TRABAJO ESPECIAL DE GRADO

EVALUACIÓN DEL IMPACTO DE LOS PARAMETROS DEL PROTOCOLO TCP EN EL RENDIMIENTO DE ACCESO A INTERNET EN REDES FIJAS

Presentado ante la Ilustre

Universidad Central de Venezuela

por el Br. Christian José Pereira Da Silva

para optar al título de

Ingeniero Electricista

Caracas, Diciembre de 2011

TRABAJO ESPECIAL DE GRADO

EVALUACIÓN DEL IMPACTO DE LOS PARAMETROS DEL PROTOCOLO TCP EN EL RENDIMIENTO DE ACCESO A INTERNET EN REDES FIJAS

TUTOR ACADÉMICO: Prof. Marco Listanti

Presentado ante la Ilustre

Universidad Central de Venezuela

por el Br. Christian José Pereira Da Silva

para optar al título de

Ingeniero Electricista

Caracas, Diciembre de 2011

DEDICATORIA

A mamma e papà,
questo è il frutto del vostro lavoro

Christian José Pereira Da Silva

EVALUACIÓN DEL IMPACTO DE LOS PARAMETROS DEL PROTOCOLO TCP EN EL RENDIMIENTO DE ACCESO A INTERNET EN REDES FIJAS

Tutor Académico en Università La Sapienza: Prof. Marco Listanti. Tesis. Caracas. Universidad Central de Venezuela. Facultad de Ingeniería. Escuela de Ingeniería Eléctrica. Ingeniero Electricista. Mención Comunicaciones. Año 2011. Hojas

Palabras Claves: TCP (Transmission Control protocol); QoS (Qualità di Servizio); Congestion Window; Receiver Window; Ottimizzazione delle prestazioni.

Resumen. En los últimos años, la introducción de nuevas infraestructuras de redes han permitidio la oferta de una importante diversidad de servicios dando resultado a una considerable simplificación y optimización de la complejidad estructural de las redes. En tal contexto, los clientes, actuales o potenciales, no cesan de incrementar sus demandas y expectativas, por lo que para ganar y retener clientes, las organizaciones necesitan encontrar un elemento que se transforme en una ventaja competitiva. Ese elemento es el servicio, en particular, la banda disponible en la sección de acceso. El siguiente trabajo de grado ha estado orientado justamente en esta dirección con el objetivo de analizar las posibles soluciones para la evaluación del desempeño del acceso a Internet desde el punto de vista del usuario final. En particular se ha utilizado un metodo experimental para evaluar tal rendimiento (velocidad alcanzable por la aplicación), a través de un test efectuado directamente sobre el pc del usuario final, todo esto con la intención de representar lo más fielmente posible la experiencia del cliente. A partir de ciertas consideraciones teóricas, el objetivo final será determinar el impacto del protocolo de transporte TCP, en particular la dimension de la congestion window en la evaluación de la calidad de acceso a Internet en redes fijas. Finalmente, el resultado de este estudio permitirá superar algunas de las limitaciones del TCP, potenciando significativamente el rendimiento (en terminos de throughput) y mejorando el funcionamiento de los protocolos en las redes.

Christian José Pereira Da Silva

EVALUACIÓN DEL IMPACTO DE LOS PARAMETROS DEL PROTOCOLO TCP EN EL RENDIMIENTO DE ACCESO A INTERNET EN REDES FIJAS

Tutor Académico en Università La Sapienza: Prof. Marco Listanti. Tesis. Caracas. Universidad Central de Venezuela. Facultad de Ingeniería. Escuela de Ingeniería Eléctrica. Ingeniero Electricista. Mención Comunicaciones. Año 2011. Hojas

Palabras Claves: TCP (Transmission Control protocol); QoS (Qualità di Servizio); Congestion Window; Receiver Window; Ottimizzazione delle prestazioni.

Riepilogo. Negli ultimi anni, l'introduzione di nuove infrastrutture di reti ha reso possibile l'offerta di una considerevole pluralità di servizi, anche molto diversi fra loro, con una conseguente semplificazione ed ottimizzazione della struttura complessiva delle reti. In questo contesto, la qualità di servizio offerta all'utente finale riveste un'importanza primaria, specialmente per quanto riguarda la banda effettivamente disponibile nella sezione d'accesso. Il presente lavoro di tesi è stato orientato proprio in questa direzione, con l'obiettivo di analizzare le possibili soluzioni per la valutazione delle prestazioni dell'accesso ad Internet in un'ottica più vicina all'utente finale. In particolare si è adottata una metodologia per valutare tali prestazioni, intese come velocità raggiungibili dall'applicazione, realizzata attraverso dei test effettuati direttamente sul pc dell'utente finale, con l'intento di rappresentare il più fedelmente possibile l'esperienza del consumatore. Partendo da determinate considerazioni teoriche che verranno prese in esame nel presente studio, l'obiettivo finale consisterà nella determinazione dell'impatto del protocollo di trasporto TCP, specificamente la dimensione della finestra iniziale di congestione sulla valutazione della qualità di accesso ad Internet da postazione fissa. La conseguenza principale derivante da questo studio consente nel superare alcune delle limitazioni del TCP, potenziando le prestazioni (in termini di throughput raggiungibile dall'applicazione) e migliorando il funzionamento dei protocolli sulle reti.

INDICE GENERALE

INDICE DE FIGURAS

INDICE DE TABLAS

INTRODUZIONE

L'esplosiva evoluzione dei nuovi servizi a banda larga, porta con sé la necessità di un forte e importante rinnovamento delle reti di telecomunicazioni, sia in termini di funzionalità, prestazioni, sistemi di gestione, monitoraggio e provisioning.

Negli ultimi anni, l'introduzione di nuove infrastrutture di reti ha reso possibile l'offerta di una considerevole pluralità di servizi, anche molto diversi fra loro, con una conseguente semplificazione ed ottimizzazione della struttura complessiva delle reti. Tuttavia, le architetture basate interamente su fibra ottica, richiedono pesanti investimenti ed è dunque necessaria una pianificazione oculata con l'obiettivo di massimizzare il rapporto tra costi e prestazioni, in modo tale da permettere, in futuro, l'evoluzione di tali infrastrutture in funzione delle nuove esigenze del mercato dei servizi.

In questo contesto, la qualità di servizio offerta all'utente finale riveste un'importanza primaria, specialmente per quanto riguarda la banda effettivamente disponibile nella sezione d'accesso. Inoltre, rendere possibile agli utenti la verifica delle effettive prestazioni offerte dagli operatori è un passaggio necessario, soprattutto se si mira a far sostenere parte dei costi delle nuove infrastrutture a chi usufruisce del servizio.

Queste esigenze si concretizzano nello studio e nello sviluppo di sistemi di valutazione delle performance di rete che dovranno conseguentemente coinvolgere gli utenti finali. Per lo sviluppo di tali sistemi si stanno aprendo molteplici implicazioni legate direttamente all'implementazione, in particolare risultano critici le norme di omogeneità e comparabilità al variare delle realizzazioni e delle condizioni al contorno della misura.

Il presente lavoro di tesi è stato orientato proprio in questa direzione, con l'obiettivo di analizzare le possibili soluzioni per la valutazione delle prestazioni dell'accesso ad Internet in un'ottica più vicina all'utente finale. In particolare si è adottata una metodologia per valutare tali prestazioni, intese come velocità raggiungibili dall'applicazione, realizzata attraverso dei test effettuati direttamente sul pc dell'utente finale. Partendo da determinate considerazioni teoriche che verranno prese in esame nel presente studio, l'obiettivo finale consisterà nella determinazione dell'impatto del protocollo di trasporto TCP, specificamente la dimensione della finestra iniziale di congestione sulla valutazione della qualità di accesso ad Internet da postazione fissa.

Il lavoro è stato realizzato presso i laboratori della ***Fondazione Ugo Bordoni*** e si articola come segue:

Nel capitolo I saranno descritti i principali meccanismi e le funzioni di base dei protocolli di telecomunicazioni nella rete di accesso; sarà analizzato il protocollo TCP nelle sue varie declinazioni implementative, con particolare attenzione alle funzionalità e alle caratteristiche di quest'ultimo sulle reti di telecomunicazioni.

Il capitolo II descrive nel dettaglio le principali implementazioni del TCP. In particolare, si darà risalto alle funzionalità, procedure e cambiamenti proposti da ciascuna versione, per capire se la loro scelta specifica di una delle versioni prese in considerazione ha un impatto rilevante sulle prestazioni dei servizi misurati.

Nel capitolo III saranno descritte alcune proposte per l'ottimizzazione delle prestazioni del protocollo TCP. Queste modifiche consentono di superare alcune delle limitazioni del TCP, adattando le caratteristiche di trasmissione in base alla dimensione delle finestre, la gestione delle perdite di pacchetto, le notifiche di congestione ed altro ancora. Tutto ciò, per capire l'impatto critico di questi possibili cambiamenti sulle performance del sistema.

Nel capitolo IV ci si sofferma su una descrizione dettagliata del Test-Bed usato per la sperimentazione. Si descriveranno i dispositivi utilizzati nella reti di accesso considerata e, nella parte finale del capitolo, tutti gli elementi hardware/software utilizzati.

Nel capitolo V si descriveranno le fasi di realizzazione del Test-Bed utilizzato per le prove sperimentali di stima di banda e successivamente si entrerà nel merito della metodologia utilizzata che consisterà nel misurare la banda effettivamente percepita dall'utente attraverso il trasferimento di file opportuni da un server ad un client.

Finalmente, nel capitolo VI verranno mostrati i risultati ottenuti in tutti gli scenari utilizzati per capire il reale impatto che ha il protocollo TCP sulle tecniche di valutazione della qualità di accesso a Internet in termini di banda offerta.

CAPITOLO I

MECCANISMI E FUNZIONI DI BASE DEI PROTOCOLLI DI TELECOMUNICAZIONI

1.1 Protocolli di Telecomunicazione nelle reti di Accesso

Un protocollo è un insieme di regole che permettono di trovare uno standard di comunicazione tra diversi computer attraverso la rete, quindi, descrive, sia il formato che un dato messaggio deve avere sia il modo con cui i vari computer devono scambiarsi i suddetti messaggi.

Esiste un protocollo diverso per ogni tipologia di rete e secondo quello che ci si aspetta dalla comunicazione. I protocolli vengono generalmente strutturati secondo la cosiddetta pila ISO/OSI o secondo il modello TCP *(Transfer Control Protocol)*/IP *(Internet Protocol).*

I diversi protocolli sono organizzati con un sistema detto "a livelli": a ciascun livello viene usato uno specifico protocollo. La divisione in livelli è fatta in modo tale che ciascuno di essi utilizzi i servizi offerti dal livello inferiore, e fornisca servizi più "ricchi" a quello superiore. L'insieme di più livelli e relativi protocolli definisce un'architettura di rete a strati, che altro non è che un'astrazione delle funzionalità logiche della rete stessa. L'implementazione informatica dei protocolli di rete definisce, all'interno dell'architettura di rete, il cosidetto software di rete, presente usualmente all'interno del sistema operativo ed elaborato dalla scheda di rete.

L'architettura di rete *Open System Interconnection* (OSI) definita dall'International Standards Organisation (ISO) è un utile strumento per comprendere e descrivere le architetture reali, come quella di Internet che illustreremo più avanti.

Figura 1.1: Architettura di rete ISO/OSI

Le comunicazioni tra due host, avvengono realmente solo nel livello più basso. I livelli superiori realizzano una comunicazione di tipo concettuale con il loro corrispondente, devono infatti utilizzare il livello sottostante, il quale a sua volta utilizza quello sotto di lui, fino ad arrivare al livello più basso. Ogni strato implementa uno o più protocolli, ognuno dei quali definisce due interfacce: la prima (di servizio), serve a definire le operazioni che lo strato fornisce a quello superiore, mentre la seconda (peer-to-peer), stabilisce le regole ed il formato per lo scambio di messaggi con il medesimo livello presente nell'host con cui si sta effettuando la comunicazione.

Analizziamo ora le funzionalità dei diversi livelli dell'architettura OSI:

Livello fisico:

Gestisce la trasmissione di flussi di bit sul canale fisico.

Livello di Collegamento:

Gestisce la comunicazione affidabile tra coppie di nodi direttamente connesse, trasmettendo e ricevendo con la necessaria sincronizzazione unità di dati dette *frame*. Nel caso di canale ad accesso multiplo

(broadcast), si occupa anche di controllare gli accesi per permetterne la condivisione.

*Livello di Rete***:**

Assegna gli indirizzi ed effettua l'instradamento logico dei dati attraverso i nodi della rete per far giungere i pacchetti all'host destinatario. Permette dunque stabilire, mantenere e terminare una connessione, garantendo il corretto e ottimale funzionamento della sottorete di comunicazione.

È responsabile di:

- Routing: scelta ottimale del percorso da utilizzare per garantire la consegna delle informazioni.
- Gestione della Congestione: evitare che troppi pacchetti arrivino allo stesso router contemporaneamente.
- Conversione dei dati: quando si fa il passaggio fra una rete ed un'altra con diverse caratteristiche. Deve quindi:
    - tradurre gli indirizzi,
    - valutare la necessità di frammentare i dati se la nuova rete ha una diversa Maximum Transmission Unit (MTU).
    - valutare la necessità di gestire diversi protocolli attraverso l'impiego di gateway.

La sua unità dati fondamentale è il *pacchetto*.

Livello di Trasporto:

Realizza la comunicazione tra due processi garantendo la consegna affidabile dei dati. Per consegna affidabile si intende che i dati non vengono persi né alterati, sono inoltre mantenuti in ordine e senza duplicati.

Si occupa anche di effettuare la frammentazione dei dati provenienti dal livello superiore in *segmenti* e trasmetterli in modo efficiente ed affidabile

cercando di ottimizzare l'uso delle risorse di rete e di prevenire la congestione.

Livello di Sessione:

Integra i diversi flussi di trasporto di una medesima applicazione.

D'altra parte, in questo livello vengono definite tutte le regole per aprire e chiudere una connessione logica e quelli necessari per il trasferimento dei dati.

Livello di Presentazione:

Rappresentazione dei dati. L'obiettivo è trasformare i dati forniti dalle applicazioni in un formato standardizzato e offrire servizi di comunicazione comuni. Esso consente di gestire la sintassi dell'informazione da trasferire.

Livello di Applicazione:

Servizi di rete. In particolare definisce: il tipo di messaggio scambiato, la sintassi del messaggio ed il significato delle informazione che contiene. L'obiettivo fondamentale è interfacciare utente e macchina.

I primi tre livelli sono presenti in tutti i nodi della rete: switch e host. I livelli da quello di trasporto a quello di applicazione sono eseguiti solamente da host.

L'architettura OSI non è realmente impiegata in un contesto pratico, ma è utile per capire quelle che sono le reali architetture utilizzate. L'architettura di Internet, ad esempio, è divisa soltanto in quattro strati, le cui corrispondenze con quelli dell'architettura OSI sono evidenziate in figura:

Figura1.2: Confronto tra architettura OSI ed architettura Internet

Notiamo che i primi due livelli dell'architettura OSI sono riuniti in un unico livello nell'architettura di Internet che è il livello di collegamento. Inoltre i livelli OSI di rete e di trasporto corrispondono rispettivamente ai livelli rete e di trasporto di Internet. Infine, il livello applicazione di Internet riunisce i rimanenti livelli OSI e può utilizzare le funzionalità, oltre che del livello di trasporto, anche dei livelli più bassi.

Vediamo le funzionalità dei quattro strati dell'architettura Internet:

Livello di collegamento:

Gestisce tutti i dettagli hardware per l'interfacciamento fisico con la rete.

Livello di rete:

Effettua l'instradamento dei pacchetti attraverso la rete per far giungere i dati al destinatario.

Il servizio di consegna dei dati offerto da questo livello a quello superiore non è affidabile in quanto possono giungere dati non ordinati, o in più copie, o anche non arrivare affatto perché sono scartati da router intermedi.

I blocchi di dati inviati e ricevuti da questo livello sono detti *datagram*.

Livello di trasporto:

Gestisce la consegna dei dati tra due host per conto dello strato di applicazione. In questo strato sono presenti il *Transmission Control Protocol* (TCP) ed il *User Datagram Protocol* (UDP). Il servizio di consegna dei dati offerto dal TCP al livello superiore è affidabile. UDP invece fornisce un servizio più semplice che si limita ad inviare i pacchetti tra due host, senza però offrire alcun tipo di garanzia sull'effettiva ricezione dei dati a destinazione. In questo caso, eventuali criteri di affidabilità richiesti dovranno essere implementati nello strato di applicazione. La maggior parte delle applicazioni utilizza TCP.

Livello di applicazione:

Coordina i vari dettagli riguardanti le specifiche applicazioni.

1.1.1 Protocolli di Applicazione

Al livello applicativo i protocolli maggiormente utilizzati in Internet sono l'HTTP (*Hyper Text Transfer Protocol)* per la trasmissione di informazioni attraverso il Web, e FTP *(File Transfer Protocol)* per il trasferimento di file tra macchine remote con architetture diverse.

1.1.1.1 Connessione FTP

Tra gli obiettivi principali di FTP ci sono :

- Promuovere la condivisione di file.
- Trasferire dati in maniera efficiente ed affidabile.
- Incoraggiare l'uso indiretto o implicito di computer remoti.
- Risolvere in maniera trasparente incompatibilità tra differenti sistemi di stoccaggio.

FTP è basato su TCP. Utilizza **due connessioni separate** per gestire comandi e dati. Per la gestione ed il trasferimento di comandi utilizza un **canale di controllo** e per il trasferimento dei file utilizza un **canale dati**.

La connessione da parte del client determinerà l'inizializzazione del canale di controllo attraverso il quale client e server si scambieranno comandi e risposte. Lo scambio effettivo dei dati, richiederà, invece, l'apertura del cosiddetto canale dati.

FTP crea un nuovo canale dati per ogni file trasferito all'interno della sessione utente (non è persistente), mentre il canale di controllo rimane aperto per l'intera durata della sessione (è persistente). Entrambi canali sono delle connessioni TCP.

Tale modello è descritto dalla RFC 959 ed è rappresentato come segue:

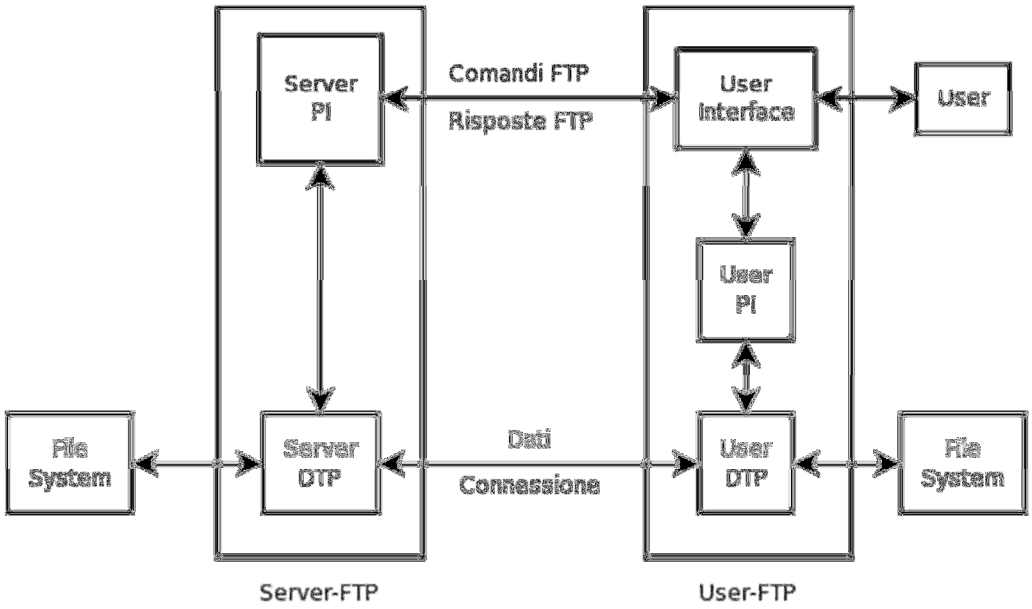


Figura 1.3: Modello RFC 959

1.1.1.2 Connessione HTTP

HTTP fu sviluppato, in origine, per ridurre le inefficienze di FTP. L'obiettivo principale, consisteva nell'avere un meccanismo di interazione client-server più veloce. È stato pensato esplicitamente per trasferire dati di qualsiasi genere, essenzialmente in formato HTML, localizzati grazie ad una stringa URL *(Uniform Resource Locator),* tra client e server web. Ogni URL ha due

componenti: il nome dell'host del server che ospita l'oggetto e il percorso dell'oggetto. Ad esempio, l'URL:

http://www.uniroma1.it/facolta/default.php

http://www.uniroma1.it → nome dell'host.

facolta/default.php → sarebbe il percorso.

L'HTTP funziona su un meccanismo richiesta-risposta (client-server), cioè, il client esegue una richiesta ed il server restituisce la risposta. Nell'uso comune il client corrisponde al browser ed il server al sito web. Vi sono quindi due tipi di messaggi HTTP: messaggi richiesta e messaggi risposta.

HTTP differisce da FTP principalmente per il fatto che le connessioni vengono generalmente chiuse una volta che una particolare richiesta (o una serie di richieste correlate) è stata soddisfatta. Questo comportamento rende il protocollo HTTP ideale per il World Wide Web, in cui le pagine molto spesso contengono dei collegamenti (*link*) a pagine ospitate da altri server.

1.1.2 Protocolli di Trasporto

Come già anticipato, il livello di trasporto converte i messaggi che riceve da un processo applicativo mittente in pacchetti, segmentandoli se necessario e aggiungendo a ciascun frammento un'intestazione di trasporto. Successivamente il segmento viene passato al livello di rete, dove viene incapsulato all'interno di un pacchetto *(datagram)* ed inviato a destinazione.

Una rete TCP/IP, ed Internet in particolare, mette a disposizione due protocolli a livello di trasporto: TCP e UDP.

Il principale compito di questi protocolli è il multiplexing e demultiplexing, ovvero l'estensione del servizio di consegna tra sistemi terminali (host-to-host) ad uno di consegna tra processi in esecuzione sui sistemi terminali (process-to-process).

1.2 TCP (Transfer Control Protocol)

È un protocollo orientato alla connessione, cioè consente a due terminali (un client e un server) di controllare lo stato della trasmissione per il successivo trasferimento dati. Questa procedura è conosciuta come *handshake,* attraverso il quale i due terminali stabiliscono a che velocità e con che protocolli colloquiare. Inoltre parole, l'entità TCP residente nel sistema mittente, prima di iniziare il trasferimento dell'informazione, deve instaurare una connessione con l'entità TCP residente nel sistema di destinazione. Successivamente, i due processi applicativi si possono scambiare i dati.

Altre caratteristiche importanti di questo protocollo sono:

- Possibilità di rimettere in ordine i datagrammi provenienti dal protocollo IP;
- Possibilità di verificare il flusso di dati per evitare una saturazione della rete (controllo di flusso);
- Possibilità di inoltrare sulla stessa connessione un flusso di dati bidirezionale (full-duplex) attraverso un canale logico in ingresso ed uno in uscita;
- Possibilità di disporre di un buffer di invio e di ricezione;
- Possibilità di adattamento della frequenza: TCP è *"rate-adaptive",* cioè la frequenza di trasferimento dei dati è capace di adattarsi alla condizione di congestione della rete e alle capacità del receiver;
- Possibilità di formattare i dati in segmenti di lunghezza variabile;
- Possibilità di multiplare i dati, cioè, far transitare su una stessa linea dati provenienti da applicazioni diverse. Tali operazioni sono realizzate grazie ai concetti di "*porta"* e "*socket*".

  Le porte ed i socket permettono di individuare e gestire quali sono gli host in comunicazione e tramite quale protocollo si comunicano.

Le Porte:

Ogni processo locale viene identificato in una connessione TCP tramite una porta.

Una porta è rappresentata, all'interno di un pacchetto TCP o UDP, da un campo a 16 bit che può assumere un valore tra 0 e 65535 e identifica un particolare canale per la trasmissione. In questo modo è possibile instaurare simultaneamente più comunicazioni, cosicché due applicazioni possono comunicare l'una con l'altra indipendente dal fatto che sulla rete stiano avvenendo altre partecipazioni.

E' possibile suddividere le porte in tre categorie:

Well Known Ports: il cui valore va da 0 a 1023 sono assegnate a specifici protocolli dalla IANA (*Internet Assigned Number Authority)*;

Registered Ports: il cui valore va da 1023 a 49151, sono registrate a nome delle società che hanno sviluppato specifiche applicazioni;

Dynamic and/or Private Ports*:* il cui valore va da 49152 a 65535, non sono gestite da nessun organo di controllo, e vengono assegnate dinamicamente, dal sistema operativo, quando un client si connetta ad un host remoto.

Socket:

Rappresenta il punto di connessione per una comunicazione. Viene identificato univocamente da un indirizzo IP ed un numero di porta, ovvero da una coppia <*indirizzo IP, porta*>.

Con l'indirizzo IP viene identificata l'interfaccia di rete. Una volta identificata questa interfaccia, i pacchetti saranno inviati all'applicazione a cui si riferisce la porta.

