loro destinazione; nella maggior parte dei prodotti in commercio questo ruolo è implementato da Ethernet 802.3.

Access Point
I frame 802.11 devono essere convertiti in altri tipi di frame per essere trasmessi. Questo compito è svolto da dispositivi chiamati Access Point, i quali performano un'azione di *bridging* dalla rete wireless alla rete via cavo.

Wireless *Medium*
Per spostare dati da stazione a stazione, lo standard utilizza metodologie senza fili: inizialmente vennero impiegate due diverse frequenze radio (RF) e una a livello infrarossi, ma le prime ebbero ottennero maggior successo e sono poi state sviluppate e potenziate nelle versioni successive di 802.11.

Stazione
Le reti sono costruite per trasferire dati da stazione a stazione. una stazione è un dispositivo calcolatore munito di un'interfaccia di rete wireless, tipicamente un laptop o desktop computer.

2.2 Data Frame

Per quanto riguarda la modalità con cui i dati vengono trasmessi a livello 802.11 possiamo individuare tre tipi di frame: *Data Frame, Control Frame* e *Management Frame.*

I Data Frame sono coloro che trasportano dati da stazione in stazione, la cui composizione vedremo più avanti nel dettaglio; i Control Frame vengono utilizzati in congiunzione con i primi per effettuare operazioni di pulizia, acquisizione di nuovi canali e riconoscimento di avvenuta ricezione; i Management Frame effettuano operazioni di supervisione: vengono utilizzati per entrare e uscire dalle reti wireless e per spostare associazioni da Access Point ad Access Point.

Vediamo nel dettaglio la composizione di un Data Frame 802.11.
Il primo campo che troviamo in testa al frame è chiamato Frame Control. Ha dimensione pari a 2 Byte e al suo interno contiene informazioni fondamentali per l'identificazione e la corretta trasmissione del pacchetto: indica la versione del protocollo in uso, il tipo di frame (Data, Control, Management), a quale Distribution System è destinato il frame (ToDS e FromDS, sul cui utilizzo torneremo più avanti), se il frame è stato ritrasmesso

o meno, se è stato processato da WEP e così via.

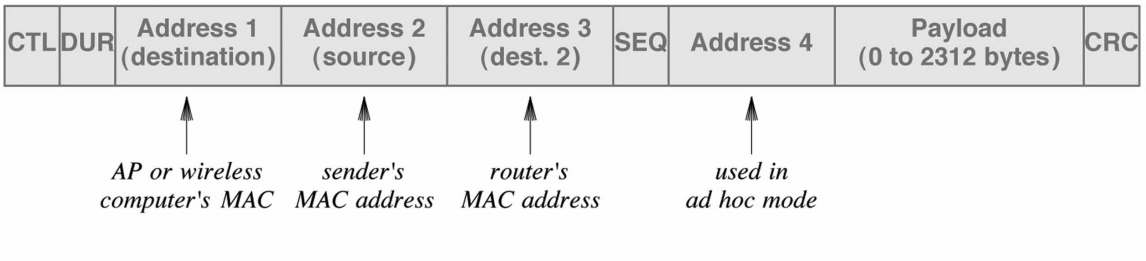

Figura 2.2: Frame 802.11

Duration è il secondo campo del frame e serve a garantire l'atomicità dell'azione di trasmissione. Esso specifica infatti il tempo richiesto per il trasferimento dei dati attraverso il *medium*, permettendo ad altre stazioni in ascolto di settare correttamente il proprio NAV (Network Allocation Vector) per stabilire quanto tempo ancora devono attendere per poter usufruire di quello stesso *medium*.

La caratteristica peculiare del frame 802.11 è che contiene quattro indirizzi piuttosto che due. Come questi indirizzi vengono interpretati dipende da come sono settati i bit nei campi ToDS e FromDS visti prima: nel caso più semplice, ovvero in cui entrambi i bit sono settati a zero, Address1 identifica il nodo destinazionee Address2 la sorgente, in quanto significa che un nodo sta inviando direttamente ad un altro; nel caso più complesso entrambi i bit DS sono settati a 1, ovvero il frame considerato viene da un nodo wireless che risiede su un'altra rete ed è destinato ad un nodo wireless su un'altra rete, Address1 identifica la destinazione finale, Address2 identifica la sorgente più vicina, Address3 identifica la destinazione intermedia e Address4 identifica la sorgente originale.

2.3 802.11 su Access Point

Approfondiamo il concetto di access point 802.11, fondamentale per la comprensione del progetto trattato in questo documento.
Un AP non è altro che un *bridge* tra una connessione wireless e una connessione via cavo, e in quanto tale ha determinate caratteristiche.

Tutti gli access point sono provvisti di almeno due interfacce di rete, una wireless in grado di interpretare i dettagli di 802.11 e una seconda interfaccia in grado di connettersi via cavo. Quest'ultima interfaccia è quasi sempre una porta Ethernet, in modo da garantire affidabilità di servizio sfruttandone la grande diffusione, ma a volte è possibile trovare anche interfacce WAN o DSL.

I bridge sono forniti di una certa quantità di memoria sotto forma di buffer, che consente di mantenere record dei frame mentre vengono trasferiti tra due interfacce e di

conservare le associazioni tra i MAC Address e le porte in una serie di tabelle interne.

L'interfaccia di rete degli AP è basata su TCP/IP: esse permettono di settare l'indirizzo IP, eventuale netmask e una gestione delle routes. I livelli di configurazione permessi all'interno dell'access point dipendono dalla sofisticatezza dell'interfaccia considerata. Talvolta possono anche offrire servizi come DHCP o NAT (Network Address Translation), specialmente gli access point utilizzati nelle abitazioni che si connettono ad un ISP (Internet Service Provider) tramite modem.

La sicurezza è una delle preoccupazioni principali per un AP, in quanto si trova in posizione ideale: essendo gateway verso la rete via cavo sono il dispositivo perfetto per mettere in atto politiche di sicurezza come ad esempio WEP o WPA.
Il metodo di controllo degli accessi maggiormente impiegato dagli access point è il *MAC Address Filtering*, ovvero la creazione di una lista di indirizzi MAC il cui accesso alla rete è permesso; i dispositvi non compresi in questa lista si vedranno negare l'accesso.

Capitolo 3

Progetto ABPS

Always Best Packet Switching (ABPS) è uno schema di comunicazione wireless *cross-layer* che permette a dispositivi mobili di sfruttare concorrentemente le proprie interfacce di rete (NIC). Esso è infatti in grado di determinare in ogni momento quale sia l'interfaccia di rete più appropriata per poter usufruire di un servizio di trasmissione ottimale, ovvero che rispetti al massimo le restrizioni imposte dal QoS (Quality of Service).

Il sistema utilizza un'architettura software distribuita il cui modulo principale è costituito dal *Client Proxy*, che esegue all'interno del nodo mobile.
Il funzionamento di ABPS può essere schematizzato nel seguente modo: all'interno del nodo mobile esegue un sistema di monitoraggio in background il cui compito è di identificare nuove possibili connessioni e di configurare le interfacce di rete; quindi il Client Proxy identifica l'interfaccia migliore basandosi su quelle correntemente attive. In realtà tutte le NIC possono essere utilizzatte concorrentemente, ed è quinti possibile passare da una rete all'altra ad ogni istante nel caso la performance di una scheda cali o la connessione fallisca [15].

