Alma Mater Studiorum · Università di Bologna

SCUOLA DI SCIENZE
Corso di Laurea Magistrale in Informatica

ANALISI PER VIA SIMULATIVA DEL PROTOCOLLO TCP A RISTRASMISSIONE ASIMMETRICA ANTICIPATA SU WIFI

Tesi di Laurea in Sistemi Mobili

Relatore:
Chiar.mo Prof.
Vittorio Ghini

Presentata da:
Paolo Alberti

Sessione II
Anno Accademico 2012/2013

Indice

Elenco delle figure

Elenco delle tabelle

Introduzione

Nel corso dell'ultimo secolo, i progressi nelle tecnologie hanno portato a una profonda e radicale modifica sul modo di comunicare; l'impatto delle comunicazioni wireless è stato e continuerà a essere profondo, la rivoluzione tecnologica è evidente nella crescita del mercato della telefonia mobile. Nel 1990, il numero di utenti nel mercato cellulare era intorno agli 11 milioni; oggi si parla di bilioni [21]; ci sono diversi fattori che hanno portato a questo boom: il cellulare è comodo, conveniente, lo puoi portare sempre con te, è in grado di comunicare con qualsiasi stazione fissa. La tecnologia ha contribuito ad aumentare questo successo: oramai questi dispositivi mobili sono piccoli e maneggevoli, la batteria dura di più, le prestazioni sono aumentate, il wireless permette di essere sempre connessi a Internet, mandare messaggi istantanei, leggere e-mail e usufruire di tutti quei servizi che oggi Internet mette a disposizione. A questo, d'altro canto, va aggiunto il fatto che la maggior parte degli accessi alla rete ora parte perlopiù da dispositivi con interfaccia wireless, e la richiesta di prestazioni sempre migliori è in continuo aumento; inoltre, le reti wireless sono più soggette a interferenze rispetto alle classiche reti cablate.
Il seguente elaborato vuole mostrare una possibile ottimizzazione della trasmissione dati su rete wireless agendo sul protocollo TCP. In particolare, fino a questo momento è stato applicato il sistema ABPS solo per la comunicazione VoIP su Wifi (che sfrutta come protocollo di trasporto quello UDP); applicando queste tecniche, che agiscono sul primo hop di comunicazione tra due end system, a un protocollo come il TCP, sarà possibile ottimizzare il processo di ritrasmissione di un pacchetto perso.
Le modifiche apportate non vengono fatte su un sistema reale, bensì viene utilizzato il simulatore Omnet++ e il framework INET che ci forniscono tutte le caratteristiche di un ambiente simulato, simile alla realtà. L'elaborato è diviso in capitoli:

- nel primo è necessario dare un'idea del contesto in cui ABPS agisce come fattore

fondamentale per quanto riguarda lo sviluppo della tematica nell'ambito complessivo di questo lavoro. Si ragionerà di ABPS tenendo presente che l'architettura si basa su tecnologie e protocolli che nel seguito saranno analizzati;

- nel secondo viene presentato lo scenario simulativo e come l'architettura ABPS e RWMA sono stati implementati fino a questo momento, considerando l'implementazione fatta in Omnet++/INET sul protocollo UDP;

- nel terzo capitolo viene presentato il protocollo TCP in generale e come è stato implementato all'interno del simulatore;

- il quarto capitolo è quello fondamentale, in cui verranno spiegati con dovizia di particolari tutte le modifiche apportate per la realizzazione e la gestione del nuovo protocollo di ritrasmissione dei pacchetti, chiamato TCP RenoRWMA;

- nel quinto capitolo sono stati raccolti brevi test utili per dimostrare l'effettivo funzionamento della modifica e per un'analisi comparativa della stessa con la versione base del TCP;

- infine, il sesto e ultimo capitolo, è quello dedicato alle conclusioni e ai vari spunti per ulteriori modifiche al sistema implementato.

Capitolo 1

ABPS

Prima di concentrarci sull'obiettivo principale di questa tesi, è necessario introdurre e soffermarsi sull'architettura ABPS, base principale da cui ne scaturisce tutta la discussione successiva; in particolare, nel capitolo ci soffermeremo sull'introduzione di questo modello, il fine per cui è stato introdotto, lo stato dell'arte, le tecnologie e i protocolli usati.

1.1 Scenario

Ad oggi, le tecnologie introdotte non consentono a un dispositivo dotato di più interfacce di rete eterogenee (NICs, Network Interface Cards) di poterle sfruttare contemporaneamente, bensì di sceglierne una e di utilizzare quella scelta per l'invio di dati. L'architettura ABPS (Always Best Packet Switching) [17,18] viene introdotta per poter sfruttare al massimo le potenzialità del dispositivo in modo da utilizzare nello stesso tempo tutte le interfacce di rete di cui è dotato, instradando il pacchetto da inviare su quella che, al momento dell'invio, si reputa essere l'interfaccia di rete migliore; in questo modo, ABPS permette alle applicazioni mobili di creare politiche di bilanciamento di carico (load balancing) e metodi di recupero della comunicazione per supportare al massimo i servizi multimediali (come ad esempio il VOIP), garantendo la massima interattività richiesta e un basso valore di pacchetti persi. Prendendo in considerazione un'applicazione VOIP, questa sarà in grado di gestire più interfacce, scegliendo ogni volta quella più adatta all'invio del pacchetto corrente ed ognuno di questi viene monitorato in modo da ot-

tenere informazioni sulla loro effettiva ricezione da parte dell'access point, al fine di una possibile ritrasmissione.

1.2 Architettura ABPS

Gli obiettivi principali per cui è stata introdotta l'architettura ABPS sono:

- limitare le emissioni elettromagnetiche dei dispositivi mobili, riducendo rischi dovuti a esposizioni prolungate; questo viene ottenuto scegliendo a runtime tecnologie di comunicazione a medio raggio (ovvero IEEE 802.11 b/g/n), al fine di trasmettere la stessa quantità di dati ma irradiando energia fino a due ordini di grandezza inferiore di una tecnologia ad alto raggio (UMTS, WiMAX);

- ottenere una maggior banda, favorendo la comunicazione con un access point a media-corta distanza e usando contemporaneamente tutte le NIC del dispositivo, per un trasferimento in parallelo dei dati;

- provvedere a una sufficiente interattività della comunicazione, implementando un protocollo cross-layer che permette di controllare la perdita dei pacchetti e la ritrasmissione degli stessi su una tratta alternativa;

- utilizzare le esistenti infrastrutture di rete senza introdurre cambiamenti alle stesse, implementando i protocolli necessari solamente sui nodi;

- riuscire ad aggirare le limitazioni imposte dalla crescente presenza di NAT e firewall, utilizzando risorse esterne e protocolli specifici. In situazioni di host coperti da NAT simmetrico, è necessaria la presenza di un server d'appoggio pubblicamente raggiungibile.

Per ottenere tali propositi, l'architettura è stata realizzata come nella figura (Figura 1.1)

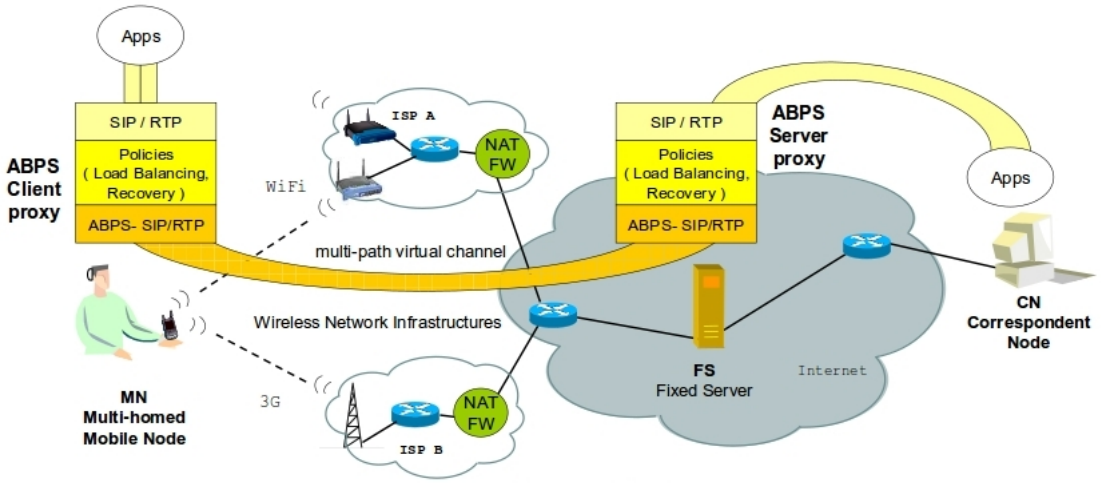

Figura 1.1: Architettura ABPS

Il sistema prevede l'uso di un server proxy esterno alle wireless access networks e non mascherato da un firewall; ogni nodo mobile (MN) possiede un local communication manager (client proxy) che ha il compito di mantenere il collegamento, attraverso le diverse NICs, con un server proxy sul Fixed Server (FS); il proxy server, quindi, è la rappresentazione del mobile node verso il mondo esterno: è incaricato di mantenere la comunicazione sempre attiva con il MN, senza la necessità di utilizzare il Mobile IP, integrando gli opportuni accorgimenti per aggirare il problema della presenza dei firewall.
Le applicazioni sul nodo mobile utilizzeranno un multi-path virtual channel tra client proxy e server proxy per comunicare con il resto del mondo; attraverso questa modalità, il nodo mobile utilizza le infrastrutture esistenti, senza necessità di modifiche su di queste, e implementando i protocolli necessari solo nei due proxy.

Associato all'architettura ABPS viene introdotto il Robust Wireless Medium Access (RWMA), meccanismo utilizzato per garantire la Quality of Service (QoS) del collegamento (soprattutto per un'applicazione VOIP) e un basso valore di pacchetti persi.

1.2.1 RWMA

RWMA è un meccanismo cross-layer applicato ad ABPS, il cui compito principale è quello di raccogliere informazioni da diversi layer dello stack ISO/OSI per monitorare a run time la qualità della trasmissione. Questo meccanisco viene utilizzato principalmente per applicazioni VOIP che, solitamente, impiegano come protocollo di comunicazione il protocollo UDP, in cui la QoS e la perdita dei pacchetti sono paramentri essenziali (si richiede che il ritardo di trasmissione, in un'applicazione VOIP, deve essere inferiore ai 150ms mentre il numero di pacchetti non deve superare il 3%); tuttavia, si pensa di poter applicare RWMA anche per applicazioni che impiegano il protocollo TCP, come ad esempio un'applicazione di streaming audio.

In particolare, le principali differenze tra RWMA e gli altri approcci esistenti sono riassumibili in questi due punti:

1. il meccanismo di monitoraggio è in grado di stabilire se il pacchetto UDP (o TCP) viene perduto durante il "percorso" tra la NIC scelta per l'invio e l'access point a cui il nodo mobile è connesso e avverte il livello applicazione se il pacchetto è stato perduto oppure no;

2. l'applicazione può decidere se ritrasmettere o meno il pacchetto, stabilendo anche la nuova scheda di rete da cui ritrasmetterlo.

Le cause, per cui i requisiti in una applicazione VOIP o in uno streaming audio non vengono rispettati, possono essere molteplici: dalla congestione della rete al crash dei dispositivi di rete utilizzati; a questi fattori vanno aggiunte le problematiche legate al mondo wireless, come ad esempio multi-path, fading, rifrazione, attenuazione, rumore, etc.
Lo scenario si complica se si considera che il nodo può essere mobile; in questo caso, alle difficoltà viste in precedenza, vanno aggiunte quelle dovute al fatto che il nodo, muovendosi, può perdere il collegamento con l'access point a cui era connesso o nella migliore delle ipotesi il collegamento può essere disturbato. Queste complicazioni sono la causa maggiore di introduzione di errori a livello di bit, trasformando un bit da 1 a 0 o viceversa, o addirittura di perdite consecutive di pacchetti che portano a una degradazione della comunicazione.

Gli errori sono gestiti in maniere differenti e si distinguono, perciò, tre principali tipi di notifiche:

1. notifiche generate dal livello network: un buon numero di errori di comunicazione è notificato al mittente attraverso messaggi ICMP. Ad esempio, un router è in grado di accorgersi se il destinatario non è più raggiungibile e in tal caso, notifica il mittente attraverso un messaggio ICMP di tipo 0 (network unreachable error) o di tipo 1 (host unreachable error); tuttavia, non si entrerà in dettaglio in questo tipo di errore;

2. notifiche generate dal livello applicazione di un end system: un router, presente nel percorso tra mittente e destinatario, non è in grado di notificare alcuni tipi di errori che si verificano durante la comunicazione. Ad esempio, se un router elimina un pacchetto in quanto la rete è congestionata, ciò non viene comunicato dal router al mittente; questa situzione viene risolta attraverso degli algoritmi di livello applicazione nei due end-system: il pacchetto perso viene identificato attraverso un identificatore che il mittente assegna al pacchetto in fase di invio, in combinazione con lo scambio di pacchetti ACK o NACK nel caso in cui il pacchetto venga rispettivamente consegnato a destinazione oppure venga scartato (o perso).

3. notifiche generate dal livello MAC wireless: nello standard IEEE 802.11 \b\g\n, il ricevente invia un acknowledgment (ack) al mittente per comunicare che il pacchetto è stato ricevuto correttamente dall'access point a cui è connesso; quando il pacchetto o l'ack viene perso, il livello MAC del mittente ritrasmette il pacchetto fino al numero massimo di ritrasmissioni possibili, ad esempio sette; all'ottavo tentativo, nel caso in cui il mittente non riceva ack dall'AP verso cui abbiamo trasmesso il pacchetto, il livello MAC del mittente lo elimina "silenziosamente", senza generare alcuna notifica. Il frame perso non raggiungerà mai il destinario e solo in questo momento i due end system si accorgeranno della perdita, che verrà gestita come detto nel punto 2.
   Si è deciso, quindi, di fare in modo che il livello MAC del mittente, all'ottavo tentativo, non elimini semplicemente il pacchetto, ma crei una notifica che comunica l'eliminazione del pacchetto anche ai livelli superiori a quello MAC.

RWMA è il sistema middleware, posto in entrambi gli end system, che si occupa di gestire proprio quest'ultimo tipo di errori; è costituito di tre parti principali:

- Trasmission Error Detection: è il componente principale del sistema. Si occupa di monitorare ogni singolo datagramma UDP inviato attraverso il collegamento wireless, individuando se il pacchetto è stato ricevuto dall'access point oppure no; una volta stabilito questo, TED notifica a ULB (UDP Load Balancer) il mancato trasferimento del pacchetto o l'avvenuta consegna, attraverso una notifica IP_NOTIFY_FAILURE o IP_NOTIFY_SUCCESS;

- UDP Load Balancer (ULB): questo componente riceve le notifiche da parte del TED e decide se il pacchetto UDP di cui ha avuto notifica deve essere ritrasmesso oppure no; inoltre, in caso di ritrasmissione, stabilisce quale interfaccia wireless di rete utilizzare;

- Monitor: questo componente si occupa di controllare e configurare le interfacce wireless del nodo mobile, comunicando a ULB quali sono attive in ogni momento. Più precisamente, il monitor opera come un'applicazione separata che configura dinamicamente le interfacce di rete wireless e le regole di routing; quando una interfaccia di rete viene configurata, e quindi può essere utilizzata per l'invio e la ricezione di pacchetti, oppure diventa inattiva, il monitor comunica all'UDP Load Balancer le nuove configurazioni, attraverso una Reconfiguration Notification.

In particolare, il TED lavora a livello MAC del protocollo IEEE 802.11 \b\g\n ed ha come compito principale quello di monitorare ogni datagram UDP (vale anche per TCP) inviato dall'end system mittente all'end system ricevente sopra un collegamento wireless configurato e attivo, al fine di stabilire se il pacchetto inviato ha raggiunto il primo AP oppure è stato scartato dal livello MAC. TED notifica a UDP Load Balancer attraverso la First-Hop Notification (Figura 2.2) lo stato della trasmissione.
UDP Load Balancer, invece, riceve un pacchetto RTP dall'applicazione VOIP, lo incapsula in un datagram UDP e lo invia all'end system ricevente; nel momento in cui viene configurata una nuova interfaccia di rete, il Monitor informa l'ULB di quest'ultima attraverso una Reconfiguration Notification. A questo punto, come previsto anche dal protocollo UDP, per ogni interfaccia di rete configurata, ULB mantiene un socket UDP

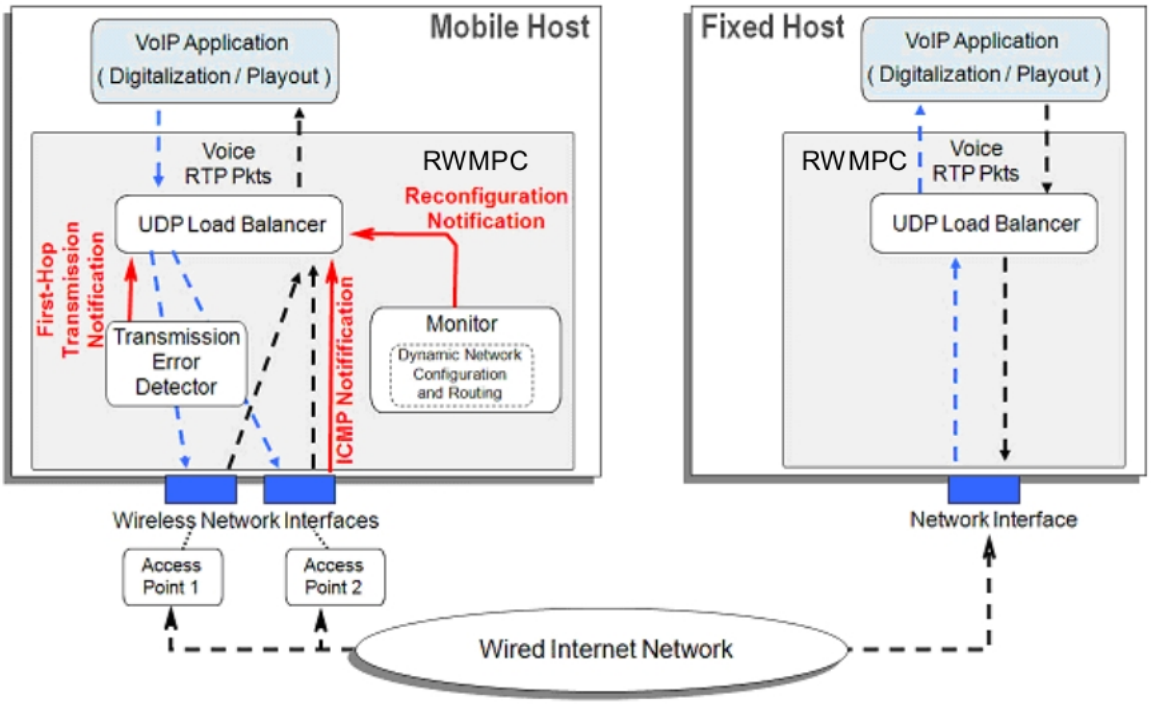

Figura 1.2: RWMA

associato alla scheda di rete attraverso la chiamata bind. In questo modo, ULB può utilizzare le interfacce per la comunicazione dei pacchetti; durante la comunicazione, ULB, inoltre, riceve altri due tipi di messaggi: i messaggi ICMP e, come detto in precedenza, i messaggi provenienti dal TED.

Capitolo 2

Prerequisiti di progettazione

In questo capitolo, verrà analizzata l'implementazione di RWMA realizzata fino a questo momento, considerando solo quella fatta nel simulatore Omnet++ e non tenendo presente quella realizzata in un kernel Linux version 2.6.27.4, in quanto le due risultano essere simili nella loro progettazione logica.

In particolare, si utilizza il simulatore Omnet++ [22] *nella versione 4.1 che fornisce l'ambiente di simulazione e INET* [23]*, framework contenente diversi protocolli wireless e wired, inclusi UDP, TCP, SCTP, IP, IPv6, Ethernet, PPP, 802.11, MPLS, OSPF e molti altri; la trattazione di Omnet++ e di Inet presi singolarmente viene lasciata al lettore, consigliando di dare un'occhiata ai manuali tecnici di entrambi.*

2.1 Lo scenario

Lo scenario simulativo adottato per lo sviluppo della tesi prevede una nodo mobile wireless dotato di più interfacce di reti, che si sposta tra le vie di un centro cittadino in presenza di vari AP e si consideri, inoltre, che il nodo mobile comunica attraverso il protocollo TCP con un server proxy oppure che stia eseguendo un'applicazione VoIP su UDP.

Come si può vedere nella Figura 2.1 e nella Figura 2.2 mobile node e server proxy si trovano in due reti differenti, collegate da due canali di comunicazioni wired; entrambe le reti, sono formate da diverse sottoreti, costituite dai classici network device, quali router, wireless access point, server dhcp, nodi fissi e nodi wireless.