La componente *porta* è contenuta nell'intestazione dell'unità di dati TCP, mentre la componente *IP_Address* è contenuta nell'intestazione dell'unità dati di IP. Dunque, tutte le sessioni di comunicazione in atto tra due specifici sistemi useranno lo stesso indirizzo IP di sorgente e di destinazione mentre saranno distinte solo a strato TCP. È bene sottolineare

il fatto che un numero di porta è relativo soltanto al protocollo considerato: una determinata porta per il protocollo TCP è diversa dallo stesso numero di porta per il protocollo UDP, cioè, si tratta effettivamente di porte diverse, anche se in genere viene utilizzato lo stesso numero di porta per un servizio che gestisce entrambi i protocolli.

Le condizioni di base per instaurare una connessione TCP sono due:

- Apertura passiva lato server, la quale indica al sistema operativo su quale porta vengono accettate le connessioni;

- Apertura attiva lato client, che richiede al sistema operativo l'assegnamento di una porta per connettersi all'host remoto;

Vediamo in maniera più dettagliata lo schema per instaurare detta connessione:

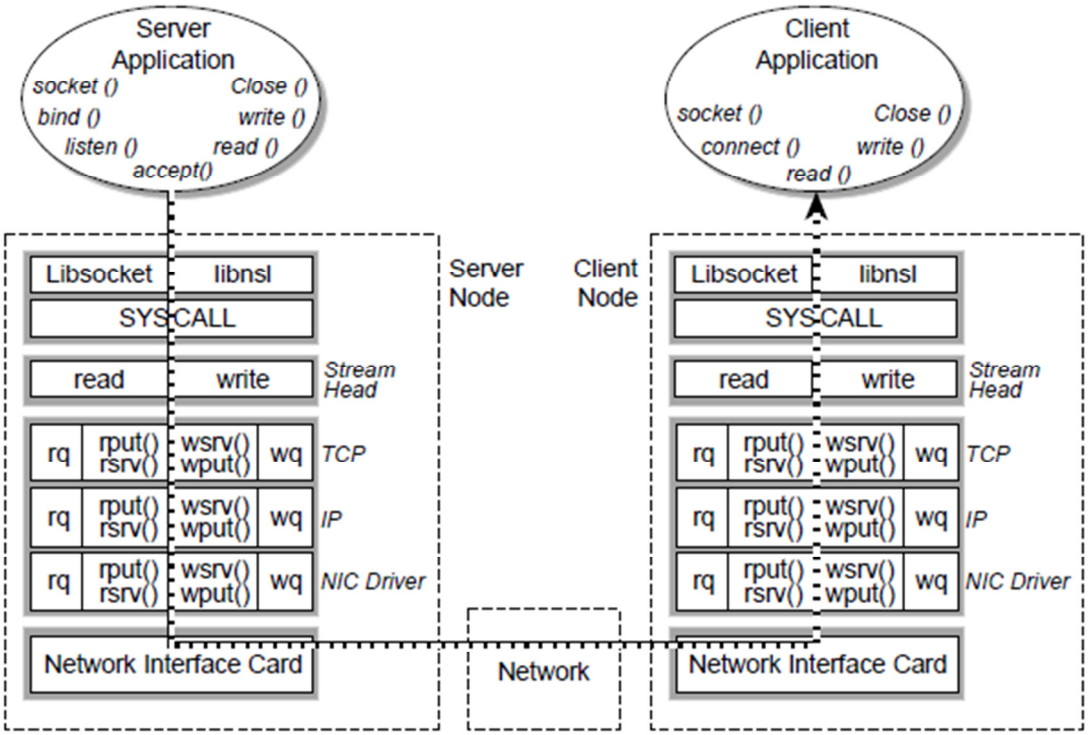


Figura1.4: Schema sui concetti di porta e socket

1. L'applicazione server apre un socket (questo attiva il sistema operativo) e si lega a una porta dello strato di trasporto. Inizia ad ascoltare, e rimane in attesa finché un client si connette. Una volta che il client si connette, il server completa la sincronizzazione, stabilisce la connessione, e finalmente sia server che client si possono comunicare.

2. Il server invia un messaggio "riempimento di buffer" che viene scritto dopo nel socket.

3. Il messaggio viene spezzato e i pacchetti vengono creati. Successivamente, sono inviati dal header per il lato di lettura di ogni modulo tramite la rutina *rput*. Se il modulo è congestionato, i pacchetti vengono immessi nella rutina di servizio per un possibile processo di ritrasmissione. Ogni modulo di rete anteporre il pacchetto con un'intestazione adeguata.

4. Una volta che il pacchetto raggiunge il NIC (*Network interface card:* scheda di rete), il pacchetto viene copiato dalla memoria di sistema alla memoria NIC e, attraverso l'interfaccia fisica, e immesso in rete.

5. Il client legge il pacchetto nella memoria NIC e viene generata una interruzione con l'obiettivo di copiare il pacchetto nella memoria del sistema e trasmetterlo attraverso la pila protocollare come è mostrato a destra nel nodo client.

6. Ogni modulo legge l'intestazione corrispondente per determinare le istruzioni di elaborazione del pacchetto e verso dove dovrebbe essere inviato. Durante la trasmissione ed una volta analizzato il pacchetto, le intestazioni sono eliminati per ciascun modulo.

7. L'applicazione client legge il messaggio mentre il pacchetto è processato e tradotto in un nuovo messaggio "riempiendo il client - leggere il buffer".

D'altra parte, TCP ha delle chiamate *Tabelle hash*, che sono strutture dati usate per cercare informazioni associata allo stato di socket per ogni pacchetto in ingresso, per controllare lo stato della macchina e per eseguire altre operazioni per conservare la connessione (aggiornamento delle sequenze numeriche, aggiornamento delle finestre, RTT, Timer, ecc).

Originariamente, TCP era stato progettato per sistemi dove le reti erano relativamente lenti rispetto alla potenza di elaborazione della CPU. Con l'avanzamento del tempo, dato che le reti sono cresciute ad un ritmo superiore a quello delle CPU, l'elaborazione TCP attualmente rappresenta il vero collo di bottiglia nella rete. Per sistemare questi problemi, attualmente stanno emergendo alcuni approcci innovativi con l'obiettivo di aumentare la velocità di elaborazione del TCP e ridurre le latenze di rete

1.2.1 Struttura dei segmenti TCP

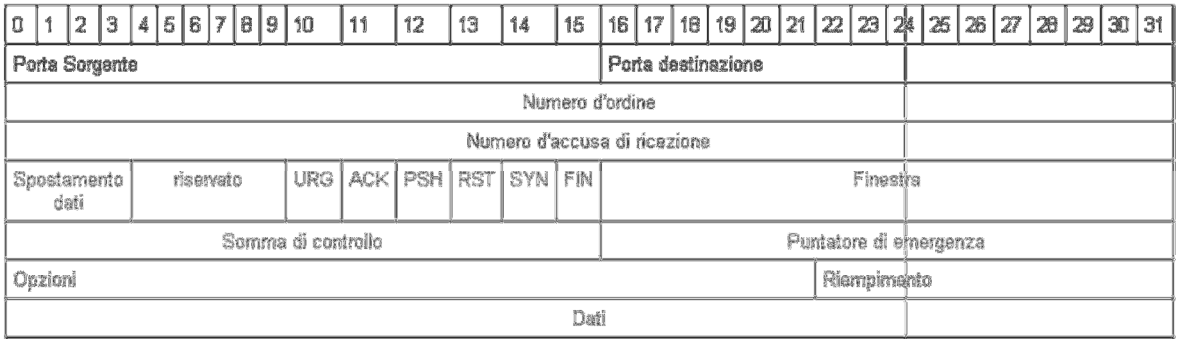


Figura1.5: Formato segmento TCP

In particolare:

- **Porta sorgente** (16bit): definisce l'indirizzo logico del processo sorgente dei dati;
- **Porta destinazione** (16bit): definisce l'indirizzo logico del processo destinatario dei dati;
- **Numero d'ordine o di sequenza** (32bit): numero di sequenza in trasmissione. Contiene il numero di sequenza del primo byte di dati contenuto nel segmento a partire dell'inizio della sessione TCP. La numerazione dei segmenti è effettuata non numerando i segmenti stessi ma gli ottetti in essi contenuti, i numeri di sequenza sono quindi numerati da 0 a 2^{32}-1 (esaurito questo spazio la numerazione ricomincia da zero).
  Tale numero d'ordine serve a verificare che nessun segmento sia perso.

- **Numero di ricezione o ACK** (32bit): numero di sequenza in ricezione. Questo campo contiene il numero di sequenza del prossimo pacchetto che il sistema si aspetta di ricevere.
- **Header Length** (4bit): rappresenta il numero di parole di 32 bit contenute nell'intestazione di TCP. Tale lunghezza può variare da 5 a 15 parole (cioè, da una lunghezza minima di 20 byte a una massima di 60 byte).
- **Reserved** (6bit): bit non utilizzati e predisposti per sviluppi futuri. Generalmente dovrebbero essere settati a zero.
- **Flags** (6x1bit): bit utilizzati per il controllo del protocollo. Rappresentano delle informazioni supplementari:

  URG: viene posto a 1 quando il pacchetto deve essere trattato urgentemente;

  ACK: viene posto a 1 quando il pacchetto è un'accusa di ricezione;

  PSH (PUSH): viene posto a 1 quando l'applicazione esige che i dati forniti vengono trasmessi e consegnati all'applicazione ricevente prescindendo dal riempimento delle memorie allocate fra applicazione TCP e viceversa;

  RST: viene posto a 1 quando un malfunzionamento impone il reset della connessione;

  SYN: viene posto a 1 solo nel primo segmento inviato durante l'handshaking;

  FIN: viene posto a 1 quando la sorgente ha esaurito i dati da trasmettere;

- **Finestra** (16bit): indica la dimensione della *finestra di ricezione*; permette di conoscere il numero di byte che il ricevitore è in attesa di ricevere;

- **Somma di controllo** (Checksum o CRC): contiene l'informazione di controllo che permette all'entità TCP ricevente di verificare la correttezza del segmento ricevuto;
- **Puntatore di emergenza** (16bit): identifica il numero di sequenza del byte che delimita superiormente i dati che devono essere consegnati urgentemente al processo ricevente;
- **Opzioni** (dimensione variabile): sono presenti raramente per usi di protocolli avanzati;
- **Riempimento**: contiene sempre degli zeri. Serve come riempimento per far si che l'intestazione abbia una lunghezza multipla di 32bit;
- **Data**: rappresenta il carico utile o *payload* da trasmettere cioè la PDU *(Protocol Data Unit*) proveniente dal livello superiore.

1.2.2 Connessione TCP

Una connessione TCP offre un servizio:

- **Full-duplex,** il flusso di dati tra due host a livello di applicazione può verificarsi contemporaneamente in due direzioni.
- **Punto-punto,** la connessione ha luogo tra un singolo mittente ed un singolo destinatario (il multi-cast con TCP non è possibile).

Una volta effettuato l'*handshaking*, il processo client manda un flusso di dati attraverso la socket e, quando questo hanno attraversato la porta, sono nelle mani del TCP in esecuzione sul client, il quale dirige i dati al buffer d'invio della connessione da cui preleverà i dati volta per volta.
TCP accoppia ogni porzione di dati client con un'intestazione, andando a formare segmenti che successivamente vengono passati al sottostante livello di rete, dove sono incapsulati separatamente in datagrammi IP e inviati in rete. Quando viene ricevuto un segmento TCP, i dati presenti in esso vengono ospitate nel buffer di ricezione della connessione TCP dove l'applicazione destinataria può prelevarli.

L'unità dati dello estratto TCP (*segment*), si compone da un'intestazione (32byte) e da un campo informativo che contiene i dati di utente (payload), cioè:

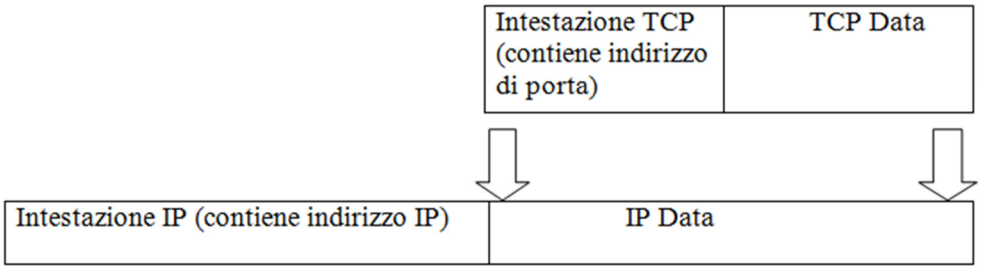


Figura1.6: Incapsulamento dati TCP nel segmento IP

1.2.2.1 Instaurazione di una connessione TCP

Prima che la fase di trasferimento delle informazioni possa avere inizio; le due entità TCP che interagiscono devono sincronizzarsi scambiandosi il proprio numero di sequenza in trasmissione iniziale. Questa sequenza rappresenta il numero a partire dal quale tutti i byte trasmessi saranno sequenzialmente numerati.

Ogni volta che si instaura una nuova connessione, il numero di sequenza in trasmissione non può iniziare da un dato valore fisso, ma viene scelto in modo opportuno attraverso un meccanismo di sincronizzazione reciproca detto "*three ways handshake*":

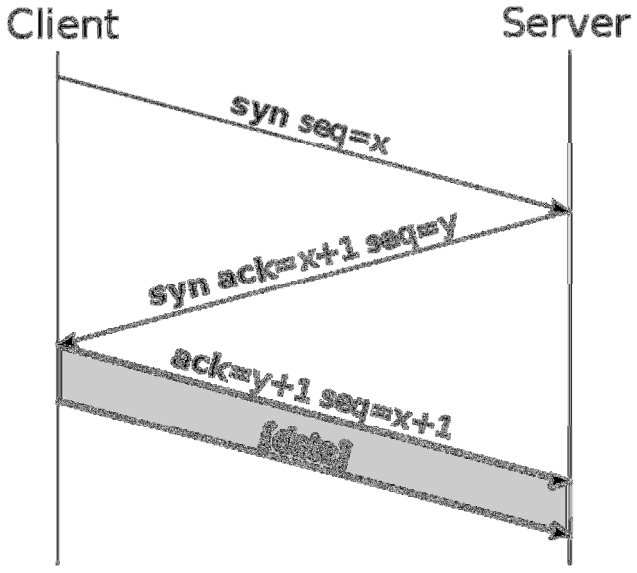


Figura 1.7: Three ways handshake

Questa procedura di instaurazione della connessione TCP può essere sintetizzat in tre punti:

1.- TCP sul lato client invia un segmento, che non contiene dati a livello applicativo (detto SYN) al TCP sul lato server con il bit **SYN** posto a 1 e il numero di sequenza iniziale (*seq=x*), ottenuto dal contatore, nel campo numero di sequenza. Tale segmento viene incapsulato in un datagramma IP e inviato al server.

2.- Quando il datagramma IP contenente il segmento TCP SYN arriva al server, quest'ultimo estrae il segmento dal datagramma, alloca i buffer e invia un segmento, detto **segmento SYNACK** di connessione garantita al client TCP (anche questo segmento non contiene dati applicativi). Nell'intestazione di tale segmento il bit **SYN** è posto a 1, il campo di riscontro assume il valore (*x + 1)* e il numero di sequenza iniziale del server (*seq= y*), ottenuto dal contatore, nel campo numero di sequenza.

3.- Alla ricezione del segmento **SYNACK** anche il client alloca il buffer e le variabili alla connessione. L'host client invia quindi al server un altro segmento che riscontra il segmento di connessione garantita al server. Tale operazione viene svolta dal client ponendo il valore (*y + 1)*, nel campo di riscontro dell'intestazione del segmento TCP. Il bit **SYN** è posto a zero.

Il terzo segmento non sarebbe, idealmente, necessario per l'apertura della connessione in quanto già dopo la ricezione da parte di cliente del secondo segmento, entrambi gli host hanno espresso la loro disponibilità all'apertura della connessione. Tuttavia esso risulta necessario al fine di permettere anche al server una stima del *Time-Out* (TO) iniziale, come tempo intercorso tra l'invio di un segmento e la ricezione del corrispondente ACK.

1.2.2.2 Rilascio di una connessione TCP

Dato che una connessione TCP è full-duplex, ogni direzione deve essere chiusa indipendentemente dall'altra. La regola è che ciascun nodo possa mandare un pacchetto di **FIN** quando ha finito di spedire dati. Quando un nodo riceve un **FIN,** deve avvisare al livello applicazioni che la controparte ha chiuso quella direzione del flusso di dati. Tale ricezione significa che non ci saranno più flussi di dati in quella direzione. Tuttavia, l'altro lato può, ancora mandare dati dopo

aver ricevuto un **FIN** e quindi possono esistere ***connessioni aperte a metà***, in cui solo uno dei due terminali ha chiuso la connessione e non può più trasmettere, ma può (e deve) ricevere i dati dall'altro terminale. Lo scenario è mostrato come segue:

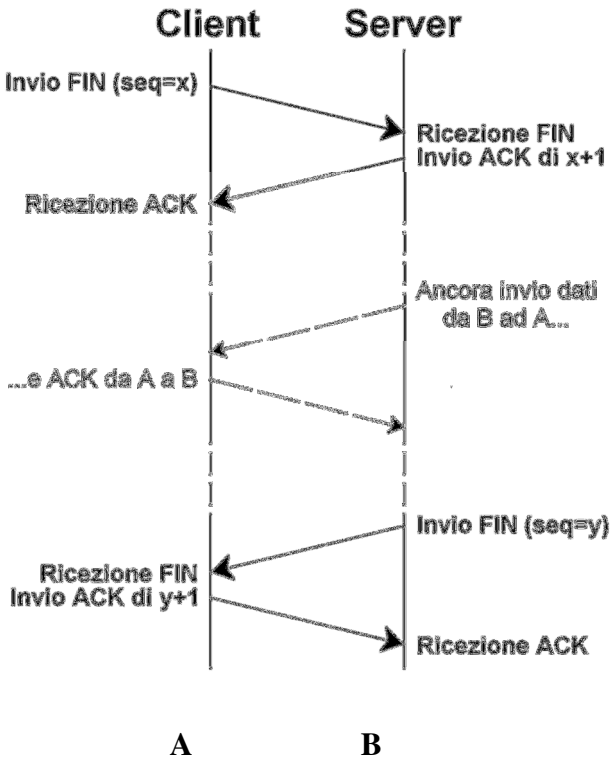


Figura1.8: Chiusura connessione TCP

Di conseguenza, la chiusura della connessione si può effettuare in due modi: con un *handshake a tre vie*, in cui le due parti chiudono contemporaneamente le rispettive connessioni, o con uno a *quattro vie*, in cui le due connessioni vengono chiuse in tempi diversi.

L'handshake a 3 vie è omologo a quello usato per l'apertura della connessione, con la differenza che il flag utilizzato è il **FIN** invece del **SYN**. Un terminale invia un pacchetto con la richiesta **FIN**, l'altro risponde con un **FINACK**, ed infine il primo manda l'ultimo ACK, e l'intera connessione viene terminata.

L'handshake a 4 vie invece viene utilizzato quando la disconnessione non è contemporanea tra i due terminali in comunicazione. In questo caso uno dei due terminali invia la richiesta di **FIN**, e attende l'ACK. L'altro terminale farà poi lo stesso generando quindi un totale di 4 pacchetti.

Inoltre, è importante definire due parametri:

1.- Il *Keepalive Timer;*

2.- Il *Time Wait Tamer;*

Il *Keepalive Timer* è utilizzato per mantenere attiva (per un certo tempo) una connessione senza traffico. Il timer viene re-inizializzato alla ricezione di ciascun segmento. Se questo timer scade, l'host controlla lo stato della connessione inviando un segmento di prova, se l'altro host non invia la conferma, la connessione è terminata.

L'uso di questo timer è discutibile perché può causare la chiusura di connessioni valide in caso di congestione o guasti temporanei.

Invece, il *Time Wait Tamer* viene utilizzato per garantire che una volta rilasciata una connessione, questa non possa essere riaperta prima che tutti i suoi segmenti circolante siano spirati (siano muori o arrivati a destinazione). Per assicurare questo, ogni host che rilascia una connessione la mantiene per un certo periodo di tempo nello stato *Time Wait Tamer.* Questa variabile è generalmente impostata al doppio della vita massima di un pacchetto

1.2.3 Controllo e Recupero d'errore

In generale, un meccanismo per il ricupero di errore ha alla base il seguente concetto:

1.- Il mittente emette un'unità dati e inizializza un contatore.

2.- Quando il destinatario riceve quella unità dati invia un riscontro (ACK).

3.- Se il mittente non riceve tale riscontro entro un certo tempo TO, assume che l'unità dati si sia persa e la ritrasmette.

4.- Il meccanismo funziona anche in caso di perdita dell'ACK.

Nel protocollo TCP, tale meccanismo è reso possibile grazie all'utilizzo delle finestre in trasmissione e ricezione che operano a livello di ottetto (byte); quest'ultimi sono numerati sequenzialmente a partire dal numero pseudo casuale ISN *(Initial Sequence Number)* scelto durante l'handshake iniziale. La finestra in trasmissione *(TxWindow)* specifica il numero di byte che un'entità TCP può inviare senza ricevere riscontro mentre che quella in ricezione *(RcvWindow)*, specifica al mittente un'indicazione sullo spazio libero disponibile nel buffer del destinatario. Quando non ci sono segmenti dati o segmenti ACK in transito, le due finestre di uno stesso canale logico, quella di trasmissione e quella di ricezione, coincidono, cioè la quantità di dati del flusso di byte in uscita che il mittente ha il permesso di inviare coincide con la quantità di dati del flusso di byte in ingresso che il destinatario è in grado di ricevere.

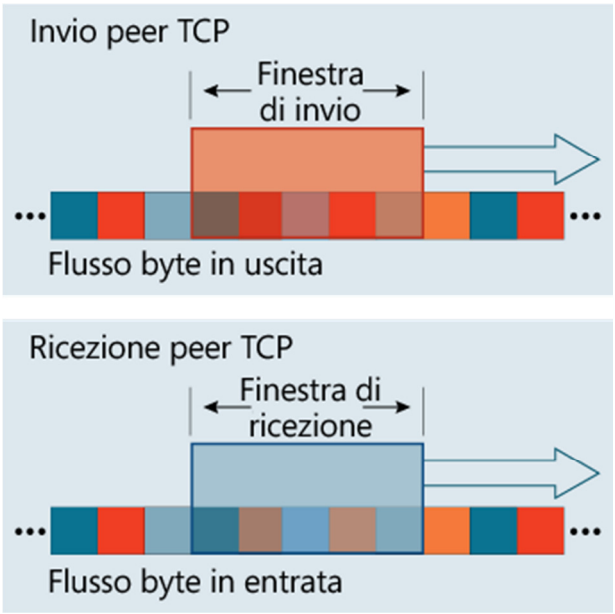


Figura 1.9: Corrispondenza tra finestra di invio e finestra di ricezione

Quando il destinatario riceve dei dati, invia al mittente delle conferme (ACK), indicando i byte che sono stati ricevuti correttamente. In ogni ACK, il campo Windows indica il numero di byte rimanenti nella finestra di ricezione. In

TCP i riscontri sono di tipo cumulativo, ovvero il destinatario comunica a quella mittente il prossimo byte che si aspetta di ricevere (nel campo *Acknoledgment Number*), significando cosi che tutti i precedenti byte sono stati ricevuti con successo. Quando i dati sono inviati, confermati ed estratti dall'applicazione, sia la finestra di trasmissione che quella di ricezione scorrono verso destra come illustrato in figura:

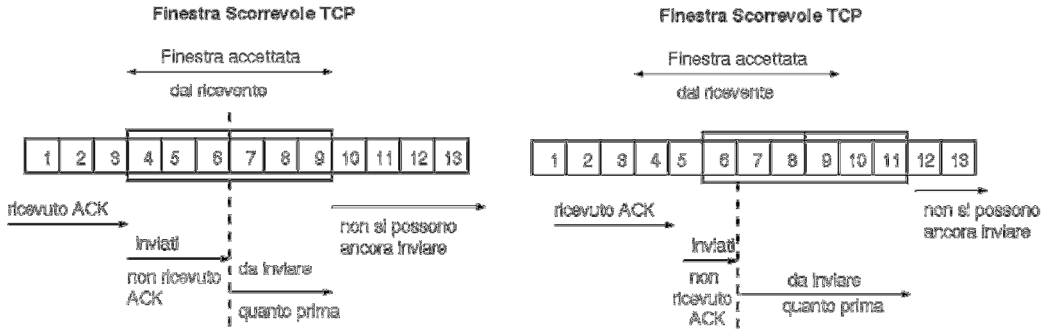


Figura 1.10: Scorrimento della finestra in Trasmissione

È rilevante sottolineare che la dimensione della finestra in trasmissione non è necessariamente costante e viene comunicata **dinamicamente** dall'entità TCP destinataria. Per questo motivo tale finestra viene chiamata in TCP "Advertised Window", ovvero finestra "comunicata" o pubblicizzata. L'entità mittente userà il suddetto valore fino a che non riceverà dal destinatario un successivo segmento con una diversa dimensione della finestra.