Lo scopo di un simile servizio è di garantire al nodo mobile la fruibilità ininterrotta dei servizi di comunicazione, la riduzione al minimo della perdita di informazioni, elevata autonomia e riduzione dei costi.
Sfruttando l'eterogeneità delle interfacce di rete disponibili su ogni nodo mobile è infatti possibile limitare l'emissione di onde elettromagnetiche scegliendo ad ogni istante il range di connessione ottimale (IEEE802.11/b/g/n), migliorare la disponibilità di *bandwidth* permettendo un uso parallelo delle NIC, fornire ulteriori garanzie sull'affidabilità della trasmissione implementando meccanismi di rilevamento anticipato degli errori e ritrasmissione preventiva, utilizzare infrastrutture wireless già esistenti senza bisogno di apportarvi alcuna modifica e implementando i protocolli necessari solo sugli end-system, e superare le limitazioni imposte dalla diffusa presenza di firewall strutturando i protocolli senza ricorrere a soluzioni fittizie.

3.1 Robust Wireless Multipath Channel

Per quanto riguarda gli scopi di questo documento la componente di ABPS più rilevante è sicuramente il RWMPC, un meccanismo relativo a QoS il cui obiettivo principale è quello di garantire alle applicazioni VoIP un basso delay di trasmissione e una perdita di pacchetti limitata al minimo.

Il design RWMPC è basato su un approccio cross-layer in cui le diverse informazioni provenienti dai diversi livelli dello stack ISO/OSI vengono utilizzate per monitorare attivamente la comunicazione wireless utilizzata per la trasmissione della voce [16].
Possiamo riassumere il meccanismo come segue: il monitor alla base del sistema rileva l'eventuale perdita dei datagram UDP durante la trasmissione tra una NIC del nodo mobile e un Access Point Wirelesse notifica tale perdita all'applicazione da cui proviene quel determinato datagram; tale applicazione può decidere se ritrasmettere il datagram perduto utilizzando una differente interfaccia.

Quindi il meccandimo RWMPC monitora il *first-hop* nella comunicazione wireless per stabilire se essa effettivamente rispetta i requisiti QoS relativi alla perdita di pacchetti e alla qualità dell'interazione.
L'assunzione che il nodo mobile possegga più interfacce wireless è fondamentale in quanto RWMPC sfrutta le sue capacità di connettersi a più di una rete (multi-homing) per poter stabilire di volta in volta qual è la più adatta per la ritrasmissione del datagram. Se si registra in fenomeno di perdita di pacchetti, RWMPC li ritrasmette utilizzando un collegamento alternativo tra quelli disponibili sulle interfacce di rete attualmente connesse alla rete.

Il meccanismo RWMPC é essenzilmenrte un intermediario, un *middleware*, stanziato in entrambi i terminali di una comunicazione VoIP. Per una questione di rilevanza con il progetto trattato, analizziamo il middleware che si trova all'interno del nodo mobile: esso è incaricato di incapsulare i dati in datagram UDP per trasmetterli (e se necessario ritrasmetterli) al nodo di destinazione.
Questo sistema esegue sopra un sistema Linux ed è formato dalle seguenti componenti:

Transmission Error Detector
: È la componente più importante ed opera a livello MAC del protocollo 802.11. Il suo compito è di monitorare ogni singolo datagram UDP che attraversa il collegamento wireless per rilevare l'avvenuta trasmissione o eventuali fenomeni di perdita dei pacchetti. TED notifica inoltre la componente ULB inserendo un messaggio di errore nella coda dei messaggi del socket utilizzato epr inviare il datagram UDP.

UDP Load Balancer
: Questa componente riceve la notifica di errore da TED e decide se il datagram perduto debba essere rispedito o meno. Inoltre, basandosi sulla notifica ricevuta decide quale interfaccia di rete sia piú idonea alla trasmissione dei successivi datagram.

Monitor

È respondabile del monitoraggio e della configurazione delle interfacce di rete all'interno del nodo mobile e della notifica a ULB dell'interfaccia al momento più idonea alla trasmissione.

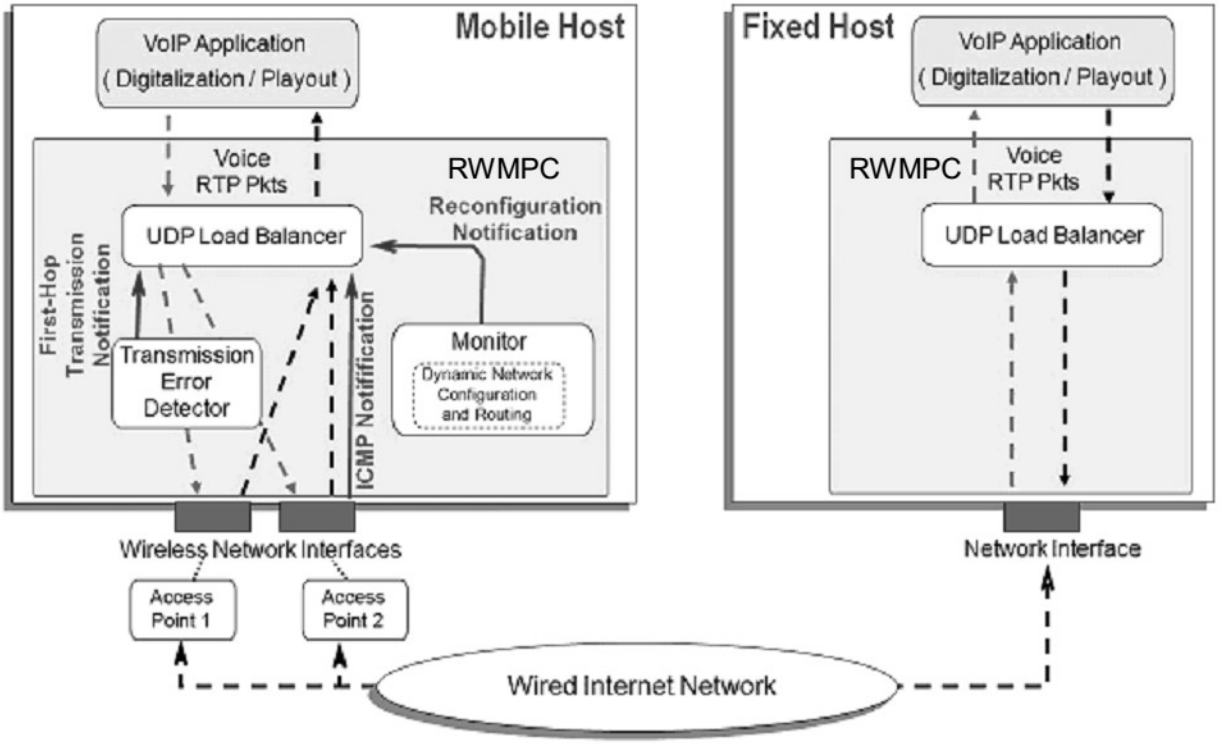

Figura 3.1: Struttura interna di RWMPC

3.2 TED su TCP

Partendo dall'idea proposta da ABPS di un Transmission Error Detector per UDP, ne è stata realizzata una versione adatta al protocollo TCP, il quale già presenta funzioni di controllo dell'affidabilità.
Lo scopo di tale realizzazione è quello di minimizzare le tempistiche di ritrasmissione dei pacchetti inserendo un controllo a livello MAC tra il nodo che trasmette i dati e l'access point, in modo da avere una notifica anticipata della perdita di un pacchetto.
Dato che la gestione della ritrasmissione di TCP avviene sfruttando un timer (come visto in precedenza), una modifica del gestore del timer e l'inserimento di un evento di ritrasmissione anticipata hanno reso possibile la realizzazione di un meccanismo di TED per TCP.