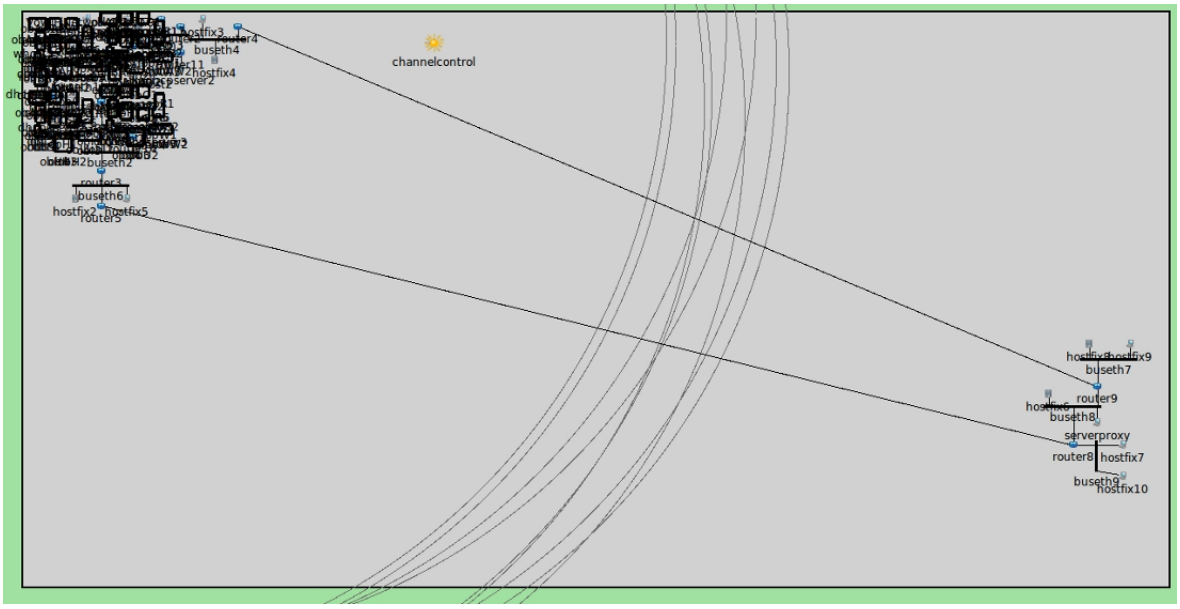

Figura 2.1: Scenario simulativo

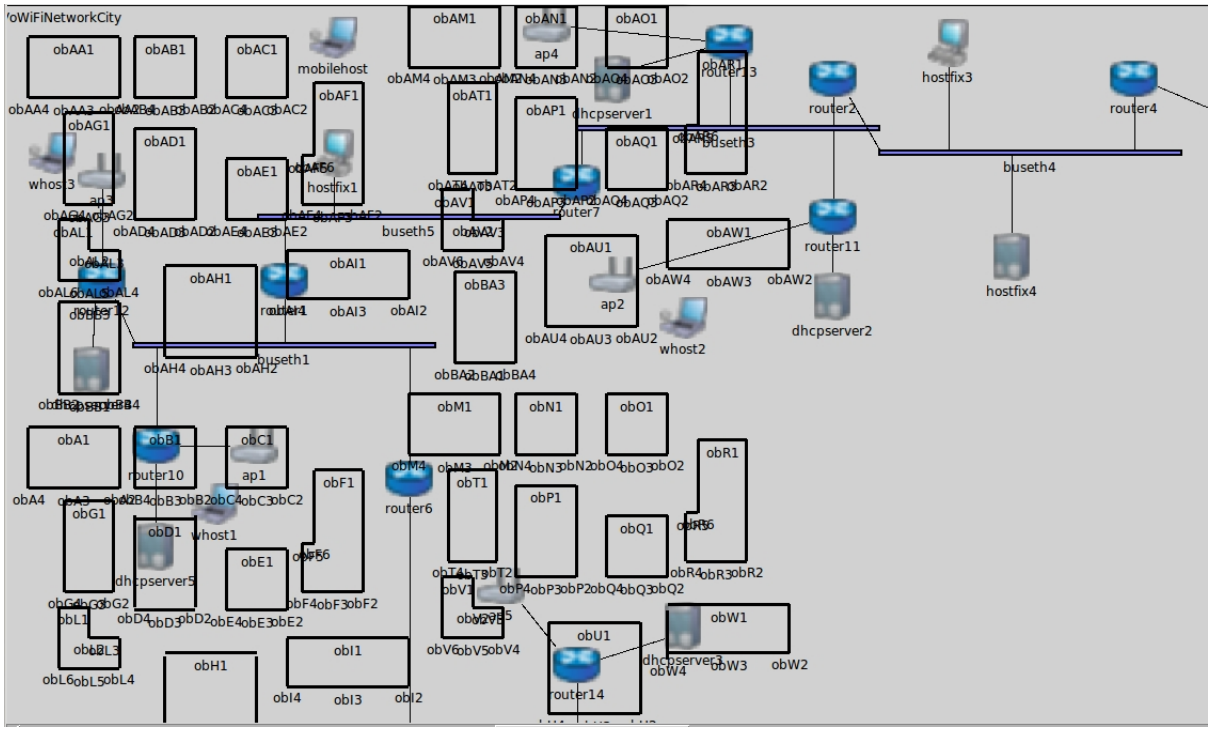

Figura 2.2: Dettaglio dello scenario simulativo

Alle classiche reti LAN su ethernet, vengono introdotte reti Wireless LAN (WLAN) basate sulla specifica IEEE 802.11, comunemente conosciuta come WiFi. Si consideri, ora, il nodo mobile, fondato sullo standard wireless, in cui sono state apportate delle modifiche che vengono esplicitate nella prossima sezione.

2.2 Il nodo mobile

Il nodo mobile è un semplice host, messo a disposizione da OMNET++\INET, a cui sono state fatte delle modifiche per fare in modo che questo sia dotato di più interfacce di rete wireless (nel nostro caso al massimo 2) e quindi garantire il controllo del bilanciamento di carico, ovvero la gestione dei pacchetti da trasmettere, o eventualmente da ritrasmettere, tramite le diverse interfacce disponibili.

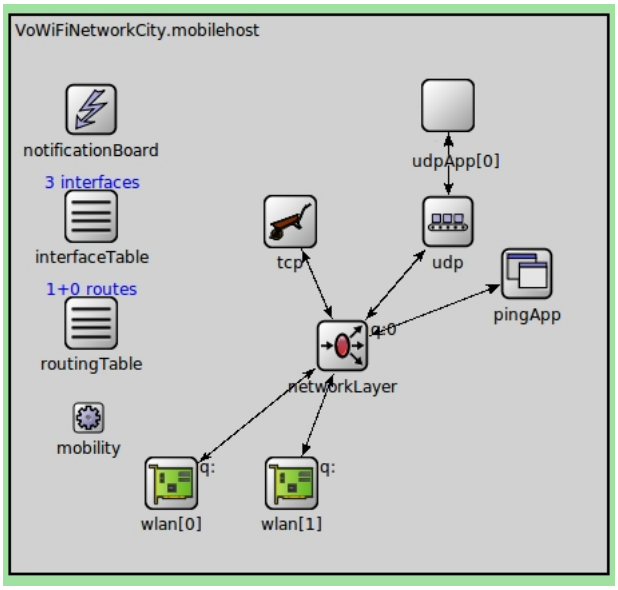

Figura 2.3: Il nodo mobile

Questo è costituito dai moduli:

- Notification Board: modulo di inet che lavora secondo un modello publish-subscribe e viene utilizzato dagli altri moduli per notificare ed essere notificati sul verificarsi di un evento, come ad esempio il cambiamento della tabella di routing, il cambiamento dello stato di una interfaccia di rete, etc.

- Interface Table: questo modulo mantiene tutte le informazioni riguardanti le interfacce (eth0, wlan0, ecc) presenti nell'host, in particolare il nome, maximum transmission unit, diversi flag, il datarate, etc. Le schede di rete durante la fase di inizializzazione devono registrarsi a questo modulo e se un qualsiasi altro modulo vuole lavorare con l'interface table module, deve in un primo momento ottenere un puntatore da esso e solo successivamente può accedere ad esso attraverso una semplice chiamata C++;

- Routing Table: questo modulo gestisce le routing table per IPv4 e viene acceduto dai moduli che sono interessati a lavorare con le rotte dei pacchetti (soprattutto il livello IP). Anche in questo caso, durante la fase di inizializzazione, il modulo carica le tabelle di route dai file *.irt e mette a disposizione chiamate a funzione per richiedere, aggiungere, cancellare e trovare le rotte migliori per un dato indirizzo IP.

- Mobility: è il modulo che si occupa di gestire il movimento del nodo attraverso modelli differenti (es: Passeggiate Aleatorie, etc), oppure, nel caso in cui il nodo sia fisso, viene utilizzato per calcolare la trasmissione wireless, in quanto legata alla posizione del nodo.

Ai moduli precedenti, che non sono stati modificati nella realizzazione della tesi e vengono messi a disposizione da INET, vengono aggiunti i successivi blocchi, che implementano i sette livelli dello standard ISO/OSI e le funzionalità di RWMA; si prenda in considerazione i primi due strati dello standard, quello fisico e quello datalink, che nel framework INET vengono implementati in un unico modulo, formato da queste quattro parti:

1. Agent
2. Management
3. MAC
4. Radio

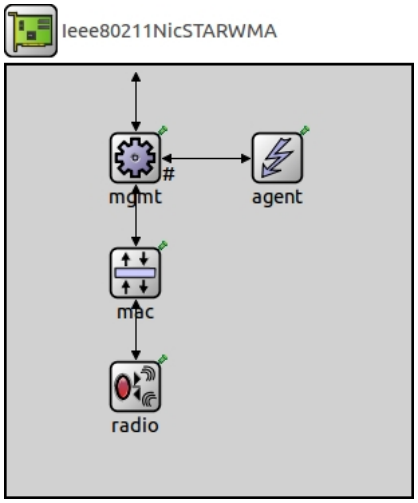

Figura 2.4: WNIC 802.11 in INET

L'**agent** è il modulo che si occupa di gestire il comportamento del livello annesso (management); in particolare ordina (attraverso messaggi command) a quest'ultimo di condurre operazioni quali la scansione dei canali radio, l'autenticazione o l'associazione con un Access Point; il livello management si limita a eseguire questi comandi per poi riportare il risultato all'agent.

Il **manager** si occupa di incapsulare e decapsulare i messaggi per/dal MAC, scambiare frame di gestione con altre station o AP. I frame Probe Request/Response, Authentication, Association Request/Response, etc., sono creati ed interpretati dal manager ma trasmessi e ricevuti attraverso il MAC. Durante la scansione è il manager che cambia periodicamente canale e memorizza le informazioni ricevute dai beacon e probe response.

Sono proprio questi due livelli che, assieme, realizzano una componente fondamentale per RWMA: il monitor; l'ULB deve essere a conoscenza di quali e quante schede wireless sono attualmente configurate ed utilizzabili per inviare pacchetti. Il monitor è stato implementato affinché segnali al livello applicazione quando una scheda viene correttamente associata ad un AP, o quando questa associazione viene meno. In caso di associazione o dissociazione, oppure in caso di perdita di troppi beacon, il monitor crea e invia un messaggio di tipo ReconfNot (Reconfiguration Notification) all'ULB, notificando l'evento.

Il **modulo MAC [19]** si occupa della trasmissione dei frame secondo il protocollo CSMA/CA[1]. Riceve dati e management frame dai livelli alti, e li trasmette. Quando un datagram IP arriva al MAC della WNIC, il pacchetto viene inviato attraverso l'interfaccia fisica su canale wireless.

Il protocollo IEEE 802.11 è correttamente implementato su INET, compreso il sistema di notifica della corretta ricezione di un frammento da parte dell'AP, attraverso il frame di controllo ACK. Inoltre vengono anche gestiti autonomamente dei timer (con l'uso di self-message) e viene controllato il numero di ritrasmissioni che il pacchetto ha subito, in modo che il livello MAC consideri il pacchetto perso dopo un certo periodo di tempo senza aver ricevuto risposta, oppure dopo aver superato il limite massimo di ritrasmissioni possibili; purtroppo in INET, il sistema non si occupa di segnalare l'avvenuta trasmissione o la perdita di un frammento ai livelli superiori come invece dovrebbe (il meccanismo è segnalato come "da implementare", ed è possibile che venga affrontato dagli sviluppatori di INET in un prossimo futuro).

Per tale motivo è stata apportata una modifica a questo livello introducendo il TED, componente essenziale in RWMA: una volta ricevuto ack dall'AP per un pacchetto oppure aver superato il numero massimo di ritrasmissioni concesse (in generale, 7), il TED comunica ai livelli superiori (fino al livello applicazione) l'esito della trasmissione del frame attraverso un pacchetto di tipo IP_NOTIFY_SUCCESS in caso di successo o di tipo IP_NOTIFY_FAILURE in caso di "perdita".

Il **blocco fisico (radio)** si occupa della trasmissione e ricezione dei frame. Modella le caratteristiche del canale radio e determina se un frame è stato ricevuto o no correttamente (ad esempio nel caso subisca errori a causa del basso potere del segnale o interferenze nel canale radio). I frame ricevuti correttamente sono passati al MAC.

[1] CSMA/CA (Carrier Sense Multiple Access/ Collision Avoidance: è l'algoritmo distribuito di contesa del canale utilizzato dallo strato MAC nello standard IEEE 802.11; è basato sulla regola per cui se una stazione deve trasmettere un frame MAC, si mette in ascolto del canale e se il canale risulta libero allora trasmette. In caso contrario aspetta la fine della trasmissione in corso, esegue una nuova scansione del canale e nel caso in cui risulta libero, inizia la fase di trasmissione, altrimenti si pone in uno stato di attesa. In questo algoritmo non viene fatta la rilevazione della collisione (ecco perché Collision Avoidance) a differenza dell'algoritmo CSMA/CD (Collision Detection), in quanto non adatta per le reti wireless.

A questo punto si sale di livello, arrivando a quello network.

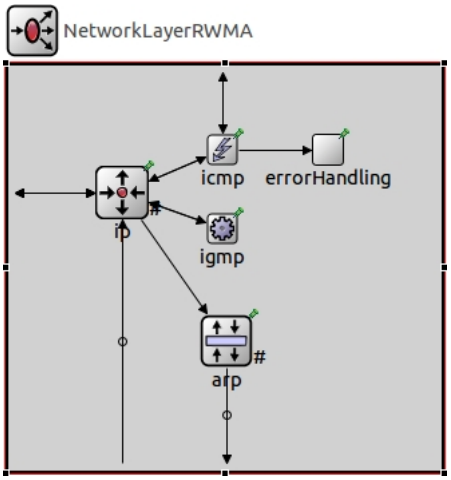

Figura 2.5: Network Layer

Anche in questo caso, il modulo è costituito di cinque blocchi:

Modulo ICMP, insieme al **modulo errorHandling**, è il blocco che implementa il protocollo ICMP (Internet Control Message Protocol), il quale si occupa di trasmettere informazioni riguardanti malfunzionamenti, informazioni di controllo o messaggi tra i vari componenti di una rete di calcolatori. I due blocchi non sono stati modificati nella realizzazione di RWMA, ma vengono mantenuti quelli di inet.

Modulo IGMP è il blocco responsabile di implementare il protocollo IGMP (Internet Group Management Protocol), usato dagli host e dai router vicini per stabilire i componenti di gruppo multicast; anche in questo caso, come nel precedente, non viene utilizzato quello fornito da inet.

Modulo IP è il blocco più importante ed implementa il protocollo IP; tutti i pacchetti dei livelli superiori (TCP, UDP, ICMP) vengono incapsulati in un datagram IP e passato ai livelli inferiori. In particolare, durante la trasmissione, il pacchetto proveniente dal livello trasporto viene incapsulato in un datagram IP e controllato nella sua dimensione; se questa risulta essere maggiore del Maximum Transmission Unit (MTU) della scheda di rete utilizzata, allora viene frammentato in più blocchi, inviato al modulo

ARP e viene deciso il percorso (route) che il datagram (o tutti frammenti) dovrà seguire. Nel caso in cui l'interfaccia utilizzata dal mobile node sia una 802.x, allora viene effettuata una ARP-resolution, in modo da ottenere l'indirizzo MAC del destinatario; se, invece, l'interfaccia utilizzata è quella p2p, allora il frammento o il datagram IP viene inviato direttamente alla scheda, senza effettuare una ARP-resolution. Queste operazioni vengono fatte ogni qualvolta il pacchetto di livello trasporto ha, nel proprio campo protocol, il valore IP_PROT_UDP, gestito dalla classe IP (.cc,.h.ned).

Invece, nel caso in cui il pacchetto di livello trasporto abbia nel campo protocol il valore IP_PROT_UDP_RWMA, lo stesso richiede il servizio RWMA e viene gestito dalla classe IPRWMA (.cc, .h, .ned), sottoclasse di IP. A questo punto, oltre a essere fatte tutte le operazione come in precedenza, il modulo IP invia un pacchetto di tipo IPNotify all'ULB (che si vedrà essere implementato a livello applicazione), nel quale si notifica che questo è arrivato con successo al modulo IP, e ne restituisce un identificativo univoco, necessario per una eventuale ritrasmissione.
In fase di ricezione, il modulo IP riceve il pacchetto dalla rete e lo gestisce in questi 3 step:

1. assemblare i frammenti: nel caso in cui il pacchetto ricevuto sia diviso in frammenti, questi vengono prima bufferizzati e, nel momento in cui arriva l'ultimo frammento, vengono assemblati;

2. estrarre il pacchetto di livello trasporto: il pacchetto assemblato viene "pulito" dalle informazioni accessorie e viene creato il datagram di livello trasporto;

3. consegnare il pacchetto al procollo di livello trasporto appropriato: il livello IP è collegato ai moduli e quindi ai vari protocolli di livello superiore (UDP,TCP, etc.) attraverso un gate, uno per ogni protocollo; il modulo IP mantiene una mappa <gate::protocollo> e proprio attraverso questa mappa consegna il pacchetto al protocollo di livello trasporto più appropriato.

Il **modulo ARP** implementa l' ARP (Address Resolution Protocol), protocollo che permette la traduzione da indirizzo IP a indirizzo MAC. Se un nodo A vuole inviare un pacchetto IP a un nodo B di cui non conosce l'indirizzo MAC, invia un messaggio ARP-request in broadcast inserendo il suo indirizzo IP e il suo MAC, in modo tale che ogni

nodo nella sottorete locale possa aggiornare le proprie mappe. Il nodo, di cui è stato richiesto il MAC, risponde con un ARP-response direttamente al nodo richiedente. Il nodo A che, a questo punto, ha ricevuto la risposta, aggiorna la propria ARP cache in modo tale che al prossimo invio di un nuovo pacchetto a B, non dovrà più fare ARP-Request e attendere la risposta, in quanto l'indirizzo MAC, ora, è contenuto nella sua cache.

Salendo di livello, si incontrano i moduli di livello trasporto; in questo capitolo, si tratterà il modulo UDP, lasciando il modulo TCP ad uno studio più approfondito nel prossimo capitolo.

Il **modulo UDP** è un blocco molto semplice che rende disponibili i servizi offerti dal livello network alle applicazioni UDP. Due aspetti fondamententali devono essere presi in esame in questo modulo:

- i socket UDP, struttura utilizzata per gestire facilmente l'invio e la ricezione di pacchetti UDP, generati dalle applicazioni UDP; l'oggetto fornisce procedure per creare, configurare e utilizzare correttamente il socket UDP, che mantiene importanti informazioni quali l'indirizzo IP locale, l'indirizzo remoto, la porta locale e la porta remota. Nel momento in cui l'applicazione UDP invia un messaggio, questo non può essere passato direttamente dall'applicazione al modulo UDP, ma incarica il socket di incapsulare il messaggio e passarlo a UDP; le operazioni duali vengono fatte durante la ricezione di un pacchetto. In altre parole, il soket si pone come interfaccia tra l'applicazione e il livello trasporto UDP.

- il secondo aspetto, non meno importante, è quello che all'interno del modulo UDP, il pacchetto proveniente dall'applicazione può seguire due "percorsi": il primo è quello standard, ovvero l'applicazione UDP non sfrutta l'architettura RWMA e quindi consegna i suoi pacchetti ad UDP, che li incapsula in un frame UDP inserendo nel campo protocol il valore IP_PROT_UDP; questo evita che l'applicazione sia costretta sempre e comunque a richiedere un servizio che magari non è realizzabile (es: il nodo non sia connesso attraverso una rete e wireless, ma su cavo). La seconda strada, invece, prevede che l'applicazione UDP usa l'architettura RWMA, consegnando i suoi pacchetti a UDPRWMA (.cc, .h, .ned), sottoclasse di UDP, che li incapsula in un frame UDP, inserendo nel campo protocol il valore

IP_PROT_UDP_RWMA; in entrambi i casi, i frame creati vengono passati al livello IP che, come si è detto in precedenza, li gestisce in modo differente.

Infine, salendo ancora un passo si trova il livello applicazione; qui, oltre alle varie applicazioni messe a disposizione da inet, si trova il cuore del sistema RWMA, ULB, il modulo dove tutti i messaggi di notifica vengono inviati per essere analizzati. Il modulo nel quale è stata implementata la componente ULB è chiamato **ULBRWMA** (.cc, .h, .ned) ed è una sottoclasse di UDPBasicApp[2].

Quando un'applicazione vuole inviare un pacchetto (RTP o di qualsiasi altro tipo) usufruendo del servizio RWMA, deve utilizzare la chiamata sendToUDPRWMA()[3] ereditata dalla classe ULBRWMA; come comprensibile dal nome della funzione, questa si comporta in modo analogo alla chiamata sendToUDP()[4], che è la funzione originale utilizzata per inviare un pacchetto con il protocollo UDP. La versione implementata nella tesi si differenzia da quest'ultima perché, oltre ad occuparsi di inviare il pacchetto, lo configura affinché sia riconoscibile dai livelli inferiori come pacchetto RWMA. Nel momento in cui l'applicazione invia il pacchetto a UDP tramite sendToUDPRWMA(), una copia di questo viene messo in una coda in attesa che il livello IP faccia pervenire il suo identificatore. Quando questo messaggio, contenente l'identificativo, arriva alla classe ULBRWMA, il pacchetto in attesa viene estratto dalla coda per essere inserito in una mappa dove resterà fino alla segnalazione del corretto invio o finché l'ULB deciderà di scartarlo.

La chiamata sendToUDPRWMA(), al contrario della sua controparte realizzata su Linux, non è bloccante. Questo è dovuto al fatto che in OMNeT++, basato su un meccanismo di message passing, l'unico modo che abbiamo per richiedere un servizio/informazione è fare una richiesta con un pacchetto attraverso i vari livelli (simili a comparti stagni).

[2]UDPBasicApp è una sottoclasse di UDPAppBase, e si occupa di gestire i comportamenti base di tutte le applicazioni UDP. Non possiede un gran numero di funzioni, ma contiene in sé già i campi basilari necessari a gestire una comunicazione UDP, come le porte o gli indirizzi.

[3]void sendToUDPRWMA (cPacket *msg, const IPvXAddress &srcAddr, int srcPort, const IPvXAddress &destAddr, int destPort).

[4]void sendToUDP (cPacket *msg, int srcPort, const IpvXAddress &destAddr, int destPort) è una funzione definita in UDPAppBase.

Inoltre, l'applicazione gestisce le varie schede di rete di cui il nodo mobile è costituito; in particolare, l'applicazione mantiene una mappa contenente l'indirizzo IP e lo stato della scheda di rete, comunicati attraverso il pacchetto Reconfiguration Notification inviato dal Monitor; la funzione utilizzata in ULBRWMA è chooseSrcAddrRWMA() che si occupa di restituire l'indirizzo dell'interfaccia più adatta per inviare il pacchetto. La funzione controlla la presenza di schede WORKING e, se almeno una di queste è presente, la restituisce terminando. Altrimenti, cerca e restituisce una scheda SUSPECTED. Nel caso non ce ne siano né dell'uno né dell'altro tipo, ritorna un indirizzo indefinito, che verrà riconosciuto dalla funzione di invio e farà scartare il pacchetto; questa gestione dell'interfaccia di rete risulta essere di primaria importanza perché, nel momento in cui viene ricevuta la notifica IP_NOTIFY_FAILURE (ricevuta quando il pacchetto non arriva al primo hop, come detto in precedenza), ULB può decidere se ritrasmettere o meno il pacchetto perso e nel caso in cui decida di ritrasmetterlo, può cambiare interfaccia se ve n'è un'altra configurata.