Adesso, osserviamo la differenza fra la finestra massima di ricezione e quella corrente:

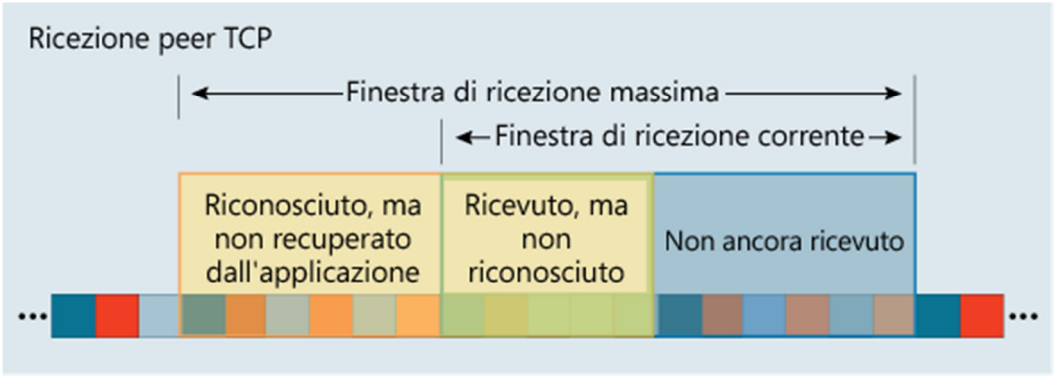


Figura 1.11: Finestra di ricezione TCP

La finestra massima di ricezione ha una dimensione fissa. La finestra di ricezione corrente ha dimensione variabile e corrisponde alla quantità rimanente di dati che il destinatario autorizza il mittente a inviare. La dimensione della finestra di ricezione corrente coincide con il valore del campo *Window* degli ACK restituiti al mittente ed è la differenza fra la dimensione massima della finestra di ricezione e la quantità di dati che sono stati ricevuti e confermati ma non ancora estratti dall'applicazione.

Anche se un controllo di flusso in base al meccanismo di finestra scorrevole è relativamente semplice, queste ha diversi obiettivi in conflitto. Per esempio, il throughput di un flusso TCP deve essere massimizzato. Ciò richiede in sostanza che la dimensione della finestra scorrevole sia lo "più grande" possibile. D'altra parte, se questa finestra è troppo grande, c'è una alta probabilità di perdita di pacchetti perché la rete e il ricevitore hanno delle risorse limitate. Così, una minimizzazione di tali perdite richiederà di ridurre la finestra di scorrimento. Pertanto, il problema fondamentale è quello di trovare un valore ottimale per la finestra (che di solito è indicato come finestra di congestione: *CongWindow*) che fornisca un buon rendimento e che non sovraccariche la rete e/o il ricevitore.

1.2.3.1 Stima Round Trip Time (RTT) e Time-Out (TO)

Il RTT misurato di un segmento (detto anche *SampleRTT*), è la quantità di tempo che intercorre tra l'istante d'invio del segmento TCP (ossia quando viene passato ad IP) e quello di ricezione dell'ACK relativo a tale segmento.

Il meccanismo di recupero d'errore dipende notevolmente del chiamato TO. L'entità mittente, dopo aver inviato un segmento, aspetta un tempo pre-definito (TO) e, se non riceve un riscontro, assume che il segmento si sia perso.

La scelta di un TO opportuno è molto importante in quanto:

- se venisse scelto molto piccolo, l'entità trasmittente rinvierebbe senza scopo un segmento che non è andato perduto ma che ancora è in viaggio, sprecando risorse e peggiorando la congestione del traffico globale della rete;
- se venisse scelto troppo grande, l'entità trasmittente dovrebbe aspettare inutilmente prima di poter assumere che un segmento sia stato perso e, le prestazioni dei singoli flussi possono essere ridotte notevolmente.

Per adattarsi a ritardi variabili, TCP utilizza un algoritmo di ritrasmissione adattivo che controlla e regola su ogni connessione il valore di TO in base a tale approccio. Tale tempo massimo è scelto dinamicamente, in maniera tale che possa variare adeguatamente nel tempo per adattarsi al traffico presente nella rete.

Per poter operare, TCP deve conoscere il TO, dunque è necessario valutare il RTT. Perciò, TCP definisce un RTT medio (*EstimatedRTT*) calcolato con un'operazione di media tra successive misure dei *SampleRTT*. L'operazione di media utilizzata è una *"media pesata mobile"* (si dice pesata perché con il parametro α si decide che peso dare ai campioni e mobile perché viene aggiornata per ogni nuova missura di *SampleRTT*):

$$EstimatedRTT = (\alpha) \times SampleRTTprecedente + (1-\alpha) \times SampleRTTnuovo$$

Il valore di α raccomandato nella RFC 2988 è di 1/8, in modo di reagire lentamente alle variazioni di ritardo. Valori minimi di α comportano un veloce aggiornamento di *EstimatedRTT*, mentre valori più elevati comportano una sempre più marcata insensibilità a brevi variazioni del ritardo di trasferimento.

Oltre ad avere una stima di RTT è molto importante tenere presente anche la variabilità di tale parametro (*DevRTT*). Sempre la RFC 2988 definisce questo parametro e lo aggiorna attraverso la seguente formula:

$$DevRTT = (1-\beta) \times DevRTT + \beta \times |SampleRTT - EstimatedRTT|$$

Il valore suggerito per β è 1/4.

Se i valori si *SampleRTT* presentano fluttuazioni limitante, allora *DevRTT* sarà piccolo, viceversa, in caso di notevoli fluttuazioni.

È importante sottolineare che il RTT di un pacchetto di dati che è stato ritrasmesso non è utilizzato nel calcolo della media e varianza RTT, e quindi non ha alcun impatto sulla stima del TO.

In definitiva, si imposta il TO pari a *EstimatedRTT* più un certo margine di sicurezza che deve essere grande quando c'è molta fluttuazione nei valori di *SampleRTT* e piccolo in caso contrario. La formula è la seguente:

$$TimeOut = EstimatedRTT + 4 \times DevRTT$$

1.2.4 Controllo di Flusso

Agli stremi delle connessione TCP si impostano dei "socket buffer" per la connessione, nei quali vengono posizionati i byte corretti ricevuti in sequenza. In questi buffer, il processo applicativo associato leggerà i dati, ma non necessariamente nell'istante in cui arrivano. Può dunque accadere che il mittente tenga una produzione di dati molto più veloce rispetto alla velocità di processo del ricedente.

Per evitare ciò, TCP offre un servizio di ***controllo di flusso*** alle proprie applicazioni. Tale meccanismo, attuato in modo coordinato dalle due entità TCP, tende a limitare il flusso di dati trasmessi in funzione delle risorse a disposizione nei sistemi terminali, indipendentemente dal traffico presente in rete. Dunque, lo scopo è evitare che un mittente invii dei segmenti TCP ad un destinatario che, in quel momento, non è in grado di riceverli a causa di indisponibilità di risorse (sia di elaborazione sia di memorizzazione).

Per attuare tale meccanismo, il mittente mantiene una finestra di ricezione scorrevole, *"RcvWindow"* (la stessa descritta nel controllo di errore), la quale fornisce un'indicazione dello spazio libero disponibile nel buffer destinatario.

Per evitare che l'entità mittente mandi in overflow il buffer di ricezione del destinatario *(RCVBuffer),* deve essere verificata la seguente disuguaglianza:

$$LastByteRcvd - LastByteRead \leq RcvBuffer;$$

nella quale, le variabili *LastByteRcvd* e *LastByteRead,* sono mantenute e gestite dall'entità destinataria. La prima si riferisce al numero dell'ultimo byte nel flusso dati che proviene dalla rete e che è stato copiato nel buffer di ricezione dell'entità destinataria. La seconda, indica il numero dell'ultimo byte nel flusso di dati letto a partire dal buffer, dal processo applicativo dell'entità destinataria. La finestra di ricezione viene impostata alla quantità di spazio disponibile nel buffer:

$$RcvWindow = RcvBuffer - (LastByteRcvd - LastByteRead)$$

Tale valore è comunicato, nel campo Window, dall'entità destinataria ogni volta che quest'ultima emette un segmento ACK verso quella mittente. A sua volta l'entità mittente mantiene due variabili*, LastByteSent* e *LastByteAcked,* la cui differenza rappresenta la quantità di dati inviati e non ancora riscontrati. Per evitare di mandare in overflow il buffer dell'entità destinataria, il mittente deve mantenere tale differenza al di sotto del valore *RcvWindow:*

$$LastByteSent - LastByteAcked \leq RcvWindow$$

Dunque l'entità destinataria può variare dinamicamente la dimensione della finestra in funzione delle sue esigenze.

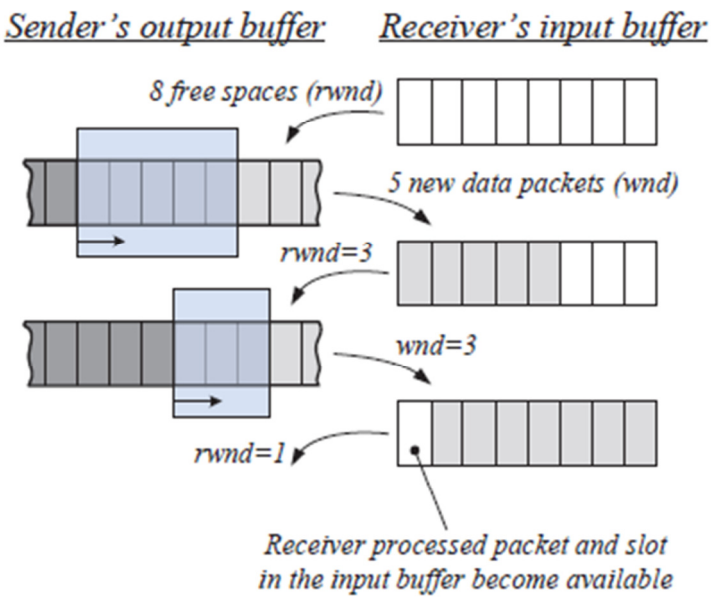


Figura 1.12: Meccanismo di adattamento delle finestre

La figura 1.12 illustra questo concetto in modo schematico. Una volta stabilita la connessione, il ricevitore informa il mittente circa la dimensione del socket buffer (nel esempio riportato, la finestra di ricezione è inizialmente di 8 segmenti dati). Il mittente trasmette una porzioni *(wnd = 5)* di segmenti dati, che come già anticipato, non deve superare la dimensione della finestra in ricezione. Nel caso in cui il ricevitore non possa gestire velocemente i dati, il mittente dovrà diminuire il tasso di invio per arrivare infine alla sincronizzazione con la velocità di elaborazione del ricevitore (nel esempio, *rwnd=3 e rwnd=1*).

Inoltre, è importante introdurre un'altro parametro chiamato *Persist Timer*. Tale parametro serve per prevenire il seguente problema (deadlock):

- Il riceiver manda un ACK con *RxWindow = 0* (fermati), in seguito manda un secondo pacchetto con la nuova dimensione della finestra, ma questo secondo pacchetto viene perso (sia per gli errori del canale sia per problemi di congestione).
- Ora sender e riceiver stanno aspettandosi l'uno l'altro.
- Quando il *Persist Timer* si azzera il sender manda un pacchetto per sondare il riceiver. La risposta è la nuova dimensione della *RxWindow*.

Se è ancora zero, si rimette in funzione il *Persist Timer* e il ciclo ricomincia.

- Se non è zero il mittente può inviare nuovi dati.

1.2.5 Controllo di Congestione

Per capire il funzionamento degli algoritmi per il controllo di congestione è stata introdotta una seconda finestra, detta *finestra di congestione (CongWindow).* Tale finestra indica la quantità di dati che possono essere inviati senza che la rete subisca una congestione. Impone un vincolo alla quantità di dati trasmessi e non ancora riscontrati dal mittente, ovvero i dati che sono in viaggio sulla rete in grado di accettare la consegna senza diventare congestionati.

Nelle implementazioni del TCP il criterio seguito è il seguente: quando si inizia una trasmissione, il mittente inizia ad "esplorare" lentamente la rete per determinarne le capacità del percorso. L'idea è quella di iniziare a trasmettere lentamente e, se tutto va bene, accelerare, salvo poi rallentare se le cose andassero male, cioè se si rileva una congestione. Lo scopo è, quindi, adattare la velocità di emissione alle condizioni della rete.

Gli eventi che indicano al server la perdita di dati trasmessi (probabilmente una congestione) sono:

1. *ACK non riscontrati allo scadere del TO*.
2. *ACK duplicati ricevuti 3 volte*.

La ricezione di tre o più ACK duplicati può essere dovuta a le seguenti situazioni:

- La rete non mantiene l'ordine dei pacchetti e, di conseguenza, è possibile che il destinatario decida inviare un ACK duplicato in conformità con l'arrivo di un segmento che non segue la normale sequenza. Attualmente, tutte l'implementazioni moderni di TCP generano un ACK duplicato quando viene ricevuto un segmento fuori d'ordine.

- Quando si perde qualche segmento dati. In questo caso TCP riceve segmenti fuori d'ordine e quindi genera ACK duplicati.
- Quando si presenta un picco di ritardo nella rete, cioè, il RTT di un pacchetto è aumentato improvvisamente provocando la scadenza del TO e di conseguenza la ritrasmissione del segmento in conflitto. Dato che il segmento in questione era già stato ricevuto correttamente, la ricezione del suo ACK genera un ACK duplicato.

Nella prossima parte discuteremo alcuni algoritmi di base per il controllo della congestione che sono state proposte per estendere la specifica TCP. Come vedremo, questi algoritmi non solo conservano l'idea di trattare la rete come una scatola nera, ma anche fornire un buon livello di precisione per rilevare la congestione e prevenire il collasso. Tuttavia, la soluzione del problema della congestione introduce nuovi incertezze che portano alla sottoutilizzazione della rete. Qui ci concentriamo soprattutto sul problema della congestione stessa e gli approcci di base per migliorare efficacemente il trasferimento dei dati.

1.2.6 Estensione TCP per reti con alto BDP

In caso di connessioni TCP su canali con grande capacità gli algoritmi visti non incrementano la finestra di trasmissione sufficientemente veloce per utilizzare al meglio la banda della connessione.

TCP è lento in catturare la banda a disposizione delle reti ad alte prestazioni, sopratutto negli reti a larghe distanze. Due questioni comunemente identificate come le ragioni di questo comportamento sono:

1.- Dimensione di buffer limitato (sia lato mittente sia quello ricevente). Questo impone un certo limite superiore della finestra, e quindi una limitazione nel throughput massimo raggiungibile.

2.- Altre questioni TCP: perdite multipli di pacchetti, incapacità di distinguere tra congestione e perdite di pacchetti a caso, l'uso di piccoli

segmenti, il tempo impiegato per fare una ritrasmissione o il valore iniziale della soglia (*SSThreshold*).

Alcuni centri di investigazione hanno studiato questi problemi basandosi su tre approcci:

- Modifiche degli algoritmi di controllo di congestione,
- Miglioramenti nel dimensionamento dei buffer,
- Analisi su come viene fatto il trasferimento TCP.

Modificare gli schemi di controllo di congestione può portare a significativi benefici per le applicazioni e per la rete e, può anche affrontare altre questioni importanti, come l'individuazione di congestione del traffico che non risponde. Modifiche di TCP, tuttavia, è un compito estremamente difficile oggi, visto i milioni di impianti già esistenti sulla base di TCP. Nonostante, quello che sembra essere la migliore opzione è cercare possibili miglioramenti nel dimensionamento dei buffer.

Dunque, un primo vincolo è che la dimensione di questi buffer non possono superare la memoria che il sistema operativo rende disponibile per tale connessione. Un secondo vincolo è che il buffer di minore dimensioni, deve essere lo sufficientemente grande in modo che il flusso di segmenti dati possa saturare il percorso di rete (link utilizzato al 100%). Questo ci porta ad una fondamentale concetto in ogni protocollo di trasporto controllato a finestra: il BDP *(Bandwith Product Delay)*. Il BDP è di fondamentale importanza perché determina la dimensione del socket buffer che massimizza le prestazioni di un trasferimento TCP. Questo parametro è stato analizzato basso diverse interpretazioni in passato e, quindi, non è ancora chiaro qual dovrebbe essere la dimensione del buffer per raggiungere al valore massimo di throughput.

Il BDP misura la quantità di dati che sono “in viaggio” in un determinato momento, in altre parole i dati non riscontrati che TCP deve gestire. Tale quantità deve essere massimizzata per aumentare la portata.

Un canale può essere largo (grande larghezza di banda) o stretto (piccola larghezza di banda), oppure corto (RTT piccolo) o lungo (RTT elevato). I canali larghi e lunghi sono quelli che hanno la capacità (BDP) maggiore. Esempi di canali a grande capacità sono quelli satellitari o le reti geografiche (WAN) che comprendono tratti intercontinentali in fibra ottica.

Si è visto in precedenza come la quantità di dati che un'entità TCP può inviare (finestra di trasmissione) è fortemente limitata dalla finestra di congestione e ricezione, ovvero:

$$TxWindow = min(CongWindow, RcvWindow)$$

Se la finestra di trasmissione è limitata dalla *CongWindow*, viene detto che il trasferimento TCP è limitato dalla congestione, invece, se la finestra è limitata dalla *RcvWindow*, si dice che il trasferimento TCP è limitato dalla dimensione dei buffer. Un trasferimento TCP può essere limitato dalla congestione in rete oppure dalla dimensione dei buffer in periodi di tempo diversi.

Dato che nelle reti con un'elevata banda e latenza la *CongWindow* assume valori maggiori di 64KB (ovvero la dimensione massima della *RcvWindow*), è evidente come la finestra di ricezione rappresenta il vero fattore limitativo. Quest'ultima come già anticipato, rappresenta la quantità di memoria che il ricevitore ha a disposizione, quindi dipende dalla dimensione del buffer (*RCVBuffer*), dove:

$$RcvWindow \leq RCVBuffer$$

Allora, possiamo definire il throughput di trasferimento come:

$$Throughput = min(CongWindow, RcvWindow) / RTT$$

In TCP, il campo Window (16bit), stabilisce la massima dimensione della finestra di ricezione (2^{16} = 65536 byte). Nelle reti con elevato BDP (anche utilizzando l'intera finestra), risulta difficile sfruttare l'intera banda disponibile con un'unica sessione.

In questo senso, per migliorare le prestazione, la RFC 1323 introduce la cosiddetta *"Window Scale Option"* che permette alle due entità TCP di comunicare, nella fase di instaurazione, un fattore di scala che estende logicamente la dimensione massima del campo Window fino a 2^{32} byte. Tale fattore è trasportato nel segmento SYN e resta costante per tutta la connessione.

Sebbene la dimensione della finestra di ricezione rappresenta un fattore fondamentale per il throughput del TCP, un altro fattore importante per determinare le prestazioni ottimali è la velocità con cui l'applicazione estrae i dati accumulati nella finestra di ricezione (il tasso di estrazione dell'applicazione). Se l'applicazione non estrae i dati, la finestra di ricezione comincerà a riempirsi e il destinatario dovrà comunicare una dimensione della finestra corrente più piccola. Quindi, per ottimizzare le prestazioni del TCP, la finestra di ricezione di una connessione dovrebbe essere configurata ad un valore tale da riflettere sia la capacità del canale che il tasso di estrazione dell'applicazione.
A questo punto, una domanda importante è: dato un certo percorso di rete, con determinate caratteristiche strutturali (capacità del percorso, RTT, dimensione del buffer di rete) e dinamiche (banda disponibile, ritardi di accodamenti), per quale valore della dimensione del socket buffer, il Throughput diventa massimo?

Ecco allora che viene presa in considerazione la capacità del canale logico *(BDP),* dato dalla seguente formula:

$$BDP = BandaTotaleDisponibile \times RTT$$

Una volta stabilita la connessione vengono trasmessi pacchetti ad una capacità **C**, causando un ritardo fisso **τ**. Questi pacchetti sono inseriti in un buffer di rete che può memorizzare fino a **B** pacchetti e dopo scartati in modo Drop Tail (ogni pacchetto viene gestito in modo identico) fino a destinazione. Inoltre, sia **ρ**

l'utilizzazione media del percorso**,** la banda totale disponibile su questo link è definita da:

$$BandaTotaleDisponibile = C \times (1 - \rho)$$

Definiamo il valore massimo di throughput come **MFT** *(Maximun Feasible Throughput).*

Se l'*RcvWindow* del proprio sistema operativo è inferiore al BDP, non potremmo sfruttare in pieno il link perché il nostro client non può inviare al server gli ACK velocemente. Quindi, conviene dimensionare la finestra di ricezione con un valore pari ad almeno il BDP, essendo questo la soluzione ottimale per ottenere determinate prestazioni in termini di traffico, evitando sia sottodimensionare la rete, costringendo il nodo mittente ad interrompere il processo trasmissivo e a mettersi in attesa della ricezione di un messaggio di ACK, che di sovradimensionarla con il rischio di congestionare la rete.

Dunque, in tanti riferimenti viene stabilito che per ottenere le massime prestazioni e quindi garantire la saturazione del percorso di rete, la dimensione del socket buffer deve essere uguale al BDP del percorso, dove la *BandaTotaleDisponibile* deve essere uguale alla propria capacità del link e, il ritardo, il minimo RTT misurato (*Tm*):

$$S = min(TXBuffer, RCVBuffer) \rightarrow S = BandaTotaleDisponibile \times RTT = C \times Tm$$

In pratica, un percorso di rete porta sempre qualche traffico di attraversamento (*cross traffic*), e quindi ad una $BandaTotaleDisponibile < C$. Alcune interpretazioni che sono stati dati al BDP sono:

- *BDP1:* $S = C \times Tm$;
- *BDP2:* $S = C \times Te$, dove *Te* è la somma di tutti i ritardi medi lungo il percorso che tiene conto sia del ritardo di accodamento sia del ritardo fisso .
- *BDP3:* $S = BandaTotaleDisponibile \times Tm$;
- *BDP4:* $S = BandaTotaleDisponibile \times Te$;

- *BDP5: S = BTC x Ta;* dove BTC *(Bulk Transfer Capacity)* è il throughput medio di un trasferimento TCP limitato da congestione, cioè, il tasso medio di dati (bit per secondo) di un singolo trasferimento sul percorso in questione, determinato dalla finestra di congestione e per l'RTT medio del trasferimento. Inoltre, *Ta* è l'RTT medio di tutto il percorso dopo che il trasferimento è iniziato;
- *BDP6: S = S_∞ (*dove S_∞ > *maxCongWindow),* dove S_∞ è un valore sufficientemente grande in modo che sia sempre maggiore della finestra di congestione. Quindi, il significato di BDP6 è che la connessione deve essere sempre limitata dalla congestione. Naturalmente, in assenza di informazioni sul percorso risulta difficile sapere quanto è grande la finestra di congestione e quindi qual sarebbe il suo valore massimo.

Le tecniche comunemente utilizzati per stimare questa capacità logica di trasferimento sono: il *Work Around Daemon (WAD)*, che utilizza una serie di ping per misurare il minimo RTT *(Tm)* impostando la dimensione del socket Buffer come in BDP1. Un altro approccio simile è quello adottato dalla realizzazione *NLANR Auto-Tuning FTP*, dove il dimensionamento del socket buffer è basato sulla media e non sul minimo, raggiungendo ad un BDP vicino a BDP2.

Finalmente, l'uso efficace della rete dipende non solo dal modo in cui una singola applicazione TCP può utilizzare le capacità di rete, ma anche sul modo in cui essa collabora con altre applicazioni di trasmissione dati attraverso la rete stessa. Raggiungere al throughput effettivo e quindi sfruttare pienamente un certo percorso di trasmissione dipende di numerosi variabili. Perciò, quello che sembra più ragionevole è modificare opportunamente la configurazione di entrambi stremi di comunicazioni cambiando alcuni di questi parametri. Tra i principali parametri abbiamo:

1. *Dimensione della finestra:* aumentare la dimensione della finestra sia sul lato mittente si sul lato ricevente.

2. *Dimensione del Buffer di Rete:* regolare la quantità di memoria del sistema e quindi la quantità di dati che possono essere gestite sulla connessione TCP.
3. *Dimensione del socket Buffer:* impostare i limiti di memoria per ogni connessione TCP in modo tale di garantire il pieno utilizzo di link.
4. *Attivare l'opzione Large Window Extensions:* consente l'uso della "*Window Scale Option"* estendendo logicamente la dimensione della finestra.
5. *Attivare il cosiddetto TCP Time Stamps:* fornisce una misura più accurata del RTT evitando possibili errori. Tale opzione consente al mittente scrivere l'istante di partenza del segmento trasmesso, in modo tale da permettere al destinatario scrivere nel segmento di risposta (l'ACK) l'istante di partenza a cui il riscontro si riferisce. In trasmissione basterà operare una semplice differenza per ottenere una buona stima dell'RTT.
6. *Univocità del numero di sequenza:* il numero di sequenza in TCP è di 32 bit e dunque, se si inviano dati molto velocemente, può accadere che un segmento numerato con un certo valore sia ancora in viaggio quando un successivo segmento viene emesso con il medesimo numero di sequenza (si è completato un ciclo). Inoltre esiste la possibilità di confondere segmenti di una connessione con quelli di una connessione precedente. Il *TTL (Time To Live)* del protocollo IP e il meccanismo di numerazione di sequenza pseudo-casuale risolvono il secondo problema. Invece, la prima situazione esposta richiederebbe che il *TTL* venisse scelto in modo coerente con la velocità di trasmissione, cosa che risulta difficile.