Tuttavia il problema principale risiede nell'identificazione del pacchetto perduto: scendendo lo stack dei livelli è possibile avere un riscontro tra i pacchetti in quanto l'incapsulamento ne fornisce una modalità di tracciamento, ma risalendo lo stack per la notifica dell'errore non si ha alcun modo per identificarlo correttamente tranne il payload vero e proprio.
A causa di ciò è stato aggiunto un nuovo parametro alla struttura *sk_buff*, così da avere un valore di riferimento che transita attraverso lo stack: quando viene inviato un nuovo segmento TCP attraverso la funzione del kernel *tcp_write_xmit*, associamo un ID crescente al nuovo campo skbID del sk_buf; questo valore durante la trasmissione sará replicato attraverso i vari strati del protocollo, mantenendo inalterato il riferimento tra il socket TCP e i pacchetti che arrivano a livello MAC [17].

Durante l'operazione di ritrasmissione anticipata è fondamentale evitare di aggiornare valori di conteggio dei pacchetti ritrasmessi. TCP infatti effettua una raccolta di dati sulla trasmissione per effettuare controlli di congestione e gestione del flusso, e tali valori non devono essere falsati durante la trasmissione anticipata in quanto si presume essa sia scatenata da interferenze del canale.

Capitolo 4

Progettazione e Implementazione

4.1 Obiettivi del progetto

Ciò che si vuole proporre è un meccanismo di ottimizzazione della comunicazione tra un nodo mobile e una rete wireless, coinvolgendo non gli endpoint agli estremi della rete ma l'Access Point Wireless a cui il dispositivo mobile è connesso. Ispirandosi ai concetti proposti da ABPS, si vuole sfruttare l'idea che, nel momento in cui la perdita di informazioni avviene tra il nodo mobile e l'Access Point, quest'ultimo possa essere in grado di effettuare la ritrasmissione prima che la notifica di avvenuta perdita raggiunga l'endpoint sorgente. L'intero meccanismo è stato pensato per il protocollo TCP, in quanto la presenza di funzionalità di *error detection* e di ritrasmissione rende la realizzazione più immediata.

Lo studio della possibilità di realizzare tale funzionalità ha portato all'ideazione di una struttura in due parti distinte: la creazione di un'applicazione in grado di salvare i pacchetti in transito verso il nodo mobile per ritrasmetterli in caso di perdita, e una funzionalità a livello kernel che sia in grado di notificare l'avvenuto errore di trasmissione. Quest'ultima funzionalità riprende l'idea presentata nel Transmission Error Detector dell'identificazione a livello MAC dell'avvenuta trasmissione.

All'interno di questo documento è illustrata la realizzazione della prima parte del progetto, ovvero l'applicazione. Essa è stata realizzata grazie a meccanismi di filtraggio di pacchetti e in seguito appoggiata su un meccanismo di simulazione di Error Detection che ne permette il corretto testing. Lo scambio di informazioni avviene nel seguente modo: l'applicazione salva una copia dei pacchetti tcp in transito verso il nodo mobile, per poi reimmetterli nella coda di trasmissione del kernel passandoli al simulatore; quest'ultimo decide arbitrariamente se trasmettere correttamente il pacchetto oppure "perderlo" per poi notificare in seguito l'applicazione del rilevamento di un errore. Quest'ultima, una volta ricevuta la notifica dal livello inferiore, ritrasmette il pacchetto.

4.2 Strumenti utilizzati

4.2.1 Costruire l'Access Point

Innanzitutto per realizzare il meccanismo desiderato è necessario disporre di un Access Point basato su tecnologie Open Source che permetta la manipolazione del traffico di rete che lo attraversa.
A questo scopo è stato usato un laptop di categoria eeePC con due interfacce di rete (wlan0 e eth0) su cui è stata installata la versione Wheezy di Debian Linux (kernel 3.2), e il programma demone *hostapd* che permette la trasformazione del terminale in un Access Point Wireless.

Hostapd è uno strumento che esegue in background e che permette la creazione e la gestione di Access Point Wireless e la gestione dell'autenticazione [18]. Tramite la creazione di un bridge (in questo caso) tra le interfacce di rete esistenti e la compilazione di un file di configurazione *hostapd.conf* è stato quindi installato l'AP IEEE802.11 desiderato a cui i dispositivi terminali e i nodi mobili si possono connettere per la trasmissione di informazioni.

Perchè non utilizzare OpenWRT?

Openwrt è una distribuzione Open Source GNU/Linux per dispositivi embedded, costruito come un sistema operativo completamente funzionate e modificabile per i router [19]. Lavorare su questo tipo di firmware tuttavia rende necessario implementare la struttura desiderata tutta a basso livello, senza avere la possibilità di creare un'applicazione gestibile a livello utente per poterla modificare adattandola ai propri scopi.
L'implementazione del TED risulterebbe quindi analoga, ma si perderebbe la semplicità data dalla possibilità di controllare tutte le operazioni a livello applicativo.

4.2.2 Netfilter e Iptables

Netfilter è un framework per la manipolazione dei pacchetti all'interno del kernel di linux. Il suo funzionamento ha alla base una serie di *hooks* logicamente collegati a funzioni di callback che permettono di seguire i pacchetti che scendono e risalgono attraverso lo stack dei protocolli.
Nel momento in cui un pacchetto attraversa un determinato hook, il framework controlla se a quel livello c'è qualche processo in attesa di ricevere il pacchetto e glielo passa.

Architettura

Consideriamo l'architettura di Netfilter per il IPv4, che presenta 5 diversi hook.
I pacchetti, dopo aver passato un primo controllo di sanità per assicurarsi che non ci sia

nessun tipo di corruzione, vengono passati all'hook NF_IP_PRE_ROUTING del framework. Dopo averlo attraversato entrano nella porzione di codice relativa al routing, dove si stabilisce se un pacchetto è destinato ad un'altra interfaccia o ad un processo locale. Nel primo caso viene attivato l'hook NF_IP_FORWARD, mentre nel secondo caso NF_IP_LOCAL_IN.
Il pacchetto infine, prima di essere instradato correttamente nella rete, attiva l'hook NF_IP_LOCAL_OUT.

I moduli del kernel possono registrarsi per ascoltare ogni hook: a ognuno di loro viene dato un valore di priorità e nel momento in cui l'hook viene attivato, ciascuno dei moduli avrà possibilità di accedervi e manipolare i dati in base all'ordine stabilito [20] .
Le azioni che i moduli possono performare sul pacchetto una volta terminate le operazioni di modifica o visualizzazione dei dati in esso contenuti sono:

1. NF_ACCEPT: il pacchetto continua la traversata normalmente;
2. NF_DROP: il pacchetto viene scartato;
3. NF_STOLEN: il pacchetto viene prelevato dalla coda di trasmissione, interrompendone l'avanzata;
4. NF_QUEUE: il pacchetto viene accodato e reso accessibile allo userspace;
5. NF_REPEAT: il pacchetto continua la traversata, passando nuovamente per l'ultimo hook attivato.

Iptables

Iptables un sistema di selezione pacchetti realizzato per il framework netfilter. Questo tool è basato su una serie di tabelle, ciascuna contenente le cosiddette "catene" o *chains*.

Una catena è una lista di regole, formate da una o piú condizioni sul pacchetto e da un'azione da intraprendere nel caso le condizioni si verifichino. Grazie alle regole è quindi possibile specificare numerosi parametri con cui filtrare i pacchetti tra cui sorgente, destinazione, protocollo, interfaccia, e così via.

Le quattro tabelle standard di iptables sono *filter*, *NAT*, *mangle* e *raw*. *Filter* viene utilizzata per aggiungere e rimuovere regole dalla tabella di filtraggio del kernel, rappresentando un ottimo strumento per la realizzazione di firewall; *NAT* gestisce appunto il Network Address Translation; *mangle* é utilizzata per per la generica manipolazione dei pacchetti prima di instradarli nella rete; *raw* viene usata principalmente per marcare e tracciare determinati pacchetti [21].