Ancora, in ULBRWMA viene implementato il protocollo DHCP (Dynamic Host Configuration Protocol). Nel momento in cui l'ULB riceve dal monitor un messaggio di tipo ReconfNot, questo significa che una scheda si sta associando o dissociando da un AP, e quindi aggiorna la sua map; lo status dell'interfaccia, a seguito di tale notifica, può passare dallo stato attuale a uno stato WORKING oppure DISABLED. Nel caso in cui passa ad uno stato WORKING, viene attivata la procedura DHCP: il nodo mobile invia un pacchetto DHCP di tipo DHCPDISCOVER inserendovi l'indirizzo MAC dell'interfaccia in questione. La richiesta viene quindi trasmessa al server DHCP utilizzando il protocollo UDP attraverso una comunicazione unicast; il server quando riceve un pacchetto DHCPDISCOVER, cerca un indirizzo disponibile tra quelli in suo possesso e se ne trova almeno uno, lo spedisce in risposta al client assieme all'indirizzo MAC dell'interfaccia mittente, inserendolo in un pacchetto DHCP di tipo DHCPREPLY. Il client, ricevuto tale pacchetto, estrae le informazioni e assegna l'indirizzo IP all'interfaccia con indirizzo MAC pari a quello presente nel pacchetto. Una volta configurata, l'interfaccia è raggiungibile dalla rete e può essere utilizzata dall'host per comunicare; il server è configurato, tramite file .ini, con l'indirizzo IP 128.128.128.128 (un indirizzo noto ai client) e un set di indirizzi IP relativo alla sottorete in cui si trova.
I nodi che vogliono configurare una propria interfaccia, dopo aver trasmesso il messaggio DHCPDISCOVER, cambiano temporaneamente l'indirizzo dell'interfaccia da configurare

e dalla quale riceveranno la risposta del server; questo è necessario affinché il client possa ricevere la risposta, altrimenti un nodo wireless spostandosi in un nuova subnet diventa irraggiungibile a causa dell'indirizzo IP non conforme alla nuova sottorete. L'indirizzo temporaneo che usa è 64.64.64.64 che è noto al server e quindi, il server quando risponde, trasmette il messaggio DHCPREPLY all'indirizzo 64.64.64.64.

Altre notifiche possono far cambiare lo status di una scheda e sono quelle provenienti dal TED. Quando un frammento viene inviato con successo (e la notifica IP_NOTIFY_SUCCESS arriva all'ULB), l'interfaccia (se necessario) viene definita WORKING. Se invece il frammento non viene spedito, all'arrivo della notifica IP_NOTIFY_FAILURE, la scheda di rete utilizzata diventa SUSPECTED oppure se era già SUSPECTED diventa DISABLED.

2.3 Il proxy server

Il proxy server (Figura 2.6) è stato inserito in una rete diversa da quella del nodo mobile, poiché svolge anche la funzione di secondo end-sytem nella comunicazione VoIP.

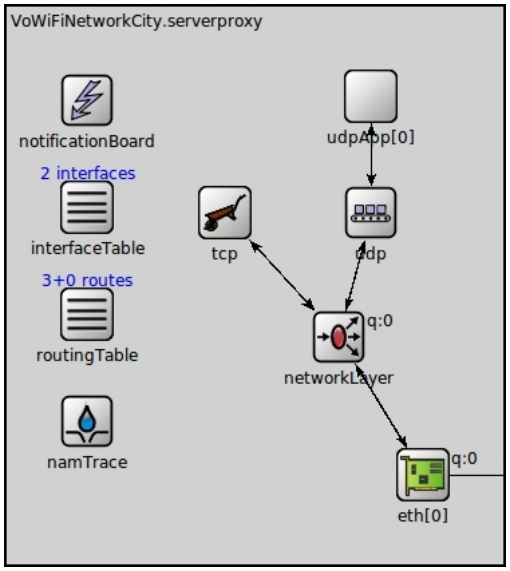

Figura 2.6: Proxy Server

Il proxy server si differenzia dal nodo mobile per due modifiche principali:

- la prima, banale, è quella per cui il proxy server non dispone di schede wireless, ma è collegato alla rete attraverso schede ethernet.

- la seconda, più sostanziale, è quella in cui è stato implementato un nuovo modulo load balancer (ULB) dell'architettura RWMA, chiamato ULBRWMAServerProxy, che presenta le funzionalità di instradatore per i dati. Il modulo ULBRWMAServer-Proxy è utilizzato dall'applicazione proxy, denominata UDPBasicAppForServer-Proxy, che viene eseguita da un nodo fisso collegato via cavo alla rete. Il load balancer gestisce una mappa <ID::indirizzo IP> in cui inserisce l'ID e l' indirizzo IP dei client che si appoggiano al proxy server per comunicare. Ogni volta che il proxy server riceve un messaggio, il suo modulo load balancer, prima di passarlo all'applicazione proxy, invoca il metodo updateAddrClient(), che aggiorna le informazioni della mappa relative al client. Il metodo estrae ID e indirizzo IP del mittente. Se tale ID non è presente nella mappa, aggiunge una nuova entry (ID,indirizzo IP); se invece è già presente, per mantenere aggiornata l'informazione sull'indirizzo IP, sovrascrive il vecchio indirizzo con quello appena letto. In questo modo i client sono sempre raggiungibili dal proxy server anche se cambiano indirizzo IP.
  Come si era già accennato, il server si comporta anche da secondo end system della comunicazione. L'applicazione UDPBasicAppForServerProxy che trasmette messaggi voce, in sostanza è la stessa dei client, ma si è dovuta creare una nuova classe poiché questa deve estendere quella del nuovo load balancer; l'applicazione genera messaggi che vengono instradati dal load balancer verso un destinatario scelto a caso fra quelli presenti nella mappa.

Capitolo 3

Trasmission Control Protocol

In questo capitolo si continua con l'analisi di un altro prerequisito di progettazione: il protocollo TCP e le sue caratteristiche principali. Nella prima parte viene fatta una panoramica sul protocollo TCP, ponendo attenzione ai meccanismi con cui il protocollo gestisce la congestione dei pacchetti, controlla il flusso di dati ed effettua la ritrasmissione di quelli perduti. Nella seconda parte viene presentato come il TCP è realizzato in Omnet++, con l'aiuto di INET.

3.1 Il protocollo TCP

3.1.1 Caratteristiche generali

Il TCP [2, 5] (Transmission Control Protocol) è un protocollo di livello trasporto. A differenza di UDP, è orientato alla connessione (o connection oriented), poiché due nodi, prima di trasmettere dati tra loro, devono instaurare una connessione; la fase di inizio della connessione, in cui vengono trasferite informazioni preliminari, per poi poter garantire un servizio affidabile di consegna dei pacchetti, viene detta handshake (o stretta di mano). E' un protocollo stream oriented, ovvero i dati trasmessi vengono visti come un flusso di byte ordinati e numerati e, inoltre, offre un servizio full-duplex, ovvero il mittente può inviare dati al destinatario e viceversa, ma più precisamente i flussi dati con direzioni diverse possono coesistere contemporaneamente; ciò significa che un nodo A può inviare pacchetti a un nodo B e nello stesso tempo può ricevere pacchetti da un nodo C. Ciò che non è possibile fare, invece, è un invio multicast: un mittente non può

trasferire dati a più destinatari in una sola operazione; se ipotizziamo che un generico host debba notificare qualcosa ad altri tre host, esso dovrà necessariamente instaurare una connessione TCP separata per ogni destinatario. La connessione è point-to-point, cioè relativa ad una sola sorgente e una sola destinazione. Le caratteristiche principali del TCP sono:

- trasferimento affidabile dei dati: tutti i dati inviati da un mittente sono recapitati al destinatario senza errori. Può succedere che, per vari motivi (errore tra i collegamenti, disturbi esterni alla comunicazione, etc.), qualche pacchetto può contenere errori oppure vada perduto nella rete. A questo punto il TCP si occuperà di rilevare questa mancata ricezione e procederà a ritrasmettere il pacchetto perso; questa caratteristica è presente in TCP e non in UDP che, invece, non garantisce un trasferimento affidabile dei dati;

- corretto ordinamento dei pacchetti: può accadere che i pacchetti contenenti le informazioni subiscano ritardi all'interno della rete (per via del congestionamento della stessa o perché due pacchetti possono seguire due rotte differenti per arrivare a destinazione, etc.) con il conseguente risultato che arrivano a destinazione non in ordine. Il compito del TCP è proprio quello di riassemblare il messaggio in maniera corretta, ordinando i pacchetti per il numero di sequenza contenuto nei pacchetti stessi e inoltrarlo al livello superiore;

- controllo del flusso: il TCP stabilisce se un host più veloce nella trasmissioni dati rischia di mandare in overflow il buffer di ricezione di un host più lento. Può accadere, infatti, che l'applicazione mittente generi i pacchetti con una frequenza molto maggiore rispetto a quella con cui l'host destinazione riesce a ricevere, con il conseguente riempimento del buffer di ricezione e della successiva eliminazione dei pacchetti da parte del destinatario. Per evitare questo, il TCP abbassa la frequenza di trasmissione del mittente al fine di equilibrare le prestazioni dei due host;

- controllo della congestione [4, 7]: anche in questo caso, il TCP si accorge che c'è un congestionamento diffuso della rete, e quindi impone all'host mittente di diminuire la frequenza di trasmissione dei pacchetti per evitare di aggravare la situazione.

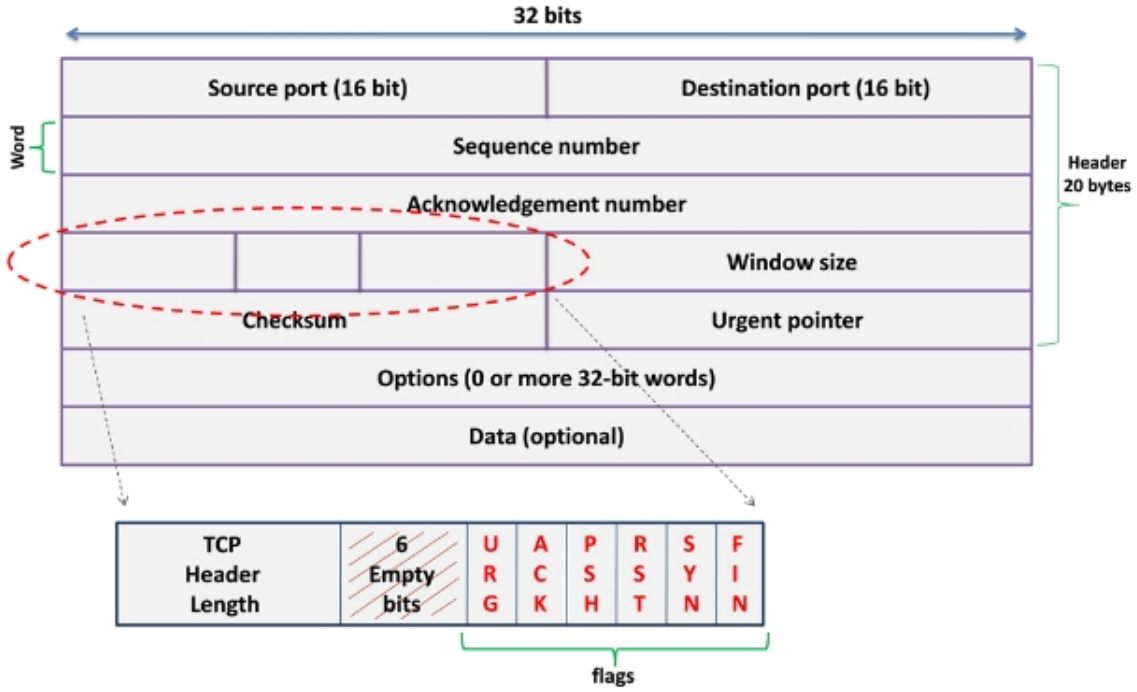

Figura 3.1: Pacchetto TCP

3.1.2 Il pacchetto TCP

La struttura di un pacchetto TCP (Figura 3.1) prevede:

- i campi Source Port e Destination Port, necessari per il corretto multiplexing e demultiplexing da parte delle singole applicazioni su un host;
- il campo Checksum viene utilizzato per capire se un pacchetto è arrivato a destinazione con errori di trasmissione oppure no;
- i campi Sequence Number e Acknowledgement Number, entrambi di lunghezza pari a 32 bit, contengono rispettivamente il numero di sequenza del pacchetto che si sta inviando e il numero del pacchetto che è stato ricevuto e di cui si sta comunicando l'acknowledgement. Sono i campi più importanti dell'header di un pacchetto TCP in quanto su di essi si basa il trasferimento affidabile dei dati; si ipotizzi, ad esempio, di dover trasmettere dal nodo A al nodo B un messaggio di lunghezza 2000 byte e che il MSS (Maximum Segment Size) sia fissato a 500 byte. Il messaggio

viene frammentato in quattro segmenti; il numero di sequenza del primo segmento sarà 0, del secondo segmento 500, del terzo segmento 1000 e infine del quarto 1500. Il campo Aknowledgement Number viene utilizzato dal nodo B verso il nodo A in risposta all'arrivo dei frammenti inviati; ad esempio, il nodo A invia il primo segmento (di 500 byte) al nodo B (frammento con sequence number posto a 0), il quale riceve il frammento correttamente e invia un pacchetto di ACK al nodo A, contenente nel campo Acknowledgement Number il valore 501 e così via per i frammenti successivi.
Si consideri, ora, un'altra situazione, ovvero lo scenario in cui l'host di destinazione abbia ricevuto correttamente il primo (0-500 byte) e il terzo (1000-1500 byte) frammento, ma non il secondo (500-1000 byte). In questa situazione anomala l'host destinatario scriverà nel campo Acknowledgement Number del pacchetto di ACK il numero 501, in quanto il campo fa riferimento al primo byte successivo tra quelli arrivati in maniera ordinata. Una volta ricevuto il secondo pacchetto (500-1000 bytes), il destinatario, accorgendosi che il terzo frammento è già stato inviato, scriverà nel numero di riscontro il valore 1501 che fa riferimento al quarto e ultimo segmento dei dati totali.

- il campo Header length è formato da 4 bit ed è usato per specificare la lunghezza totale dell'intestazione del pacchetto; viene utilizzato quando il campo opzioni ha dimensioni variabili, e quindi permette di calcolare la grandezza dell'intestazione per ogni singolo pacchetto;

- il campo Window Size è un campo di lunghezza 16 bit ed è viene utilizzato per il controllo di flusso descritto in seguito;

- il campo Option è l'unico facoltativo, ha una lunghezza variabile e viene utilizzato per l'abilitazione di particolari versioni del protocollo;

- il campo Flag ha lunghezza 6 bit ed ognuno di essi può essere visto come un campo o a se stante. Proprio questi flag vengono utilizzati per implementare il servizio di ritrasmissione in caso di perdita e per effettuare l'handshake iniziale. Il bit ACK indica, se settato a 1, che il segmento ingloba un riscontro di un pacchetto ricevuto precedentemente senza errore. RTS,CTS e FIN vengono utilizzati per il processo di handshake e per la chiusura della connessione. Il campo PSH, poco usato, impone

all'host che riceve il pacchetto di inoltrarlo immediatamente al livello applicazione. Il campo URG, anch'esso poco usato, indica all'host di destinazione che il segmento contiene dei pacchetti che il livello applicativo mittente ha marcato come urgenti.

3.1.3 Handshake a tre vie

Come detto in precedenza, il TCP è un protocollo connection oriented, ovvero due nodi che vogliono comunicare tra loro, prima di poter trasferire dei dati devono instaurare la connessione. Questa viene stabilita attraverso un algoritmo chiamato handshake a tre vie, in quanto consta di tre fasi principali (come in Figura 3.2):

- il client invia al server un segmento vuoto, senza dati, con il bit SYN del campo Flag posto a 1 e con valore del campo sequence number inizializzato a un valore CLIENT_ISN, scelto in maniera pseudocasuale tra un range di valori possibili;
- il server riceve il segmento e risponde al client con un altro segmento vuoto con il bit SYN posto a 1, con campo di riscontro (Acknowledgement Number) settato al valore CLIENT_ISN+1 e con numero di sequenza settato a un altro valore iniziale SERVER_ISN, scelto questa volta in modo pseudocasuale dal server.
- il client invia al server un nuovo segmento vuoto, senza dati, con il bit SYN posto a 0, con campo di riscontro (Acknowledgement Number) settato al valore SERVER_ISN+1 e con numero di sequenza settato al valore CLIENT_ISN+1.

Lo stesso algoritmo viene utilizzato anche per la chiusura della connessione; in questo caso, però, consta di quattro fasi ed è per questo che viene chiamato handshake a tre vie (FIN+ACK) oppure handshake a quattro vie (Figura 3.3). In particolare:

- il client invia al server un segmento con il bit FIN del campo Flag settato a 1, e si pone nello stato FIN_WAIT_1;
- il server invia al client un riscontro e il client, una volta ricevuto ACK, entra nello stato FIN_WAIT_2;
- il server invia al client un segmento con il bit FIN del campo Flag settato a 1;
- il client riscontra il segmento inviando un ACK al server ed entra nello stato TIME_WAIT; dopo un tempo variabile la connessione viene chiusa.

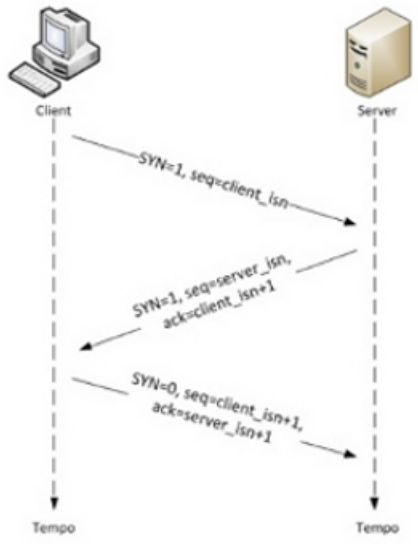

Figura 3.2: Handshake a tre vie

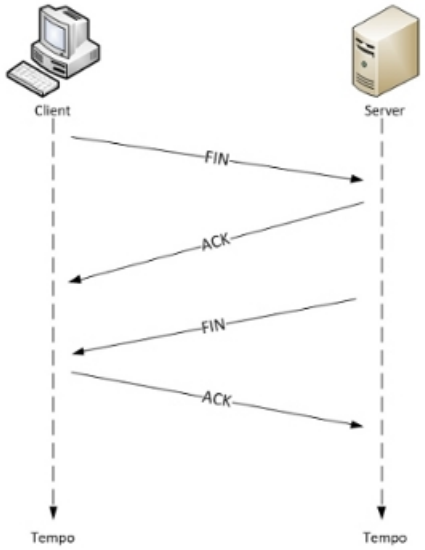

Figura 3.3: Handshake a tre vie (FIN+ACK)

3.1.4 Controllo di flusso

Si supponga una comunicazione tra un nodo A e un nodo B; nel momento in cui il nodo A invia un pacchetto al nodo B questo, al momento della ricezione, viene posto in

un buffer di ricezione, inizializzato durante la fase di handshake e letto periodicamente dall'applicazione, che ne processa il contenuto. Se la lettura dell'applicazione sul nodo B è molto più lenta dell'invio da parte del nodo A si può verificare un overflow del buffer di ricezione; il controllo di flusso serve proprio ad evitare questo problema. Entrambi i nodi tengono traccia all'interno di una variabile, detta finestra di ricezione (o Received Windows) dello spazio libero del buffer di ricezione nel nodo corrispondente; questa variabile conterrà un valore dinamico calcolato nel seguente modo:

- il nodo destinatario mantiene due puntatori: Last Read Byte e Last Received Byte, che contengono rispettivamente il valore dell'ultimo byte letto dall'applicazione destinazione dal buffer e il valore dell'ultimo byte ricevuto dalla destinazione e inserito all'interno del buffer. Durante la fase di invio dell'ACK, il destinatario pone nel campo Windows Size del pacchetto il valore della Received Windows data da

$$Received\,Windows = Buffer\,Size - (Last\,Read\,Byte - Last\,Received\,Byte)$$

  dove Buffer Size è la dimensione massima del buffer di ricezione.

- il nodo mittente, alla ricezione dell'ACK, aggiorna la Received Windows leggendo il valore contenuto nel campo Windows Size del pacchetto ricevuto; inoltre, alloca due ulteriori variabili Last Sent Byte e Last Verified Byte di ovvio significato. Queste due variabili sono utilizzate per adattare la trasmissione dei dati allo spazio libero nel buffer di ricezione del destinatario, creando pacchetti con dimensioni che rispettano la seguente formula

$$Last\,Sent\,Byte - Last\,Verified\,Byte \leq Received\,Window$$

Questo sistema di controllo di flusso è estremamente semplice e garantisce ottimi risultati; la versione descritta, tuttavia, richiede un piccolo correttivo dato dal caso limite, in cui il buffer di ricezione è pieno e quindi la finestra di ricezione ha valore pari a 0. L'host mittente, nel momento in cui riceve all'interno del campo Windows Size il valore 0, interrompe immediatamente l'invio dei dati per evitare l'overflow del buffer. Nel frattempo l'applicazione destinazione andrà a leggere i dati nel buffer, con il conseguente svuotamento dello stesso. L'host destinazione però non potrà notificare all'host mittente che il buffer si sta svuotando, dato che non vengono più generati i pacchetti ACK nei

quali viene inserita l'informazione sulla finestra di ricezione. In questo modo l'host A non potrà in nessun modo capire che il buffer del destinatario si è svuotato e di fatto rimane bloccato. Per ovviare a questa criticità, le specifiche TCP impongono all'host mittente di inviare un pacchetto della dimensione di un byte, per permettere al destinatario di notificare il nuovo stato del buffer.

3.1.5 Trasferimento affidabile dei dati

Come detto in precedenza, il TCP garantisce che tutti i dati inviati da un mittente sono recapitati al destinatario senza errori; lavorando in una rete, per di più wireless, si possono verificare eventi che non garantiscono il trasferimento affidabile dei dati. Tra questi, i principali sono:

- corruzione: il pacchetto ricevuto dal destinatario contiene degli errori;
- timeout: il segmento inviato dal mittente viene ricevuto dal destinatario "fuori orario" oppure non viene ricevuto per niente;
- arrivo fuori ordine: i pacchetti, inviati dal mittente verso il destinatario, possono seguire rotte differenti nella rete e quindi potrebbero arrivare in un ordine diverso rispetto a quello utilizzato dal mittente in fase di trasmissione.