   È necessario dunque soddisfare la seguente relazione:

$$\frac{2^{31}}{B} \geq MSL;$$

   dove B, misurato in byte/s, è il ritmo di trasmissione e MSL *(Maximum Segment Life),* rappresenta il massimo tempo di vita di un segmento. Implementazioni di TCP più moderne scelgono per MSL un valore di 30 secondi. Allora, facendo un calcolo, per velocità di 100Mbps bisognarebbe scegliere un valore di *MSL* minore di 170 secondi mentre per connessioni

a 1Gbps e a 10Gbps il valore di *MSL* dovrebbe essere minore rispettivamente di 17 e 1.7 secondi.

L'RFC 1323 propone di utilizzare l'opzione *timestamp* anche qui, consentendo, con lo stesso meccanismo spiegato in precedenza, di evitare confusioni tra segmenti che hanno lo stesso numero di sequenza.

7. *Attivare l'opzione Selective Acknowledgments (SACK):* la perdita di segmenti molteplici può avere una effetto distruttivo sulla efficienza del TCP. Infatti il TCP utilizza ACK cumulativi che confermano il più alto numero di sequenza arrivato correttamente. Questo può portare a ritrasmissioni non necessarie in quanto il mittente non ha piena conoscenza dei segmenti che sono arrivati correttamente. Quando si verificano perdite molteplici, il mittente può solamente attendere che scada il TO per rimandare l'intera sequenza di pacchetti non confermati. Ciò, comporta ad una sostanziale diminuzione della portata.

   La soluzione è ricorre ad una strategia di ri-emissione selettiva (SACK) che utilizza riscontri selettive invece che cumulativi permettendo indicare al trasmettitore esattamente quali sono i pacchetti persi che devono essere ritrasmessi. Ciò permette al mittente di continuare a spedire segmenti (sia nuovi che ritrasmissioni) ad un'appropriata velocità.

È importante sottolineare che, realizzare misure sul PC dell'utente finale e fare delle modifiche, ha una stretta dipendenza dal tipo di sistema operativo su cui si sta lavorando.

CAPITOLO II

IMPLEMENTAZIONI DEL TCP

2.1 Implementazioni TCP

2.1.1 Versione TCP Berkeley

È stata la prima versione di TCP che tentò di evitare la congestione di 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, denominata "*Slow Start Threshold" (SSThresh).*

In pratica, all'avvio della connessione la *CongWindow* (che regola il ritmo di spedizione) può essere inizializzata ad un valori pari a 2 volte la dimensione massima di un segmento in emissione (SSMS). Noi invece ipotizziamo che la *CongWindow* è inizializza da 1 segmento, ossia una quantità di dati molto piccola. Dato che molto probabilmente la banda disponibile è molto maggiore, nella prima parte della connessione ogni volta che il mittente riceve un riscontro riferito a nuovi dati (non un riscontro duplicato), la *CongWindow* è aumentata di un segmento (per essere precisi di SMSS).

$$CongWindow = CongWindow + SMSS$$

Ciò che accade è che il mittente invia il primo pacchetto, riceve il rispettivo ACK e porta la *CongWindow* al valore 2. Vengono dunque inviati adesso due segmenti che una volta riscontrati, entrambi faranno salire la finestra di congestione a 4. In questo modo, la quantità di dati inviati raddoppia ogni RTT, comportando, di fatto, un incremento esponenziale (politica di incremento moltiplicativa). Tale crescita esponenziale avviene poiché si ritiene che all'inizio di ogni trasferimento il canale trasmissivo sia più libero, e quindi si cerca di inviare all'inizio i pacchetti più grossi. Questa fase della connessione è conosciuta

come **Slow Start (Partenza Lenta)** e rappresenta l'impulso iniziale necessario per portare la connessione allo stato di equilibrio.

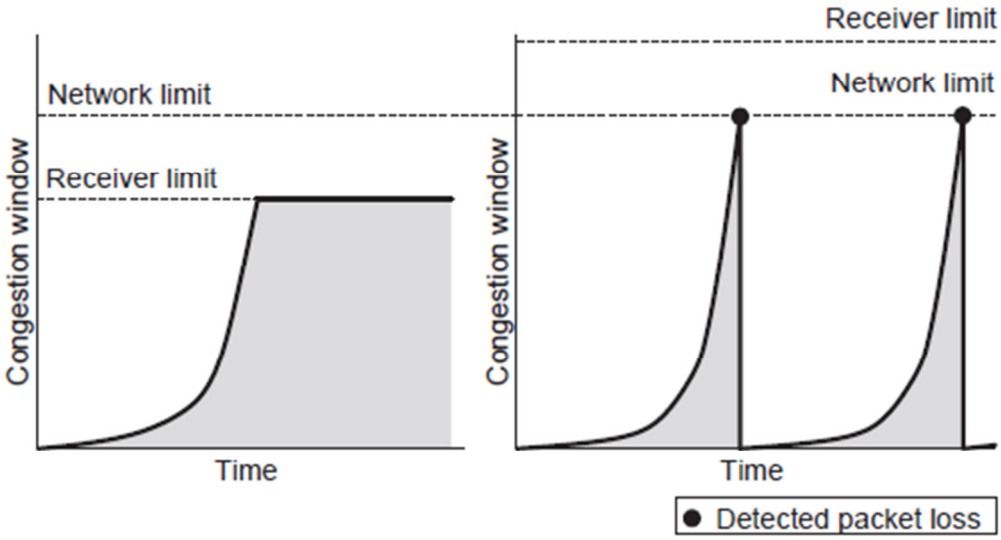


Figura 2.1: Fase Slow Start (Partenza Lenta)

Allora, è chiaro che, quando le risorse di rete disponibili sono inferiori ai limiti imposti dal ricevitore, l'efficacia, nel lungo termine dell'algoritmo Slow Start è molto bassa.

D'altra parte, il valore di *SSThresh* viene impostato all'inizio della connessione ad un valore molto alto (la RFC 2001 consiglia di inizializzare tale variabile ad un valore pari a 65535byte). Dunque, al essere questo valore troppo elevato, nella prima parte della connessione si rimane nella fase di Slow Start relativamente a lungo.

Quando la *CongWindow* raggiunge il valore di *SSThresh,* la connessione entra nella fase **Congestion Avoidance**, in cui la *CongWindow* è incrementata linearmente di un segmento per finestra completa trasmessa, ogni riscontro non duplicato ricevuto. La crescita è di tipo lineare per cercare di raggiungere il livello di congestione il più lentamente possibile.

$$CongWindow = CongWindow + 1/CongWindow$$

Essa mira a migliorare l'efficacia TCP in reti con risorse limitate, cioè, dove la rete diventa un collo di bottiglia di trasmissione reale e quindi la capacità della risorsa considerata è inferiore alla domanda che investe alla stessa risorsa.

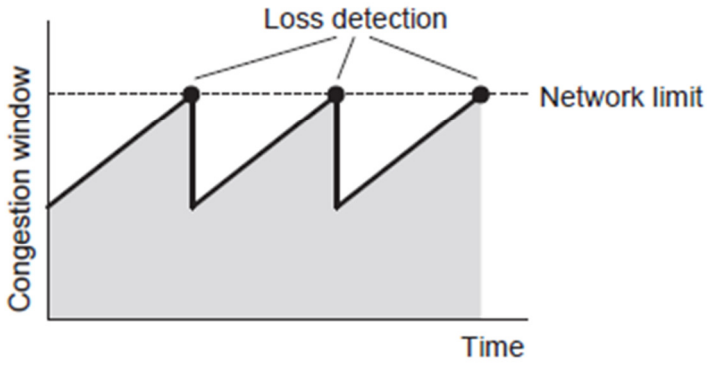


Figura 2.2: Fase Congestion Avoidance

Quando si verifica il **primo evento di perdita**, ossia lo **scadere del TO,** viene effettuata la seguente strategia:

1) Si inizializza una nuova soglia a partenza lenta pari a:

Inizialmente la RFC 2001 consigliava di inizializzare la variabile *SSThresh* ad un valore non superiore alla metà della finestra di congestione, cioè, *SSThresh = CongWindow/2*. Tuttavia, la RFC 2581 ha specificato che la *SSThresh* deve essere posta ad un valore minore o uguale di "*max (FlightSize/2, 2*SMSS)",* dove *FlightSize* è la quantità di dati emessi dal mittente ma non ancora riscontrata (cioè, i dati ancora in "viaggio"). Noi ipotizziamo *SSThresh = max (FlightSize/2, 2*SMSS).*

2) Si pone *CongWindow = 1 segmento* e, si procede con la tecnica a partenza lenta fino a che:

$$CongWindow = SSThresh.$$

In questo modo la connessione riparte dalla fase Slow Start e vi rimane fino a che *CongWindow* non raggiunge il nuovo valore di *SSThresh,* momento in

cui entra nella fase di Congestion Avoidance. Per il resto della connessione, tutte le volte che scade il TO, ciò che succede è quello descritto precedentemente.

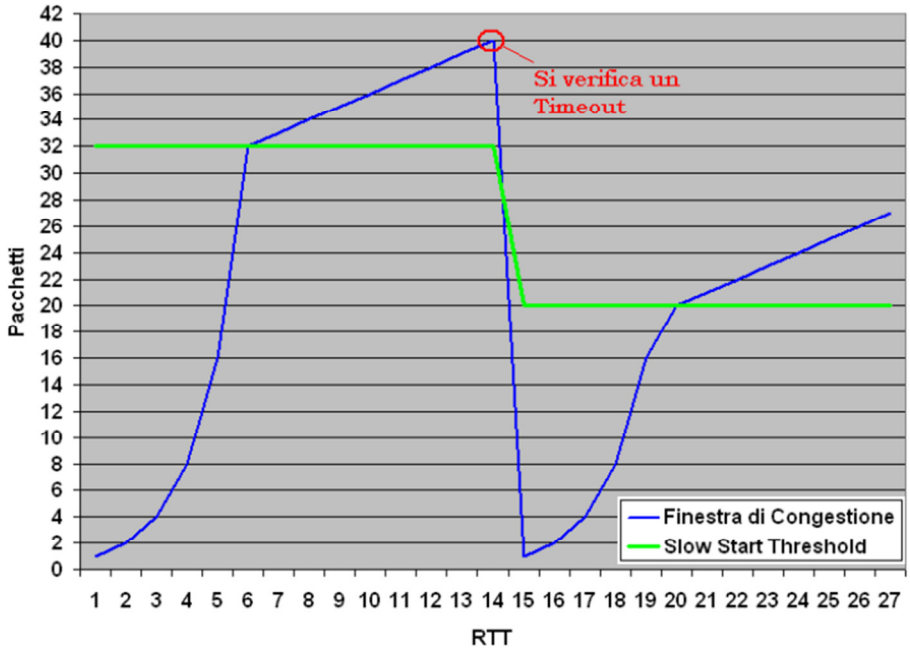


Figura2.3: Andamento della finestra di congestione nel TCP Berkeley

2.1.2 Versione TCP Tahoe

Come abbiamo visto, nella versione Berkeley l'unico evento che può essere considerato come perdita di un pacchetto è lo scadere del TO. Il TCP Tahoe (evoluzione di Berkeley) introduce un meccanismo chiamato **Fast Retransmit** con lo scopo di migliorare le prestazioni del controllo di flusso**.** In pratica esso aggiunge un nuovo tipo di perdita: la ricezione da parte dal mittente di un certo numero di ACK identici tra di loro (ACK duplicati).

Dunque, se il mittente riceve diversi ACK duplicati capisce che qualcosa è andato storto è può prendere i provvedimenti necessari.

I pacchetti spediti dal mittente successivamente a quello andato perduto (prima dell'arrivo del terzo ACK duplicato), se arrivano correttamente, possono essere salvati nel buffer del ricevitore e in questo modo, una volta arrivato il pacchetto ritrasmesso verranno riscontrati tutti i pacchetti con un singolo ACK. È

importante sottolineare che, inizialmente, le vecchie implementazioni di TCP scartavano anche tutti i segmenti successivi a quello errato e dunque il mittente doveva ritrasmetterli tutti.

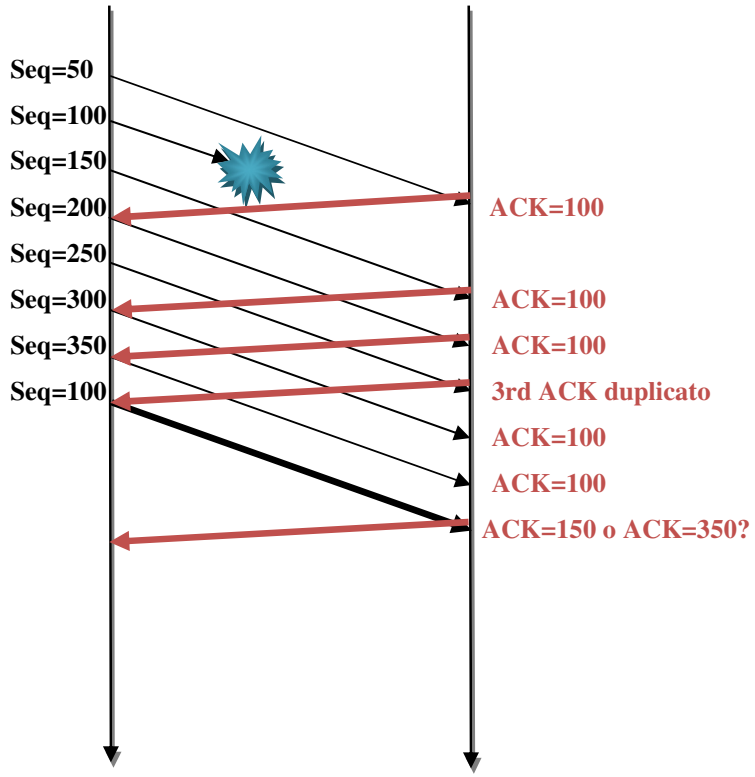


Figura 2.4: Arrivo di 3 ACK duplicati

Per quanto riguarda l'aggiornamento delle due variabili dopo dell'arrivo di tre o più ACK duplicati, il trasmettitore si comporta come se fosse scaduto un TO e ritrasmette subito quello che si ritiene essere un segmento perso, quindi il TCP Tahoe si comporta esattamente come il Berkeley.

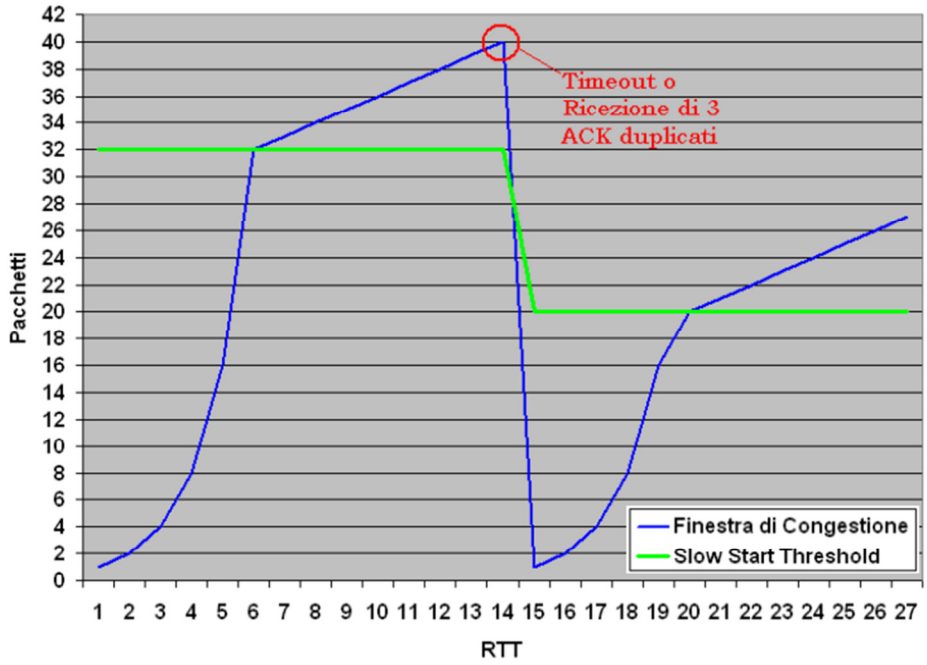


Figura2.5: Andamento della finestra di congestione nel TCP Tahoe

L'idea di base è supporre che una perdita identificata da tre ACK consecutivi sia essenzialmente una perdita occasionale e che quindi non indichi necessariamente la presenza di una vera e propria situazione di congestione.

Il principale problema del utilizzo di questo algoritmo riguarda il fatto di chiudere la finestra di congestione ad un segmento e iniziare con la fase Slow Start. Questa soluzione diventa troppo "aggressiva" inducendo a significativi cambiamenti (ritmo di invio, RTT, dimensione del buffer di rete, ecc.), con conseguenti variabilità nelle perdite di pacchetti. Dobbiamo ricordare che quello che interessa è massimizzare l'utilizzo della rete e non degradare inutilmente il comportamento di TCP riducendo drasticamente il numero di pacchetti immessi in rete causando perdite di throughput significative.

2.1.3 Versione TCP Reno

TCP Reno è la versione che introduce l'algoritmo chiamato **Fast Recovery,** comportando una differente reazione del mittente a seconda che si verifiche la ricezione di 3 ACK duplicati o lo scadere del TO. Analizzando infatti 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 TO, mentre la ricezione di 3 o più ACK duplicati indica probabilmente che un pacchetto è stato ricevuto corrotto, o non è stato ricevuto affatto, a causa del rumore del canale. Quindi il fatto che comunque qualcosa arrivi al ricevitore rende difficile pensare ad una rete congestionata. Invece, lo scadere del TO indica che il ricevitore non ha ricevuto nulla per un lungo periodo di tempo, ossia proprio quello che si verifica quando una rete è congestionata ed uno o più nodi della rete hanno una fila di attesa molto lunga.

Quindi, l'algoritmo di **Reno** fa una netta distinzione tra i due casi:

- In caso di perdita dovuta al TO, il TCP Reno lo interpreta come una probabile congestione e assume che la rete non è in grado di far passare nessun altro pacchetto. In questo senso, si comporta come il TCP Tahoe, cioè, pone la *CongWindow= 1 segmento*, la soglia *SSThresh= max (FlightSize/2, 2*SMSS)* e si porta in fase Slow Start.

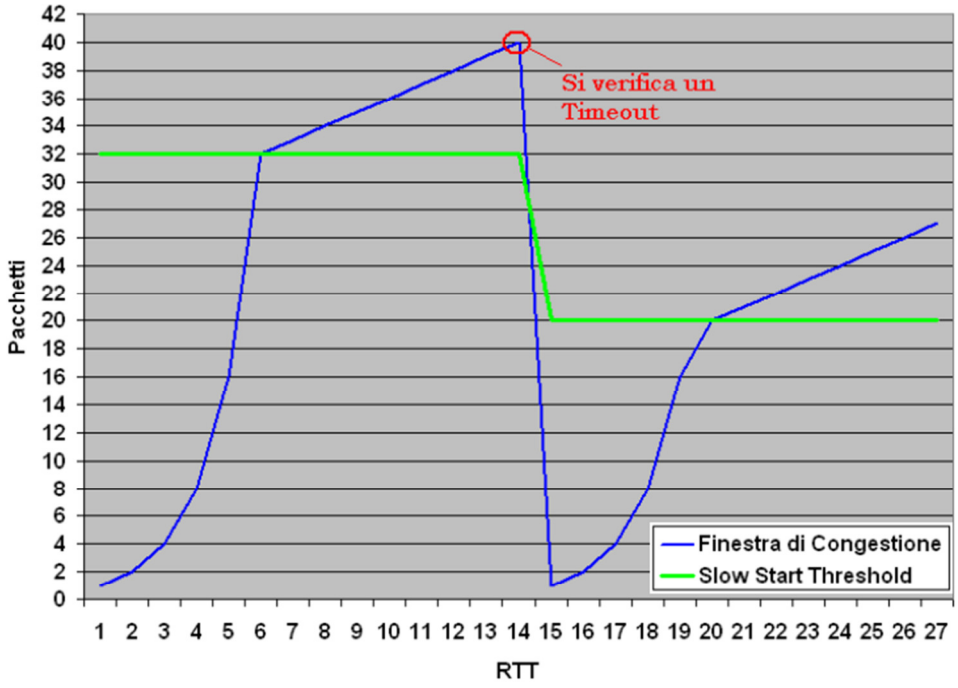


Figura2.6: Andamento della finestra di congestione nel caso di perdita dovuta al TO

- Quando invece l'evento perdita è generato dalla ricezione di 3 ACK duplicati, il TCP Reno assume che la rete è ancora in grado di trasferire qualcosa e quindi:

  1. Il valore soglia *(SSThresh)* viene impostato come:

  $$SSThresh = max\ (FlightSize/2,\ 2*SMSS),$$

  2. Una volta fatta la ritrasmissione del segmento perso, la variabile *CongWindow* viene incrementata per tenere conto dei tre segmenti giunti a destinazione fuori ordine e viene impostata come:

  $$CongWindow = SSThresh + 3xSMSS,$$

  A questo punto, la *CongWindow* viene aumentata di 1 segemento per ogni ulteriore ACK duplicato che viene ricevuto. Se le condizioni espresse dal nuovo valore della *CongWindow* e di quello della *RcvWindow* lo consentono, viene inviato un nuovo segmento. Una volta ricevuto l'ACK non duplicato (che riscontra tutti i segmenti spediti successivamente al pacchetto perso), la *CongWindow* viene di nuovo abbassata al valore di *SSThresh* e il ritmo viene rallentato, portandolo al tipico incremento lineare di Congestion Avoidance.

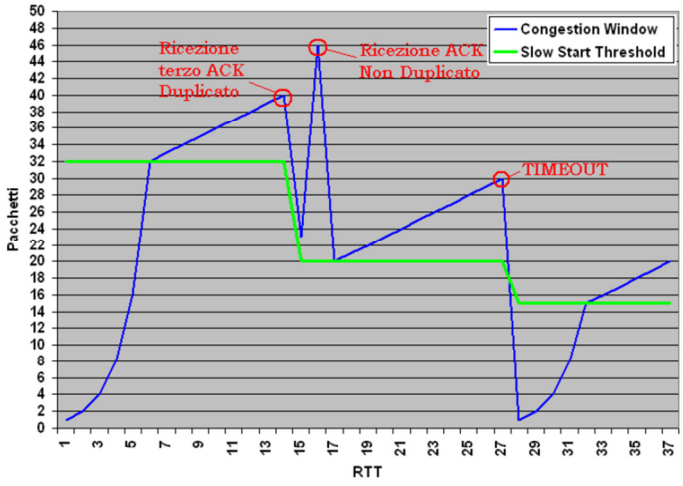


Figura2.7: Andamento della finestra di congestione nel caso di ricezione di 3 ACK duplicati

Questo algoritmo risulta molto efficiente nel caso in cui un singolo segmento venga perso o arrive corrotto dal rumore. Infatti questo viene immediatamente ritrasmesso all'arrivo del terzo ACK duplicato. Poi, impostando *CongWindow= SSThresh + 3xSMSS* e aumentando il ritmo di un segmento per ogni ACK duplicato, si permette di ritrasmettere tutti i dati già trasmessi ma successivi al pacchetto perso e arrivare così in breve tempo a trasmettere un nuovo pacchetto.

A causa della sua semplicità e suoi livelli di prestazioni forniti, Reno è in genere lo standard più utilizzato per il controllo della congestione TCP nella maggior parte dei sistemi operativi installati. Tuttavia, ci sono una vasta gamma di ambienti di rete in cui Reno ha prestazione insufficiente. Ad esempio, nel caso in cui si perdano o arrivono corrotti due o più segmenti della stessa finestra di trasmissione.