Le *chains* presenti all'interno di iptables sono cinque, ma ogni tabella utilizza la propria versione: PREROUTING (prima della decisone del routing riguardo alla destinazione del pacchetto), INPUT (il pacchetto è destinato all'host locale), OUTPUT

(pacchetti in uscita dall'host locale), FORWARD (destinato ad una destinazione diversa da quella locale), POSTROUTING (pacchetti gestibili dopo la desisione del router).

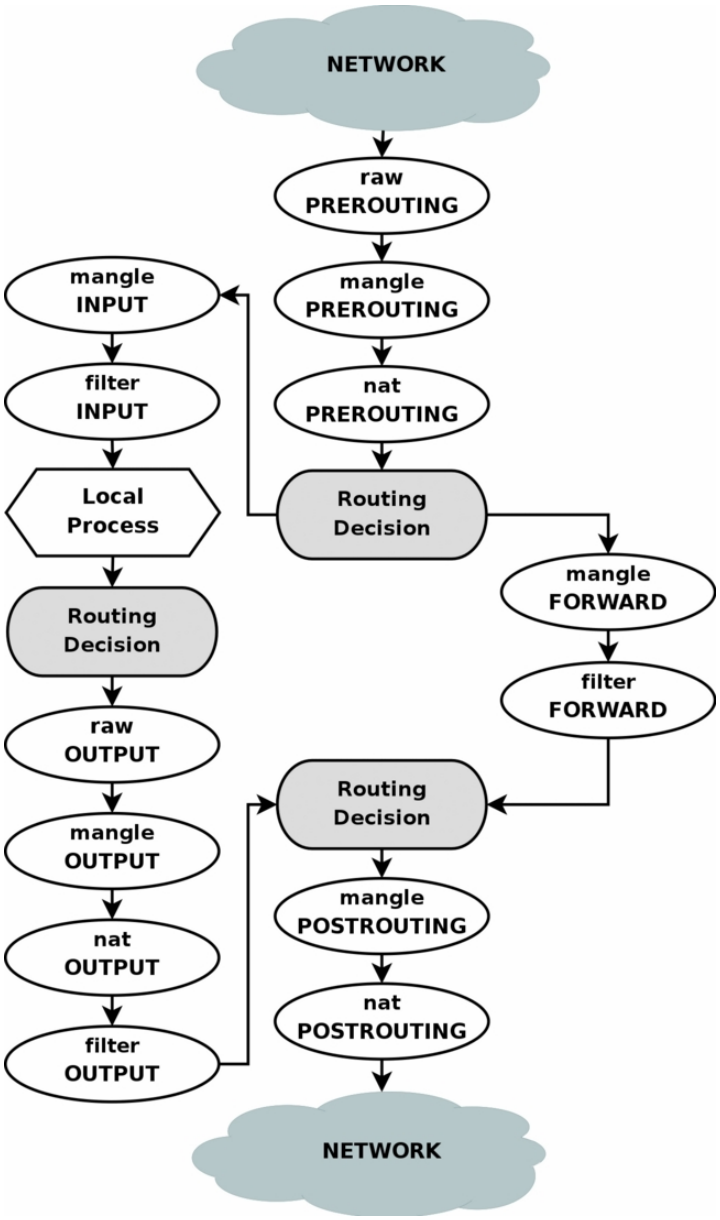

Figura 4.1: Struttura di regole e tabelle all'interno di iptables.

Gestione delle Code

È possibile, per un'applicazione, manifestare il proprio interesse a gestire i pacchetti accodati registrandosi all'hook NF_QUEUE.

Quando cioò avviene i pacchetti in transito vengono accumulati nella coda e viene permessa la gestione da parte dell'applicazione. Quest'ultima può effettuare le manipolazioni desiderate ma *deve* reimmettere il pacchetto nella coda di trasmissione del kernel assegnandogli uno dei verdetti visti in precedenza, altrimenti verrà scartato di default.

Frammentazione

Nel momento in cui un pacchetto è troppo grande per essere trasmesso, viene suddiviso in frammenti prima di essere spedito; il destinatario avrà il compito di riassemblare i frammenti ricostruendo il pacchetto.

Il problema di questo procedimento risiede nel fatto che solo il frammento iniziale contiene gli header completi di tutte le informazioni, mentre quelli successivi presentano solo un sottoinsieme delle intestazioni (IP senza i campi dei protocolli addizionali).
Netfilter, grazie alle funzionalità offerte da *nf_defrag_ipv4()* è in grado di ricomporre i frammenti prima che raggiungano il codice che si occupa di filtrarli. È inoltre possibile stabilire una regola per i frammenti successivi all'header utilizzando il flag **-f**.

4.3 Struttura dell'applicazione

L'intera applicazione è basata sulla libreria *libnetfilter_queue* offerta da netfilter, che contiene tutte le funzioni necessarie al recupero e alla manipolazione dei pacchetti IP in trasmissione; lo svantaggio principale dell'utilizzo di tale strumento è che le strutture dati sono estremamente specifiche per il problema trattato, e il recupero dei dati necessari alla ritrasmissione richiede un po' di manipolazione attenta dei puntatori.
Prima di cominciare è necessario definire la regola di iptables che permette lo storage dei pacchetti tcp in arrivo sulla rete wireless (in questo caso sull'interfaccia di bridging per AP `br0`) all'interno della coda Netfilter di manipolazione NF_QUEUE:

```
iptables -t mangle -A POSTROUTING -p tcp -o br0 -j NFQUEUE --queue-num 10.
```

A questo punto è possibile eseguire l'applicazione con `sudo ./preempt_retransmit 10` per avviare la ritrasmissione anticipata.

L'applicazione sostanzialmente è divisa in due rami (senza contare un terzo ramo costituito dal TED Simulator): il thread incaricato di prendere i pacchetti dalla coda NF_QUEUE, farne una copia e reinserirli nella coda di trasmissione del kernel, e il thread incaricato di restare in ascolto per segnalazioni provenienti dal TED Simulator ed eventualmente ritrasmettere i pacchetti persi. Essi sono rispettivamente `copy_and_send_pkt` e `listen_and_retransmit`.

È ora necessario illustrare le strutture dati utilizzate all'interno dell'applicazione, pri-

ma di mostrarne il funzionamento, tenendo nota che data la presenza i multithreading esse sono gestite tramite `mutex`:

id_to_retransmit
: Ovvero "lista dei pacchetti perduti". Tale struttura è una lista fornita di puntatori al primo e all'ultimo elemento, e contiene i dati necessari all'identificazione dei pacchetti IP che necessitano di ritrasmissione anticipata. Ogni pacchetto IP è identificato univocamente dalla coppia *id, checksum*, che vengono salvati dal TED Simulator una volta trasmesso il pacchetto. È necessario prestare molta attenzione in quanto tale scelta di identificazione é stata fatta a livello applicativo, nel momento in cui l'applicazione di sovrappone ad un effettivo TED si potrebbe dover adottare un altro metodo. Questo tipo di identificazione tramite coppia di elementi é stato adottato per evitare il rischio di ritrasmettere erroneamente il pacchetto sbagliato: il campo 'id' del datagram IP viene utilizzato per identificare univocamente il datagram ma nel caso di frammentazione tutti i frammenti presenteranno lo stesso id, in quanto viene usato per la ricomposizione del datagram originale. Dato che questo tipo di ricomposizione avviene nei terminali della connessione (e non all'interno dei router della rete) prima di passare il datagram a livello transport, si è pensato di affiancare a questo parametro anche il checksum del datagram per ottenere un identificatore univoco in tutto e per tutto.

pkt_node
: Ovvero "lista dei pacchetti copiati". Anch'essa è una lista fornita di puntatori al primo e all'ultimo elemento, e contiene le copie dei pacchetti trasmessi che hanno attraversato la coda NF_QUEUE per poi essere messi correttamente in trasmissione. Tale struttura servirà a recuperare e ritrasmettere i pacchetti IP persi e segnalati dal TED. Tale lista viene svuotata periodicamente dai pacchetti che stanziano per più di un determinato intervallo di tempo, oltre evitare lo spreco di risorse. Ispirandosi alla struttura dei socket tale tempo é stato settato equivalente a TIME_WAIT, ovvero `2*TimeToLive` (il tempo massimo di vita dei datagram IP).