Il TCP attua diversi accorgimenti per ovviare a questi errori:

1. riscontro: nel momento in cui il destinatario riceve un pacchetto corretto, il ricevente notifica al mittente l'avvenuta ricezione attraverso uno speciale pacchetto detto ACK (Acknowledgement); la mancata ricezione di un segmento ACK può causare l'invio di pacchetti duplicati;
2. numeri di sequenza: sia i segmenti che gli ACK hanno dei numeri identificativi univoci all'interno del campo sequence number; questo permette di comunicare quale o quali segmenti contengono errori, quali sono arrivati fuori ordine o non sono arrivati affatto, etc.;
3. timer: segmenti e ACK possono essere persi durante la trasmissione. TCP gestisce questa fonte di errori impostando un timeout quando vengono inviati i dati; nel

caso in cui il timeout scada prima dell'arrivo del riscontro, i segmenti vengono inviati nuovamente. TCP gestisce quattro diversi timer per ogni connessione:

- retransmission timer: gestisce i timeout di ritrasmissione, che si hanno quando si è terminato l'intervallo tra l'invio di un datagramma e la ricezione del segmento acknowledgement. Il valore del timeout non è statico, ma viene calcolato dinamicamente a seconda della velocità e della congestione della rete; in particolare viene determinato misurando il tempo necessario al segmento per arrivare al destinatario, aggiungendo quello dell'ACK per ritornare al mittente; questo tempo viene chiamato round trip time (RTT). Nel protocollo viene effettuata una media degli RTT e ne consegue un valore atteso chiamato smoothed round-trip time o SRTT.
  Nel momento in cui viene inviato un datagram, il retransmission timer viene settato con il valore SRTT calcolato; se l'ACK arriva entro lo scadere del timer, ciò presuppone che il pacchetto è arrivato a destinazione, allora il timer viene annullato; mentre se il timer scade, il datagram viene inviato nuovamente e il timer settato ancora con un RTO (Retransmission Timeout) modificato, che usualmente si incrementa esponenzialmente fino a un limite massimo, oltre il quale si presume un fallimento della connessione.
- timed wait: nel momento in cui viene interrotta la connessione TCP, è possibile che alcuni datagram tentino di accedere alla porta legata alla connessione terminata; il timed wait ha lo scopo di impedire che la porta appena chiusa venga aperta di nuovo per ricevere questi datagrammi, che, conseguentemente, vengono scartati. Questo timer è usualmente impostato al doppio del tempo di vita massimo di un segmento (lo stesso valore del campo Time to live in un header IP).
- persist timer: gestisce un evento molto raro. Come precedentemente detto, si può verificare la situazione in cui il buffer di ricezione del nodo destinatario sia colmo e quindi che il campo windows size contenga il valore zero, costringendo il nodo mittente a mettere in pausa la trasmissione. Si è, inoltre, specificato che per riattivare la comunicazione, il nodo mittente manda un messaggio di un byte a cui il nodo destinatario risponde inviando la dimensione del buffer, oramai svuotata; tuttavia, si può verificare che questi messaggi vengano persi,

causando un ritardo infinito nel riavvio della comunicazione.
Il persist timer viene utilizzato per fare in modo che nel caso in cui il messaggio di un byte venga perduto, il nodo attende lo scadere del timer e poi invia nuovamente il segmento di un byte ad intervalli predeterminati, per assicurarsi che la macchina riceva il messaggio e stabilire se è ancora bloccata. La macchina ricevente rinvia il messaggio di finestra di ampiezza zero ogni volta che riceve uno di questi segmenti di stato, se è ancora intasata, altrimenti inoltra un messaggio con il nuovo valore di finestra e la comunicazione viene ripristinata.

- keep-alive timer: manda un pacchetto vuoto a intervalli regolari per assicurare che la connessione con l'altra macchina sia ancora attiva. Se non si è ricevuta risposta dopo che è stato inviato il messaggio, prima che scada il timer di idle, si presume che la connessione sia interrotta. Il keep-alive timer è usualmente impostato da un'applicazione con valori che variano fra 5 e 45 secondi. Il timer di idle è usualmente impostato a 360 secondi.

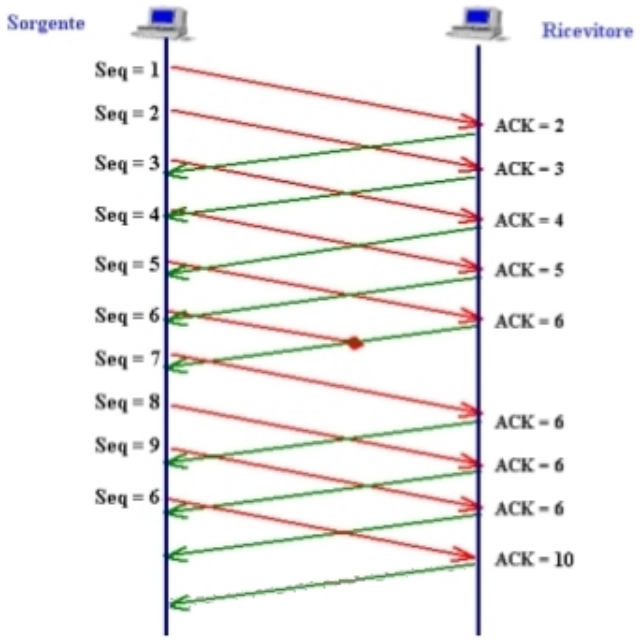

Figura 3.4: Trasferimento dati e controllo di errore in TCP

Si analizzi, ora, ciò che accade, considerando una comunicazione da una sorgente verso una destinazione (Figura 3.4):

- il nodo mittente invia il primo segmento, che viene ricevuto correttamente dal destinatario; per non abbassare l'efficienza della comunicazione, il nodo mittente non sempre attende l'ACK dal nodo destinatario per il pacchetto inviato, così capita che nel momento in cui il nodo mittente ha "pronto" un nuovo pacchetto, lo invia. Alla ricezione dell'ACK-2, il trasmettitore interrompe il retransmission timer del segmento impostato sul primo pacchetto;

- i pacchetti (2),(3),(4),(5), come per il pacchetto (1), vengono inviati e ricevuti correttamente dal destinatario, il quale notifica con i rispettivi ACK-3, ACK-4, ACK-5, ACK-6;

- il sesto segmento (6) viene scartato dalla rete, e dato che per questo non perviene nessun ACK, il suo timer continua ad incrementarsi. La ricezione dei segmenti (7), (8), (9), tutti fuori sequenza, comporta nel ricevente il loro mantenimento nel buffer di ricezione, e l'invio di riscontri contenenti sempre lo stesso valore ACK-6, ad indicare che manca ancora il segmento (6);

- alla ricezione del terzo ACK duplicato (in quanto già ricevuto), il trasmettitore attua la procedura di fast retransmit, e reinvia il segmento (6) senza attendere lo scadere del timer (che viene resettato, e fatto ripartire). Questa volta il segmento è ricevuto correttamente, ed il ricevente ricompone il messaggio, svuota il buffer di ricezione, invia ACK del prossimo segmento atteso (il 10), ed usa questo valore per riscontrare tutto quanto ricevuto fino a quel momento;

- il trasmittente, alla ricezione dell'ACK, rimuove dal buffer di trasmissione tutti i segmenti riscontrati ed elimina i rispettivi timer.

Nell'esempio precedente non si è tenuto conto di alcuna finestra che modifica il ritmo di invio dei segmenti. Ma come si è detto, il meccanismo a finestra scorrevole determina, istante per istante, il numero massimo di bytes che possono essere trasmessi consecutivamente verso il destinatario senza ricevere riscontri, e consente a un nodo poco veloce nella gestione dei segmenti ricevuti, di adeguare la velocità di tramissione alle proprie capacità. Infatti, il ricevente può variare la dimensione della finestra di trasmissione nel

corso della connessione, impostando il valore del campo Windows Size che compare nella intestazione TCP dei pacchetti inviati in occasione degli ACK.

Un altro aspetto basilare per il TCP è il calcolo del valore di RTT (Round Trip Time), da assegnare al retransmission timer. Questo valore può variare secondo molti fattori, principalmente dovuti alla congestione del traffico di rete e dal percorso scelto per la consegna dei pacchetti. Inizialmente TCP misura il valore di RTT inviando all'end system ricevente un pacchetto con un particolare numero di sequenza, aspettando la risposta.

Nella formula sottostante verrà utilizzata M per denotare il valore di RTT misurato. Le specifiche originali di TCP prevedono che il protocollo aggiorni un valore di RTT stimato (Smoothed RTT Estimator) per ogni misurazione effettuata. Questo valore viene indicato come R e calcolato nel seguente modo:

$$R = \alpha * R + (1 - \alpha) M$$

dove α è un fattore fisso di smoothing prefissato a 0,9. Il 90% del nuovo valore è dato da precedenti stime, mentre solo il 10% è determinato dalla nuova misurazione. Dato questo smooth estimator, che cambia a seconda del valore di RTT, RFC 793 raccomanda che il valore di RTO (retransmission timeout) venga impostato a

$$RTO = R * \beta$$

dove β è il delay variance factor con valore prefissato 2.

A questo metodo di calcolo, si affianca l'algoritmo di Karn che affronta il problema di ottenere stime accurate del valore di RTT per i messaggi inviati tramite protocollo TCP. Teoricamente, risulta difficile ottenere stime accurate di questo valore a causa di ambiguità sui segmenti ritrasmessi. Il valore RTT viene stimato calcolando il tempo intercorso tra l'invio di un segmento e la ricezione dell'ACK corrispondente, ma i pacchetti ritrasmessi possono causare ambiguità: il segmento ACK potrebbe essere la risposta al primo invio o a una seguente ritrasmissione. L'algoritmo ignora i valori di RTT calcolati per i segmenti ritrasmessi al fine dell'aggiornamento del Round Trip Time Estimator: questo valore viene aggiornato limitandosi a quei segmenti che sono stati spediti una volta sola. Questa implementazione semplicistica dell'algoritmo può portare a notevoli problemi: consideriamo cosa succede quando viene inviato un segmento TCP

dopo un drastico aumento di ritardo della trasmissione. Usando il valore precedente di Round Trip Time Estimator, TCP valuta un timeout e, allo scadere di quest'ultimo, rinvia il pacchetto. Se l'algoritmo non calcola il nuovo RTT valutando anche i pacchetti ritrasmessi, il protocollo continuerà a inviarne altri senza tener conto dell'aumento di delay del canale. Una soluzione a questo problema è stata trovata utilizzando una strategia di backoff nei timeout: viene calcolato un timeout iniziale e, nel caso scada a causa di ritrasmissioni, viene incrementato di un fattore 2 (new timeout = old timeout $*$ 2). Questo algoritmo ha dato notevoli risultati in scenari con alti valori di pacchetti persi.

3.1.6 Controllo della congestione

Una delle principali cause di degradazione delle prestazioni delle reti è senza dubbio la congestione, che si verifica quando per un certo intervallo di tempo il traffico offerto generato dai terminali è maggiore della possibilità di smaltirlo da parte della rete stessa. Infatti, ogni volta che un nodo interno alla rete riceve dati più velocemente di quanto riesca a smaltirli, è costretto a creare una coda e mettere alcuni e pacchetti in attesa. Pur senza entrare nello specifico della gestione delle code e della teoria del traffico, è facile intuire quanti inconvenienti possano creare alle prestazioni della rete il perdurare di una situazione di congestione in diversi nodi. Oltre ad aumentare in modo notevole il ritardo dei pacchetti, può infatti succedere che dei buffer si saturino e scartino dei pacchetti: il terminale sorgente provvederà dunque a ritrasmetterli, aumentando ancora di più il carico offerto alla rete. Ancora peggiore è il caso in cui un pacchetto che passa molto tempo nella coda di un router venga ritenuto perso dall'host che lo ha generato (a causa dello scadere del Timeout): il mittente lo ritrasmetterà anche se in realtà non è necessario, sprecando così notevoli risorse.

Le diverse versioni di TCP che sono state sviluppate nel corso degli anni si differenziano principalmente proprio per il modo di reagire a degradazioni delle prestazioni della rete, siano esse dovute a congestioni o a rumorosità del canale. Nelle prime versioni del TCP (fino al 1986), sostanzialmente non veniva preso alcun provvedimento per evitare i problemi causati dalle congestioni. Nel 1986 fu introdotto il TCP Berkeley, che cominciava ad affrontare il problema introducendo gli algoritmi Slow Start e Congestion Avoidance [8]. Successivamente, nel 1988, il TCP Tahoe aggiunse a questi anche l'algo-

ritmo Fast Retransmit [8], e nel 1990 con il TCP Reno fu introdotto l'algoritmo Fast Recovery [8]. Nel 1995 questo venne leggermente modificato (TCP New Reno [9]), e per molti anni rimase il protocollo più utilizzato nella rete. Il TCP Vegas [11] introdurrà una nuova filosofia, cercando di prevenire la congestione e rallentando il ritmo di invio dei dati prima che si verifichino eventi di perdita, ma questo lo renderà poco compatibile con le altre versioni. Con l'avvento delle wireless LAN furono sviluppate nuove versioni che meglio si adattavano al nuovo canale, come TCP Westwood [10] e Veno [16]. L'approccio seguito dal TCP per effettuare il controllo della congestione è di fare si che ogni mittente limiti il ritmo con cui immette traffico nella sua connessione in funzione della congestione in rete percepita. Ovviamente, se un terminale percepisce che c'è poca congestione nel percorso tra sé e la destinazione, il TCP aumenta il suo ritmo di trasmissione, mentre lo riduce se sente che alcuni nodi sono congestionati. Il primo argomento da affrontare è sapere come il TCP si accorge che la rete è congestionata e come fa per regolare il ritmo di trasmissione. Si consideri in prima battuta come il mittente regola la quantità di dati che immette nella rete: per fare ciò il TCP utilizza una nuova variabile detta finestra di congestione (Congestion Windows). La finestra di congestione (Cwnd) impone una limitazione addizionale alla quantità di traffico che un host può inviare, oltre a quella che già abbiamo analizzato per il controllo del flusso. Riprendendo la terminologia precedentemente usata, l'ammontare di dati non riscontrati che un terminale può avere durante una connessione TCP risulta:

$$Last\ Byte\ Sent - Last\ Byte\ Acked \leq minCwnd, RcvWin$$

Per capire se la rete è congestionata o meno, il TCP utilizza i cosiddetti eventi di perdita, che possono essere o lo scadere di un timeout o, nelle versioni con l'algoritmo Fast Retransmit, la ricezione di tre ACK duplicati: quando c'è congestione, infatti, uno o più buffer dei router lungo il percorso si possono saturare fino a dover scartare dei pacchetti. Il mittente si accorge del datagram perso e quindi della congestione grazie appunto agli eventi di perdita e può dunque agire di conseguenza. Come già accennato, il modo di reagire a queste situazioni cambia a seconda delle versioni del TCP: ne vengono presentate le caratteristiche.

3.1.6.1 TCP Berkeley

Il TCP Berkeley (Figura 3.5) fu la prima versione che tentò di evitare di congestionare la rete con un eccessivo traffico. Esso, tuttavia, non prevedeva di cercare di capire lo stato della rete e poi regolarsi di conseguenza, ma semplicemente modificava il ritmo di invio dei dati quando questo raggiungeva una certa soglia (Ssthresh: Slow Start Threshold). In pratica quando la connessione viene avviata, il valore di Cwnd che regola il ritmo di spedizione è inizializzato ad un valore pari a 1 MSS, ossia una quantità di dati molto piccola. Considerato che molto probabilmente la banda disponibile è molto maggiore, nella prima parte della connessione il valore di Cwnd viene aumentato esponenzialmente per ogni Round Trip Time. Ogni volta che il mittente riceve un ACK, il valore di Cwnd viene modificato: $Cwnd = Cwnd + 1MSS$. Accade che il mittente invia il primo pacchetto, riceve il rispettivo ACK e porta la Cwnd al valore 2. Vengono dunque inviati adesso 2 pacchetti che, una volta riscontrati, faranno salire la finestra di congestione a 4: in questo modo la quantità di dati inviata raddoppia per ogni RTT. Questa fase della connessione è conosciuta come Slow Start. Quando la variabile Cwnd raggiunge il valore di Ssthresh, la connessione entra nella fase di Congestion Avoidance, che riduce il ritmo di trasmissione incrementando la finestra di congestione di un solo MSS ogni RTT: si ottiene così una crescita lineare e non più esponenziale della quantità di dati spediti. In particolare, per ogni ACK ricevuto, si ha: $Cwnd = Cwnd + \frac{1}{Cwnd}$. Il valore di Ssthresh viene impostato, all'inizio della connessione, a un valore molto alto: dunque, nella prima parte della connessione si rimane nella fase di Slow Start relativamente a lungo. Quando si verifica il primo evento di perdita, ossia lo scadere del Timeout, il valore di Ssthresh viene portato a $Ssthresh = \frac{Cwnd}{2}$ e $Cwnd = 1MSS$. In questo modo la connessione riparte dalla fase di Slow Start e vi rimane fino a che Cwnd non raggiunge il nuovo valore di Ssthresh, quando cioè entra in fase Congestion Avoidance. Per il resto della connessione, tutte le volte che scade il timeout, ciò che succede è: $Ssthresh = \frac{Cwnd}{2}$ e $Cwnd = 1MSS$.

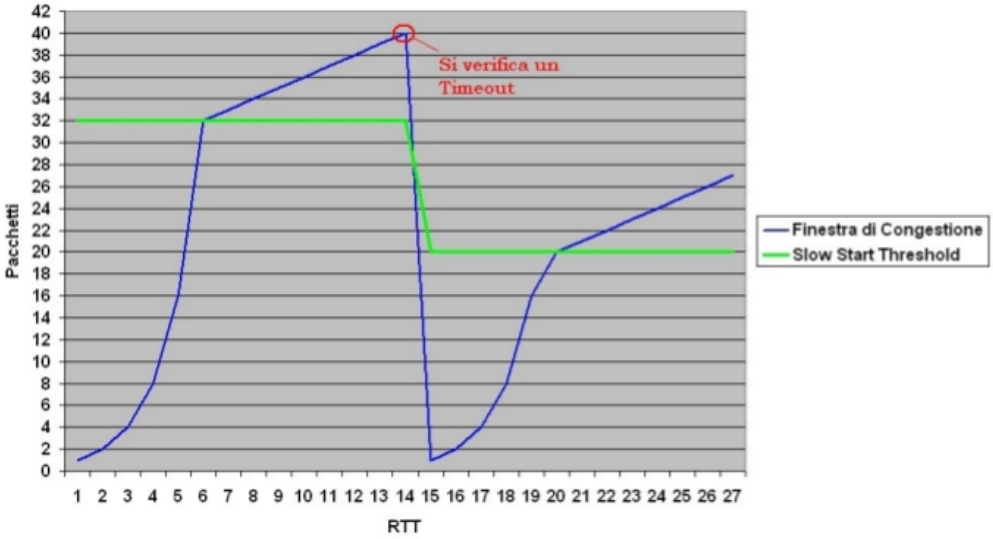

Figura 3.5: TCP Berkeley

3.1.6.2 TCP Tahoe

Il TCP Tahoe (Figura 3.6) introduce l'algoritmo chiamato Fast Retransmit. Oltre allo scadere del timeout, la perdita del pacchetto è data dalla ricezione da parte del mittente di un certo numero di ACK duplicati (o DUPACK), che riporta $Ssthresh = \frac{Cwnd}{2}$ e $Cwnd = 1MSS$.

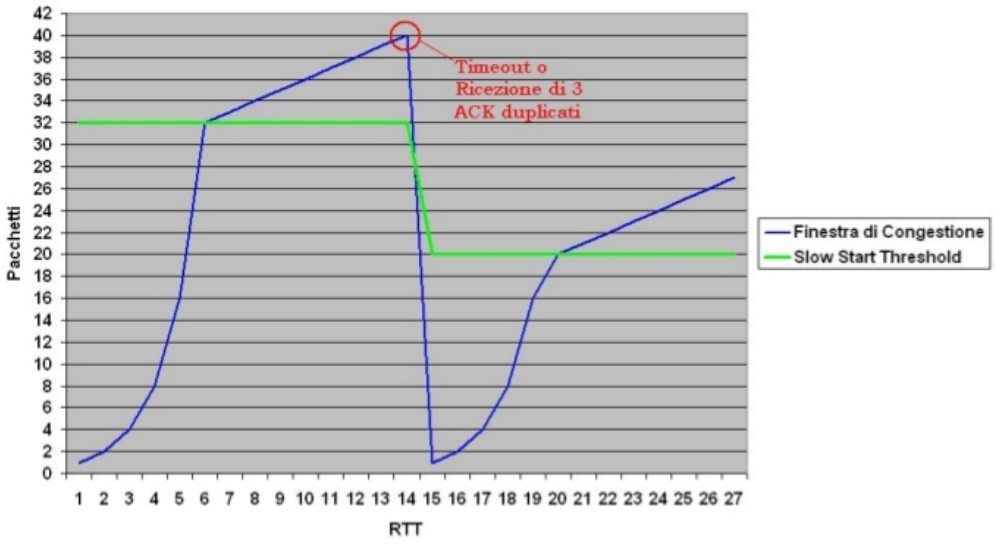

Figura 3.6: TCP Tahoe

3.1.6.3 TCP Reno

Il TCP Reno è la versione che introduce l'algoritmo chiamato Fast Recovery, che comporta una differente reazione del mittente a seconda che si verifichi la ricezione di 3 ACK duplicati o lo scadere del Timeout. Analizzando i motivi che portano al verificarsi dei due eventi, si può notare che è più probabile che ci si trovi di fronte ad una congestione della rete quando scade un timeout, mentre la ricezione di più ACK duplicati indica probabilmente che un pacchetto o è stato ricevuto corrotto o non è stato ricevuto affatto, a causa del rumore nel canale. Si ricordi, infatti, che il destinatario invia ACK quando riceve i pacchetti dal mittente, il che porta a pensare ad una rete non congestionata. Viceversa lo scadere del timeout indica che al ricevitore non è arrivato nulla per un lungo periodo, ossia proprio quello che si verifica quando uno o più nodi nella rete hanno liste di attesa nelle code molto lunghe. Dunque la versione Reno del TCP se riceve 3 ACK duplicati porta $Ssthresh = \frac{Cwnd}{2}$ e ritrasmette il pacchetto perso. Poi imposta la finestra di congestione a $Cwnd = \frac{Cwnd}{2} + 3MSS$.

A questo punto la Cwnd viene aumentata di 1 MSS per ogni ulteriore ACK duplicato che viene ricevuto, fino a che non arriva l'ACK non duplicato che riscontra tutti i dati spediti successivamente al pacchetto perso. Così Cwnd viene di nuovo abbassata al valore

di ssthresh e anche il ritmo viene rallentato, portandolo al tipico incremento lineare di Congestion Avoidance.