Per capire cosa accade vediamo la seguente figura:

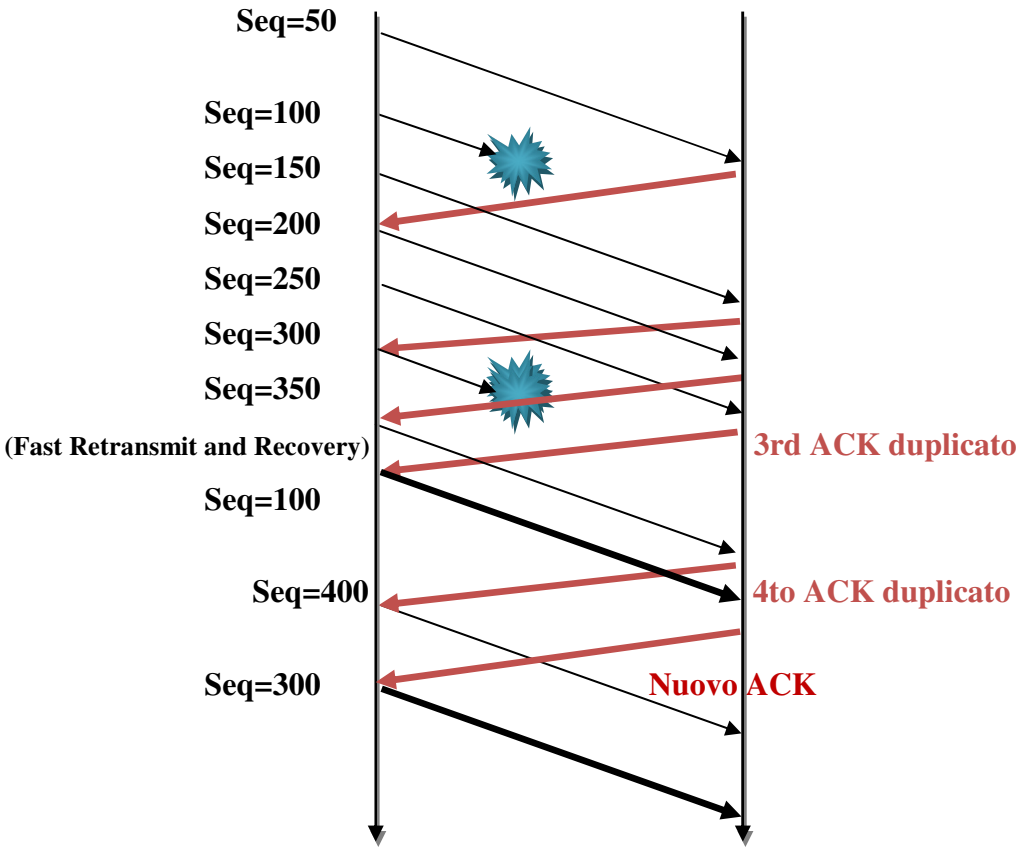


Figura2.8: Perdite multiple di pacchetti in una singola finestra di dati

Come si vede vengono persi due pacchetti nella stessa finestra, il numero 100 e il numero 300. Al terzo ACK duplicato per il pacchetto 100, il TCP Reno avvia la Fast Retransmit e la Fast Recovery, ritrasmettendo il pacchetto 100, portando la *CongWindow= SSThresh + 3xSMSS* e aumentando il ritmo di 1 SMSS per ogni ulteriori ACK duplicati. Quando riceve il primo ACK non duplicato (che in questo esempio riscontra fino al pacchetto 250), entra in Congestion Avoidance, rallentando il ritmo di invio a *CongWindow = CongWindow + 1/CongWindow*.

Con il ritmo di invio di Congestion Avoidance passerà una notevole quantità di tempo prima che vengono ricevuti gli altri tre ACK duplicati che segnaleranno al mittente la perdita del pacchetto 300, causando spesso lo scadere del TO dello stesso.

Per risolvere questo problema furono apportate delle modifiche che portarono allo sviluppo di una nuova versione chiamata TCP NewReno.

2.1.4 Versione TCP NewReno

TCP NewReno cerca di migliorare le prestazioni introducendo alcune modifiche al TCP Reno. Tali modifiche riguardano l'impostazione del valore di *SSThresh* e il miglioramento della reazione del protocollo nel caso in cui vengono persi due o più segmenti all'interno della stessa finestra.

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, mentre noterà la mancanza degli altri solo allo scadere del TO.

Per risolvere questo problema, TCP NewReno introduce il concetto di "*ACK parziali"*, ossia gli ACK che riscontrano pacchetti intermedi, e non gli ultimi pacchetti che necessiterebbero riscontro, dopo che è stata già iniziata la fase di Fast Recovery.

Il meccanismo degli ACK parziali richiede che venga memorizzata dal mittente una variabile speciale di stato *(recover),* per ricordare il numero di sequenza dell'ultimo pacchetto dati che ha inviato.

Quindi, le differenze rispetto all'algoritmo tradizionale di Fast Retransmit/Fast Recovery risiedono quindi nella possibilità di utilizzare questi ACK parziali e di tenere aggiornata la variabile *recover*. Fast Recovery termina quando arriva un ACK completo, quindi con la conferma di un numero di sequenza non inferiore a quello contenuto nella variabile *recover*, oppure alla scadenza del TO.

Inizialmente, il TCP NewReno si comporta come il TCP Reno quando riceve 3 ACK duplicati, portando:

$$SSThresh = max\ (FlightSize/2,\ 2*SMSS),$$

e la finestra di congestione a:

$$CongWindow = SSThresh + 3xSMSS.$$

Successivamente, se il primo ACK non duplicato che riceve riscontra tutti i segmenti inviati (incluso quello il cui numero di sequenza è memorizzato nella variabile *recover*), allora si esce dallo schema di Fast Recovery e si provvede ad eliminare della *CongWindow* l'incremento precedentemente operato:

$$CongWindow = min\ (SSThresh,\ FlightSize + SMSS).$$

Nella RFC 2582, il valore della *CongWindow* viene abbassato al valore di *SSThresh* (stesso comportamento di TCP Reno).

Altrimenti, se il mittente riceve un ACK parziale che riscontra solo alcuni pacchetti inviati prima di entrare in Fast Recovery, il TCP NewReno interpreta questo fatto come una ulteriore perdita e ritrasmette immediatamente il pacchetto successivo a quello riscontrato dall'ACK parziale. Oltre alla ritrasmissione viene operata una riduzione della *CongWindow* corrispondente al numero di segmenti

giunti a destinazione (indicati dall'arrivo dell'ACK parziale) ed un aumento di un segmento per tenere conto dell'arrivo a destinazione dei dati che hanno generato l'ACK parziale. Tale ridimensionamento della *CongWindow* viene fatto in modo di assicurare che alla fine del Fast Recovery una quantità di dati circa pari a *SSThresh* sarà ancora in sospeso nella rete. Inoltre, ma solo per il primo ACK parziale giunto al mittente, viene anche re-inizializzato il timer di ritrasmissione. Finalmente, si rimane nella fase di Fast Recovery, aumentando la *CongWindow* di 1 segemento per ogni ulteriore ACK duplicato che viene ricevuto e se le condizioni espresse dal nuovo valore della *CongWindow* e di quello della *RcvWindow* lo consentono, viene inviato un nuovo segmento.

Per capire meglio vediamo il seguente esempio:

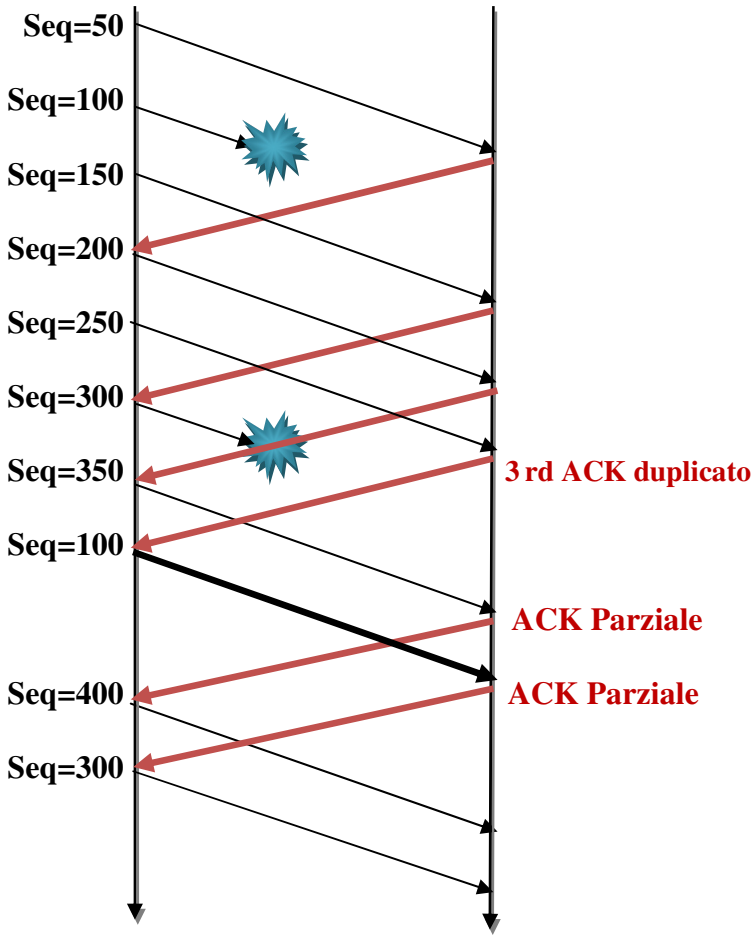


Figura 2.9: TCP NewReno

La ricezione del ACK parziale riscontra unicamente i pacchetti 200 e 250. Il TCP NewReno ritrasmette allora subito il pacchetto 300, ridimensiona la finestra e resta nella fase di Fast Recovery aumentando di un segmento la *CongWindow* per ogni ulteriore ACK duplicato relativo al secondo pacchetto perso.

Questo algoritmo entrerà in Congestion Avoidance solo quando sia stato ricevuto l'ACK che riscontra tutti i pacchetti inviati prima di entrare in Fast Recovery.

Perdite multiple all'interno di una singola finestra di invio potrebbero inutilmente far invocare tantissimi volte la procedura di Fast Retransmit. Per risolvere questo problema sono state proposte alcune modifiche al TCP NewReno originale. In particolare, si utilizza una nuova variabile, denominata *send_high*, il cui valore iniziale corrisponde al primo numero di sequenza inviato e nella quale, dopo ogni TO, viene memorizzato il numero di sequenza più elevato tra quelli trasmessi. Inoltre, se il mittente riceve tre ACK duplicati ed ancora non si trova in modalità Fast Recovery, si verificano i numeri di sequenza riconosciuti dagli ACK duplicati. Qualora tali numeri di sequenza risultano inferiori a quello memorizzato nella variabile *send_high*, il mittente non compie alcuna azione. Viceversa, la variabile *SSThresh* viene impostata secondo l'equazione:

$$SSThresh = max\ (FlightSize/2,\ 2xSMSS),$$

il numero di sequenza più elevato tra quelli dei dati trasmessi viene memorizzato in *recover* e si continua poi con la procedura precedentemente illustrata.

Oltre a ciò, l'algoritmo NewReno potrebbe comunque provocare l'invocazione dell'algoritmo di Slow Start, ad esempio nel caso in cui siano stati persi tantissimi pacchetti all'interno della stessa finestra di invio oppure se il TO è impostato di poco superiore al valore di RTT.

2.1.5 Versione TCP Vegas

TCP Vegas parte da un approccio diverso rispetto a TCP Reno e TCP NewReno. Questo protocollo di trasporto include modifiche alla strategia di ritrasmissione basata su una misura del RTT più accurata e l'utilizzo di nuovi meccanismi per la rilevazione della congestione durante Congestion Avoidance e Slow Start. Si propone infatti non di reagire ad un evento di perdita ma di rallentare il tasso di trasmissione dei dati prima che se verifiche uno. In altre parole, cerca, di evitare che si crei una congestione della rete piuttosto che prendere provvedimenti quando la congestione si è già verificata.

TCP Vegas mantiene le regole di equità nei confronti degli altri utilizzatori della rete, correggendo in maniera moltiplicativa la frequenza di invio quando si rileva una congestione (applica infatti Slow Start quando si rileva un TO). Inoltre, è in grado di operare con qualsiasi altra valida implementazione del TCP e le modifiche richieste si limitano al mittente.

TCP Vegas si basa sull'unione di tre tecniche. La prima riguarda un nuovo meccanismo di ritrasmissione avente lo scopo di reagire più rapidamente alla perdita di un segmento. La seconda realizza un meccanismo per anticipare la congestione e aggiustare di conseguenza il ritmo di invio dei dati. La terza tecnica modifica la procedura di Slow Start, in modo da evitare possibili perdite nella fase iniziale.

Vegas estende il meccanismo di ritrasmissione della versione Reno tenendo traccia dei tempi di arrivo degli ACK e stimando in questo modo i RTT. All'arrivo di un ACK duplicato, se la differenza tra il tempo corrente ed il *timestamp* (istante di partenza del pacchetto) risulta maggiore del TO, allora il segmento mancante viene immediatamente ritrasmesso senza dover aspettare l'arrivo del terzo ACK duplicato come visto, ad esempio, in Reno. Inoltre, il tempo che intercorre tra l'invio di un pacchetto e la ricezione dell'ACK viene verificato anche per l'arrivo dei primi due nuovi ACK successivi all'invio dei dati nel caso che si tratti di una ritrasmissione. Se questo intervallo di tempo è più ampio del valore di TO, allora TCP Vegas ritrasmette un altro segmento. Con

questo nuovo meccanismo si riesce a ridurre il tempo di reazione del protocollo alle perdite di segmenti.

Il secondo meccanismo innovativo implementato da TCP Vegas è quello che gli consente di prevenire le congestioni senza dover aspettare di rilevare delle perdite nei segmenti inviati. L'idea fondamentale è quella di avere la banda pienamente utilizzata e tenere sotto controllo la quantità di occupazione dei buffer di rete lungo il percorso. Infatti buffer troppo pieni provocherebbero perdite di segmenti ma, allo stesso tempo, buffer troppo vuoti non permetterebbero opportuni adattamenti della frequenza trasmissiva in situazioni di aumenti della banda disponibile. Dunque, è necessario che il mittente conosca in ogni istante lo stato della rete. Vegas cerca di quantificare, non un parente, ma un numero assoluto di pacchetti accodati nei buffer di rete in funzione della velocità di trasmissione prevista e quella effettiva. Per fare questo, osserva i cambiamenti del RTT dei segmenti già inviati. Se il mittente osserva che l'RTT diventa particolarmente ampio, deduce che la rete si sta per congestionare e diminuisce il ritmo di invio. Invece, se l'RTT risulta breve, significa che la rete non ha problemi e la finestra viene aumentata.

Per realizzare questo controllo, Vegas cerca di rilevare l'imminente congestione comparando, il throughput misurato con il throughput atteso. La finestra di congestione è incrementata solo se questi due valori sono vicini, cioè se c'è abbastanza capacità di rete tale che il throughput aggregato possa essere effettivamente raggiunto. Inoltre, viene ridotta se il throughput misurato è considerevolmente più basso di quello atteso, dato che questa condizione viene assunta come un segnale di congestione.

TCP Vegas utilizza alcune formule e parametri per fissare queste tassi:

- Calcolo del throughput atteso:

$$Expected = CongWindow / Base\_RTT,$$

dove *CongWindow* è il valore attuale della finestra di congestione e *Base_RTT* è il più piccolo RTT misurato per i segmenti inviati.

Generalmente il segmento con il RTT minimo corrisponde al primo inviato, poiché i router non devono ancora sopportare il peso della nuova connessione.

- Una ulteriore operazione è quella di calcolare il throughput reale una volta ogni RTT, rilevando il tempo trascorso tra l'invio di un segmento contenente un certo ammontare di dati ed il ricevimento del corrispondente ACK, e dividendo poi il tempo impiegato per i byte trasmessi. In pratica, un modo semplice per stimare questo parametro è tramite il calcolo della velocità di trasferimento attuale utilizzando il valore corrente dell'RTT:

  Actual = CongWindow / RTT,

Chiamiamo *diff = (Expected – Actual),* la differenza tra questi due tassi (il throughput atteso e throughput reale).

Questa differenza, tramite opportuni aggiustamenti lineari della frequenza reale di invio dei segmenti, deve rimanere sempre all'interno di un intervallo prestabilito:

$$\alpha < diff < \beta$$

dove α e β sono due costanti con valori tipici 1 e 3 o 2 e 4, che devono rappresentare rispettivamente la situazione di avere troppi oppure pochissimi pacchetti nei buffer di rete lungo il percorso verso il ricevente.

Quando la rete è congestionata i nodi creano delle code dove mettono in attesa i pacchetti che arrivano. Basandosi sulla formula de Little possiamo ricavare una stima del numero N di pacchetti presenti in una coda. Così otteniamo:

$$RTT = Base\_RTT + N/Actual,$$

da qui possiamo trovare N:

$$N = Actual \times (RTT - Base\_RTT) = diff \times Base\_RTT.$$

In particolare, l'aggiornamento della finestra viene fatto come segue:

- Se diff < α → CongWindow = CongWindow +1;
- Se $\alpha \leq diff \leq \beta$ → *CongWindow* = *CongWindow* (il sistema è considerato essere in uno stato costante e non viene applicata nessuna modifica alla finestra di congestione);
- Se diff > β/ Base_RTT → CongWindow = CongWindow –1;

Tale finestra viene controllata per tenere una quantità di pacchetti (N) all'interno dei limiti [α, β].

L'ultimo meccanismo introdotto da TCP Vegas riguarda alcune modifiche da apportare alla procedura di Slow Start. Cerca di dedurre la banda disponibile di una connessione in modo tale da introdurre un meccanismo di "Congestion Detection" anche in questa fase. In pratica, per essere in grado di rilevare ed evitare una congestione incipiente durante tale fase (dovuta alla crescita troppo veloce della *CongWindow* e, conseguentemente, del traffico generato, con il rischio di oltrepassare la banda a disposizione), Vegas consente una crescita esponenziale solo a RTT alterni. Tra un RTT e l'altro la finestra di congestione rimane ad un valore fisso in modo tale da poter fare un confronto tra il throughput atteso e quello attuale. Quando il throughput missurato si trova, di una certa quantità (la dimensione di un segmento) al di sotto di quello atteso, significa che si è in una situazione di congestione, motivo per cui Vegas passa direttamente alla fase di Congestion Avoidance.

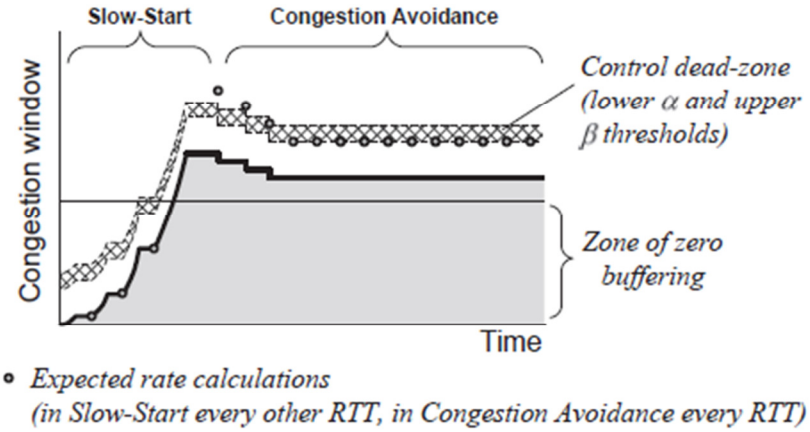


Figura 2.10: Andamento della finestra di congestione nel TCP Vegas

TCP Vegas, grazie alle sue capacità predittive offre uno schema per evitare il verificarsi di perdite dovute a congestione. Le versioni, Reno e NewReno del TCP inviano pacchetti sulla connessione con una frequenza sempre più elevata, attendendo di rilevare una perdita per comprendere quando la rete è divenuta congestionata e ridurre così la frequenza di invio. Si può quindi dire che questi protocolli sono ***reattive***, poiché hanno bisogno di creare perdite per trovare la banda disponibile della connessione. TCP Vegas invece utilizza un approccio ***proattivo*** cercando di percepire la congestione osservando i cambiamenti di throughput, quindi definisce una politica di aggiustamento della finestra di congestione con l'obiettivo di controllare il ritmo di invio del mittente prima che siano esperimentati perdite.

D'altra parte però non è detto che il metodo utilizzato sia quello giusto, ovvero il più adatto per stimare la banda disponibile sul canale. Come tutti i protocolli di trasporto che effettuano la stima della banda nel mittente, TCP Vegas soffre in situazioni di asimmetria del percorso. Se infatti le capacità del canale dal mittente al ricevente sono diverse da quello inverso, non è detto che la misura del RTT costituisca un preciso indicatore del livello di congestione del percorso in avanti.

La versione Vegas del TCP non ha ricevuto molti consensi, soprattutto a causa del suo comportamento in presenza di altri terminali che utilizzano TCP Reno o New Reno. In questo caso, l'algoritmo Vegas viene "asfissiato" dalle altre versioni più "aggressive" che finiscono con l'occupare tutta la banda disponibile. Esso invece si "sacrifica" per non creare congestione riducendo continuamente il suo ritmo di invio, mentre che gli altri stanno approfittando la maggior banda disponibile.

TCP Vegas non è inoltre in grado di reagire adeguatamente a perdite di tipo diverso da quelle dovute a congestione, situazioni nelle quali viene ugualmente invocata la procedura di Slow Start.

2.1.6 Versione TCP BIC e CUBIC

Nel tentativo di creare un meccanismo ottimo di controllo di congestione ed in grado di scalare bene in qualsiasi rete con elevato BDP, è stato progettato TCP BIC con l'obiettivo di rinnovare le potenzialità del prtocollo TCP/IP e diventare una delle basi più solide su cui costruire l'Internet del futuro.

Questo algoritmo è una estensione di NewReno con una fase operativa supplementare (*Rapid Convergence*). Questa fase scopre rapidamente, attraverso una ricerca binaria (Search Binary), le dimensioni ottimali della finestra di congestione (il valore corrispondente alle risorse di rete disponibili), basandosi sulla rilevazione di una perdita di pacchetti come indicazione di superamento di tale finestra.

TCP BIC definisce quattro parametri fondamentali:

- ***Minimum window size (Wmin)****:* è la dimensione minima della finestra di congestione (corrispondente ad una velocità di trasmissione senza perdita di pacchetti). Inizialmente è impostata ad uno.
- ***Maximum window size (Wmax)****:* è la dimensione massima della finestra di congestione (corrispondente alla massima velocità di trasmissione consentita dalla rete). Inizialmente viene impostata ad un valore arbitrario di altezza.
- ***Target window size (Tw)****:* è il valore medio tra la *Minimum window size* e la *Maximum window size*.
- ***Current window size****:* è la dimensione della finestra di congestione corrente, il cui obiettivo è raggiungere la grandezza della "target window".

Allora, l'andamento della finestra nel TCP-BIC può essere rappresentata così:

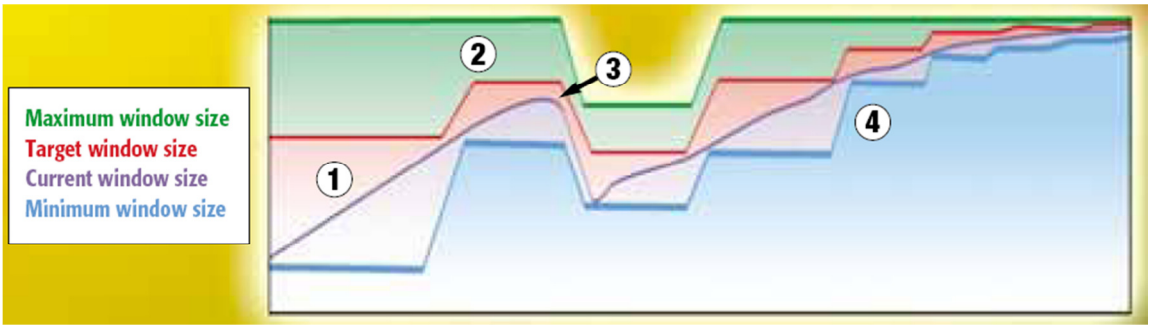


Figura 2.11: Andamento della finestra nel TCP BIC

Questo protocollo è costituito da due algoritmi:

• crescita "additiva";

• crescita "binary search":

Crescita "additiva"

Come già anticipato, la *Wmin* può essere stimata come la dimensione della window alla quale non si verifica nessuna perdita di pacchetti. Se la *Wmax* è conosciuta, allora possiamo fissare la dimensione della "target window": che in BIC viene posta a metà tra il valore massimo e quello minimo.

Mentre la rete gestisce con successo i pacchetti dati (ad esempio, il mittente riceve tutti i riscontri durante l'ultimo RTT) e quando tra la dimensione della *Wmin* e il punto intermedio *(Tw),* c'è una differenza elevata, la finestra di congestione corrente cresce in modo agressivo (esponenziale) con l'intento di raggiungere al punto medio del range di ricerca *(Tw).*

Tale strategia di incremento della finestra viene denominata "*additive increase"* (corrispondente al **1** nella figura).

Anche se un vero algoritmo di ricerca binaria dispone di un tempo di convergenza molto veloce, in una rete con elevato BDP si può creare lo stesso problema che è stato scoperto in Slow Start, cioè se la finestra di congestione è aumentata troppo veloce, un gran numero di pacchetti possono essere persi. Per

questo motivo, aumentare la dimensione della window direttamente al punto intermedio potrebbe comportare una crescita troppo aggressiva.
Quindi, in alcuni riferimenti, viene fissato un valore S_{max} che limite superiormente l'incremento della dimensione della finestra corrente, cioè, invece di aumentare tale finestra direttamente al punto intermedio al prossimo RTT, si aumenta di un valore pari a S_{max} finché la distanza tra *Wmin* e *Tw*, non sia inferiore di S_{max}. Una volta superato questo vincolo, l'incremento torna ad essere al punto intermedio.