Arriviamo dunque alla descrizione vera e propria del funzionamento dell'applicazione.

L'attività di `copy_and_send_pkt` è basata sulla funzione *nfqueue_loop()*, che grazie alle funzioni offerte di *libnetfilter_queue* si mette in ascolto sulla coda specificata come parametro in ingresso alla funzione (nel nostro caso 10) con una funzione bloccante di tipo *recv()*, e ogni volta che sulla coda NF_QUEUE viene registrato un nuovo pacchetto scatena la funzione di callback incaricata di gestirlo.
Tale funzione, denominata *nfqueue_cb()*, basa il suo funzionamento sulla struttura dell'-header IP definita come:

```
struct ipheader {
      unsigned char ip_hl:4, ip_v:4;
      unsigned char ip_tos;
```

```
unsigned short int ip_len;
unsigned short int ip_id;
unsigned short int ip_off;
unsigned char ip_ttl;
unsigned char ip_p;
unsigned short int ip_sum;
unsigned int ip_src;
unsigned int ip_dst; };
```

Il procedimento per recuperare i dati dall'header IP richiede l'utilizzo delle strutture dati offerte da netfilter, dalle quali si estraggono tutte le informazioni per poter creare un pacchetto IP idoneo alla ritrasmissione. L'header del pacchetto fornito dalla funzione nativa *nfq_get_msg_packet_hdr* tuttavia non ritorna l'header del pacchetto IP, ma l'header del pacchetto visto da Netfilter, che contiene i dati relativi alla coda.
Per ottenere un pacchetto IP è necessario operare nel seguente modo:

```
struct nfq_data *nfa;
.
.
nfq_get_payload (nfa, &buffer);
.
.
struct ipheader *iph = (struct ipheader *)buffer;
```

Una volta ottenuti i dati a cui siamo interessati possiamo copiarli nella lista di pacchetti vista in precedenza e reiniettare il pacchetto in analisi all'interno della coda di trasmissione del kernel, operazione che nel nostro caso viene effettuata da una funzione appartenente al TED Simulator, come vedremo in seguito. Tale funzione emetterà un verdetto sul pacchetto tramite la procedura offerta da netfilter *nfq_set_verdict()*. Quest'ultima procedura non processa il pacchetto direttamente, ma prende come parametro la coda NF_QUEUE che contiene i pacchetti attualmente processati e la posizione del pacchetto in tale lista, per assegnargli un verdetto che ne stabilirà la destinazione (es. NF_ACCEPT, NF_DROP) e lo farà continuare o meno nella traversata dello stack. Per tale motivo non è possibile emettere un verdetto più volte sullo stesso pacchetto: una volta che il pacchetto è uscito dalla coda, l'id dell'header di netfilter contenuto nella struttura `nfqnl_msg_packet_hdr` non è più uno strumento valido per la sua identificazione, in quanto il pacchetto non si trova più al suo interno. Dovrà essere quindi utilizzato un altro metodo, come vedremo in seguito.

Il centro della ritrasmissione è quindi incentrato sul thread `listen_and_retransmit`. Esso è costantemente in ascolto sul file descriptor di output di una `pipe` fornita dal `main`, che è la controparte del file descriptor su cui il TED Simulator notifica l'avvenuta perdita di un pacchetto. Una volta ricevuta la notifica della necessaria ritrasmissione

estrae il pacchetto dalla lista *pkt_node* e ne effettua la trasmissione tramite la funzione *send_missing_pkt()*.

```
struct id_to_retransmit* id_pkt_to_retransmit = (struct
    id_to_retransmit*)malloc(sizeof(struct id_to_retransmit));
struct pkt_node* lost_pkt;
/**Blocking read on the pipe fd*/
int nbytes = read(fd[0], id_pkt_to_retransmit, sizeof(struct
    id_to_retransmit));
/**Extract data needed to find the packet inside the pkt_node
    list*/
uint16_t id, checksum;
id=id_pkt_to_retransmit->id;
checksum=id_pkt_to_retransmit->checksum;
pthread_mutex_lock(&mutex_to_queue);
/**Find packet inside the list*/
lost_pkt = extract_to_retransmit(checksum, id);
pthread_mutex_unlock(&mutex_to_queue);
/**Call sender*/
send_missing_pkt(lost_pkt);
free(id_pkt_to_retransmit);
```

Infine, *send_missing_pkt()* è incaricata di utilizzare il socket speciale SOCK_RAW e il protocollo speciale IPPROTO_RAW per inviare il pacchetto. Questa particolare impostazione del socket permette di inviare datagram IP già completi di header senza che il kernel li impacchetti ulteriormente; è un passo fondamentale in quanto altrimenti la ritrasmissione non può avvenire, perchè il datagram verrebbe trattato come una stringa di dati da trasmettere e trattato di conseguenza mentre discende lo stack di livelli OSI. I dati per la trasmissione alla giusta destinazione vengono recuperati dall'elemento *pkt_node* prelevato in precedenza e il datagram 'raw' viene ritrasmesso:

```
sendto(sd_raw, lost_pkt->payload, lost_pkt->msglen, 0, (struct
    sockaddr *)&dest, sizeof(dest));
```

4.3.1 Problemi da risolvere

Il socket di tipo 'raw' effettua un nuovo invio del pacchetto, ma ciò implica anche che il pacchetto ritrasmesso verrà catturato nuovamente dalla nostra trasmissione, copiato e reso pronto nuovamente per la ritrasmmisione. Con un meccanismo di questo tipo su una connessione con problemi di congestione si rischia di ricadere in un ciclo di tentativi di invio.

La connessione TCP che parte dall'end-system, in un caso di simile ciclo, avrebbe comunque la possibilità di ricevere la segnalazione di errore allo scadere del RTT e tentare la ritrasmissione, e nel caso di continuo fallimento troncare la connessione.

È in ogni caso necessario trovare un modo per identificare i pacchetti già ritrasmessi ed evitare di reinserirli in coda, in modo che non abbiano in effetto negativo sulla trasmissione.

Capitolo 5

Testing

5.1 Simulazione del Trasmission Error Detector

La simulazione del Transmission Error Detector avviene all'interno dell'applicazione sottoforma di due funzioni principali: *fake_sender()*, e *wait_and_signal_loss_function()*.

La prima funzione è basata sulla macro `DROP_PKT` definita in cima all'applicazione, che rappresenta il rate con cui i pacchetti devono essere scartati per imporne la perdita. Questa funzione infatti è incaricata dall'applicazione di effettuare il reinvio dei pacchetti, ma si comporta in modo da scartare un pacchetto ogni volta che il rate di perdita di pacchetti stabilito viene raggiunto.
I pacchetti scartati vengono identificati tramite le informazioni *id* e *checksum* presenti all'interno dell'header IP e accodati in una lista che servirà al TED Simulator per notificarne la perdita. Le operazioni di scarto o corretto invio dei pacchetti vengono effettuati tramite la funzione nativa di netfilter *nfq_set_verdict()* dando come target NF_DROP o NF_ACCEPT.