Questo metodo risulta molto efficiente nel caso in cui un singolo segmento venga perso o arrivi corrotto dal rumore: infatti, questo viene immediatamente ritrasmesso all'arrivo del terzo ACK duplicato. Poi, impostando $Cwnd = \frac{Cwnd}{2} + 3MSS$ e aumentando il ritmo di 1 MSS per ogni ACK duplicato, si permette di ritrasmettere velocemente tutti i dati già inviati ma successivi al pacchetto perso e arrivare così in breve tempo a spedire un nuovo pacchetto. Se l'evento perdita è invece lo scadere di un timeout, il TCP Reno lo interpreta come una probabile congestione e si comporta come il TCP Tahoe: pone $Ssthresh = \frac{Cwnd}{2}$ e $Cwnd = 1MSS$ e si porta in fase Slow Start. Il sistema di ritrasmissione del TCP Reno risulta particolarmente valido se viene perso un singolo pacchetto, ma può essere particolarmente inefficiente nel caso in cui si perdano o arrivino corrotti due o più segmenti della stessa finestra di trasmissione, in particolare nelle implementazioni in cui i pacchetti ricevuti fuori ordine vengono salvati nel buffer del ricevitore. Per capire cosa accade si valuti il seguente esempio (Figura 3.7)

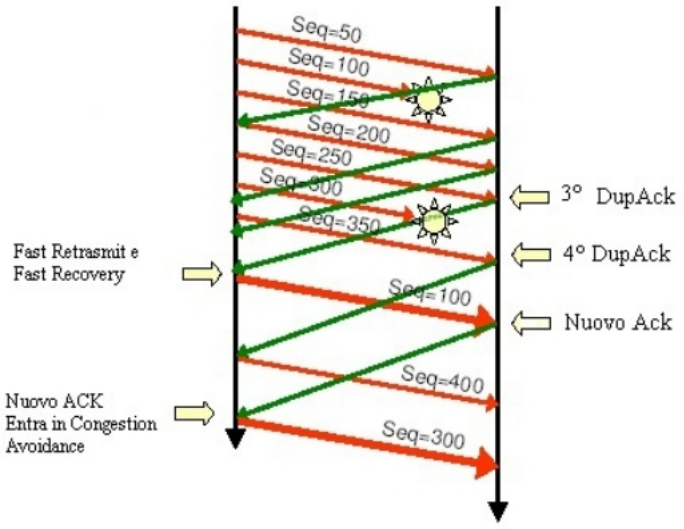

Figura 3.7: Perdite multiple in TCP Reno

Il mittente invia una serie di pacchetti verso il destinatatio, ma i pacchetti con sequence number 100 e 300 vengono persi; al terzo ACK duplicato per il pacchetto (100), il TCP Reno avvia la Fast Retransmit e la Fast Recovery, ritrasmettendo il pacchetto

100, portando $Cwnd = \frac{Cwnd}{2} + 3MSS$ e aumentando il ritmo di 1 MSS per ogni ulteriore ACK duplicato. Quando il mittente riceve il primo ACK non duplicato (che in questo caso riscontra fino al pacchetto 250), entra in Congestion Avoidance, rallentando il ritmo di invio $Cwnd = Cwnd + \frac{1}{Cwnd}$. Con il ritmo di invio della Congestion Avoidance passerà una notevole quantità di tempo prima che siano ricevuti gli altri tre DUPACK che segnaleranno al mittente la perdita del pacchetto 300, causando spesso lo scadere del timeout dello stesso. Per risolvere questo problema furono apportate delle modifiche che portarono allo sviluppo di una nuova versione chiamata TCP NewReno.

3.1.6.4 TCP New Reno

Il TCP NewReno introduce alcune modifiche al TCP Reno per potenziare le prestazioni: tali modifiche riguardano l'impostazione del valore di Ssthresh e il miglioramento della reazione del protocollo nel caso in cui vengano persi due o più segmenti all'interno della stessa finestra. Per quanto riguarda l'impostazione del valore di Ssthresh, questo viene calcolato quando viene instaurata una nuova connessione utilizzando l'equivalente in bytes del prodotto $ritardo * ampiezza$ della banda. Per valutare l'ampiezza della banda disponibile viene utilizzato un algoritmo chiamato Packet Pair che si basa sugli intertempi di arrivo al mittente degli ACK. Come abbiamo già visto, se due o più segmenti trasmessi nella stessa finestra vengono persi, il TCP Reno si accorgerà rapidamente solo della mancanza del primo, a mentre noterà la mancanza degli altri solo allo scadere del Timeout. Per risolvere a questo problema, il TCP New Reno introduce il concetto di ACK parziale, ossia un ACK che riscontra alcuni, ma non tutti, i pacchetti inviati prima di entrare in fase di Fast Recovery. Il TCP New Reno si comporta esattamente come il TCP Reno quando riceve 3 ACK duplicati, portando $Ssthresh = \frac{Cwnd}{2}$ e $Cwnd = \frac{Cwnd}{2} + 3MSS$.

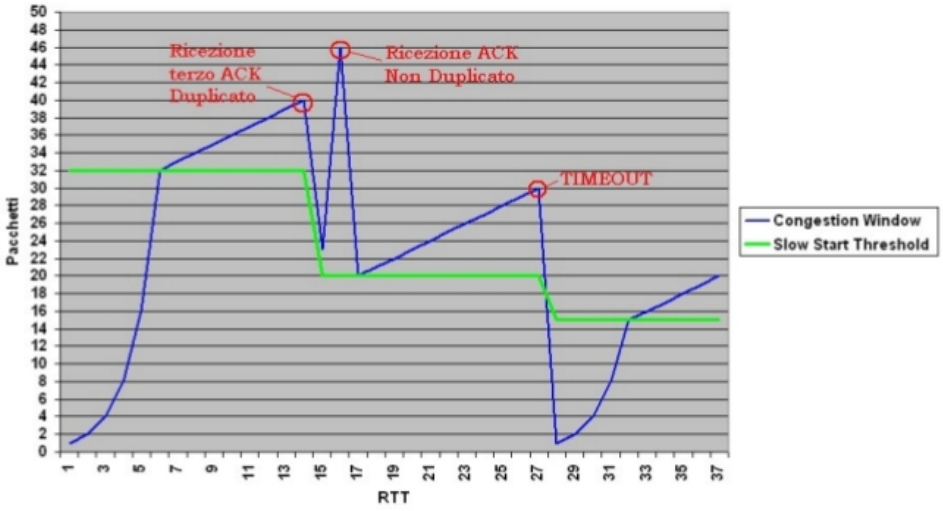

Figura 3.8: TCP Reno e NewReno

Successivamente si comporta ancora come il TCP Reno se il primo ACK non duplicato che riceve riscontra tutti i pacchetti inviati prima di entrare in Fast Recovery. Altrimenti, se riceve un ACK parziale che riscontra solo alcuni pacchetti inviati prima di entrare in Fast Recovery, il TCP New Reno interpreta questo fatto come una ulteriore perdita e ritrasmette immediatamente il pacchetto successivo a quello riscontrato dall'ACK parziale rimanendo nella fase di Fast Recovery. Come indicato nell'esempio precedente (sezione TCP Reno), l'ACK parziale riscontra solo i pacchetti (200) e (250): il TCP New Reno ritrasmette allora subito il pacchetto (300) e resta nella fase di Fast Recovery aumentando di 1 MSS Cwnd per ogni ulteriore ACK duplicato relativo al secondo pacchetto perso. Il TCP New Reno entrerà in Congestion Avoidance solo quando riceverà l'ACK che riscontrerà tutti i pacchetti inviati prima di entrare in a a Fast Recovery.

3.1.6.5 TCP Westwood

L'idea principale che caratterizza il TCP Westwood (Figura 3.9) è di ottenere una stima della banda disponibile della connessione, misurando la velocità con cui arrivano i pacchetti di ACK. Per banda disponibile, si vuole indicare la velocità con cui i dati vengono attualmente trasferiti. Questo è un compito alquanto semplice rispetto alla medesima misurazione nel momento della creazione della connessione. La stima della

banda viene poi usata per una corretta regolazione della congestion window (Cwnd) e slow-start threeshold (ssthresh) dopo un episodio di congestione:

1. Quando 3 ACK duplicati sono stati ricevuti dal mittente:
   $ssthresh = \frac{(BW*RTTmin)}{segsize}$ e $Cwnd = ssthresh$;
2. Quando scade il timeout:
   $ssthresh = \frac{(BW*RTTmin)}{segsize}$ e $Cwnd = 1$;
3. Quando gli ACK sono ricevuti con successo: Cwnd aumenta come nell'algoritmo Reno.

La stabilità della rete non richiede che il flusso dei dati venga ridotto in presenza di un singolo indicatore di congestione. In particolare, per prevenire la congestione, è necessario che i flussi di dati abbiano un qualche meccanismo di controllo per non aumentare il traffico di pacchetti in presenza di un crescente drop rate degli stessi. Con il TCP Westwood, il numero di pacchetti inviati viene ridotto in conseguenza di misurazione sulla banda disponibile quando avviene un fenomeno di congestione. Quindi, in presenza di gravi congestioni della rete, questa riduzione potrebbe essere più drastica del modello Reno, mentre con rallentamenti minori anch'essa risulterà meno marcata. Questa caratteristica può sicuramente migliorare la stabilità e l'utilizzo della rete rispetto al modello Reno.

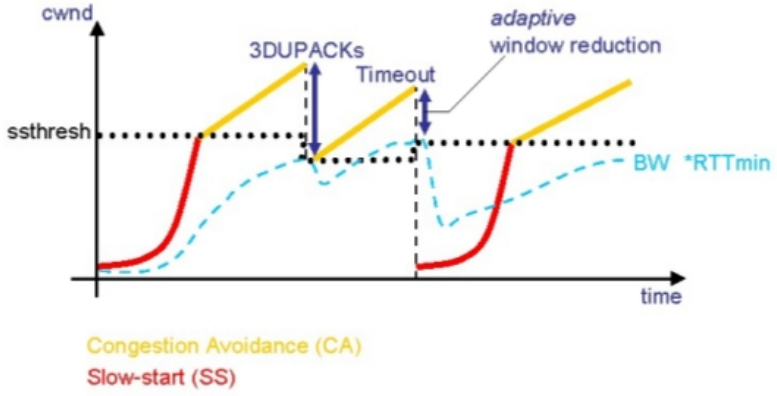

Figura 3.9: TCP Westwood

3.1.6.6 TCP Vegas

La versione Vegas del TCP si basa su un principio nettamente diverso da quelle analizzate finora: si propone infatti non di reagire ad un evento di perdita ma di rallentare il ritmo di invio dei dati ancor prima che se ne verifichi uno. Cerca, insomma, di evitare che si crei una congestione della rete piuttosto che prendere provvedimenti quando la congestione è già avvenuta. E' dunque necessario che il mittente conosca in ogni istante lo stato della rete, e per fare questo osserva i cambiamenti degli RTT dei segmenti già inviati. Ovviamente, se il mittente si accorge che RTT diventa particolarmente ampio, deduce che la rete si sta per congestionare e diminuisce il ritmo di invio. Viceversa, se RTT risulta breve vuol dire che la rete non ha problemi e la finestra viene aumentata. In particolare l'algoritmo di controllo della congestione del TCP Vegas è così concepito:

$$\begin{cases} Cwnd = Cwnd + 1 & se\ diff < \frac{\beta}{BaseRTT} \\ Cwnd = Cwnd & se\ \frac{\alpha}{BaseRTT} < diff < \frac{\beta}{BaseRTT} \\ Cwnd = Cwnd - 1 & se\ diff > \frac{\beta}{BaseRTT} \end{cases}$$

con

$$diff = \frac{Cwnd}{BaseRTT} - \frac{Cwnd}{RTT}$$

dove Base RTT è il più piccolo valore di RTT osservato e RTT è quello attuale, mentre α e β sono due costanti.

3.1.6.7 TCP Sack

E' purtroppo una realtà che la perdita di segmenti non contigui sia una grande problematica. La soluzione di estendere l'algoritmo della finestra scorrevole di base e del TCP con un'opzione addizionale, che permette ai dispositivi di mandare ACK per singoli segmenti non contigui. Questa opzione, introdotta in RFC 1072 e raffinata in RFC 2018 è chiamata TCP selective acknowledgement (SACK) [12, 13, 14]. Per poter utilizzare SACK, i due dispositivi partecipanti alla connessione devono supportare questa opzione, attivando Selective Acknoledgement Permitted nel segmento SYN utilizzato per stabilire la connessione. Fatto questo, entrambi i dispositivi potranno aggiungere l'opzione Selective Acknowledgement ai segmenti TCP. Ogni dispositivo modifica la sua coda di

ritrasmissione in modo tale che i segmenti includano un flag di Selective Acknowledgement. Se questo flag viene impostato a 1, vuol dire che il mittente ha ricevuto un SACK per il segmento: nel caso vengano persi pacchetti precedenti nella finestra di invio, il segmento non verrà reinstradato.

Si consideri ora l'esempio (Figura 3.10) di come viene gestita la perdita di un segmento:

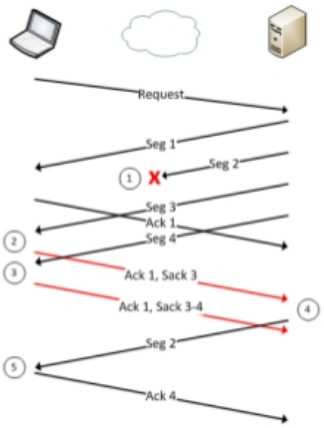

Figura 3.10: Gestione delle perdite in TCP Sack

1. il client effettua una richiesta e il server risponde inviando il segmento (1) il (2) e il (3), ma il frame (2) viene perso;

2. il client realizza che è stato perso un segmento tra il primo e il terzo. Manda un ACK duplicato per il segmento 1 e include l'opzione SACK indicando di aver ricevuto il segmento 3;

3. il client riceve il segmento 4 e manda un altro ACK duplicato per il segmento 1. Questa volta espande l'opzione SACK, mostrando che ha ricevuto con successo i segmenti 3 e 4;

4. il server riceve gli ACK duplicati del client per il segmento 1 e 3. Da questo deduce che è stato perso il segmento 2 quindi viene ritrasmesso. Il prossimo e SACK ricevuto dal server indica che il client ha ricevuto anche il segmento 4, quindi non è necessaria un'altra ritrasmissione;

5. il client riceve il segmento 2 e manda un ACK per indicare che ha ricevuto tutti i dati.

3.1.6.8 TCP Cubic

Cubic [14] è l'attuale versione di TCP implementata su sistemi linux e fornisce un e ottimo servizio in presenza di grandi reti dove possono sussistere alternativamente stati di grave congestione a periodi di elevata banda disponibile. Prima di approfondire il discorso su Cubic è necessario fornire alcune informazioni sul suo predecessore, BIC e sul suo modo di gestire la grandezza della finestra di trasmissione. BIC usa un algoritmo di ricerca binaria (binary search algorithms), ponendo la dimenzione della finestra tra i valori Wmax e Wmin che rappresentano rispettivamente la dimenzione della finestra quando il TCP ha rilevato la perdita di un pacchetto e la dimenzione della finestrea quando non si è verificata nessuna perdita per un RTT; la ricerca di questo punto medio tra Wmax e Wmin ha intuitivamente senso in quanto, se si è rilevata la perdita di un pacchetto nel tempo in cui la finestra è tra Wmax e Wmin, questo significa che la capacità della rete è pari ad un valore tra Wmax e Wmin, a meno di cambiamenti repentini della rete stessa. Dopo che la dimenzione della finestra è stata posta sul punto medio, si possono verificare due situazioni:

- non si hanno perdite di pacchetti, il che implica che la rete può gestire maggior traffico; allora, il punto medio dovrebbe essere il nuovo Wmin e bisognerebbe far ripartire la ricerca binaria tra il nuovo minimo e Wmax; tuttavia, aumentare la finestra da Wmin al valore intermedio (e quindi al nuovo Wmin) può risultare gravoso in un solo RTT; così, nel caso in cui la distanza tra il vecchio Wmin e punto medio, rappresentante il nuovo Wmin, sia maggiore di una costante Smax, BIC incrementa la finestra Cwnd solo di quest'ultimo valore, avendo una crescita lineare;

- si hanno perdite di pacchetti, ma allora la rete non è in grado di gestire il traffico generato dai nodi; di conseguenza, il punto medio divenda il nuovo Wmax e viene ripetuta la ricerca binaria tra Wmin e il nuovo massimo.

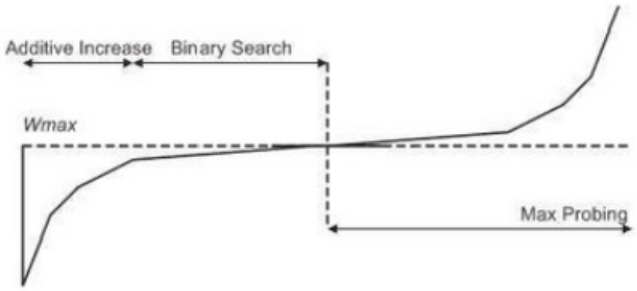

Figura 3.11: TCP Bic

Se la grandezza della finestra cresce oltre il massimo, ne deve essere trovato uno nuovo; BIC entra nella fase chiamata *Max Probing* nella quale usa una funzione di crescita speculare a quella precedente per trovare il valore intermedio, in cui l'incremento parte lento e lineare per poi continuare, nel caso non si trovi un nuovo massimo, in maniera esponenziale (Figura 3.11).

Tuttavia BIC può risultare molto aggressivo per un TCP, specialmente con RTT molto bassi o per reti con ridotte prestazioni. CUBIC si presenta come variante a BIC utilizzando una funzione di crescita della finestra leggermente diversa: la finestra di congestione viene determinata dalla funzione

$$W_{cubic} = C * (t-K)^3 + Wmax$$

dove C è detto scaling factor, t è il tempo intercorso dall'ultima riduzione della finestra, $K = Wmax * \sqrt[3]{\frac{\beta}{C}}$ e β è un fattore moltiplicativo costante che viene applicato alla riduzione della finestra durante un evento di perdita.

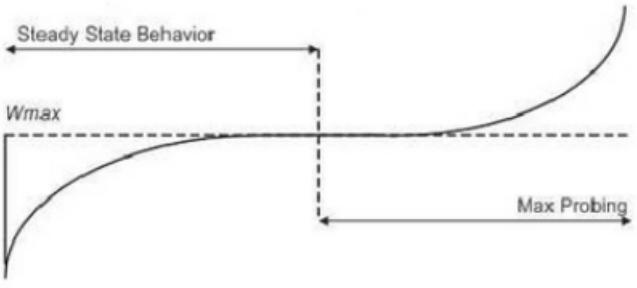

Figura 3.12: TCP Cubic

La Figura 3.12 mostra il comportamento di CUBIC: la finestra cresce molto velocemente dopo un evento di perdita, finchè non si avvicina a Wmax, dove, in quest'intorno, rallenta la crescita fino ad avvicinarsi quasi allo zero. Sopra questo valore, CUBIC inizia la fase di probing, dove la finestra inizia a crescere prima lentamente per poi incrementarne il ritmo allontanandosi da Wmax; questa crescita lenta nell'intorno di Wmax assicura la stabilità del protocollo, mentre quella veloce dopo questo valore ne assicura la scalabilità.

Inoltre, la funzione cubica assicura l'equità intraprotocollo tra diversi flussi di dati; per osservare questo si consideri ad esempio due flussi di dati che vogliono attraversare lo stesso percorso end-to-end. I due flussi convergeranno ad un equo utilizzo della banda, dato che arriveranno ad utilizzare lo stesso fattore moltiplicativo β. Quindi il flusso con la Wmax maggiore si ridurrà maggiormente in caso di perdita, mentre aumenterà più lentamente se la banda è disponibile; i due flussi eventualmente, convergeranno allo stesso utilizzo di banda.

La funzione offre anche una certa equità anche rispetto all'RTT, anche perchè la crescita della finestra non è dominata da quest'ultimo ma dal fattore t.

3.1.6.9 TCP Veno

La versione Veno [16] di TCP è stata sviluppata al fine di una migliore qualità di trasmissione su reti wireless. In questi scenari, la difficoltà maggiore sta nel riconoscere il motivo della perdita dei pacchetti, i quali possono essere persi non solo per congestione nella comunicazione, ma semplicemente a causa di rumori e interruzioni di connessione attribuibile perlopiù a cause ambientali. Sarebbe necessario distinguere tra perdite dovute a congestione e quelle relative a interferenza, cui da ora ci riferiremo per mezzo del termine *random losses*. Veno nasce come una unione di Reno e Vegas per una corretta gestione delle random losses. Reno tratterebbe queste perdite come una manifestazione della congestione della rete con la conseguente riduzione della velocità di trasmissione dei dati, degradando inutilmente le prestazioni della comunicazione. Per arginare questo problema, si è pensato di utilizzare il sistema di Vegas, in cui viene stimata la possibile congestione della rete tramite un calcolo degli RTT sui pacchetti già inviati. In particolare, all'aumentare del valore degli RTT, si presume un possibile congestionamento della rete e, quindi, Vegas reagisce diminuendo il ritmo con cui invia pacchetti. Appli-

cando questo sistema a Reno è possibile capire se una perdita è dovuta a congestione o meno, agendo di conseguenza. Nel caso di perdita, se il calcolo degli RTT di Vegas non prevede una rete congestionata, è possibile reagire e ad essa interpretandola come random losses. Veno utilizza un sistema simile a Vegas per la previsione dello stato della rete, associandolo a una versione Reno leggermente modificata. Nel caso di ricezione di 3 ACK duplicati, Reno modifica ssthresh impostandolo a $\frac{cwnd}{2}$ per poi associare a Cwnd il valore di $ssthresh + 3$. Per ogni ulteriore ACK duplicato ricevuto aumenta il valore di Cwnd di 1; nel momento in cui arriva l'ACK corrispondente Reno imposta Cwnd al valore di ssthresh per ripartire con la trasmissione corretta. Veno modifica solo il calcolo iniziale di ssthresh nel caso in cui il sistema, da previsione, non risulti congestionato. In questo caso invece di portare ssthresh a $\frac{cwnd}{2}$ viene invece ridotto a $cwnd * \gamma$, dove γ è tipicamente un valore maggiore di $\frac{1}{2}$. Così facendo si ha una riduzione minore della finestra.

Veno quindi non modifica Reno nel caso in cui il pacchetto risulta perso per scadenza di timer, ma solamente su ACK duplicati agendo sul calcolo del nuovo ssthresh durante le fasi di fast retransmit e fast recovery.

3.2 TCP in Omnet++/Inet

Il modulo TCP implementa il protocollo TCP nella sua totalità; è connesso con il livello IP con due tipologie di gates:

- ipIn e ipOut, che rappresentano rispettivamente il gate di ingresso e di uscita per il livello IPv4.
- ipv6In e ipv6Out, che rappresentano rispettivamente il gate di ingresso e di uscita per il livello IPv6.

Inoltre, è connesso al livello applicazione attraverso l'array di gates appIn[] e appOut[]; il modulo TCP può servire svariate applicazioni contemporaneamente: il modulo dell'applicazione k-esima è connesso al modulo TCP attraverso le porte appIn[k] e appOut[k].

Applicazione e modulo TCP comunicano tra loro attraverso lo scambio di messaggi; l'applicazione invia al modulo TCP messaggi di tipo TCPCommand, come ad esempio:

- TCP_C_OPEN_ACTIVE: per aprire una connessione attiva, utilizzata nel caso in cui l'applicazione è un client;
- TCP_C_OPEN_PASSIVE: per aprire una connessione passiva, utilizzata principalmente nel caso in cui l'applicazione è un server;
- TCP_C_SEND: per inviare dei dati;
- TCP_C_CLOSE: non ci sono più dati da inviare e quindi è possibile chiudere la connessione;
- TCP_C_ABORT: per abortire la connessione;
- TCP_C_STATUS: per richiedere lo stato in cui si trova il TCP.