Crescita "binary search"

Una volta che la finestra corrente ha raggiunto il target senza che ci sia stato alcun evento di perdita di pacchetti, si passa successivamente alla fase *"binary search"*.

Allora, se non si verificano perdite, *Wmin* aumenta fino al valore della *Tw*, la quale adesso è impostata al valore medio tra la nuova finestra minima e la massima (rimasta fissa al valore iniziale). Successivamente la finestra corrente continua a crescere per raggiungere la nuova finestra *Target*.

Questa tecnica è chiamata crescita *"binary search" (corrispondente al **2** nella figura)* e permette una ricerca della massima larghezza di banda. Inoltre, ne consegue che la funzione di crescita della dimensione della window è di tipo logaritmico, ovvero la pendenza della funzione gradatamente decresce quando si è vicini al valore di target.

Se si verificano perdite di pacchetti (per esempio, tre ACK duplicati vengono ricevuti), la dimensione corrente della window diventa il nuovo massimo *(corrispondente al **3** nella figura)* e la finestra ridotta (fino al valore assunto della finestra corrente) dopo la perdita, sarà il nuovo minimo. La metà tra questi nuovi valori diventerà il nuovo target.

La giustificazione di questo approccio è che la rete tipicamente è soggetta a perdite nell'intorno del nuovo massimo ma non nell'intorno del nuovo minimo.

Tipicamente il numero di pacchetti perduti è proporzionale alla dimensione dell'ultimo incremento della finestra prima della perdita.

La combinazione di queste due tecniche (crescita "additiva" + crescita "binary search") viene detta "*binary increase*".

"Max Probing"

In quest'ultima fase (**4** in figura), la finestra corrente ricomincia a crescere ma in modo meno aggressivo finché la differenza tra la finestra massima e minima è circa nulla. In tale modo si riesce a raggiungere la velocità massima consentita dalla rete senza eventi di perdita di pacchetti.

Le fasi descritte (*"additive increase"* + *"binary search"* + *"max probing"*), possono essere rappresentate nella figura 2.12, in cui si mostra l'andamento temporale della finestra di congestione (ovvero la finestra corrente vista in precedenza):

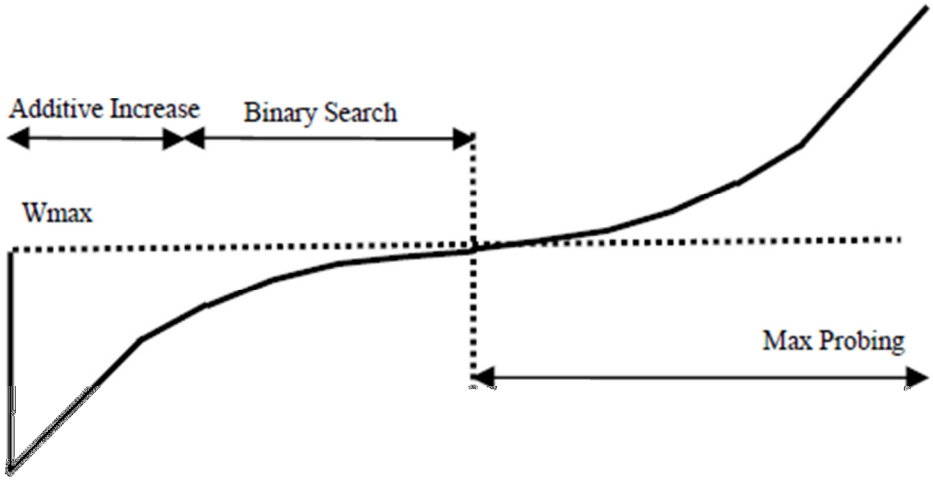


Figura 2.12: Comportamento della *CongWindow* in TCP BIC

Tale incremento della *CongWindow* può essere sintetizzato teoricamente come segue:

CongWindow =

$$\begin{cases} CongWindow + \frac{1}{CongWindow} & CongWindow < Wmin \\ CongWindow + \frac{Tw - CongWindow}{CongWindow} & Tw - CongWindow < Smax \\ CongWindow + \frac{Smax}{CongWindow} & Tw - CongWindow > Smax \\ CongWindow + \frac{Smin}{CongWindow} & CongWindow > Wmax \end{cases}$$

Sebbene BIC ha buone prestazioni (scalabilità, correttezza ed stabilità) in ambienti ad alta velocità, il suo comportamento, potrebbe ancora risultare troppo "aggressivo", in particolar modo in reti con bit rate non molto alti e per piccoli RTT.

TCP CUBIC, a differenza dell'algoritmo BIC, porta ad una crescita della finestra di congestione molto più lenta, attraverso una funzione cubica descritta in figura:

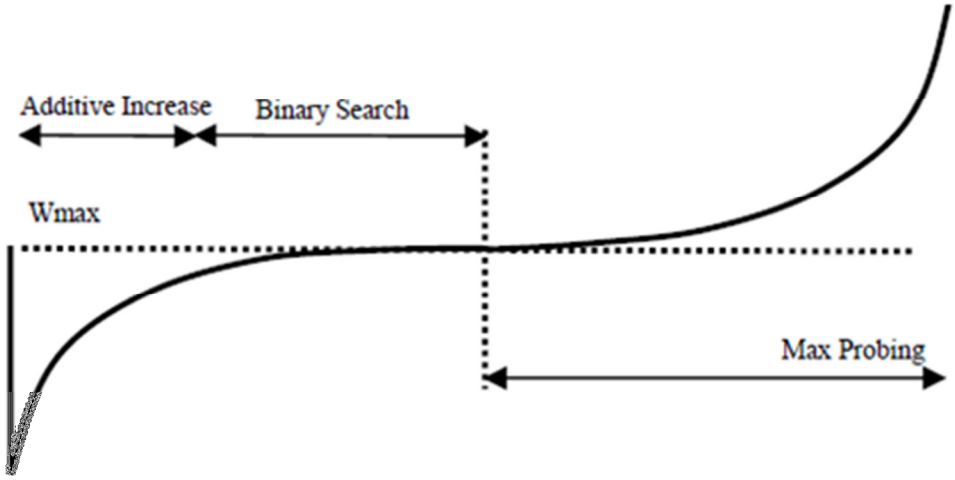


Figura 2.13: Comportamento della *CongWindow* in TCP CUBIC

In effetti, mentre la fase di *additive increase* nei due algoritmi è praticamente identica, la differenza principale concerne le fasi di "*binary search*" e "*max probing*". Nella prima, dopo aver raggiunto il valore Wmax, la finestra di congestione continua a crescere più lentamente rispetto al BIC, rendendo il protocollo più stabile e aumentando l'utilizzo di rete. Nella seconda (nelle zone in cui si hanno valori della finestra corrente distanti da quella massima), invece, la

velocità di crescita della finestra di congestione risulta molto più veloce e ciò comporta una maggiore scalabilità del protocollo.

Nello specifico, la dimensione della finestra corrente *Wcubic* è determinata dalla seguente funzione:

$$W_{cubic} = C(t - K)^3 + W_{max};$$

Dove: *"C"* è un fattore di scala, *"Wmax"* è la dimensione della finestra prima dell'ultima riduzione della stessa, *"t"* è il tempo trascorso dall'ultimo decremento e *"K"* è pari a: $\sqrt[3]{(Wmax * \beta)/C}$, dove β è un fattore moltiplicativo di decrescita dopo un evento di perdita di pacchetti.

Per capire meglio tale dinamica teorica della *CongWindow* vediamo il seguente esempio:

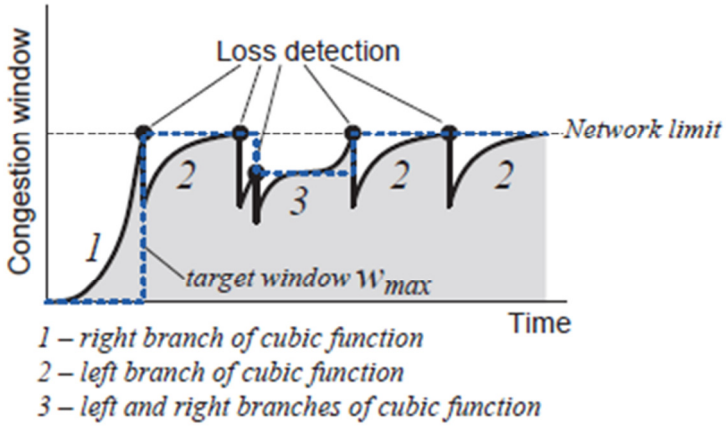


Figura 2.14: Andamento della finestra di congestione nel TCP CUBIC

Nella fase iniziale (**1**), la dimensione della massima finestra di congestione è sconosciuta e si scopre attraverso la funzione cubica. Questa scoperta è molto più prudenti della scoperta esponenziale (Slow Start) di utilizzo tradizionale. Nelle fasi successive, a seguito di una riduzione della finestra al rilevamento di una perdita di pacchetti, la finestra di congestione si avvicina con delicatezza al valore Wmax (fase **2**). Se viene rilevata una perdita prima di raggiungere il limite superiore, Wmax è aggiornata al valore della finestra corrente. Adesso, se fosse

stato rilevato un evento di congestione temporaneo, vedremo una crescita della finestra di congestione a entrambi lati (destro e sinistro) di tipo cubica (fase **3**).

Una importante differenza con gli altri standard TCP è che CUBIC non si basa sulla ricezione degli ACKs per aumentare le dimensioni della finestra, tale dimensionamento dipende unicamente dell'ultimo evento di congestione. D'altra parte come abbiamo visto, in altri standard TCP, i flussi con un RTT molto breve riceveranno degli ACK più velocemente e quindi le sue finestre di congestione cresceranno più rapidamente rispetto ad altri flussi con RTT più grande. Nonostante, CUBIC garantisce: TCP Fairness (trattamento equo di flussi distinti con stesso RTT), RTT Fairness (trattamento equo per flussi con RTT diverso).

CUBIC fornisce un meccanismo per assicurare che le prestazioni non siano peggiore rispetto alla prestazione dallo standard Reno. Questo meccanismo prevede il calcolo di una finestra di congestione supplementare *WReno*. Poiché la finestra di congestione in CUBIC può essere ridotta di un fattore β (in genere diverso a quello in CUBIC), il funzionamento adeguato per no peggiorare le prestazioni si trova aumentando la finestra di un fattore:

$$s = 3 \times (\beta - 1) / (\beta + 1)$$

Formalmente, questo può essere scritto come un aumento della *WReno* da s ogni RTT. Se CUBIC rileva che la finestra *WReno* supplementare supera la finestra principale, questa viene impostata al suo valore precedente.

Per quanto riguarda la dimensione massima della finestra di congestione, essa dipende dallo spazio che il kernel alloca nella memoria per i socket buffer di ricezione e trasmissione (valore di default 64KByte).

Il kernel è forse la parte più importante del sistema operativo. In sostanza si tratta di un codice che si occupa della gestione delle risorse presenti nel computer (hardware) e le rende direttamente accessibili agli altri programmi che girano sul PC. In oltre parole, è un mediatore che serve a far interagire il software con l'hardware.

Il kernel di Linux potrebbe anche limitare la dimensione della finestra di congestione attraverso il meccanismo di *"CongWindow moderation"* che, in ogni istante, si basa su una stima della quantità di pacchetti in viaggio verso il ricevente, dei quali il mittente non ha ancora ricevuto riscontri. Grazie a tale meccanismo, non viene ecceduto il BDP e vengono minimizzati fenomeni di elevato ritardo, causate dall'accumulo di dati nel socket buffer lato mittente.

Tuttavia, CUBIC non ha il 100% di utilizzo delle risorse di rete e può indurre un elevato numero di perdite di pacchetti nella rete (sempre che una perdita sia l'unico meccanismo di segnalazione).

2.2 Implementazione TCP/IP nei sistemi operativi più comuni

2.2.1 Windows XP

- L'algoritmo utilizzato in Windows XP è di tipo TCP Reno.
- Il suo valore di default è basato sulla velocità di linea dell'interfaccia mittente e cambia anche a seconda della versione del sistema operativo utilizzato (XP SP2 e le versioni precedenti adottano un valore di default di 17520byte, mentre il XP SP3 utilizza un valore di default pari a 65535byte).
- Finestra di ricezione fissa per ogni connessione. Questa considerazione può comportare una limitazione in termini di sfruttamento della banda a disposizione, pertanto, una limitazione importante del throughput.
- Essendo la dimensione della finestra fissa, non segue le possibili variazioni del tasso di estrazione dell'applicazioni (rischiando di mandare in overflow il buffer di ricezione del destinatario) o della congestione del canale di trasmissione.

2.2.2 Windows Seven

Per ottimizzare le prestazioni del TCP, specialmente per i canali di trasmissione ad alta capacità, nasce uno stack "*TCP/IP Next Generation",* ovvero

una riprogettazione completa delle funzionalità TCP/IP per i protocolli IPv4 e IPv6 in grado di soddisfare le esigenze di connettività e prestazioni nelle reti attuali. Tra le nuove funzionalità introdotte abbiamo le seguenti:

- Uso dell'algoritmo New Reno;
- Regolazione automatica della finestra di ricezione ("Auto-Tuning");
- TCP composto (CTCP);
- Miglioramenti per ambienti caratterizzati da un'elevata perdita di pacchetti;

La funzione di ***"Auto-Tuning",*** permette determinare la dimensione ottimale della finestra di ricezione per ogni connessione tendo in considerazione i valori del BDP sia il tasso di estrazione dell'applicazione lato destinatario. Con tale ottimizzazione, un'entità TCP in Windows Seven è in grado, quindi, di utilizzare dimensioni della finestra molto più grandi di quanto avviene in Windows XP.

L'impatto sulla rete è che un flusso di pacchetti dati TCP che normalmente sarebbero inviati con frequenza più limitata sono immessi in rete molto più velocemente, con un picco di utilizzo più largo durante il trasferimento. Quindi, in Windows Seven viene determinato automaticamente un valore ottimale della finestra connessione per connessione (potendo sfruttare dimensioni molto più grandi), e ciò importa un notevole miglioramento delle prestazioni. L'entità TCP mittente, può dunque inviare molti più segmenti dati senza bisogno di aspettare un ACK (sempre tenendo presente i vincoli imposti dal controllo di congestione e dunque dal valore della finestra di congestione).

D'altra parte, per quanto riguarda il **TCP Composto *(CTCP)***, a differenza della regolazione automatica della finestra di ricezione che consente di ottimizzare la velocità effettiva in ricezione, questo consente di ottimizzare la velocità effettiva in trasmissione, aumentando in modo sensibile la quantità di dati inviati in un dato intervallo di tempo senza influire negativamente sulle altre connessioni TCP. Come TCP Vegas, CTCP utilizza le stime di ritardo in coda come una misura della congestione: se il ritardo è piccolo, si presuppone che nessun link sul

percorso sia congestionato, e allora il mittente aumenta rapidamente la sua velocità.

In Windows 7, inoltre, è previsto anche un meccanismo di supporto al protocollo TCP teso a ottimizzare le prestazioni per particolari contesti: l'*ECN (Explicit Congestion Notification).* In pratica, quando un segmento TCP è perso (si suppone che ciò sia avvenuto a causa della congestione dei router intermedi), viene fatto un apposito controllo che determina una riduzione notevole del ritmo di invio. Grazie al ECN, i router in cui si verifica la congestione contrassegnano i pacchetti al momento dell'inoltro. I nodi TCP, che ricevono tali pacchetti, sono in grado di ridurre la velocità di trasmissione per diminuire la congestione ed impedire la perdita di segmenti, consentendo di aumentare la velocità effettiva generale.

2.2.3 Linux

- Uso dell'algoritmo TCP CUBIC;
- Come in Windows Seven, anche nei sistemi operativi Linux (con kernel dal 2.4.16 e dal 2.6.6) è implementata la funzione di Auto-tuning della finestra di ricezione, quindi riesce ad aggiustare automaticamente la dimensione del socket buffer in modo da ottenere il migliore bilanciamento tra le prestazioni del TCP e l'utilizzo di memoria del sistema. Il suo funzionamento prevede, d'altronde, che il kernel tenti di aumentare la dimensione della finestra in modo da uguagliare quella del ricevente. Se però c'è una richiesta troppo elevata di memoria del kernel, la dimensione del buffer potrebbe essere limitata o diminuita.

CAPITOLO III

OTTIMIZZAZIONE DELLE PRESTAZIONI DEL PROTOCOLLO TCP

3.1 L'uso delle Opzioni

Attualmente il protocollo TCP rappresenta un metodo standard utilizzabile in molte situazioni per la consegna affidabile ed efficiente dei dati. A questo scopo, anziché inventarsi un proprio protocollo di trasporto, in genere le applicazioni utilizzano TCP perché ha ormai raggiunto una importante maturità ed è stato oggetto di molti miglioramenti per incrementare le prestazioni e l'affidabilità.

Abbiamo visto come il collo di bottiglia presente nella rete di accesso sia uno dei principali parametri che limita considerevolmente le prestazioni in termini di banda disponibile percepita. Tuttavia, l'idea che ha dato vita alla realizzazione di questo lavoro è che ci fossero altri fattori che avessero un impatto notevole sulle misure della QoS. Infatti, oltre alle caratteristiche hardware del PC, come il carico di memoria o la velocità del disco, abbiamo già anticipato l'importanza del protocollo di trasporto; in particolare, si è sottolineato come la dimensione della finestra di ricezione abbia un ruolo chiave nella valutazione delle prestazioni, considerando che regola direttamente il flusso dei dati.

Si tenga nota dei seguenti parametri:

- **MTU (Maximum Transfer Unit)** : è la dimensione massima (in byte) di un pacchetto TCP inviato, ovvero la più grande quantità dati che può essere trasferita in un frame fisico sulla rete.

Riguardo la scelta dell' MTU invece valgono le seguenti considerazioni:

Per chi usa il PPPoA (RFC 2364, solitamente i modem ADSL USB o PCI usano questo sistema) è consigliato fissare MTU=1500byte, ovvero il valore massimo per questo tipo di connessione.

Per chi usa il PPPoE (RFC 2516, quasi tutti i modem ethernet e i router), invece il valore massimo assegnabile è di 1492.

Nel caso si abbiano problemi, ad esempio può capitare con schede di rete vecchiotte, possiamo impostare un valore più basso come 1454 o 1300.

Note importanti:

La grandezza del nostro pacchetto viene “accettata” dal primo router che incontriamo sul camino e poi viene rilasciato verso Internet, con la grandezza assegnata.

La quasi totalità della rete, ha dei router che accettano pacchetti da 1500byte senza frammentarli. Se la dimensione del nostro segmento è diversa dalla dimensione che è capace di gestire il router, quest’ultimo è costretto a frammentare il pacchetto e spedirlo in parti differenti: quindi quello che prima era solo “un dato”, può diventare “più dati”, con un più che è indefinibile, perché se capita di avere una perdita in quel momento, possono diventare 4, 8, 16, oppure più segmenti.

Perciò, se abbiamo un router non dobbiamo dimenticare di settare gli stessi valori di MTU anche lì, evitando così che il pacchetto venga "spezzato" ancor prima che passi il nostro router.

Inoltre, alcuni sistemi operativi utilizzano una procedura di auto-ottimizzazione di pacchetti automatica che, se è attivata, tenta di scoprire qual è la dimensione massima di pacchetto che può essere inviata. Se invece, è disattivata, scelgono un MTU di 576 byte per tutte le connessioni.

D’altra parte, se vogliamo modificare manualmente i valori di MTU dobbiamo disattivare tale procedura e settare il valore predeterminato da noi.

- **MSS (Maximum Segment Size)**: è la dimensione massima di un segmento. Solitamente viene calcolata automaticamente a partire dei valori di MTU. Poiché l'intestazioni (ACKs) sono in genere di 40byte, MSS è di solito 40byte inferiore di un MTU, ovvero:

  $$MSS = 1500 - 40 = 1460byte$$

- **TTL (Time To Live)**: questo valore è impostato nell'intestazione del pacchetto IP. Determina il numero massimo di "punti" attraverso i quali un pacchetto può passare prima di essere considerato perso oppure prima di essere scartato.

  Un valore troppo piccolo può causare che i pacchetti non siano in grado di raggiungere ai server distanti, mentre che un valore molto grande, potrebbe richiedere di troppo tempo per riconoscere perdite di pacchetti. Solitamente questo parametro è impostato tra 64 e 128 e non influisce direttamente sulle prestazioni, perciò è generalmente inutile modificarlo.

Le reti e le comunicazioni sono di importanza cruciale per le organizzazioni che desiderano competere in modo efficace nel mercato globale e adattarsi alle esigenze attuali della società. A tale scopo, le reti di oggi supportano diverse opzioni TCP utilizzate con l'obiettivo di ottimizzare la velocità effettiva e le prestazioni del sistema. Nell'uso di queste opzioni si deve fare attenzione alla compatibilità e all'overhead. Quindi bisogna verificare prima se essa, nelle condizione di utilizzo della rete, è realmente necessaria o se comporta solo uno spreco di banda e di tempo di calcolo.

3.1.1 TCP 1323 Options

- *TCP Window Scaling:* questa opzione permette aumentare la dimensione della finestra di ricezione sopra di 65535byte. Si consiglia di avere questa impostazione attivata se, e solo se, si vuole introdurre un valore più alto della *RcvWindow*.

- *Timestamp:* fornisce una misura più accurata del RTT evitando possibili errori. Tale opzione consente al mittente calcolare la differenza tra l'istante di partenza del segmento trasmesso e l'istante di partenza a cui il segmento di risposta (l'ACK) si riferisce.

  Lo svantaggio di usarlo è che aggiunge 12byte sulla intestazione TCP di ogni pacchetto facendo aumentare il tempo di processing dei singoli segmenti. Perciò, non sarebbe necessario abilitare questa impostazione quando la *RcvWindow* è uguale o minore a 65535. Invece, sembra utile quando la latenza varia molto (per esempio pura connessione satellitare).

I valori permessi sono compressi fra 0 e 3:

Tabella 1: TCP 1323 Options

TCP 1323 Opts	**Significato**
0	Disabilitate entrambe le opzioni
1	Abilitata solo l'opzione *Window Scaling*
2	Abilitata solo l'opzione *Timestamp*
3	Abilitate entrambe le opzioni

Questi valori possono essere modificati andando nel registro di configurazione dei vari sistemi operativi.

3.1.2 L'estensione SACK (Selective Acknowledgments)

Introdotta per la RFC 2018. È particolarmente utile per connessione che utilizzano grandi dimensioni della finestra TCP.

SACK permette al nodo mittente di identificare i blocchi consecutivi di dati correttamente ricevuti e di ritrasmettere di conseguenza solo i pacchetti persi tra un blocco e l'altro senza ridurre la finestra di congestione. Tale comportamento migliora significativamente la velocità effettiva complessiva.

La specifica RFC 2883 definisce un ulteriore utilizzo dell'opzione TCP SACK per confermare i pacchetti duplicati. Ciò consente al destinatario del segmento TCP contenente l'opzione SACK di determinare quando è stato ritrasmesso inutilmente un segmento e regolare il suo funzionamento per evitare che tale condizione si ripeta.

L'acknowledgment selettivo usa 2 opzioni TCP:

- *"SACK permitted"* è un opzione che può essere mandata nella fase di instaurazione della connessione (in un segmento di SYN) per indicare al ricevitore di utilizzare l'opzione SACK una volta stabilita la connessione.
- *"SACK"* è un opzione che viene inviata dal ricevitore quando si verifica una perdita dopo che il mittente in fase di instaurazione della connessione ha inviato l'opzione *"SACK permitted"*.

Il parametro che abilita o disabilita questa opzione è il *valore Dword* booleano *SackOpts* presente nel registro di configurazione del sistema operativo.

3.1.3 Autotuning

Il fatto che lo stack TCP/IP ha un'implementazione differente nei Sistemi operativi più diffusi ha spinto, in definitiva, a valutare anche l'eventuale impatto circa l'utilizzo di questi ultimi. Per esempio mentre in Windows XP l'implementazione TCP era stata pensata per reti a prestazioni limitate, in Windows 7 e Ubuntu si sono considerate reti ad elevate prestazioni. Uno dei punti focali su cui è caduta l'attenzione, quindi, è stata la funzionalità di Autotuning, presente in Windows 7 e Linux.

L'autotuning serve a monitorare e ad ottimizzare la connessione, quindi teoricamente, a migliorare le prestazioni. Tale regolazione automatica della finestra di ricezione intenta scoprire la dimensione "ottimale" della *RcvWindow* per ogni connessione, misurando il BDP e la velocità di recupero dell'applicazione. In teoria porta all'aumento delle performance di rete, ma se, e solo se, l'apparato (modem/router) supporta anch'esso questo autotuning, ovvero è compatibile con questa soluzione.

Visto che spesso la maggioranza dei router sono un po' meno malleabile che i sistemi operativi e, visto che questo meccanismo può introdurre ulteriori ritardi sulla rete durante la stima dei parametri necessari per l'aggiustamento della finestra, alcune fonti sostengono che con questa procedura il sistema può trovare dei problemi e andare peggio, rallentando il trasferimento dei dati o causando la perdita di connettività. D'altra parte, altre fonti considerano che questa opzione indubbiamente fornisce un contributo notevole al miglioramento delle performance TCP.