La seconda funzione, *wait_and_signal_loss_function()*, è lo starter per il thread che rappresenta il TED Simulator vero e proprio. Esso si sveglia periodicamente (ogni 5 millisecondi) e controlla se la lista in cui la *fake_sender()* ha accodato i pacchetti persi c'è qualche elemento che necessita di una ritrasmissione. Se ciò avviene viene messa all'interno di una `pipe` appositamente creata una struttura dati che contiene i dati identificatori del pacchetto che è stato perso. Tale pipe è condivisa con il thread dell'applicazione incaricato della ritrasmissione, che leggerà poi il file descriptor e recupererà le informazioni per effettuare la ritrasmissione.

```
usleep(5000);
    if(id_head.next!=NULL){
    printf("TED Simulator: an element has been lost and needs
       to be retransmitted!\n");
    /**Estract packet to be retransmitted from the head of the
       list*/
    struct id_to_retransmit* id_pkt_to_retransmit;
    pthread_mutex_lock(&mutex_to_retransmit);
    id_pkt_to_retransmit = dequeue_id();
    pthread_mutex_unlock(&mutex_to_retransmit);
    /** Sendthe data structure containing packet identifiers
       through the output side of pipe */
    write(fd[1], id_pkt_to_retransmit, sizeof(struct
       id_to_retransmit));
    /**Delete sent packet, we do not need it anymore*/
    free(id_pkt_to_retransmit);
  }
```

È possibile vedere nell'immagine che segue che, mentre il thread di filtraggio e gestione della coda continua ad eseguire ignaro di errori di trasmissione, la *fake_sender()* scarta appositamente un pacchetto. Dopo poco tempo il TED Simulator registra la perdita e fa partire il meccanismo di ritrasmissione.

```
Filtering handler: reading packets from NF_QUEUE
------- received 152 bytes ----------
hw_protocol=0x0800 hook=1 id=6
Copying received packet...
Fake Sender: I am dropping id=49002, check=32069

Filtering handler: reading packets from NF_QUEUE
TED Simulator: an element has been lost and needs to be retransmitted!
------- received 152 bytes ----------
hw_protocol=0x0800 hook=1 id=7
Retransmit pkt_id=49002 and pkt_chksm=32069 : Success
Copying received packet...
```

Figura 5.1: Rappresentazione dell'esecuzione dell'applicazione.

5.2 I Test

Per effettuare i test sono state analizzate le capacità della rete sottoposta ad una perdita forzata di un determinato numero di pacchetti e il suo comportamento una volta introdotta la ritrasmissione anticipata realizzata dall'applicazione.

I test sono stati svolti in ambiente wireless, coinvolgendo il nodo mobile e l'AP appositamente creato.
Una volta connesso il nodo mobile alla rete wireless tramite access point, sono stati recuperati dati di dimensione considerevole (abbastanza grandi da poter registrare un buon cambiamento di prestazioni) dalla rete tramite il comando `wget`.
È stato quindi osservato il tempo necessario al trasferimento dei dati e la velocità di tale trasferimento per determinare la rilevanza della ritrasmissione anticipata offerta dall'applicazione.
Sono stati effettuati tre test consecutivi, ognuno eseguito imponendo una perdita di pacchetti sempre maggiore e trasferendo il file desiderato sfruttando prima la ritrasmissione offerta di TCP, poi la ritrasmissione anticipata realizzata localmente.
Il comando `wget` è stato eseguito per ottenere un file di dimensione 23552 KB, in seguito mostriamo i risultati.

Introducendo un *packet loss* del 1% la media del tempo di trasmissione calcolata su tre tentativi è stata di 2 minuti, con velocità di trasferimento a 195 KB/s; introducendo la ritrasmissione anticipata il tempo medio di trasferimento cala seppur minimamente a 1 minuto e 48 secondi, e la velocità di trasferimento sale a 219 KB/s.

Introducendo un *packet loss* del 5% la media del tempo di trasmissione calcolata su tre tentativi è stata di 5 minuti e 6 secondi, con velocità di trasferimento a 76.9 KB/s; introducendo la ritrasmissione anticipata il tempo medio di trasferimento cala a 4 minuti e 11 secondi, fornendo un buon risultato per quanto riguarda l'apprezzabilità della modifica introdotta, e la velocità di trasferimento sale a 93.6 KB/s.

Introducendo un *packet loss* del 10% la media del tempo di trasmissione calcolata su tre tentativi è stata di 6 minuti e 25 secondi, con velocità di trasferimento a 61.2 KB/s; introducendo la ritrasmissione anticipata il tempo medio di trasferimento cala a 6 minuti e 4 secondi e la velocità di trasferimento sale a 64 KB/s.

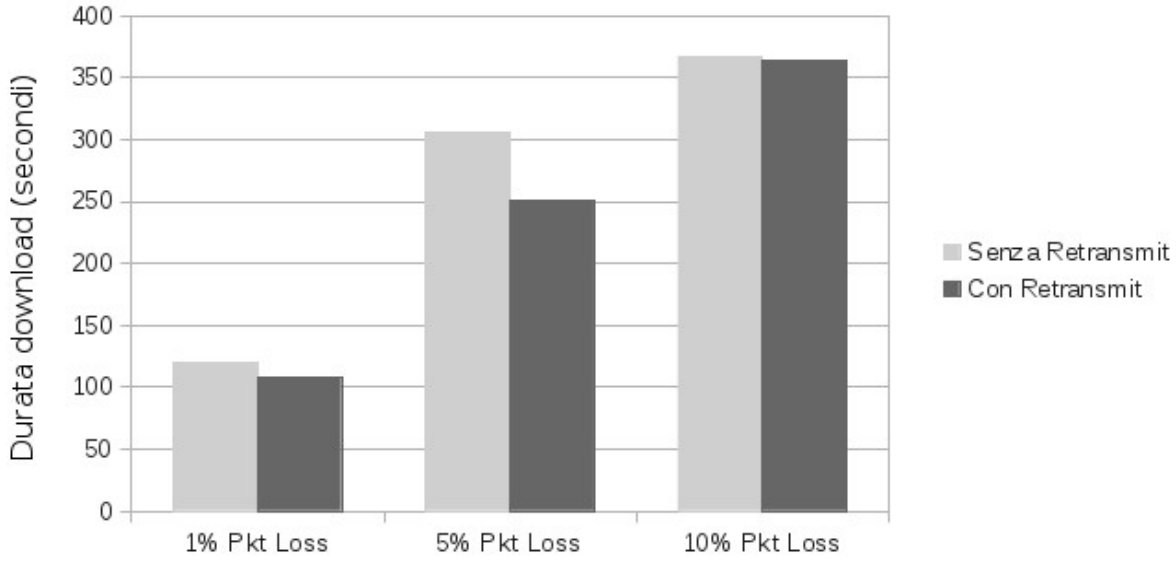

Figura 5.2: Raffigurazione in tempo dei tre test: la colonna a destra rappresenta la ritrasmissione anticipata, ed è possibile vedere il vantaggio apportato.

È possibile notare che l'applicazione realizzata porta quindi un miglioramento della trasmissione, tuttavia nel momento in cui la perdita di pacchetti si fa elevata l'*overhead* dalla gestione delle strutture dati all'interno dell'applicazione causa un rallentamento delle operazioni rendendo il risultato molto simile alla trasmissione originale.
L'idea è che si possa migliorare tale *overhead* con una gestione affinata delle strutture di liste e code all'interno dell'appliczione, ad esempio con l'impiego di ABR, per ridurre i costi di accesso al minimo, oppure migliorare la gestione dei timer interni con un calcolo puntuale in modo da non mettere il programma in attesa per tempi che risultano influire sulla trasmissione.

In conclusione i risultati dei test sono soddisfacenti, in quanto per una perdita di pacchetti non eccessiva l'applicazione riesce ad introdurre un miglioramento sensibile alla trasmissione, verificando l'ipotesi iniziale ovvero che sia possibile controllare la qualità della trasmissione verso un nodo mobile tramite Access Point Wi-Fi.