Ognuno di questi messaggi, che rappresentano essenzialmente dei comandi che l'applicazione "impone" al livello TCP, contiene al suo interno delle informazione di controllo e un identificatore univoco, *connId*, che individuano la connessione con l'applicazione. Il modulo TCP risponde ai comandi inviati dall'applicazione attraverso messaggi di tipo TCPStatus, come ad esempio:

- TCP_I_ESTABLISHED: la connessione è stata stabilita;
- TCP_I_CONNECTION_REFUSED: la connessione è stata rifiutata;
- TCP_I_CONNECTION_RESET: la connessione è stata resettata;
- TCP_I_TIME_OUT: il timer per stabilire la connessione è scaduto o è stato raggiunto il massimo numero di ritrasmissioni possibili;
- TCP_I_DATA: pacchetti dati;
- TCP_I_URGENT_DATA: pacchetti dati urgenti;
- TCP_I_PEER_CLOSED: pacchetto FIN ricevuto dall'host remoto;
- TCP_I_CLOSED: connessione chiusa normalmente;
- TCP_I_STATUS: informazioni sullo stato del TCP.

Anche questi messaggi, che rappresentano essenzialmente lo stato in cui si trovano il modulo e la connessione TCP, vengono inviati all'applicazione insieme ad alcune informazioni di controllo.

A questo punto, è evidente come il TCP sia gestito e implementato attraverso una macchina a stati finita, in cui la classe TCPConnection gestisce e implementa essa stessa la macchina a stati, mentre la classe TCPAlgorithm implementa i vari algoritmi di controllo della congestione viste in precedenza. In particolare, la classe TCPConnection gestisce le transizioni da uno stato a un altro, date dal verificarsi di un evento, e memorizza lo stato della macchina (TCB) contenente informazioni essenziali quali:

- il campo active per indicare se la connessione è attiva o passiva;
- il campo fork utilizzato quando la connessione è passiva, per indicare se è stata fatta una fork della connessione;
- il campo snd_mss rappresentante il maximum segment size;
- i campi snd_una, snd_nxt, snd_max rappresentanti rispettivamente il numero di sequenza del primo pacchetto di cui non è stato ricevuto ancora ack, il numero di sequenza del prossimo pacchetto da inviare e il massimo numero di sequenza dei pacchetti inviati (questo perchè snd_nxt può essere portato indietro in caso di ritrasmissione);
- il campo snd_wnd per indicare la finestra di invio;
- il campo snd_up per indicare il puntatore al pacchetto urgente;
- il campo snd_iss contenente il valore per l'initial sequence number;
- i campi snd_wl1 e snd_wl2 contenenti rispettivamente il numero di sequenza usato per inviare l'ultima finestra aggiornata e il numero di ack usato per inviare l'ultima finestra aggiornata;
- i campi rcv_nxt, rcv_max e rcv_up rappresentanti rispettivamente il numero di sequenza del prossimo pacchetto da ricevere, il valore della finestra di ricezione e il numero di sequenza di un pacchetto urgente ricevuto;
- il campo irs contenente il valore per l'initial receive sequence number;

- il campo dupack contentente il numero di ack duplicati ricevuti per un pacchetto;
- i campi syn_rexmit_count, sys_rexmit_timeout, fin_ack_rcvd, send_fin, snd_fin_seq, fin_rcvd e rcv_fin_seq utilizzati per gestire le fasi di handshake durante l'instaurazione della connessione.

Questi campi permettono di implementare una versione base del protocollo TCP, a cui ne vanno aggiunti eventualmente degli altri nel caso in cui i vari algoritmi di controllo della congestione ne abbiano bisogno.

Infine, ci sono ancora due "oggetti" che entrano in gioco e vengono gestiti anch'essi da TCPConnection: questi sono le istanze di TCPSendQueue e TCPReceiveQueue, rappresentanti rispettivamente i buffer di invio e di ricezione dei messaggi.

Infine, si introduce la classe TCPSocket, più di livello applicazione che di livello trasporto (e quindi TCP), ma che permette una gestione semplice della connessione TCP; in particolare si ha uno o più oggetti rappresentante il socket TCP nell'applicazione e possono essere usati i suoi metodi per aprire o chiudere una connessione TCP, o per inviare e ricevere messaggi verso/dal modulo TCP.

Le operazioni principali svolte nel modulo TCP sono:

1. Instaurazione della connessione
2. Invio (o Reinvio) dei pacchetti;
3. Ricezione dei pacchetti;
4. Chiusura della connessione.

3.2.1 Instaurazione della connessione

L'instaurazione della connessione può essere fatta in due maniere differenti:

- l'applicazione apre una porta locale per una connessione in ingresso inviando un messaggio TCP_C_PASSIVE_OPEN al modulo TCP, contenente l'indirizzo e la porta locali. L'applicazione può, inoltre, specificare se gestire una sola connessione TCP alla volta o più connessioni multiple: se il campo fork è settato a true, allora viene creato un thread con un nuovo id e, mentre il thread figlio gestisce la connessione che si sta creando, il thread padre si pone in uno stato di listening per

accettare altre eventuali connessioni; se il campo fork è settato a false, allora viene gestita una sola connessione alla volta.

- l'applicazione apre una connessione inviando un messaggio TCP_C_ACTIVE_OPEN al modulo TCP; questo crea un oggetto TCPConnection e invia un segmento SYN; a questo punto ha inizio la fase di handshake a 3 vie. Dopo 75s, se la fase di instaurazione della connessione non è terminata, scatta il timeout, viene inviato TCP_I_TIMEOUT all'applicazione e viene chiusa la connessione. Se, invece, la connessione viene instaurata, TCP invia all'applicazione TCP_I_ESTABLISHED, insieme all'indirizzo e porta remoti; altrimenti, si può verificare il caso in cui la connessione venga rifiutata dall'host remoto, per cui il TCP invia TCP_I_CONNECTION_REFUSED all'applicazione.

3.2.2 Invio dei pacchetti

L'applicazione invia i pacchetti al modulo TCP, insieme al comando TCP_C_SEND; il modulo TCP aggiunge il pacchetto nella coda di invio, controlla se il pacchetto può essere spedito oppure no, lo divide in frammenti (nel caso in cui la dimensione totale del pacchetto sia maggiore del maximum segment size) e invia questi ultimi, sottoforma di TCPSegment, al modulo IP.

Esistono tre tipologie differenti di code di invio, a seconda delle tre modalità di trasferimento dei dati:

- TCP_TRANSFER_BYTECOUNT: il segmento TCP ha payload nullo;
- TCP_TRANSFER_BYTESTREAM: l'applicazione invia al modulo TCP un array di bytes; il modulo divide l'array in pezzi di dimensioni pari alla MSS e li trasmette nel campo payload del segmento TCP;
- TCP_TRANSFER_OBJECT: l'applicazione passa al modulo TCP un oggetto di tipo cMessage; il TCP crea i segmenti necessari, in base alla lunghezza massima consentita, e invia l'oggetto come payload del primo segmento.

Per quanto riguarda le operazioni di ritrasmissione del pacchetto, di controllo di flusso e di congestione, queste vengono gestite dalla classe TCPAlgorithm, in base all'algoritmo scelto. Gli algoritmi attualmente sviluppati in INET sono TCP Tahoe, TCP Reno, TCP

New Reno e un altro algoritmo base, in cui non viene considerato e risolto il problema della congestione di rete.

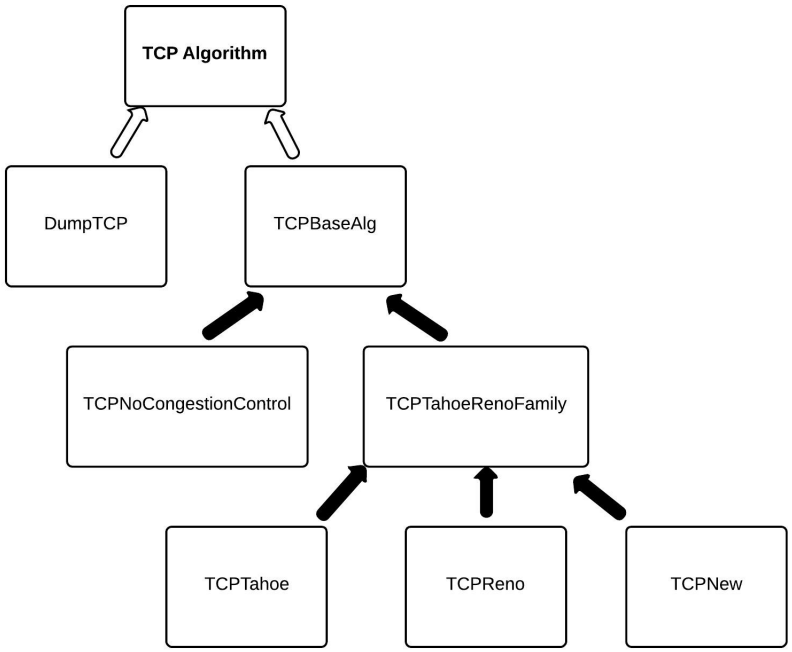

Figura 3.13: Algoritmi TCP sviluppati in INET

3.2.3 Ricezione dei pacchetti

La connessione TCP memorizza i segmenti in arrivo nella coda di ricezione; come per il lato trasmittente, anche per il lato ricevente esistono tre tipologie di code differenti, in base al tipo di trasferimento dei dati fatto dal trasmittente. Durante la fase di ricezione, se il buffer è pieno o la dimensione del pacchetto è maggiore dello spazio libero sul buffer, il messaggio viene scartato; inoltre, se il pacchetto ricevuto ha il sequence number successivo all'ultimo ricevuto (e quindi è il pacchetto che si stava aspettando), questo viene subito inoltrato all'applicazione incapsulato nel messaggio TCP_I_DATA, altrimenti viene bufferizzato in attesa di essere ricomposto e solo successivamente passato all'applicazione.

3.2.4 Chiusura della connessione

Come per l'instaurazione, anche la chiusura della connessione può essere fatta in due modalità differenti. Si consideri l'esempio di una connessione TCP tra client e server:

- l'applicazione lato client decide di chiudere la connessione, in quanto non ha più dati da inviare; comanda, attraverso il TCP_C_CLOSE, il modulo TCP di cessare la connessione. Il modulo TCP tenta di inviare tutti i pacchetti all'interno del suo buffer di trasmissione e solo successivamente inizia la fase di chiusura della connessione, ovvero le operazioni viste nell'algoritmo di handshake a 3 vie (FIN+ACK);
- il server decide di chiudere la connessione; il modulo TCP lato client riceve un segmento FIN e invia un messaggio TCP_I_PEER_CLOSED alla applicazione client, contenente l'identificatore della connessione e altre informazioni di controllo.

Durante la comunicazione, tuttavia, si possono verificare degli eventi per cui la connessione può essere resettata oppure addirittura abortita.

Capitolo 4

Progettazione e implementazione

In questo capitolo si entra nel vivo della tesi, andando a esplicare tutta la progettazione e l'implementazione della modifica, dall'idea iniziale alla realizzazione vera e propria. La modifica consiste nell'applicare il sistema RWMA anche al protocollo TCP, cosa che fino ad ora riguardava solo il protocollo UDP; la motivazione principale per cui si vuole effettuare la modifica al protocollo è la possibilità di perdite dovute a cause esterne durante le comunicazioni wireless. In questo scenario, fattori ambientali, rumori e ostacoli possono influire sulla comunicazione con l'access point, riducendo effettivamente la QoS. Test svolti mostrano che una semplice comunicazione in cui l'host mittente si muove in un ambiente domestico, può portare a perdite di pacchetti, senza che l'interfaccia di rete smetta di funzionare a causa dell'eccessiva lontananza dall'AP; si noti che allontanandosi dall'AP è possibile che si presentino delle perdite senza che comunque l'interfaccia di rete perda completamente la connessione. Scopo della nostra modifica sarà la gestione anticipata di queste perdite, riscontrandole prima dello scadere del timeout di TCP.

4.1 Progettazione

Per la realizzazione della modifica, il principio che si vuole sfruttare è il passaggio di informazioni sull'errore di invio da livello MAC a quelli trasporto. Modificando un sistema con questo ulteriore controllo, sarebbe teoricamente possibile aumentare la QoS della trasmissione su interfaccia wireless; riuscendo, quindi, a capire se il pacchetto è arrivato con successo all'AP a cui il nodo mobile è connesso o se quest'ultimo l'ha scartato per

problemi di congestione, è possibile una gestione più veloce dell'errore. La struttura base del codice sarebbe quella simile a quella per UDP vista in precedenza, con la principale differenza che, mentre con UDP il modulo ULB di RWMA si trova a livello applicazione e quindi è in questo strato che vengono effettuate tutte le decisioni riguardanti le ritrasmissioni del pacchetto in caso di non arrivo al primo hop, con TCP, il modulo ULB viene integrato all'interno del modulo TCP, lasciando l'applicazione TCP ignara su cosa sta accadendo a livello transporto e lasciando a quest'ultimo strato il compito di ritrasmettere il pacchetto. A questo, va aggiunta la difficoltà della ritrasmissione basata sui timer, come visto per le versioni standard del protocollo TCP.

Nelle sezioni successive verranno descritte dettagliatamente le modifiche apportate al codice, le classi di INET che sono state introdotto o modificate in modo da non variare pesantemente le strutture già esistenti e allontanarsi dalla specifica imposta dallo standard per il TCP.

4.2 Il "nuovo" proxy server

In questo nodo non sono state fatte modifiche sostanziali, ma viene introdotta un'applicazione TCP, fornita da INET.

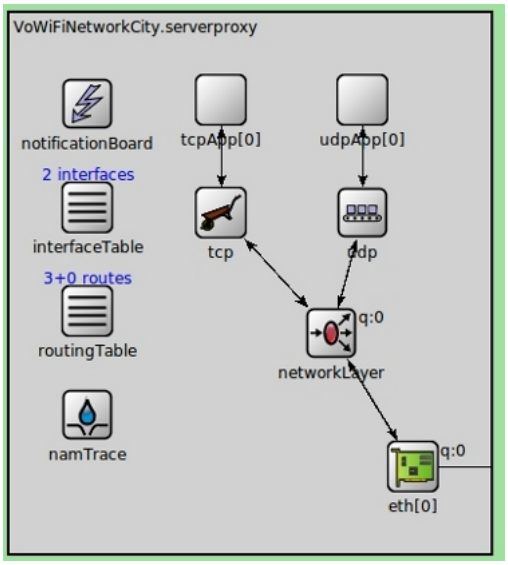

Figura 4.1: Il nuovo proxy server

In particolare, la classe utilizzata è la TCPServerHostApp (.cc, .h) che realizza una semplice applicazione con compiti principali quelli di ricevere i pacchetti inviati dal mittente (o client) e rispondere attraverso un ACK, nel caso in cui il pacchetto arrivato sia quello atteso e privo di errori.

Le modifiche sostanziali sono state fatte nell'altro end-system della comunicazione, il nodo mobile, che verrà esaminato attentamente dal livello più alto a quello più basso.

4.3 Il "nuovo" nodo mobile

Dopo la modifica apportata, il nodo mobile si presenta come in Figura 4.2

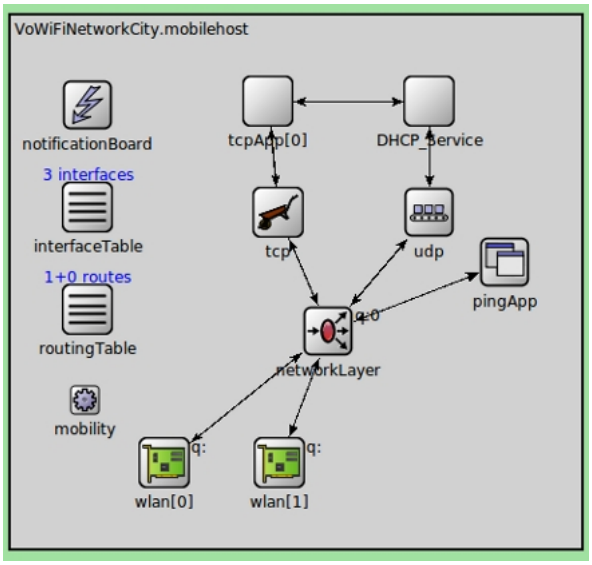

Figura 4.2: Il nuovo nodo mobile

Prima di analizzare il ramo TCP, viene affrontata la modifica riguardante il DHCP.

4.3.1 DHCP_Service

Come detto nel capitolo 2, il protocollo DHCP è "immerso" all'interno dell'applicazione UDP (implementata dalle classi ULBRWMA e UDPAppBasicAppForMultipleNics); nel nuovo nodo mobile, invece, è stato creato un modulo indipendente che realizza solo ed esclusivamente il DHCP, chiamato DHCP_Service; quest'ultimo è collegato al livello UDP,

e quindi comunicherà attraverso il protocollo UDP, ed è collegato all'applicazione TCP attraverso l'array di gate tcpAppIn[] e tcpAppOut[].
Le operazioni svolte da questo modulo sono le stesse che venivano effettuate dall'applicazione UDP, ovvero nel momento in cui il DHCP_Service riceve dal monitor un messaggio di tipo ReconfNot, questo significa che una scheda di rete si sta associando o dissociando da un AP; se lo stato dell'interfaccia, passato al modulo DHCP_Service, è WORKING, allora la scheda di rete si sta associando all'AP e quindi viene attivata la procedura DHCP (vista in precedenza).

Rispetto alla gestione precedente, sono state introdotte queste nuove operazioni:

- nel momento in cui il modulo DHCP_Service riceve il DHCPREPLY, questo comunica all'applicazione TCP il nuovo indirizzo IP e l'idetificatore dell'interfaccia che si è configurata, attraverso un messaggio di tipo IPComunication. Oltre ai valori detti in precedenza, il messaggio comunicato all'applicazione TCP contiene anche l'identificatore del client, l'indirizzo MAC della scheda di rete, il nuovo valore che ha assunto l'interfaccia (ovvero il valore di WORKING), il valore e l'indirizzo IP che l'interfaccia aveva prima di associarsi all'AP;

- nel momento in cui il modulo DHCP_Service riceve un messaggio di tipo ReconfNot in cui lo stato della scheda di rete è DISABLED, questo comunica all'applicazione TCP che l'interfaccia si è dissociata, attraverso un messaggio di tipo IPComunication; quest'ultimo contiene l'identificatore del client, l'dentificatore e l'indirizzo MAC della scheda di rete, il nuovo valore che ha assunto l'interfaccia (ovvero il valore di DISABLED), il nuovo indirizzo IP che verrà posto a NULL, il valore e l'indirizzo IP dell'interfaccia, prima di essersi dissociata dall'AP a cui era connessa.

Sono state introdotte queste operazioni per far in modo che anche l'applicazione TCP sia a conoscenza di quale interfaccia è attiva in ogni istante di tempo e con quale indirizzo IP la scheda di rete sta lavorando.

4.3.2 Applicazione TCP

L'applicazione TCP è realizzata dalla classe **TCPAppRWMA** (.cc, .h, .ned) ed esegue queste principali operazioni:

- durante la fase di inizializzazione viene creato un socket della classe TCPSocket ed effettuata una socket.bind sulla porta locale, senza specificare l'indirizzo IP locale utilizzato, in quanto questo ultimo ci verrà comunicato dal messaggio IPComunication inviato dal DHCP_Service;

- nel momento in cui l'applicazione TCP riceve il pacchetto IPComunication in cui lo stato dell'interfaccia è WORKING, allora viene settato l'indirizzo IP locale e richiamata la funzione connect sul socket configurato, specificando indirizzo IP e porta remota. E' il socket che si prende carico di instaurare la connessione, facendo partire le operazioni di handshake. Nel momento in cui la connessione è istaurata, l'applicazione TCP inizia ad inviare i pacchetti.
  Nel momento in cui, invece, l'applicazione TCP riceve il pacchetto IPComunication in cui lo stato dell'interfaccia che sta utilizzando è DISABLED, allora viene subito richiamata la funzione close sul socket configurato e viene terminata la simulazione con un endSimulation();

- la classe amministra un timer che scatta periodicamente; la gestione del timer comporta la creazione di un pacchetto di tipo GenericAppMsg, in cui viene settato il campo nome con la stringa VOWFPacket, il campo sequence number con un identificatore univoco e crescente, con dimensione variabile e specificando anche la lunghezza che la risposta ad ogni messaggio deve avere. E' stato utilizzato il messaggio GenericAppMsg in quanto l'applicazione TCP utilizzata lato server accetta solo questo tipo di messaggi.
  Una volta che il pacchetto è stato creato, questo viene inviato richiamando la funzione send del socket utilizzato.

Per gestire i pacchetti e le notifiche provenienti dal TCP si è deciso di lasciare questo compito alla classe TCPSocket; in particolare, si passa al TCPSocket un *callback object*, rappresentato dalla classe TCPAppRWMA stessa; questo permette di ridefine all'interno della classe passata come callback object le funzioni socketEstablished(), socketDataArrived(), socketFailure(), socketPeerClosed(), etc., utilizzate rispettivamente per comunicare all'applicazione che la connessione con il nodo remoto è stata instaurata, che sono arrivati dei pacchetti, che la connessione non è stata instaurata e che il nodo remoto ha chiuso la connessione.

Come detto in precedenza, mentre nel ramo e nell'applicazione UDP viene implementato ULB e quindi a livello applicazione si ha la politica di gestione della ritrasmissione dei pacchetti, nel ramo TCP le cose cambiano; l'applicazione viene mantenuta allo scuro sul reinvio dei pacchetti, che viene fatta a livello trasporto; in sintesi, nel ramo TCP, ULB è posizionato e implementato all'interno del TCP.

4.3.3 Il modulo TCP

E' questo modulo che viene maggiormente modificato per realizzare RWMA e la componente ULB; in particolare, vengono introdotte le classi **TCPRWMA** (.cc, .h, .ned), **TCPRenoRWMA** (.cc, .h) e viene modifica la classe TCPConnection (.cc, .h).

4.3.3.1 TCPRWMA

Viene creata la classe TCPRWMA, generalizzazione di TCP (Figura 4.3), in modo da ridefinire il metodo handleMessage(cMessage * msg), utilizzato per la gestione dei messaggi provenienti sia dal livello applicazione che dal livello IP; quando un pacchetto dal livello IP arriva al modulo TCPRWMA, questo viene decaspulato e il messaggio ottenuto viene discriminato, differenziando il pacchetto dati da una notifica IP di tipo IPNotify. Al contrario, questa operazione non viene fatta nel modulo TCP e un qualsiasi pacchetto proveniente da IP, contenente una notifica IP, viene considerato un segmento TCP, passato all'applicazione e quindi gestito non correttamente.
Infine, la funzione findConnForNotify() è un metodo accessorio, utilizzato per ottenere il riferimento all'oggetto TCPConnection e, successivamente, il riferimento all'oggetto TCPAlgorithm; una volta ottenuto, questo ultimo viene utilizzato per gestire la notifica proveniente da IP.