Questa soluzione può essere abilitata o disabilitata andando nel registro di configurazione del sistema operativo.

3.1.4 CTCP (TCP Composto)

È un algoritmo introdotto in Windows Vista e implementato anche in Windows seven progettato a regolare e ad ottimizzare la velocità effettiva in trasmissione. Tale elemento consente di aumentare l'utilizzo del collegamento e garantire vantaggi sostanziali in termini di prestazioni per quanto riguarda le connessioni con prodotto tra larghezza di banda e ritardo elevato.

CTCP viene utilizzato per le connessioni TCP con finestra di ricezione di grandi dimensioni, aumentando in modo sensibile la quantità di dati inviati in un determinato intervallo di tempo senza influire negativamente sulle altre connessioni. CTCP mantiene due finestre di congestione: una regolare AIMD *(Additive Increase Multiple Decrease)* ed una basata sul ritardo *("Delay-Base Window").* La dimensione della finestra scorrevole effettivamente utilizzata è la somma di questi due finestre. La finestra AIMD è aumentata nello stesso modo con cui viene incrementata in TCP Reno, mentre, se il ritardo è contenuto la seconda aumenta rapidamente per migliorare l'utilizzo della banda. Una volta rilevato un'accodamento, la finestra *Delay-Base* diminuisce gradualmente per compensare l'aumento della finestra AIMD. L'obiettivo è quello di mantenere la loro somma approssimativamente costante al valore che l'algoritmo stima (ovvero il BDP).

Allo stesso modo, questo meccanismo può essere attivato o disattivato andando nel registro di configurazione di quelli sistemi operativi che supportano tale opzione.

3.2 Miglioramenti proposti nel TCP per ottimizzare lo sfruttamento della banda

I miglioramenti non si riferiscono unicamente alle banali e semplici modifiche (sempre utili) del registro di configurazione, ma anche a modifiche più complesse nel funzionamento del protocollo TCP. Quindi, verranno sottolineati alcuni miglioramenti proposti al TCP per permettere sfruttare pienamente la banda disponibile e ottimizzare notevolmente le prestazioni del sistema.

3.2.1 Dimensionamento della RcvWindow

La finestra di ricezione rappresenta la quantità di byte in un buffer di memoria su un host di ricezione utilizzato per archiviare dati in ingresso su una connessione TCP.

La scelta ottimale di questo parametro diventa un’azione fondamentale nel tweaking di una qualsiasi connessione.

Molto spesso capita che alcuni pacchetti si "perdano" nel girare l'immensa rete di internet, per cui prima di considerare una transazione "conclusa", il mittente aspetta la conferma di ricevimento che il ricevente provvederà a mandare. Purtroppo, a causa del cosiddetto tempo di ping (latenza), ciò può portare a rallentamenti considerevoli e ad inefficienze sull’uso effettivo della capacità di rete .

È qui che entra in gioco la *RcvWindow* che, come già anticipato, stabilisce che un tot di pacchetti possano essere inviati uno dopo l'altro senza dover aspettare la conferma di ricevimento.

Purtroppo ciò comporta alcune cose:

1. Se il valore della finestra di ricezione è molto alto, si potranno avere picchi di velocità notevoli, se però un pacchetto si perde, si dovranno rinviare

tutti i pacchetti della stessa transazione, perdendo così una considerevole quantità di tempo.

2. Se è basso, si avrà una velocità più bassa e stabile ma il sistema diventerà troppo lento.

Dunque, per determinare correttamente il valore della dimensione massima della finestra di ricezione per una connessione, conviene farlo in base alle condizioni correnti della rete e in base all'uso che si fa della connessione. Ad esempio, se si scarica un file con bassi tempi di ping, sarebbe meglio utilizzare un valore di *RcvWindow* basso; mentre se i tempi di ping aumentano, sarebbe meglio utilizzare un valore *RcvWindow* maggiore.

Questo ci porta ad una fondamentale concetto in ogni protocollo di trasporto controllato a finestra: il BDP *(Bandwith Product Delay)*. Il BDP è di fondamentale importanza perché determina la dimensione della finestra che massimizza le prestazioni di un trasferimento TCP:

$$BDP = BandaTotaleDisponibile \times RTT$$

Tutto ciò significa che la finestra di ricezione è vincolata direttamente con il BDP. Dunque, se la *RcvWindow* è inferiore rispetto al prodotto della larghezza di banda e latenza a disposizione, non potremo mai sfruttare pienamente il link in quanto il client non può inviare velocemente i riconoscimenti (ACKs) indietro. Perciò, *RcvWindow* deve essere abbastanza grande da contenere il massimo BDP:

$$BDPmax = MassimaBandaDisponible \times RTTmax$$

Alcune fonti sostengono che per questioni di architetture del protocollo TCP è opportuno che la dimensione della finestra debba essere un multiplo della dimensione massima del corpo dati di un pacchetto (MSS), anche se in realtà altri dicono sia solo una leggenda. Tuttavia, visto che alla fine la dimensione della finestra è un numero piuttosto arbitrario, fare in modo che risulti un multiplo di MSS sicuramente non può farvi male.

Se ad esempio consideriamo il seguente profilo di linea di utente:

Bit_Rate= 5Mbps

RTTmedio= 10ms

La prima modifica riguarda un aumento (50%) del tempo di ping medio. Questo si fa per tener conto delle fluttuazioni del ritardo: sia per come è la sua distribuzione o sia per ulteriori rallentamenti introdotti dai vari dispositivi di rete.

RTT= 10ms x 1,5 = 15ms (RTTmedio aumentato da un 50%)

Allora, una prima approssimazione può essere scritta come segue:

Bit_Rate x RTT = 5Mbps x 15ms = 75000bit / 8 = 9375byte

Il valore ottenuto è molto vicino a quello della "finestra ottimale", ma come abbiamo detto, è opportuno che sia un multiplo di MSS. L'MSS è facilmente calcolabile:

MTU = 1500byte

MSS = 1500byte – 40byte = 1460byte

Procediamo a dividere il valore ottenuto (prima approssimazione) per l'MSS. Una volta fatta tale divisione, arrotondiamo questo valore al successivo intero pari e ri-moltiplichiamo per l'MSS:

$$\frac{RcvWindow}{MSS} = \frac{9375byte}{1460byte} = 6{,}42123 \approx 7$$

MSS x 7 = 1460byte x 7 = 10220byte

Finalmente, il valore "ottimale" della finestra sarà **10220byte.**

I settaggi della finestra di ricezione nella distribuzione Linux Ubuntu si trovano sotto la voce "*tcp_rmem"* (nella directory "proc/sys/net/ipv4") che contiene tre numeri corrispondenti, rispettivamente, al minimo, default e massimo valore (in byte) del buffer di ricezione.

Lo spazio del buffer di ricezione presente in Linux è diviso tra l'applicazione e il kernel, cioè, il TCP mantiene parte del buffer come TCP

window (e questa è la dimensione della finestra di ricezione dichiarata al sender), mentre il resto dello spazio è utilizzato come "buffer applicazione".

È presente a tale proposito un parametro specifico che regola la suddetta divisione, il "*tcp_adv_win_scale"*, in modo tale che:

$$RcvWindow = (\#bytes\ Rcv\_buffer) - (\#bytes\ Rcv\_buffer)/(2^{tcp\_adv\_win\_scale})$$

Il suo valore di default è pari a 2 e implica che lo spazio utilizzato dal "buffer applicazione" è un quarto del totale, mentre il resto è assegnato alla finestra di ricezione.

Nello specifico, la versione 2.6.31 di Ubuntu utilizza normalmente il seguente settaggio di default:

tcp_rmem= 4096 87380 3514368

Considerando i precedenti settaggi base del tcp_rmem, la *RcvWindow* oscillerà tra il valore minimo e quello di default (87380 - 87380/4 = 87380 - 21845 = 65535) senza windows scaling, e fino a 2635776byte (3514368 - 3514368/4 = 3514368 - 878592) con l'opzione attivata.

Nel detaglio:

-minimo (4096): questo dovrebbe essere generalmente un buon valore, e si dovrebbe evitare un aumento quando si verificano sporadicamente alti carichi di rete dato che il sistema può trovare dei problemi e andare peggio.

-default (87380): questo valore dovrebbe, in circostanze normali non essere modificato in quanto può causare problemi simili a quelli visti precedentemente, ma se è disponibile memoria, può aumentare.

-massimo: viene proposto come un doppio del valore specificato di default (174760). Tuttavia, si può migliorare la situazione in maniera sensibile aumentando questo valore. Un recente articolo apparso su Zdnet India consiglia di aumentare questo parametro a 16777216 in modo tale da consentire all'applicazione di gestire i dati più velocemente, migliorare la

capacità del client di inviare dei dati al server quando quest'ultimo è occupato e, finalmente, incrementare la dimensione della *RcvWindow*.

Questo giustifica la seguente scelta:

net.core.rmem_max = 16777216

net.core.wmem_max = 16777216

net.ipv4.tcp_rmem = 4096 87380 16777216

net.ipv4.tcp_wmem = 4096 65536 16777216

3.2.2 Max Duplicate ACKs

Con questo parametro indichiamo il numero di ACK duplicati che devono essere ricevuti con lo stesso numero di sequenza di dati inviati, per iniziare la fase di ritrasmissione *(Fast Retransmit)*. Lo standard TCP prevede che il numero di ACK duplicati necessari per ritrasmettere un certo segmento dati sia tre.

Sebbene questo valore non è inadeguato, non è dal tutto efficiente per arrivare in breve tempo alla trasmissione di un nuovo pacchetto. Una soluzione che solo comporterebbe una modifica andando nel registro di configurazione sarebbe impostare questo valore a due. Con questa scelta si può ridurre il tempo di reazione del protocollo alle perdite di segmenti e, di conseguenza, ritrasmettere più velocemente.

Poiché il segmento mancante nel lato ricevitore potrebbe essere stato oggetto di un certo ritardo sul percorso, il fatto che il mittente aspette adesso solo due ACK duplicati prima di procedere con la ritrasmissione, può portare ad una possibile congestione dovuto alle ritrasmissioni inutili. Tuttavia, la RFC 4138 prevede l'utilizzo di un algoritmo di controllo per affrontare questo problema. Questo algortimo è chiamato F-RTO *(Forward RTO Recovery)* e impedisce la ritrasmissione non necessaria dei segmenti TCP in modo da consentire un più rapido ripristino della normale velocità di trasmissione. Tale meccanismo funziona negli ambienti che subiscono aumenti improvvisi o temporanei del RTT, come ad esempio un client wireless che passa da un punto di accesso ad un altro.

Tutte queste funzionalità di rete sono presenti sullo stack TCP/IP.

3.2.3 Allargamento della finestra iniziale di congestione

Come abbiamo visto, lo standard TCP prevede che la finestra di congestione sia dapprima posta a un segmento e poi gradatamente sia ampliata per sfruttare la disponibilità di banda sul percorso di rete. Una modifica che comporterebbe solo una variazione nello stack TCP del mittente è quella che prevede un allargamento della finestra iniziale di congestione a un valore superiore a quattro segmenti.
Infatti ci sono due casi fondamentale che comportano attualmente il settaggio della finestra di congestione:

- l'inizio di una nuova connessione
- la scadenza del TO.

I cambiamenti proposti devono applicarsi all'inizio di una nuova connessione, dopo che si è concluso il "*three ways handshake*".

Con l'avanzamento del tempo, la dimensione media di una pagina web è più che triplicata rispetto agli anni precedenti. Si è passati da una media di 98kbyte (anno 2005) ai 390kbyte odierni. Anche il numero di elementi presenti in una pagina (testi, immagini, ecc.) è salito negli ultimi anni e si è arrivati ad una media di 80 oggetti per pagina, aspetto che contribuisce notevolmente nell'aumentare il tempo di caricamento di un certo web site. Per questi motivi, aumentare la finestra iniziale di congestione può comportare la possibilità di gestire più traffico dati, velocizzare la comunicazione e minimizzare l'impatto della latenza media presente in rete.

Idealmente, vogliamo scegliere una finestra iniziale di congestione conforme con i seguenti obiettivi:

- Ridurre al minimo il tempo medio di download.

- Minimizzare l'impatto della latenza in rete.

D'altra parte, visto che diversi studi hanno dimostrato che il 90% della ricerche sul web e l'uso di mail richiedono solitamente circa 15kbyte per l'uso di applicazioni di grandi dimensioni, alzare la finestra a 15 segmenti può ragionevolmente soddisfare questa proprietà.

L'RFC 4380 stabilisce che è conveniente fissare il limite massimo della finestra di congestione in funzione dalla dimensione massima di un segmento (MSS). Perciò, seguendo tale procedura:

- Se (MSS $\leq$ 1095byte) allora la finestra iniziale deve essere $\leq$ 15 x MSS.
- Se (1095byte < MSS < 2190byte), la finestra iniziale deve essere $\leq$ 16425byte.
- Se (2190byte $\leq$ MSS) allora la finestra iniziale deve essere $\leq$ 7,5 x MSS.

Come già anticipato, una finestra iniziale più larga può portare a notevoli vantaggi, ad esempio: per le connessioni in cui si deve trasmettere solo una piccola quantità di dati, una finestra iniziale più larga riduce drasticamente il tempo di trasmissione. Infatti molte file di dimensioni contenute possono essere scaricati in pochi segmenti in un singolo RTT.
Ma questa modifica è molto importante anche per quelle reti con un elevata latenza poiché permette di ridurre i tempi di trasmissione di qualche RTT. L'unico problema si può verificare negli scenari altamente congestionati dove una finestra iniziale allargata può comportare la perdita di alcuni segmenti dovuta alla incapacità del router per gestire grandi burst di informazione.

3.2.4 ECN (Explicit Congestion Notification)

Per garantire affidabilità nella trasmissione dei dati non sono sufficienti gli algoritmi di controllo di congestione forniti da TCP, ma vengono richiesti anche ai router dei meccanismi di controllo di congestione di supporto agli host terminali.

Tradizionalmente un router reagisce alla congestione perdendo il pacchetto in arrivo se la sua coda di trasmissione è completamente piena *(overflow)*. Quindi per ogni coda di trasmissione di un router viene configurata una lunghezza in modo da accodare pacchetti solo fino a quando non viene superata tale lunghezza massima, poi si incomincerà a perdere pacchetti. Nuovi pacchetti vengono accettati solo quando è presente nuovo spazio disponibile nella coda di trasmissione. Questa tecnica di gestione delle code prende il nome di "*Tail Drop"* e interagisce in modo poco efficiente con i meccanismi di controllo di congestione di TCP.

Un ciclo di trasmissione di dati è costituito da una certa quantità di perdite dopo che è stata raggiunta la massima lunghezza della coda di trasmissione, ciò comporta un decremento simultaneo della *CongWindow* da parte degli host terminali. Successivamente i nodi trasmittenti incrementano la dimensione della loro finestra simultaneamente fino a che non vi è di nuovo perdita di pacchetto. Questo fenomeno viene chiamato *Global Synchronization* e comporta una sottoutilizzazione della banda del link e un basso throughput.

Dunque, per affrontare questo problema sono stati studiati due metodi per informare il mittente della congestione: il primo va sotto il nome di BECN *(Backward Explicit Congestion Notification)* e consiste nell'inviare al mittente un messaggio ICMP *(Internet Control Messagge Protocol)* di notifica della congestione. Ciò richiede che gli stessi router indirizzino i pacchetti su entrambe le direzioni di trasmissione. Nei collegamenti asimmetrici però è tipico l'uso di connessioni indipendenti per i flussi nelle due direzioni.

Un altro meccanismo maggiormente utilizzato in quanto permette che i flussi nelle due direzioni viaggino in due reti separate è quello del FECN *(Forward Explicit Congestion Notification)*. FECN marca il pacchetto con uno speciale tag, quando la lunghezza media di coda del router si avvicina al suo valore massimo. Questa marca viene fatta ai pacchetti prima che ci sia realmente congestione, e questo è molto utile per protocolli come TCP che sono sensibili anche ad una singola perdita di pacchetto. In tal modo i segmenti continuano il loro naturale cammino fino al ricevitore che, esaminando il bit dell'intestazione, si accorge dell'eventuale notifica di congestione e invia un messaggio ICMP al mittente.

RFC 3168 prevede che TCP non deve reagire alle indicazione di congestione più di una volta per ogni finestra di dati (a almeno non più di una volta per RTT). La *CongWindow* dovrebbe essere ridotta solo una volta per una finestra in risposta ad una serie di pacchetti persi o marcati. Inoltre TCP sorgente non dovrebbe decrementare la soglia se questa è già stata decrementata all'interno dell'ultimo RTT. Comunque se qualche pacchetto ritrasmesso viene perso, questo viene interpretato dal sender come una nuova istanza di congestione.

L'uso del FECN comporta cambiamenti nell'implementazione TCP del mittente/ricevente ed anche nel software dei router per permettere di marcare i segmenti in modo che i nodi finali siano informati della possibile congestione (RFC 2760). Tali cambiamenti dovrebbero comportare miglioramenti delle performance TCP, ottimizzando l'utilizzazione della capacità di link e introducendo un guadagno in termine di throughput.

3.2.5 Perdite non dovute a congestione

La differenziazione fra le perdite causate da corruzione e quelle da congestione è molto importante in quanto le azioni che il mittente deve compiere nei due casi sono completamente differenti. Nel caso di corruzione, il mittente deve ritrasmettere il più velocemente possibile i segmenti mancanti senza modificare la finestra di congestione. L'implementazione standard del TCP, però,

assume che tutte le perdite siano dovute a congestione e ciò comporta nella migliore delle ipotesi l'inizio dell'algoritmo di Fast Recovery.

Perciò, un metodo che consente in ambienti di bassa qualità (elevati BER), di ottenere buoni miglioramenti di performance si basa sulla distinzione fra corruzione e congestione. Proprio a questo, l'elemento di rete che rivela un pacchetto corrotto non lo deve semplicemente scartare, ma deve segnalarlo al ricevente che poi ha il compito a sua volta di informarne il mittente. Attualmente i dispositivi a livello rete scartano i pacchetti corrotti senza informare il livello trasporto quello che accade, quindi bisogna implementare un meccanismo che invii al ricevente un messaggio a livello rete (ad esempio un messaggio ICMP) e, una volta che il livello rete ha ricevuto la notifica della corruzione, informerà il livello trasporto e questo provvederà ad avvisare il livello trasporto del mittente (ad esempio settando un bit nell'intestazione del TCP di un ACK). Tale meccanismo può essere mostrato come segue:

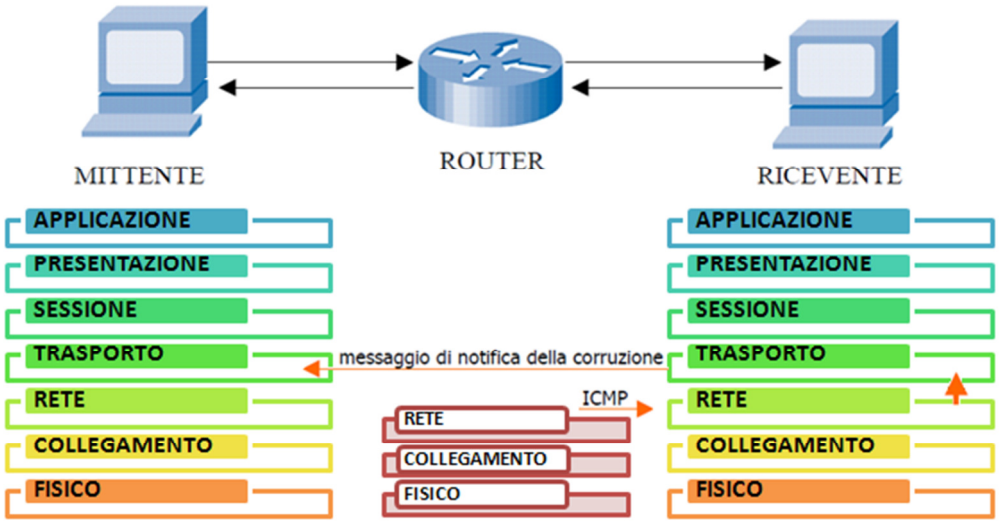


Figura 3.1: Notificazione di perdite dovute a corruzione

Dopo la ricezione della notifica della corruzione, il mittente si porrà in uno stato chiamato *"corruption experienced"* (RFC 2760) per un certo periodo di tempo (circa 2xRTT), e in questo stato il mittente ritrasmetterà i segmenti persi senza diminuire la finestra di congestione.

La corruzione di un pacchetto non è detto che interessi solo i dati in esso contenuti. Infatti, anche l'intestazione può essere modificata; ciò può portare a degli errori negli indirizzi del mittente e del destinatario. Per risolvere questo problema si può utilizzare in ogni elemento di rete un "deposito segreto" (una cache) che tenga in memoria i più recenti indirizzi dei destinatari; quando il router incontra un rate di errore sopra a una certa soglia (determinata in funzione delle caratteristiche della rete), manda un messaggio ICMP a tutti i destinatari memorizzati nella cache per informarli dello stato di corruzione e così mantenerli allarmati ante tale situazione.

Una debolezza di questo meccanismo è che se la congestione di rete si verifica allo stesso tempo della corruzione dei pacchetti, il mittente non reagirà alla congestione e continuerà ad inviare pacchetti durante i 2xRTT.

Questo approccio comporta le seguente modifiche:

- a livello di rete (ai router): devono essere in grado di generare messaggi ICMP che informano della corruzione.
- allo stack TCP/ IP del ricevitore: deve essere in grado di processare i messaggi ICMP di corruzione e convertirli in segnali di corruzione nel header TCP dei ACK duplicati.
- allo stack TCP del mittente: deve essere in grado di comprendere i segnali che lo invitano a passare allo stato *"corruption experienced"*.

CAPITOLO IV

DESCRIZIONE DEL TEST-BED

4.1 Struttura del Test-Bed

Per la sperimentazione di questo lavoro di tesi è stato utilizzato un test-bed presso il Ministero dello Sviluppo Economico – Comunicazioni, dove ha sede un distaccamento della Fondazione Ugo Bordoni. Tale test-bed, realizzato in collaborazione con l'ISCTI (Istituto Superiore delle Comunicazioni e Tecnologie dell'Informazione), si sviluppa su tre laboratori distinti tra i quali è stato creato un collegamento ad hoc in fibra ottica multimodale in prima finestra (850 nm):

- Laboratorio Trasmissioni ottiche (ufficio III ISCOM);
- Laboratorio Reti ottiche dinamiche (primo piano ISCOM);
- Laboratorio Reti di accesso (ufficio II ISCOM).

Nel laboratorio *Trasmissioni Ottiche* è attestato un collegamento all'anello ottico Roma-Pomezia; il *link,* che collega in anello Roma e Pomezia, è lungo 25 km "one way" ed è costituito da 80 fibre monomodali in terza finestra (1550 nm), delle quali 30 ds (*dispersion shifted*), 20 nzd (*non zero dispersion*) e 30 sf (*standard fiber*). A questo link ottico sono connessi quattro routers Juniper.

Nel laboratorio *Reti Ottiche Dinamiche* riportato nella figura 3.1(a) sono presenti:

- due routers Juniper M10 e due routers Juniper M10i;
- quattro pc client (ciascuno connesso ad un diverso router), utilizzati sia per lo scambio di flussi video allo scopo di testare le connessioni, sia per l'accesso remoto ai routers;
- due pc server (connessi ai due Juniper M10i), utilizzati come sorgenti e archivio per tali filmati;

- un pc connesso alle interfacce di management dei quattro routers ed usato come “driver”.

Figura 4.1: (a) Laboratorio Reti Ottiche Dinamiche, (b) Laboratorio Reti di Accesso

Nel laboratorio *“Reti di Accesso”* riportato in figura 3.1(b) sono presenti:

- tre routers **Cisco 3845;**
- un softswitch **Alcatel ESS 7450**;
- due pc client (ciascuno connesso ad un diverso router), utilizzati sia per lo scambio di flussi video allo scopo di testare le connessioni, sia per l’accesso remoto ai routers;
- un pc connesso in seriale alle interfacce di management dei tre routers ed usato come “driver” (tale pc è anche connesso alla LAN del laboratorio e vi si può accedere tramite desktop remoto).