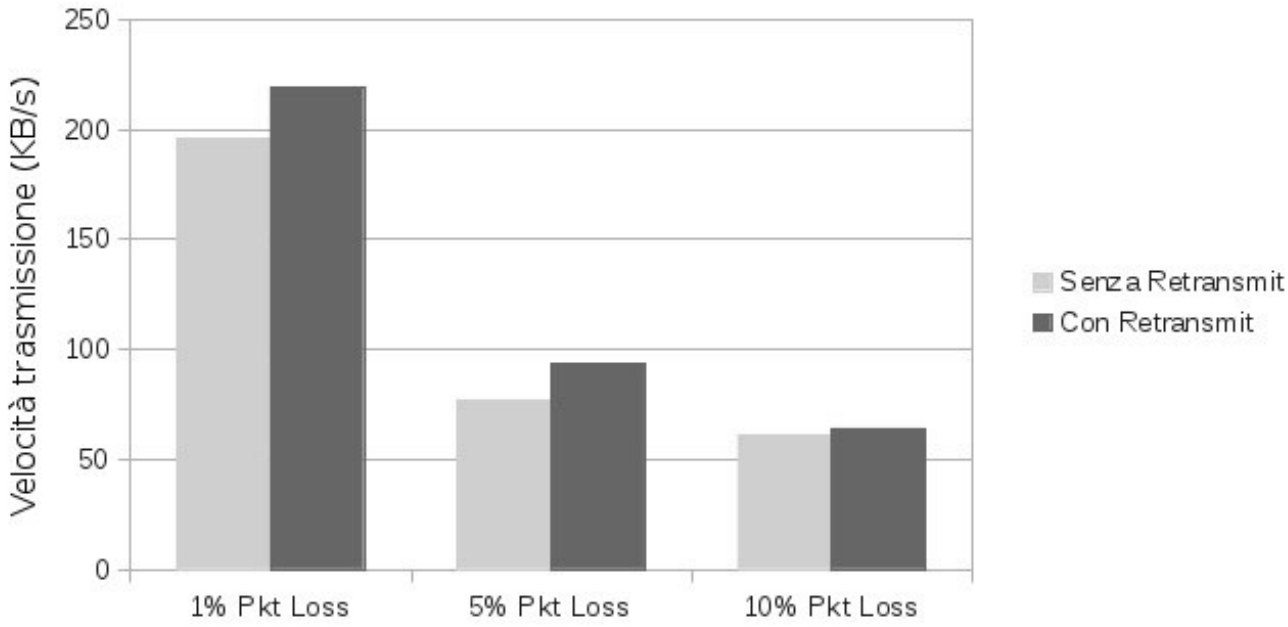

Figura 5.3: Raffigurazione in velocità dei tre test: la colonna a destra rappresenta la ritrasmissione anticipata, ed è possibile vedere come aumenta in rapporto al trasfermiento normale, dando risultati soddisfacenti.

5.3 II Test

Per effettuare il secondo test sono state ripetute le stesse operazioni del test precedente, introducendo un delay nella trasmissione prima di 50ms e poi di 200ms.
Tale delay è stato ottenuto utilizzando il comando `tc` che permette di aggiungere un ritardo ai pacchetti uscenti da una specifica interfaccia di rete:

```
tc qdisc add dev wlan0 root netem delay 100ms 10ms 25%
```

I risultati di tali test sono stati positivi: più aumenta il delay del pacchetto in uscita più è facile vedere un distacco tra la performance della trasmissione con ritrasmissione standard e quella con ritrasmissione anticipata. Nel caso del delay a 200ms si arriva anche ad ottenere un miglioramento di più del 50%. Dove nel test iniziale l'overhead della computazione al 10% di packet loss compensava i risultati della ritrasmissione anticipata provocando un miglioramento bassissimo de non quasi assente, aumentando il delay di trasmissione ciò non avviene, e si è in grado quindi di registrare un netto miglioramento

di performance. Dai grafici che seguono è facile vedere come i tempi impiegati per la trasmissione del file siano nettamente inferiori nel caso con ritrasmissione anticipata.