4.3.3.2 TCPConnection

La classe TCPConnection è stata modificata, inserendo tutte le funzioni necessarie per gestire un pacchetto che ha richiesto il servizio RWMA; in particolare, nel momento in cui il pacchetto viene inviato dall'applicazione al modulo TCP, può essere gestito in due differenti maniere:

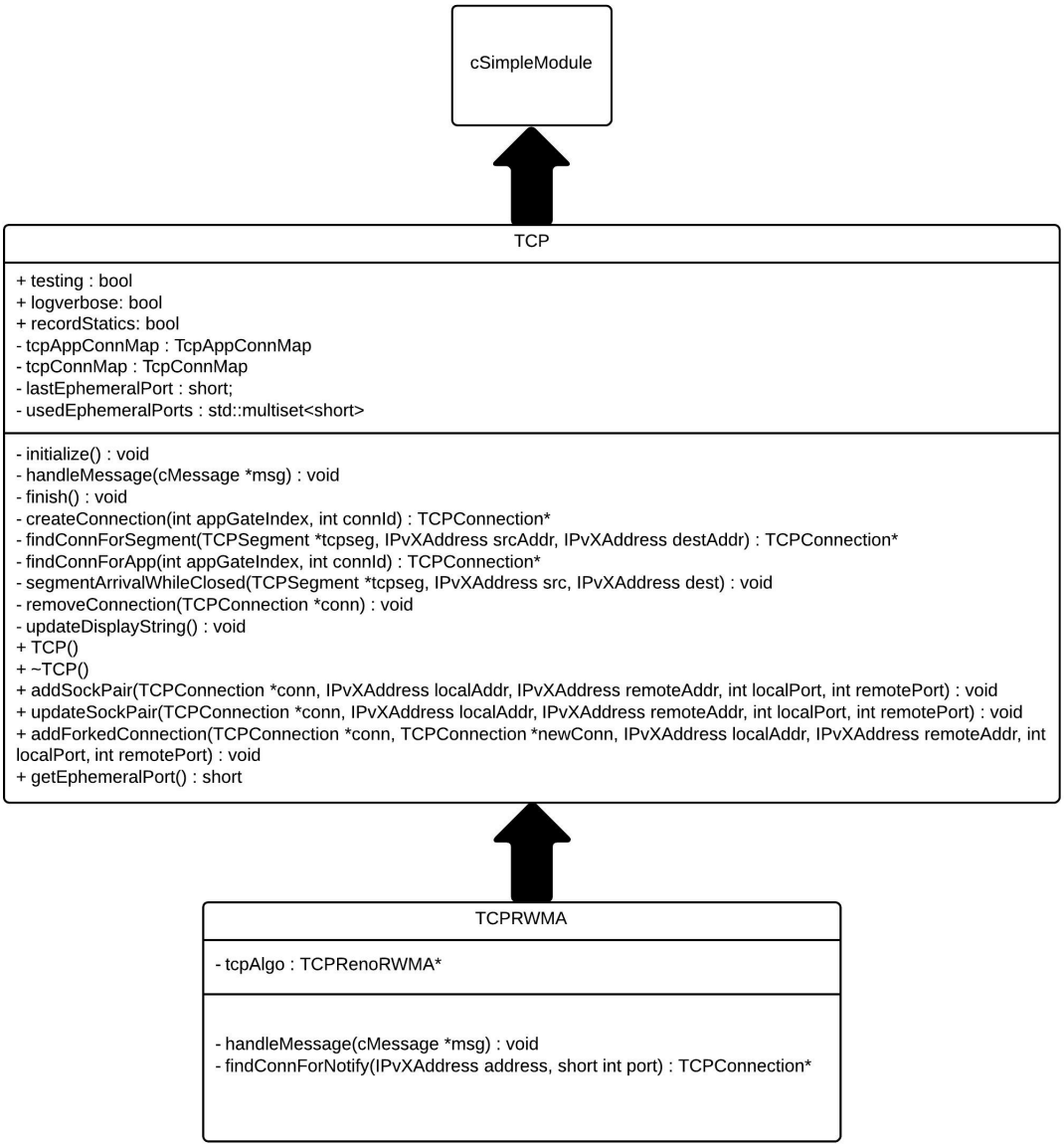

Figura 4.3: Diagramma delle classi TCP

- se l'algoritmo di gestione dei pacchetti, sottoclasse di TCPAlgorithm, usa il servizio RWMA, allora il pacchetto viene incapsulato in un segmento TCP, settando il campo protocol al valore IP_PROT_TCP_RWMA, ed inviato al modulo IP;

- se l'algoritmo di gestione dei pacchetti, sottoclasse di TCPAlgorithm, non usa il servizio RWMA, allora il pacchetto viene incapsulato in un segmento TCP, settando il campo protocol al valore IP_PROT_TCP, ed inviato al modulo IP;

Il valore del campo protocol del pacchetto risulta essere di primaria importanza a livello IP per continuare la gestione del servizio RWMA.

4.3.3.3 TCPRenoRWMA

E' la classe in cui viene implementato ULB; come si può vedere in Figura 4.4, TCPRenoRWMA è una sottoclasse di TCPReno; si è fatta questa scelta implementativa in quanto si è deciso di sfruttare le funzioni per il controllo della congestione già implementate in Reno e di ridefinire o implementare le funzioni necessarie per il rinvio dei pacchetti persi e notificati dal primo hop.

In particolare, il pacchetto inviato dall'applicazione (che richiede il servizio RWMA) viene incapsulato in un segmento TCP e inviato al modulo IP; prima di essere inviato al livello inferiore, vengono fatte le seguenti operazioni:

- i campi snd_una, snd_from e snd_nxt (puntatori al buffer di trasmissione) vengono salvati nella struttura PartialStateRWMA. In questo modo, vengono memorizzati i riferimenti al pacchetto che si sta inviando;

- la struttura PartialStateRWMA creata viene messa all'interno di una coda in attesa che il livello IP restituisca un identificatore univoco al pacchetto; una volta ricevuto tale identificatore, la struttura viene estratta dalla coda e inserita all'interno di una mappa, i cui elementi sono dati dalla coppia <id::PartialStateRWMA>.
  Gli elementi all'interno di tale mappa rappresentano tutti i riferimenti utili per una eventuale ritrasmissione dei pacchetti persi e notificati dal primo hop;

- viene attivato il timer di ritrasmissione, in caso questo non sia già stato attivato in precedenza dall'invio di un altro pacchetto di cui non si è ricevuto ancora ACK dall'altro end system.

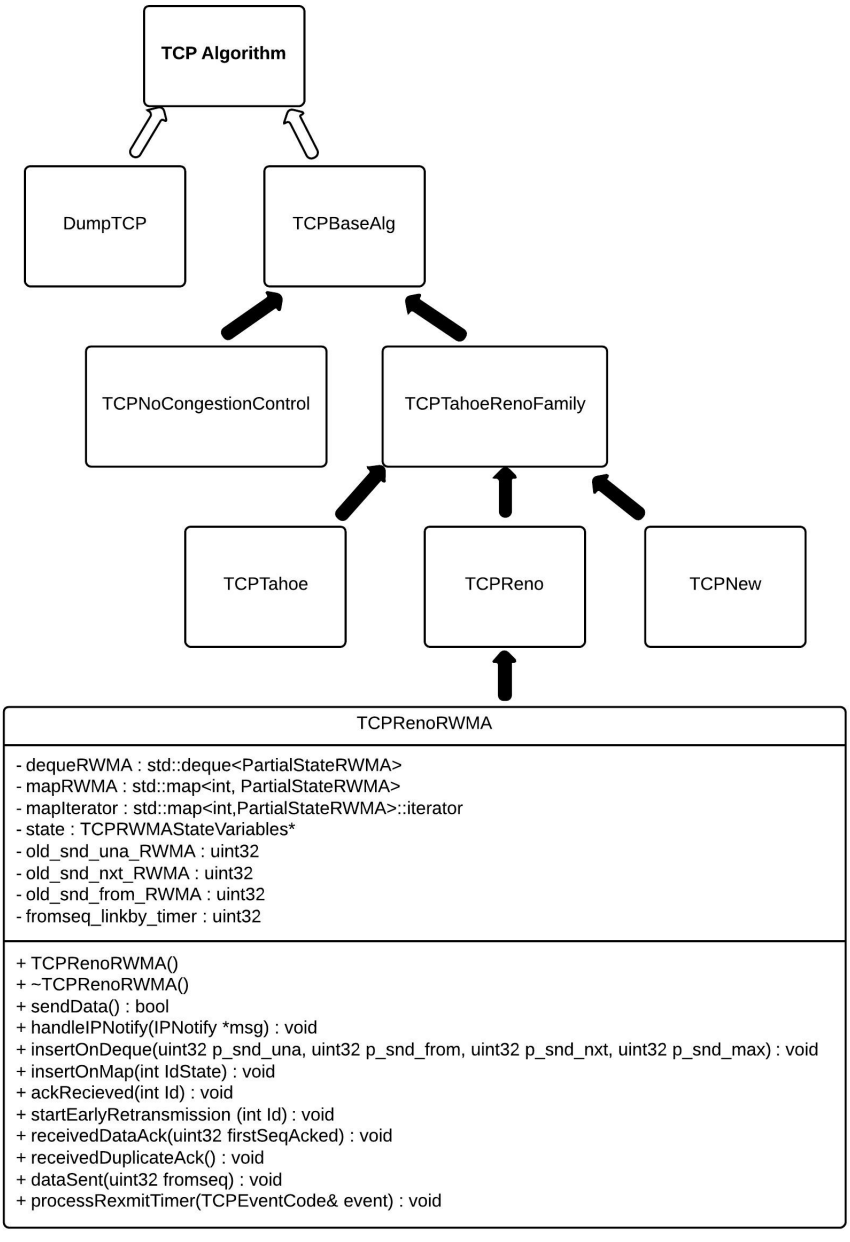

Figura 4.4: Diagramma delle classi TCPRenoRWMA

La classe gestisce le notifiche provenienti dal livello IP:

- IP_NOTIFY_RETURN_ID è la notifica contenente l'identificatore del pacchetto che si sta inviando e che è stato messo all'interno della coda in attesa di questa notifica;

- IP_NOTIFY_SUCCESS è la notifica inviata dal livello MAC e inoltrata dallo strato IP per notificare che il pacchetto inviato è arrivato correttamente al primo hop; nel momento in cui arriva questa notifica, il TCPRenoRWMA elimina dalla mappa il pacchetto notificato;

- IP_NOTIFY_FAILURE è la notifica inviata dal livello MAC e inoltrata dallo strato IP per notificare che il pacchetto inviato non è arrivato al primo hop. Nel momento in cui arriva questa notifica, il TCPRenoRWMA attiva la procedura di ritrasmissione anticipata del pacchetto; viene definita ritrasmissione anticipata perchè il pacchetto notificato viene ritrasmesso senza attendere lo scadere del timer o l'arrivo del terzo duplicate ACK, operazioni e tempi che invece vengono rispettati in uno dei qualsiasi algoritmi sviluppati per TCP.
  Si consideri l'esempio in Figura 4.5; nel momento in cui il mittente invia il pacchetto con sequence number 2, questo non arriva all'AP oppure l'AP decide di scartarlo. Il livello MAC del nodo mobile tenta di rinviare il pacchetto per almeno altre 7 volte, notificando dopo il 7° fallimento una IP_NOTIFY_FAILURE al modulo TCP; come detto in precedenza, la notifica viene gestita dall'oggeto TCPRenoRWMA che farà iniziare la procedura di ritrasmissione del pacchetto 2.

L'operazione di ritrasmissione del pacchetto a seguito di una IP_NOTIFY_FAILURE consiste di queste pochissime fasi:

- l'identificatore del pacchetto notificato viene cercato all'interno della mappa <id::PartialStateRWMA> e, una volta trovato, viene estratto dalla mappa l'elemento PartialStateRWMA ad esso legato; all'interno di questa struttura si trova i puntatori (snd_from, snd_nxt e snd_una) al buffer di trasmissione che descrivono la parte di buffer rappresentante il pacchetto notificato;

- i valori attuali dei puntatori snd_una e snd_nxt vengono memorizzati in variabili temporanee e successivamente viene assegnato al puntatore snd_nxt il valore con-

tenuto in snd_from appartentente alla struttura PartialStateRWMA trovata nel punto precedente;

- viene fatto il reinvio del pacchetto e, solo al termine del ritrasmisione, vengono ristabiliti i valori di snd_una e snd_nxt con quelli memorizzati nelle variabili temporanee;

- in ultima operazione, si deve necessariamente gestire il timer di ritrasmissione; nel momento in cui viene effettuata un rinvio a causa dell'arrivo di una IP_NOTIFY_FAILURE, il timer viene gestito nella seguente modalità, sfruttando la variabile fromseq_linkby_timer che contiene il puntatore al buffer di trasmissione da cui partire per in rinvio (e quindi, idealmente, dal pacchetto a cui il timer è collegato):

  - se il timer è legato al pacchetto inviato e di cui si è ricevuta la notifica di IP_NOTIFY_FAILURE, allora viene resettato e riattivato impostando il nuovo timeout a partire dal momento del reinvio;
  - se il timer non è legato al pacchetto inviato e di cui si è ricevuta la notifica di IP_NOTIFY_FAILURE, ma è quindi conesso ad un pacchetto inviato in precedenza, allora non viene modificato.

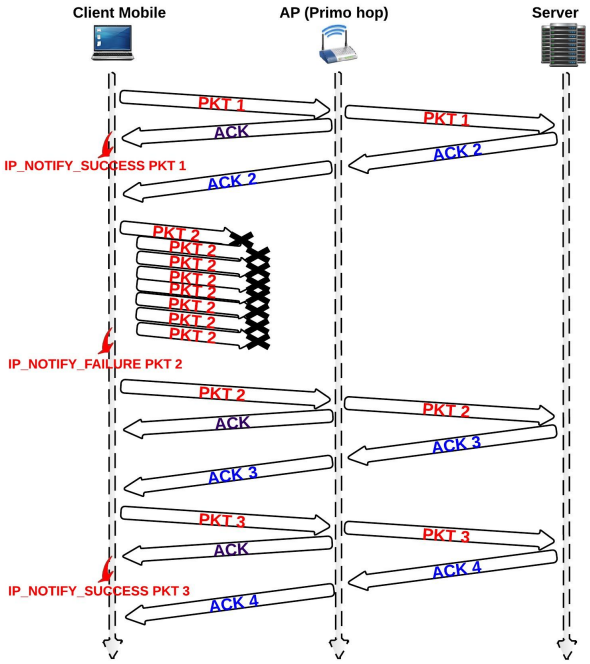

Figura 4.5: Comunicazione TCP con l'algoritmo TCPRenoRWMA

Queste stesse operazioni non possono essere fatte nelle versioni standard degli algoritmi TCP e senza l'utilizzo di RWMA; non sfruttando RWMA, non si hanno le notifiche IP_NOTIFY_SUCCESS o IP_NOTIFY_FAILURE e quindi l'algoritmo per ritrasmettere il messaggio dovrà attendere lo scadere del timer oppure la ricezione del terzo ack duplicato (Figura 4.6).

4.3.4 I moduli network layer e link layer

In ultimo, anche questi due moduli sono stati modificati, introducendo dei controlli per fare in modo che un pacchetto con campo protocol settato a IP_PROT_TCP_RWMA venga trattato alla stessa maniera di un pacchetto con campo protocol settato a IP_PROT_UDP_RWMA, a meno di campi specifici; in particolare, nel modulo network layer è stata modificata la classe IPRWMA per fare in modo che, anche in caso di pacchetto TCP con protocol IP_PROT_TCP_RWMA, venga richiamata la funzione returnID() necessaria per restituire al modulo TCP l'identificatore del pacchetto che si sta inviando; nel modulo link layer, invece, è stata modificata la classe IEEE80211MACRWMA per

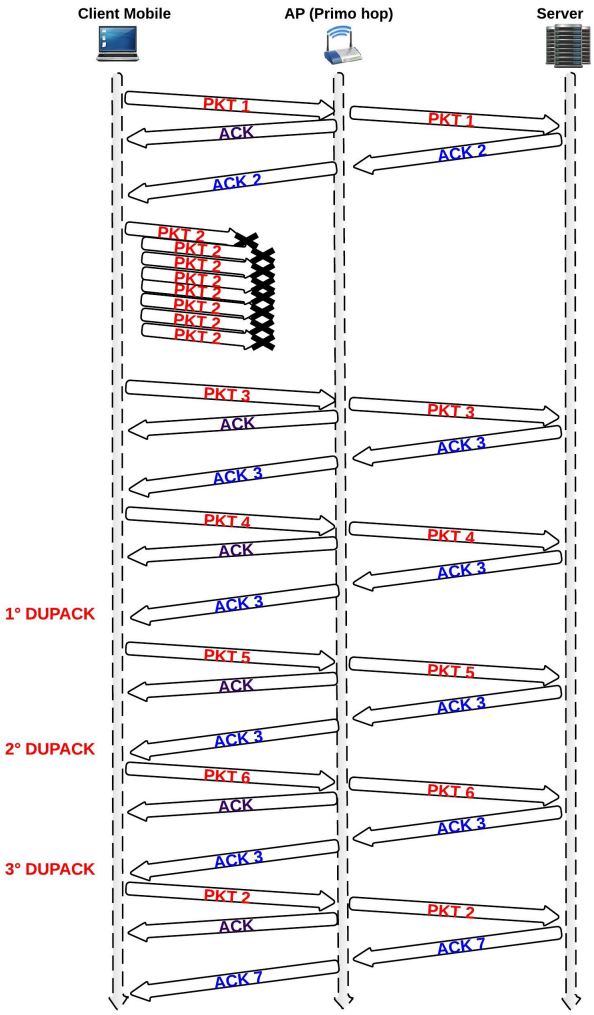

Figura 4.6: Comunicazione TCP con l'algoritmo TCP Reno

far in modo che, anche per il pacchetto TCP con protocol IP_PROT_TCP_RWMA, vengano creati i due messaggi di notifica IP_NOTIFY_FAILURE e IP_NOTIFY_SUCCESS, da inviare ai livelli superiori.

4.4 Modifiche per i test: generazione perdite e misurazione traffico

Per effettuare i test della ritrasmissione anticipata è stata modificata la classe IEEE80211-MAC (.cc), utilizzata dal modulo MAC in un AP, ed è stato introdotto il modulo FIBERLINE (.cc, .h, .ned), utilizzato per il collegamento delle due reti; in particolare, la classe FIBERLINE, che estende il DataRate channel, è stata introdotta per poter selezionare e successivamente contare tutti i pacchetti di tipo GenericAppMsg con nome VOWF-PACKET che sono destinati al server proxy. In questa maniera, si riesce a numerare i pacchetti che vengono inviati dal mittente verso il destinatario e che passano effettivamente all'interno del canale.
La modifica fatta nella classe Ieee80211Mac, invece, consiste nel fare in modo che l'AP possa scartare randomicamente dei pacchetti di tipo GenericAppMsg con nome VOWF-PACKET dal nodo mobile verso il server proxy; per ogni pacchetto che arriva all'AP, viene generato un numero random e se tale numero è minore di un valore soglia, allora non viene inoltrato verso la destinazione ma viene scartato; altrimenti se il numero generato è superiore del valore soglia, il pacchetto viene inoltrato verso la destinazione. Questo fa in modo che se il pacchetto viene scartato dall'AP (per le 7 volte definite dal MAC), quest'ultimo non invierà ACK, il livello MAC del nodo mobile notifica al TCP dello stesso la perdita con IP_NOTIFY_FAILURE e verrà iniziata la fase di ritrasmissione; se il pacchetto viene inoltrato verso la destinazione dall'AP, quest'ultimo invierà ACK, il livello MAC del nodo mobile notifica al TCP dello stesso con IP_NOTIFY_SUCCESS.

Capitolo 5

Test e risultati ottenuti

In questa sezione sono stati raccolti alcuni test. I primi sono stati svolti utilizzando RWMA e la modifca effettuata al protocollo TCP, al fine di valutare l'effettivo funzionamento del sistema; i secondi sono stati svolti utilizzando l'algoritmo TCP Reno implementato da INET, al fine di comparare, in un secondo momento, le prestazioni del TCP con e senza la modifica.

5.1 Parametri di configurazione e aspetti presi in esame

Per la simulazione viene utilizzato il file *rwmaMonitorCity.ini* (../examples/wireless/voiceoverwifiRWMA), in cui è definita la configurazione NetworkCity; questa fa riferimento alla rete definita in VoWiFiNetworkCity.ned.
Lo scenario NetworkCity è costituito dalle due reti in cui sono presenti i diversi dispositivi di rete; il nodo wireless mobile si muove lungo un percorso predefinito passando tra un certo numero di ostacoli, che rappresentano gli edifici in un contesto urbano. All'interno della stessa rete dell'host mobile sono, inoltre, presenti cinque access point collocati in modo tale da fornire una copertura disomogenea per far sì che il nodo mobile, spostandosi fra le aree con intensità di segnale diverse, possa ricevere delle notifiche di mancata trasmissione al primo hop o debba riconfigurare più volte l'interfaccia di rete, provocando così una perdita ulteriore di pacchetti, che dovranno quindi essere ritrasmessi.

Qui di seguito viene presentata parte della configurazione utilizzata, considerando solo i nodi principali della comunicazione:

- Host Mobile:
  - Indirizzo MAC schede di rete: 0A:AA:00:01:00:01 e 0A:AA:00:01:00:02
    * Applicazione: DHCP (sottoclasse di UDPBasicAppForMultipleNics)
    * Indirizzo IP temporaneo: 64.64.64.64
    * Indirizzo IP del server DHCP: 128.128.128.128
    * Porta locale: 80
    * Porta destinazione: 80

    Oltre all'applicazione DHCP è presente l'applicazione TCP:

    * Applicazione: TCPAppRWMA
    * Dimensione pacchetti: 1024B
    * Frequenza pacchetti: 40ms
    * Dimensione pacchetti di risposta: 100B
    * Algoritmo TCP utilizzato: variabile tra TCPReno e TCPRenoRWMA
    * Indirizzo IP destinazione: 128.97.1.131
    * Porta locale: 90
    * Porta destinazione: 90
- Proxy Server:
  - Applicazione: TCPSrvHostApp
  - Indirizzo IP: 128.97.1.131
  - Porta locale: 90
- Server DHCP:
  - Applicazione: UDPBasicAppForServerProxy
  - Indirizzo IP del client: 64.64.64.64
  - Porta locale: 80
  - Indirizzi IP per i client DHCP (sottorete wireless 1):
    128.96.4.132 128.96.4.133 128.96.4.134 128.96.4.135 128.96.4.136

- Indirizzi IP per i client DHCP (sottorete wireless 2):
  128.96.4.4 128.96.4.5 128.96.4.6 128.96.4.7 128.96.4.8
- Indirizzi IP per i client DHCP (sottorete wireless 3):
  128.96.6.3 128.96.6.4 128.96.6.5 128.96.6.6 128.96.6.7
- Indirizzi IP per i client DHCP (sottorete wireless 4):
  128.96.5.6 128.96.5.7 128.96.5.8 128.96.5.9 128.96.5.10
- Indirizzi IP per i client DHCP (sottorete wireless 5):
  128.96.5.132 128.96.5.133 128.96.5.134 128.96.5.135 128.96.5.136

Inoltre, sono stati realizzati utilizzando i seguenti parametri di configurazione:

1. il datarate del cavo di comunicazione tra le reti viene posto a 10Mbps;
2. il pacchetto generato dall'applicazione del nodo mobile ha dimensioni pari a 1kB ed è creato ogni 40 ms;
3. la latenza del cavo di comunicazione tra le reti può assumere i valori 10ms, 100ms oppure 300ms;
4. la perdita random del pacchetto a livello di AP può variare tra 1%, 3% e 5%

I primi due parametri sono comuni a tutti i test, mentre gli altri due variano da test a test.
Durante le prove vengono presi in considerazione:

- i pacchetti di tipo GenericAppMsg generati e trasmessi dall'applicazione del nodo mobile;
- i pacchetti di tipo GenericAppMsg provenienti dal nodo mobile e diretti al server proxy, elimitati randomicamente dall'AP;
- i pacchetti che attraversano il cavo di comunicazione tra le reti, che sono provenienti dal nodo mobile e che sono diretti al server proxy;
- i pacchetti di tipo GenericAppMsg ricevuti dall'applicazione del server proxy.

Qui di seguito viene presentata una tabella riassuntiva dei test effettuati.

		TCPRenoRWMA			TCPReno		
		PktTrasmessi (App.Nodo)	PktTrasmessi (Cavo)	PktRicevuti (App.Server)	PktTrasmessi (App.Nodo)	PktTrasmessi (Cavo)	PktRicevuti (App.Server)
Datarate 10Mbps, Perdita 5%, Pkt 1kB generati ogni 40ms	10ms	2106	2100	2100	2071	2142	2064
	100ms	1421	1415	1415	1363	1156	1124
	300ms	1379	1227	1211	1350	466	448
Datarate 10Mbps, Perdita 3%, Pkt 1kB generati ogni 40ms	10ms	0	0	0	0	0	0
	100ms	1388	1381	1381	1379	1365	1353
	300ms	1373	1270	1257	1432	645	626
Datarate 10Mbps, Perdita 1%, Pkt 1kB generati ogni 40ms	10ms	0	0	0	0	0	0
	100ms	1419	1412	1412	1414	1409	1408
	300ms	1524	1475	1459	1361	990	975

Tabella 5.1: Tabella riassuntiva

In particolare, tra i pacchetti trasmessi dall'applicazione del nodo mobile vengono contati non solo i pacchetti che effettivamente inviati e che raggiungono l'AP, ma anche quelli che rimangono nel buffer di trasmissione e non vengono inviati perchè il segnale della scheda wireless oramai è degradato o si sta degradando; infine, tra i pacchetti inoltrati dal cavo vengono contati i pacchetti duplicati che il nodo mobile ha ritrasmesso (a causa dello scadere del timer di ritrasmissione).

Dai dati emerge che il TCP modificato (TCP RenoRWMA) si comporta molto meglio su alte latenze, cioè dove le distanze tra i due end system sono molto elevate; una trasmissione con questo algoritmo sfrutta molto meglio il comportamento della ritrasmissione

anticipata, poichè la perdita del pacchetto nel primo hop viene rilevata molto prima, rispetto alla situazione in cui si utilizza solo sulla mancata ricezione dell'ACK; il risultato ottenuto è apprezzabile dai due grafici sottostanti (Figura 5.1 e Figura 5.2), in cui vengono rappresentati i test in cui si considerano un darate pari a 10Mbps, la dimensione del pacchetto pari a 1kB, il frame generato ogni 40ms, tutti e tre i possibili valori per la latenza e una perdità del 5%.

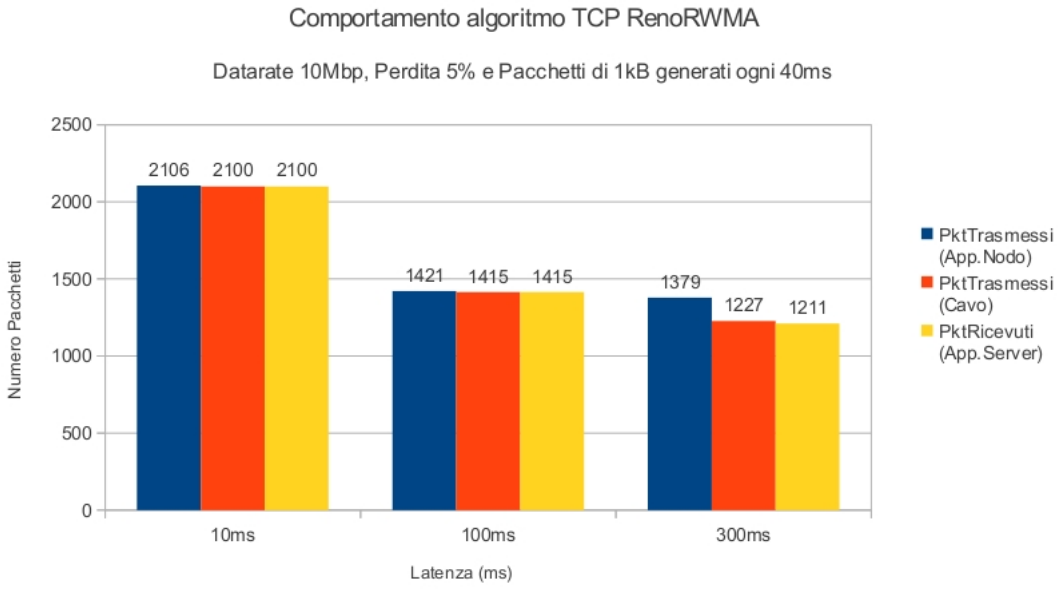

Figura 5.1: Risultato del TCP RenoRWMA con perdite del 5%

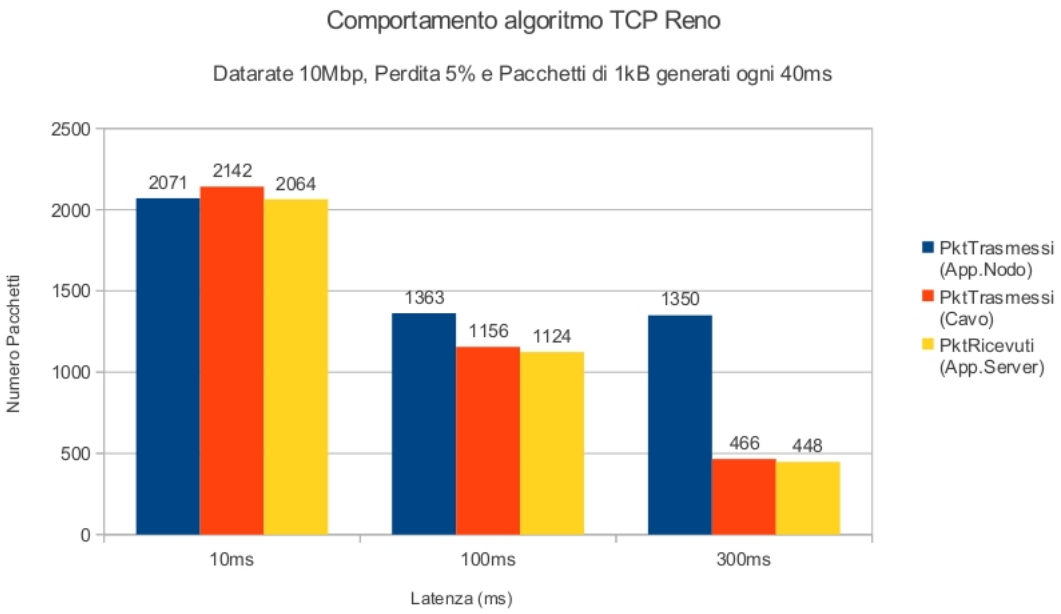

Figura 5.2: Risultato del TCP Reno con perdite del 5%

Infine, le differenze di guadagno diminuiscono tra i due algoritmi al diminuire della percentuale di pacchetti persi, non solo dovuto al fatto che si eliminano meno pacchetti a livello di AP, ma anche perchè si hanno meno perdite all'interno di una stessa finestra di trasmissione; considerando l'esecuzione dei due algoritmi con la stessa latenza del cavo, si osservi i grafici riportati di seguito, in cui si varia la percentuale di pacchetti persi.

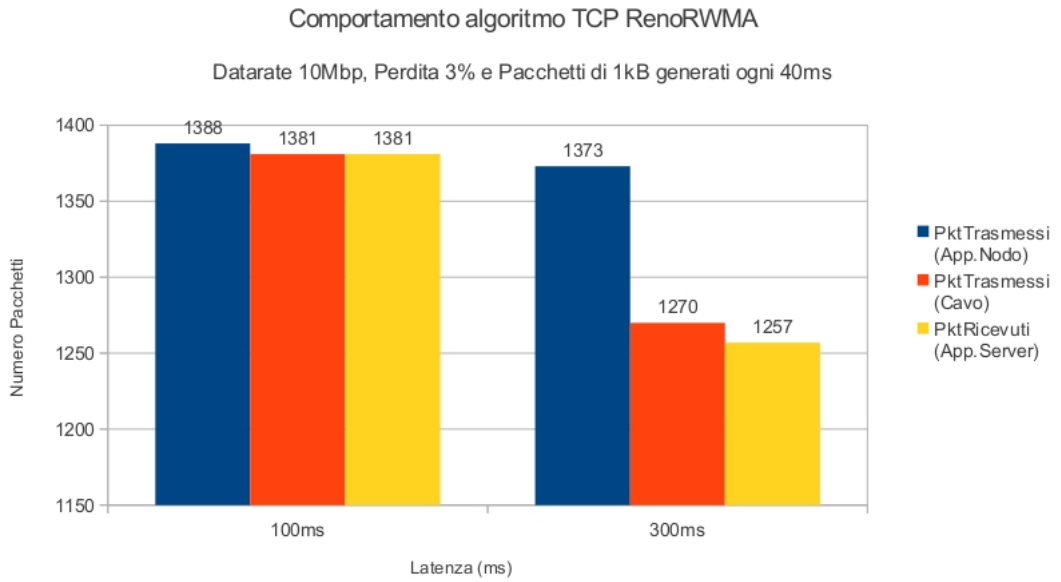

Figura 5.3: Risultato del TCP RenoRWMA con perdite del 3%

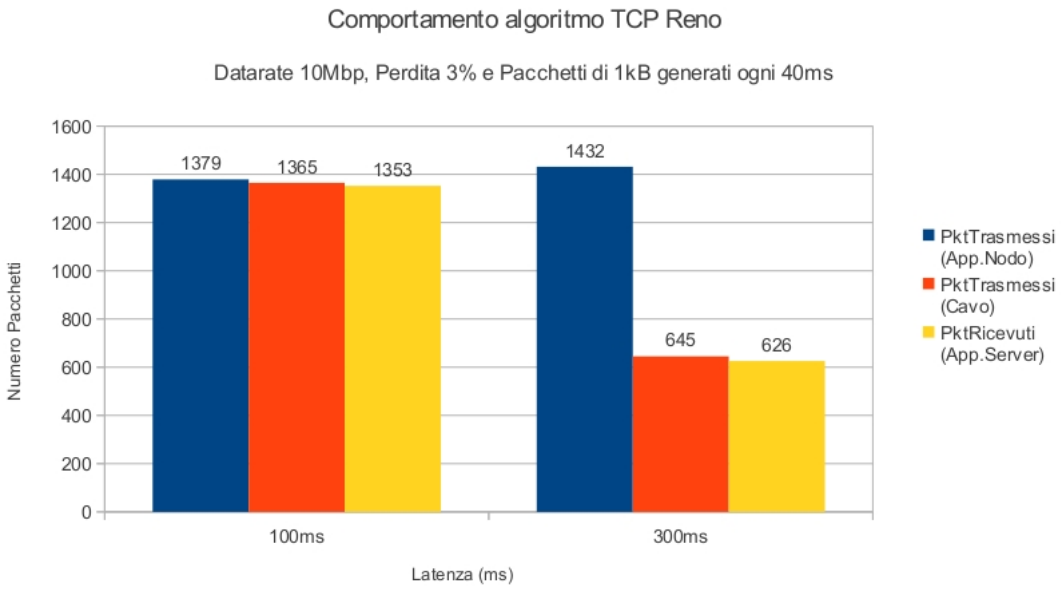

Figura 5.4: Risultato del TCP Reno con perdite del 3%

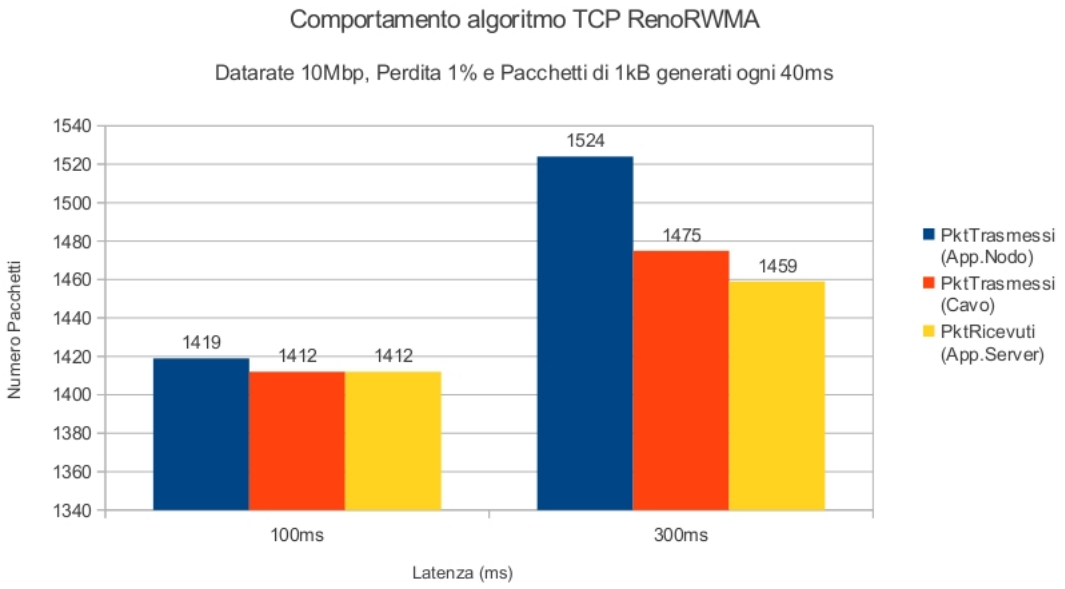

Figura 5.5: Risultato del TCP RenoRWMA con perdite del 1%

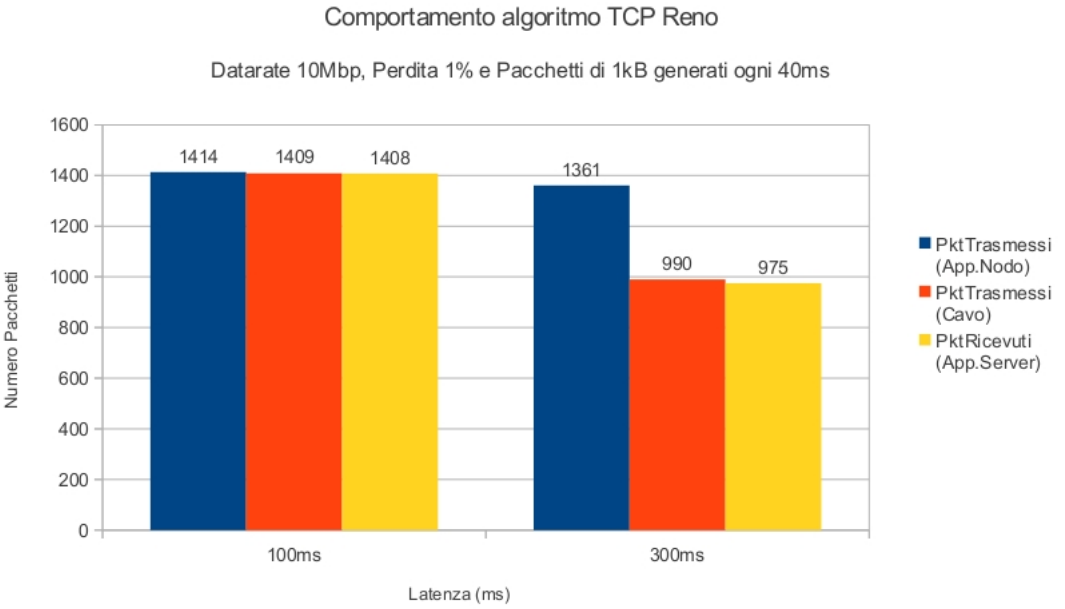

Figura 5.6: Risultato del TCP Reno con perdite del 1%

Capitolo 6

Conclusioni, problematiche e sviluppi futuri

L' obiettivo della tesi è una modifica al sistema di ritrasmissione del protocollo TCP in ambiente wireless, attraverso il simulatore Omnet++ e l'ausilio del framework INET. Dopo una prima analisi di ABPS e RWMA, allo studio di Omnet++/INET, si è passati alla conoscenza del codice di RWMA implementato sperimentalmente in Omnet++ (e anche in un kernel linux 2.6.30.5, ma questo non è stato preso in considerazione) per una trasmissione UDP; procedendo, sono state affrontate necessariamente le principali tematiche del protocollo TCP quali il controllo di flusso, la struttura del pacchetto, e maggiormente le politiche di controllo della congestione valutando le varie versioni di TCP che si sono susseguite nel tempo. Solo successivamente si è passati all'implementazione vera e propria di RWMA per una comunicazione TCP, andando a modificare o ad aggiungere parti nei vari livelli dello stack di rete. I test hanno confermato, oltre il corretto funzionamento del meccanismo, anche un buon miglioramente delle prestazioni sulle trasmissioni con latenze di rete elevate, rispetto al protocollo TCP Reno. Tuttavia, sarebbe interessante verificare se gli stessi risultati sono raggiungibili, paragonando le prestazioni dell'algoritmo di ritrasmissione creato con le altre implementazioni del protocollo, quali il TCP Cubic o TCP Veno; questo non è stato possibile farlo perchè la versione di INET utilizzata non fornisce altre implementazioni del TCP, se non il TCP Tahoe e alcuni algoritmi di trasmissione in cui non si effettua il controllo della congestione.

A questo punto si potrebbe procedere in tre modalità:

- fare un porting del sistema su versioni di INET più aggiornate che forniscono nuovi algoritmi TCP per il controllo della congestione; nel qual caso non fossero stati ancora implementati, bisognerebbe provvedere a realizzarli in modo da far test più reali;

- prevedere una modifica del processo in modo da essere compatibile con IPV6, la cui versione è implementata in un modulo differente da IPV4. Ciò renderebbe il sistema utilizzabile anche in futuro col nuovo standard emergente;

- nel breve termine, è necessaria una gestione differente del DHCP. Il primo passo di scorporare il DHCP dall'applicazione UDP è stato fatto; sarebbe necessario prevedere una ritrasmissione della richiesta di DHCP nel caso in cui la risposta, ad una DHCP DISCOVER fatta in precedenza, fosse andata perduta. Una implementazione primordiale si trova all'interno della classe DHCP.cc, ma viene fornita commentata, non essendo stata testata fino in fondo. Questo perchè, dai test svolti, si è notata la perdita di una DHCP REPLY a una richiesta DHCP;

- in ultimo, sarebbe necessaria una modifica alla gestione dei socket nell'applicazione TCP; per il momento, utilizzando una sola interfaccia di rete, viene usato un solo socket; la modifica da fare consiste nel fare in modo che vengano utilizzate contemporaneamente due o più interfacce di rete contemporaneamente, legandole a due socket differenti, in modo tale da permette una comunicazione continuativa e il più trasparente possibile nel caso in cui una delle due schede fosse disabilitata, mantenendo comunque connessioni separate.

Bibliografia

[1] Jonathan B. Spira (2003) - *20 YearsOne Standard: The Story of TCP/IP* - Disponibile in: <http://www.cbi.umn.edu/iterations/spira.pdf>.

[2] Advanced TCP/IP - *The TCP/IP Protocols Details* - Disponibile a: <http://www.tenouk.com/Module43.html>.

[3] Michele Luti (2006) - *Ottimizzazioni del tcp su reti di accesso ieee802.11b/e e valutazioni delle prestazioni di Qos.*

[4] RFC 5681 - *TCP Congestion Control (PRP STD).*

[5] W. Richard Stevens - *TCP/IP Illustrated, volume 1* - Disponibile in: <http://www.pcvr.nl/tcpip/>.

[6] RFC 2988 - *Computing TCP's Retransmission Timer (PRP STD).*

[7] Jacobson Van, Karels Michael (1988) - *Congestion Avoidance Control.*

[8] RFC 2001 - *TCP Slow Start, Congestion Avoidance, Fast Retransmit, and Fast Recovery Algorithms.*

[9] RFC 2582 - *The NewReno Modification to TCP's Fast Recovery Algorithm.*

[10] Mario Gerla, M. Y. Sanadidi, Ren Wang, and Andrea Zanella, Claudio Casetti, Saverio Mascolo - *TCP Westwood: Congestion Window Control Using Bandwidth Estimation.*

[11] Lawrence S. Brakmo, Sean W. OMalley, Larry L. Peterson - *TCP Vegas: New Techniques for CongestionDetection and Avoidance.*

[12] RFC 2018 (1996) - *TCP Selective Acknowledgment Options (PRP STD).*

[13] RFC 2883 - *An Extension to the Selective Acknowledgement (SACK).*

[14] Jeremy Stretch (2010) - *TCP Selective Acknowledgments (SACK)* - Disponibile in: <http://packetlife.net/blog/2010/jun/17/tcp-selective-acknowledgments- sack/>.

[15] Injong Rhee, Lisong Xu - *CUBIC: A New TCP-Friendly High-Speed TCP Variant.*

[16] Cheng Peng Fu, Associate Member, IEEE, and Soung C. Liew, Senior Member, IEEE (2003) - *TCP Veno: TCP Enhancement for Transmission Over Wireless Access Networks.*

[17] Giorgia Lodi, Fabio Panzieri, Antonio Messina, Vittorio Ghini, Luigi Enrico Tomaselli - *Always best packet switching for sip-based mobile multimedia services* - Dept. of Computer Science, University of Bologna, Italy.

[18] Vittorio Ghini, Giorgia Lodi, Fabio Panzieri - *Always Best Packet Switching: the mobile VoIP case study* - Dept. of Computer Science, University of Bologna, Italy.

[19] IEEE Standard for Information technology (Revision of IEEE Std 802.11-2007) - *Part 11: Wireless LAN Medium Access Control (MAC) and Physical Layer (PHY) Specifications* - Disponibile in: <http://standards.ieee.org/about/get/802/802.11.html>.

[20] William Stallings - *Wireless Communications & Networks* - William Stallings, Second Edition.

[21] The Economist - *The world in your pocket* - October 1999.

[22] András Varga and OpenSim Ltd - *OMNeT++ User Manual Version 4.2.2* - Disponibile in: <http://www.omnetpp.org>.

[23] András Varga - *INET Framework for OMNeT++ Manual* - Disponibile in: <http://inet.omnetpp.org/>.

[24] Mirco Pedrini - *TCP a ritrasmissione asimmetrica anticipata su WiFi.*