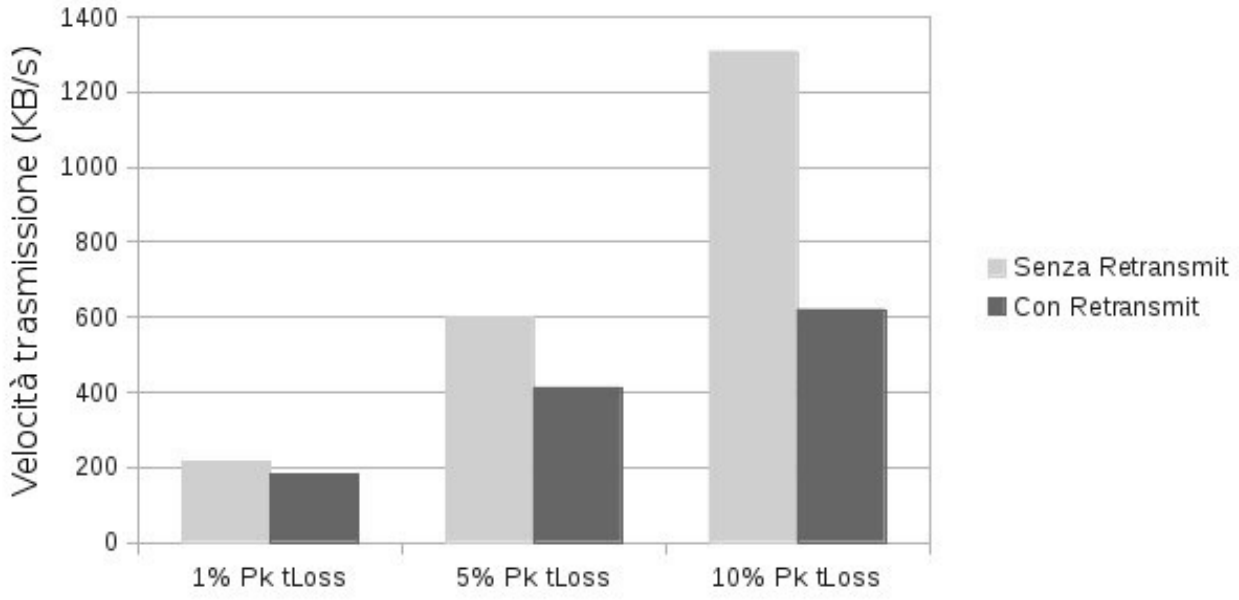

Figura 5.4: Trasmissione con 50ms di delay

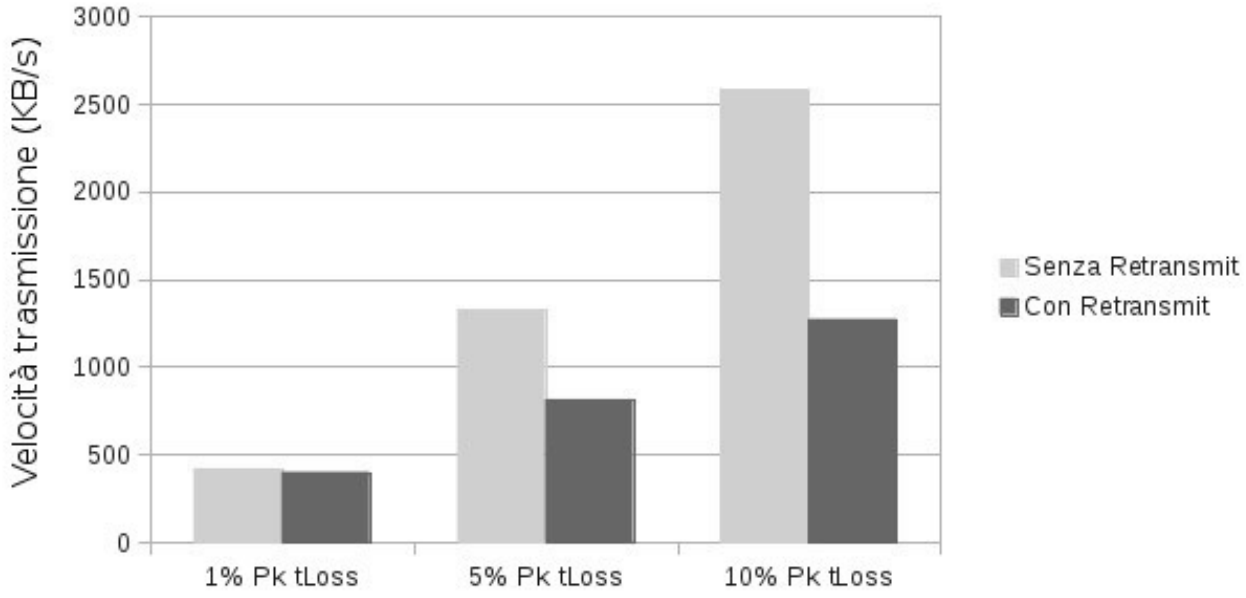

Figura 5.5: Trasmissione con 200ms di delay

Capitolo 6

Sviluppi futuri

L'applicazione realizzata per questo progetto non è altro che un primo embrione, un *proof of concept* che lascia grande spazio a miglioramenti e ampliamenti di ogni sorta.

TED lato Access Point
Il passo successivo più naturale e forse anche più interessante è la realizzazione della seconda parte di questo progetto: il porting del TED a lato Access Point. Questo permetterebbe in un futuro di staccare l'applicazione dal simulatore che l'accompagna per poter essere testata in ambiente reale in modo da poterne valutare i risultati e i vantaggi (o svantaggi) in modo concreto.

Miglioramento dell'applicazione incaricata della ritrasmissione
Un'altra linea di azione può essere quella di ottimizzare l'applicazione creata. Si può partire dalla sostituzione della struttura dati utilizzata per salvare i pacchetti copiati (attualmente una lista) con un ABR, in modo da ridurre ulteriormente il costo delle operazioni di ricerca. Un'altra modifica interessante può essere l'affinamento del meccanismo di timer con cui i paccchetti vengono eliminati dalla lista di ritrasmissione: mentre ora si basa su un timer indicativo utilizzato per tutti i pacchetti, sarebbe possibile tentare di approssimare il RTT realtivo ad ogni pacchetto per vedere se conviene un tipo di eliminazione singola ma estremamente accurata in modo da minimizzare il rischio di errori.

Miglioramento del simulatore di TED
Utilizzando un'estensiva tecnica di packet sniffing è possibile ricondurre il comportamento del simulatore di TED più vicino possibile al comportamento della macchina reale. Calcolando un RTT approssimativo e mettendosi in ascolto dei segmenti di ACK che tornano dalla sorgente è possibile individuare determinati tipi di errore e notificare l'applicazione in modo tempestivo.

Appendice A

Contenuto dei file di configurazione

Mostriamo di seguito il contenuto dei files `/etc/hostapd/hostapd.conf` e `/etc/network/interfaces`.
Il file `hostapd.conf` viene compilato per permettere ad hostapd di configurare correttamente l'Access Point Wireless. Eventualmente è possibile anche configurare una modalità di sicurezza, facendo ad esempio uso di WPA.

```
# Wireless network name
interface=wlan0
#Bridge name
bridge=br0
# Driver name
driver=nl80211
# Country code
country_code=IT
#SSID of the network
ssid=cdebianet
# Operation mode
hw_mode=g
# Set channel
channel=1
# Wpa mode
wpa=2
wpa_passphrase=mypassphrase
# Key management algorithms
wpa_key_mgmt=WPA-PSK
#Set cipher suites (encryption algorithms)
# TKIP = Temporal Key Integrity Protocol
# CCMP = AES in Counter mode with CBC-MAC
```

```
wpa_pairwise=TKIP
rsn_pairwise=CCMP
# Shared Key Authentication
auth_algs=1
# Accept all MAC address
macaddr_acl=0
```

Di seguito si mostra inoltre come è stata configurata l'interfaccia br0 nel file `/etc/network/interfaces`.

```
iface br0 inet static
bridge_ports eth0 wlan0
address 192.168.1.105
netmask 255.255.255.0
network 192.168.1.0
broadcast 192.168.1.255
gateway 192.168.1.1
dns-nameservers 192.168.1.1
dns-search localdomain
```

Bibliografia

[1] Jon Postel, *Comments on Internet Protocol and TCP*. Internet Engineering Note number 2, `http://www.rfc-editor.org/ien/ien2.txt`, 1977.

[2] Charles M. Kozierok, *The TCP/IP Guide, a comprehensive, illustrated Internet protocols reference*. No Starch Press, San Francisco, 2005.

[3] Craig Hunt, *TCP/IP Network Administration*. O'Reilly, 3rd Edition, 2002.

[4] Fred Halsall, *Computer Networking and the Internet*. Addison Wesley, 5th Edition, 2005.

[5] Information Sciences Institute of the University of Southern California, *Internet Protocol Specification*. RFC: 791, `http://www.ietf.org/rfc/rfc0791.txt`, 1981.

[6] Andrew S. Tanenbaum e David J. Wetherall, *Computer Networks*. Prentice Hall, 5th Edition, 2011.

[7] Christian Benvenuti, *Understanding Linux Network Internals*. O'Reilly, 2006.

[8] Jon Postel, *Internet Control Message Protocol*. RFC: 792, `http://www.ietf.org/rfc/rfc0792.txt`, 1981.

[9] Douglas E. Comer, *Internetworking with TCP/IP, Volume I*. Prentice Hall, 4th Edition, 2000.

[10] L. Peterson e B. Davie, *Computer Networks, a Systems Approach*. Morgan Kaufmann Publishers, 3rd Edition, 2003.

[11] Walter Goralski, *The Illustrated Network, How TCP/IP Works in a Modern Network*. Morgan Kaufmann Publishers, 2009.

[12] V. Paxson, *Computing TCP's Retransmission Timer*. RFC: 2988, `http://www.hjp.at/doc/rfc/rfc2988.txt`, 2000.

[13] Kevin R. Fall e Richard Stevens, *TCP/IP Illustrated, Volume 1: The Protocols*. Addison Wesley, 2011.

[14] Matthew Gast, *802.11 Wireless Networks: The Definitive Guide*. O'Reilly, 2002.

[15] S. Ferretti, V. Ghini, M. Marzolla, F. Panzieri, *Modeling the Always Best Packet Switching Mechanism*. Dept. of Computer Science, University of Bologna, Italy.

[16] S. Ferretti, V. Ghini, M. Marzolla, F. Panzieri, *Always Best Packet Switching: the Mobile VoIP Case Study*. Dept. of Computer Science, University of Bologna, Italy.

[17] M.Pedrini, *TCP a Ritrasmissione Asimmetrica Anticipata su WiFi*. Dept. of Computer Science, University of Bologna, Italy.

[18] Jouni Malinen, *Hostapd: IEEE 802.11 AP, IEEE 802.1X/WPA/WPA2/EAP/RADIUS Authenticator*. `http://w1.fi/hostapd/`, 2002-2013.

[19] *About OpenWRT*. `http://wiki.openwrt.org/about/start`.

[20] Rusty Russell e Harald Welte, *Netfilter Hacking HOWTO*. `http://www.netfilter.org/documentation/HOWTO/it/netfilter-hacking-HOWTO.html`, 2004.

[21] Rusty Russell, *Packet Filtering HOWTO*. `http://www.netfilter.org/documentation/HOWTO/it/packet-filtering-HOWTO.html`, 2000.