La struttura completa del test-bed è la seguente:

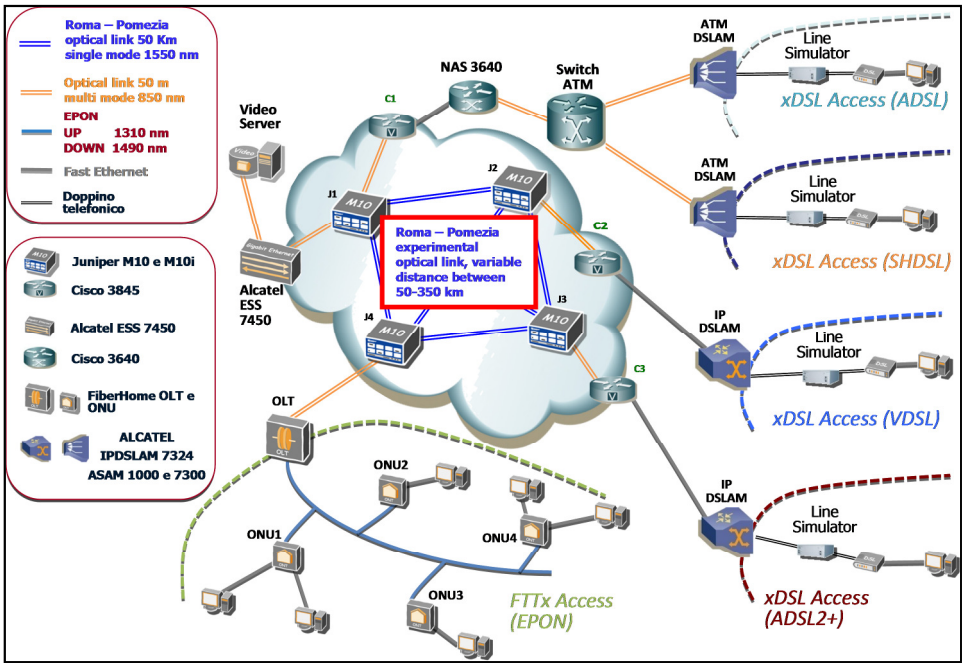


Figura 4.2: Test Bed

Il cuore del test-bed è formato da sette routers, quattro Juniper (due M10 e due M10i) e tre Cisco 3845. I routers della serie M di Juniper sono routers di fascia alta, pensati ed equipaggiati per far parte di una core network, mentre i routers della serie 3800 della Cisco sono routers di fascia media, tipicamente utilizzati come routers di accesso. Proprio per le caratteristiche degli apparati, la rete è stata progettata in modo che i quattro routers Juniper fungano da core network completamente magliata, e i tre routers Cisco siano utilizzati come nodi della rete di accesso.

Oltre ad avere caratteristiche diverse, questi routers sono dislocati in due laboratori differenti interconnessi tramite fibra ottica multimodale, sfruttando un cablaggio già predisposto tra i due piani. Il controllo dell'intera rete è stato remotizzato presso il laboratorio Reti di Accesso, e gli accessi a tutti i routers sono stati realizzati sia in remoto, dai pc host tramite Telnet, sia in seriale con un pc dedicato.

Dalla figura 4.2 si può notare la presenza di ulteriori elementi quali un router Cisco 3640, utilizzato come Nas-Ras Radius per l'autenticazione d'utente, uno switch Cisco Lightstream per i percorsi ATM, un ASAM 1000 per l'accesso ADSL, un ASAM 7300 per accessi VDSL ed SHDSL ed un ISAM 7324 per l'accesso ADSL2/2+.

Nel test-bed, infine, sono presenti due sezioni di accesso, una in rame basata su xDSL (oggetto della fase sperimentale della tesi) e una in fibra costituita da una tipica architettura FTTx, in particolare una Ethernet PON. Questa è formata da una OLT (Optical Line Termination) AN5116-03 della FiberHome, da un certo numero di ONU (Optical Network Unit) AN5006-05 anche esse della FiberHome e da uno splitter/accoppiatore passivo.

4.2 Router Cisco 3845 e sistema operativo IOS

In questo paragrafo viene riportata la struttura hardware dei Cisco e il sistema operativo IOS.

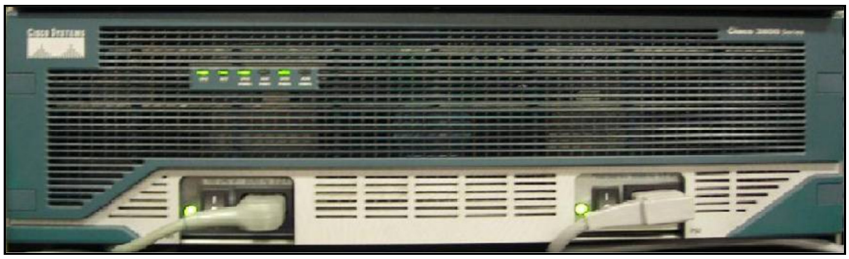


Figura 4.3: Router Cisco 3845

4.2.1 Architettura hardware dei routers Cisco

Gran parte del successo dei router Cisco fu dovuto alla somiglianza, a livello architetturale, con i normali PC. Le macchine Cisco possiedono, infatti, tutti i componenti tipici di un PC, ad eccezione dell'hard disk e del floppy drive, in quanto le tabelle di routing necessitano di essere immagazzinate in una memoria veloce (NVRAM, RAM) per ottenere una rapida esecuzione.

Una differenza sostanziale con l'architettura dei PC è la presenza di più processori la cui potenza varia a seconda del ruolo che dovrà assumere il router (router di accesso, di core, ecc.).

Completano il router le interfacce di rete tipo Ethernet, Sonet, ATM etc..

In Figura 4.4 è riportato un tipico layout di una motherboard Cisco e di seguito si analizzano le diverse tipologie di memorie presenti:

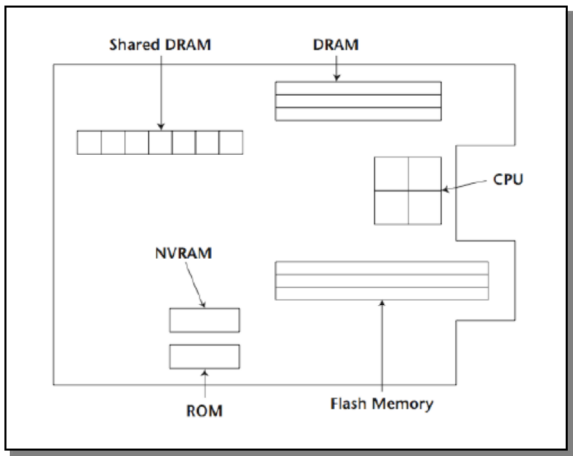


Figura 4.4: Architettura di un router Cisco

RAM/DRAM (Random Access Memory/Dynamic RAM), detta anche *working storage*. Quando il router è in funzione, la RAM contiene un'immagine del software IOS, la configurazione che sta girando in quel momento e la tavola di routing, mentre la DRAM, o *shared memory,* contiene il buffer dei pacchetti che devono essere "ruotati";

NVRAM (Non Volatile random access memory): è una memoria non volatile, che quindi si comporta come una sorta di hard disk, mantenendo i dati anche in caso di *power loss.* Viene utilizzata per contenere una copia del file di configurazione e vi è presente quindi quella che viene chiamata *startup config;*

Flash Memory: differentemente dalla NVRAM, la memoria flash può essere cancellata e riscritta più volte e contiene una o più copie del sistema operativo IOS;

ROM: è una memoria di sola lettura e pertanto contiene solo quello che viene chiamato *bootstrap program*, ovvero una configurazione minima del router che possa permettere il riavvio a seguito di una perdita improvvisa di alimentazione o di un danno al file di configurazione di startup.

4.2.2 Sistema Operativo IOS

Il sistema operativo dei routers Cisco presenta due componenti principali: il *configuration file* e il *Cisco IOS Software.* Il primo può essere modificato a seconda delle funzioni che il router deve svolgere, il secondo invece può essere solo aggiornato.

La versione di IOS implementata sui routers Cisco del test-bed è la 12.3. L'IOS Software, contenuto in un singolo file (di dimensioni variabili da 3 a 10 MB), è costruito in sezioni, che separano le funzionalità implementabili; normalmente il router viene venduto completo del pacchetto base ed a questo vengono aggiunte le ***feature packs***, ovvero porzioni di software che aggiungono al sistema operativo base funzioni più evolute.

Le modalità operative con le quali ci si può interfacciare con un router Cisco sono diverse, ciascuna con scopi, capacità gestionali e prompt differenti. Nella **Errore. L'origine riferimento non è stata trovata.** vengono illustrate schematicamente le diverse modalità di accesso e le loro funzionalità. La prima categoria è la **Boot mode**, racchiude le modalità *Setup*, utilizzata per lo startup e la configurazione base del router, ed altre due modalità utilizzate in caso di boot failures la seconda è la **User mode**, comprende la sola modalità *User EXEC* ed è utilizzata per i normali accessi al router al fine di esaminare una stretta cerchia di funzionalità del router; la terza è la **Configuration mode**, racchiude le modalità *Privilege EXEC* che garantisce un monitoraggio ed un'analisi più approfondita delle funzionalità del router, e le modalità *Global configuration* e *Specific configuration,* che offrono la possibilità di configurare il router in maniera globale o parziale.

Digitando il comando "*configure terminal*" si accede alla modalità di configurazione ed è possibile configurare il router a diversi livelli, suddivisi in menù.

4.3 Interfacce e connessioni

A questo punto vengono descritte le interfacce presenti sui routers e sugli armadi di collegamento tra i laboratori, le diverse tipologie di connettori e di cavi utilizzati.

Sia sui routers Juniper che su quelli Cisco entrambi facenti parte del test-bed sono presenti interfacce di tipo ottico e interfacce di tipo elettrico. Quelle di tipo ottico sono interfacce LC (Local Connector, connettori in plastica e di forma quadrata), invece quelle elettriche sono di tipo RJ45.

Le fibre utilizzate per le connessioni, riportate in Figura 3.7, sono sia monomodali sia multimodali, con connettori di tipo LC ed FC (Ferrule Connector, connettori in metallo di forma tonda), mentre i cavi per i collegamenti elettrici sono i classici CAT5 (supportanti Fast-Ethernet a 100Mbit/s), sia crossati sia dritti con connettori RJ45.

Figura 4.5: Connettori fibra ottica

Dopo che tutti gli elementi del test-bed sono stati interconnessi fisicamente, si è realizzato un piano di indirizzamento per le interconnessioni tra i routers. Ciascuna porta è stata indirizzata con un indirizzo progressivo, del tipo 192.168.X.Y/24 (mascheramento a 24bits).

4.4 Principali elementi hardware/software utilizzati

4.4.1 Wireshark

Wireshark è un network packet analyzer, cioè un software in grado di catturare es analizzare i pacchetti trasmessi in una rete in modo dettagliato. Inoltre, consente il riconoscimento e l'analisi di una grande varietà di protocolli, fornendo all'utente informazioni dettagliate su ciascuno di essi.

Wireshark è dotato di un'interfaccia grafica che rende l'operazione di analisi del traffico in rete semplice, veloce ed intuitiva. Si tratta di uno strumento molto potente e di indubbia utilità per l'amministratore di rete e per applicazioni ingegneristiche, ma anche per l'utente comune che vuole studiare ed imparare il funzionamento dei protocolli di rete.

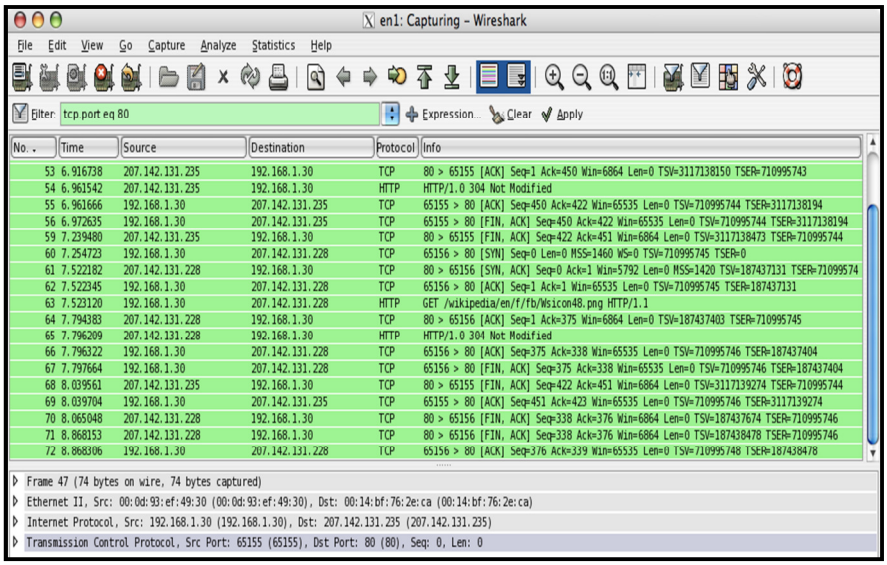


Figura 4.6: Schermata di Wireshark dopo una cattura

Vengono riportate brevemente le potenzialità di questo software:

- Può essere utilizzato con diversi tipi di rete, tra cui Ethernet 802.11, FDDI, Token Ring e ATM;
- Analizza innumerevoli protocolli a vari livelli (collegamento, rete, trasporto, applicazione);
- Permette di catturare e di salvare su file il traffico in rete per analisi future;

- Permette di importare ed esportare i dati catturati con molti altri analizzatori di rete;
- Consente di filtrare i pacchetti e di effettuare ricerche tra i pacchetti secondo molti criteri;
- Supporta la colorazione dei pacchetti basata sull'impostazione dei filtri;
- Genera utili statistiche sul traffico analizzato.

4.4.2 Server FTP

Per poter scaricare i file di una data dimensione attraverso il protocollo FTP, è stato installato e configurato un server FTP con le seguenti caratteristiche:

- CPU: 2 processori Intel ® Xeon ® 2,66GHz
- RAM 8GB
- Schede di Rete: Gigabit Ethernet Controller (Copper)
- Sistema Operativo: Ubuntu 8.04 (LTS)
- Server FTP: vsftpd versione 2.0.6

All'interno di esso, inoltre, sono stati inseriti cinque file di dimensione (in byte) pari a dieci volte i profili ADSL2+ considerati per l'esecuzione dei test (7Mbps, 10Mbps, 12Mbps, 18Mbps).

4.4.3 Client FTP

Per poter effettuare il download/upload dei file caricati sul server FTP, è stato utilizzato un PC dual boot con il ruolo di client FTP Linux; le caratteristiche hardware sono elencate qui sotto:

- CPU: processore Intel(R) Atom(TM) CPU 230 1.60GHz
- RAM: 2GB
- BIOS: Phoenix Technologies 6.0 PG; size: 128KiB capacity: 960KiB
- Scheda di rete1: Realtek Semiconductor RTL8111/8168B PCI Express Gigabit Ethernet
- Scheda di rete2: Realtek Semiconductor RTL-8110SC/8169SC Gigabit Ethernet

- Video: Intel 82945G/GZ Integrated Graphics Controller
- Audio: Intel 82801G (ICH7 Family) High Definition Audio Controller
- Hard Disk: ATA Disk WDC WD800BEVT-22 da 80GB

Nel capitolo successivo verrà descritta la metodologia con la quale il client sarà in grado di dialogare con il server per il download/upload dei file.

4.4.4 Alcatel IPDSLAM 7324

Per poter settare i diversi profili ADSL2 che verranno utilizzati per la parte sperimentale della tesi, è stato impiegato il modello "Alcatel IP DSLAM 7324 (RU)":

Figura 4.7: Alcatel IP DSLAM 7324 RU

Si tratta essenzialmente di un piccolo DSLAM che offre "servizi ADSL multipli ovvero POTS" (ADSL, ADSL2, ADSL2+) per poter supportare i servizi Triple-Play a banda larga, includendo "High Speed Internet Access" (HSIA) , "IP Television" (IPTV) e "Voice over IP" (VoIP). Le caratteristiche principali sono:

- 48 porte multi-ADSL con all'interno i "POTS splitters" e :
  - ADSL: G.992.1 Annex A
    G.992.2 G.Lite
  - ADSL2: G.992.3 Annex A
    G.992.4 G.Lite
  - ADSL2+: G.992.5 Annex A
- Possibilità di raggiungere velocità fino a 24Mbps con l'ADSL2+
- Una porta console per il "local management"
- Due interfacce 100/1000 Base-TX
- Rate adaptation
- ADSL line rate setting
- Pre-channel QoS, incluso controllo di banda, priorità e traffic shaping

Tra queste, “l’ADSL line rate setting” è stata quella più funzionale per lo svolgimento dei test, dato che ha permesso di impostare i differenti profili di linea. Ciò è stato possibile configurando manualmente da remoto la voce “xDSL Profiles Setup”, impostando il massimo rate (in kbps) in downlink e uplink, come mostrato nella Figura 4.8:

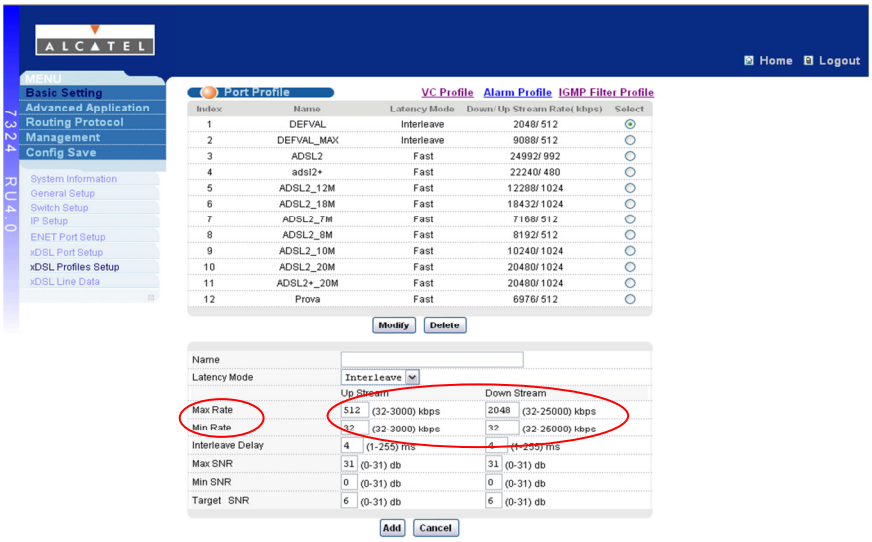


Figura 4.8: xDSL Profile Setup

4.4.5 Modem Speed-Touch

Per poter effettuare i test nella rete di accesso ho utilizzato il modem ADSL2 “SpeedTouch 516/536 v6” con le seguenti caratteristiche:

- Proprietà: ADSL, Bridge, Firewall, Flash-ROM, Router.
- Bus di comunicazione: LAN, RJ-45 WAN.
- Sistemi operative supportati: Apple Mac, Linux, Windows 2000/XP/7.
- Famiglia: BCM6348 ADSL2+ (Thomson).

Figura 4.9: Modem Speed-Touch

Il fatto di essere un modem ADSL con capacità di "bridging" significa che il router non è ristretto a un singolo utente, ma può essere utilizzato anche per connettere un'intera rete a Internet. Altre funzioni essenziali consistono nel poter essere un "DNS - server", un "DHCP – server" (per auto - connettere i computer alla LAN) e un "Firewall" (per isolare la LAN da Internet).

4.4.6 Shunra/Storm

Per poter riprodurre determinate condizioni e topologie di rete è stato utilizzato il simulatore di linea "Shunra/Storm STX-100"; si tratta, in pratica, di un potente strumento che permette di creare "un'immagine specchio" dell'ambiente di lavoro nel laboratorio per poterne valutare le prestazioni. E' composto da due elementi: 1) la "StormConsole", ovvero un software, installabile su PC (con sistema operativo Windows 2000/XP Professional), che fornisce una serie di elementi specifici (network clouds, routers, endpoints, links, ecc.) da utilizzare nella costruzione della propria WAN; 2) la "Storm Appliance" (Shunra's hardware), costituita da quattro porte Fast - Ethernet. Entrambi comunicano tra loro via TCP/IP utilizzando una delle quattro porte.

Per la fase sperimentale della tesi ho "disegnato" un semplice scenario di rete utilizzando lo "Shunra/Storm's add-in Visio objects"; tale scenario è composto da una "WAN cloud" e da due porte che rappresentano i due end-point (client e server):

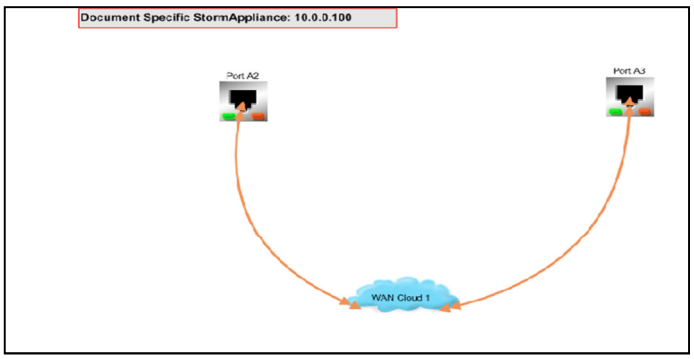


Figura 4.10: Scenario simulato di rete

Per poter cambiare i parametri caratterizzanti il nostro path, come la latency, il packet loss e il link faults, è sufficiente cliccare due volte sulla nuvola per far apparire la schermata mostrata in figura 4.11:

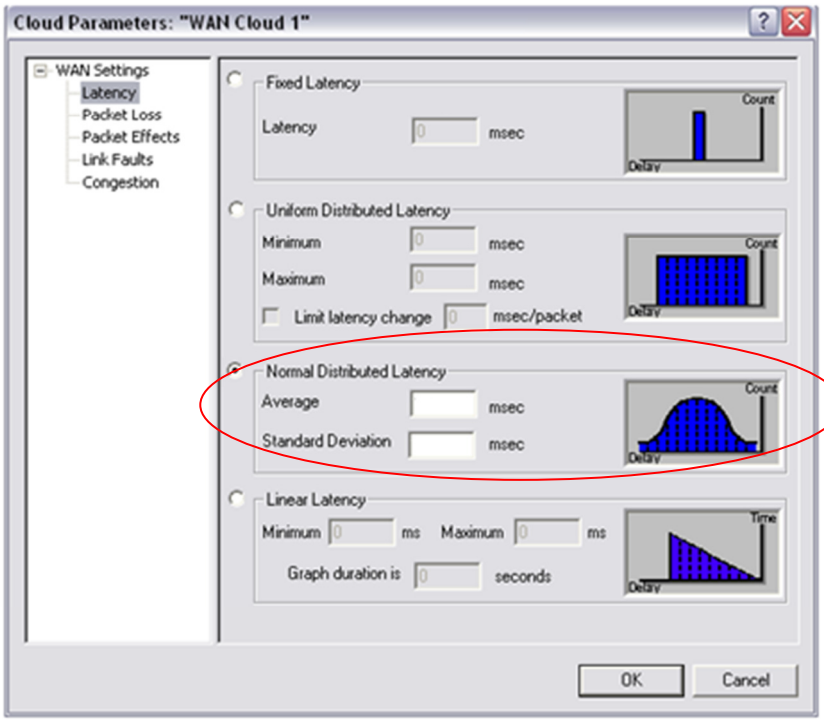


Figura 4.11: Cloud Parameters

L'unico parametro importante alla fine della fase sperimentale sarà la latency, che verrà variata di volta in volta a seconda dei ritardi considerati per i vari profili ADSL.

CAPITOLO V

REALIZZAZIONE DELLA RETE DI ACCESSO E CARATTERIZZAZIONE SPERIMENTALE DELLE PROVE

5.1 Descrizione del lavoro

Abbiamo già visto come il parametro più interessante dal punto di vista dell'utente finale nella valutazione della qualità di accesso a Internet sia la banda disponibile percepita, ossia il throughput utilizzabile dalle applicazioni di utente. L'obiettivo del presente lavoro consiste, d'altronde, nel monitorare le prestazioni dell'utente proprio attraverso questo parametro. Si è anche discusso sia sul modo più intuitivo per misurarlo, cioè attraverso la stima del tempo di download di un file da un server a un client, sia su come possa avvenire lo scambio dei dati tramite i protocolli applicativi utilizzati nell'accesso ad Internet. Per muoversi in questo contesto si è allestito un server FTP, secondo le modalità introdotte nel quarto capitolo, contenenti dei file (la cui tipologia e dimensione saranno affrontate dettagliatamente nei paragrafi successivi) scaricabili da un client. Inoltre, su quest'ultimo è stato installato il sistema operativo Ubuntu dove si ha fatto la modifica della dimensione della finestra iniziale di congestione con l'obiettivo di valutare l'impatto di questa variazione sulle misure di qualità di accesso ad Internet da postazione fissa.

Rimane ora da gestire l'altro parametro (introdotto nel primo capitolo) strettamente legato alla capacità del canale logico e, in particolare, al throughput di linea: il BDP. Data la sua importanza nel costituire un limite superiore per la velocità di trasferimento dati del protocollo TCP, l'idea di base è stata quella di variarlo nei diversi test con l'intento di analizzare il comportamento delle tecniche di stima di banda considerate. La variazione del BDP si è ottenuta modificando sia il profilo di linea (attraverso il pc remoto collegato all' IPDSLAM 7324 RU descritto nel precedente capitolo), che la latenza del collegamento (tramite lo Shunra). Si è proceduto, quindi, alla messa a punto della rete di accesso di riferimento per le prove sperimentali.

5.2 Topologia di rete

La topologia di rete considerata vuole rappresentare un segmento di una Wide Area Network (WAN) ed è costituita da due sezioni principali: quella di *accesso,* che comprende il percorso dal PC client fino all'IPDSLAM, e quella *metro/core*, che va dal DSLAM fino al PC server, come illustrato nella Figura 5.1:

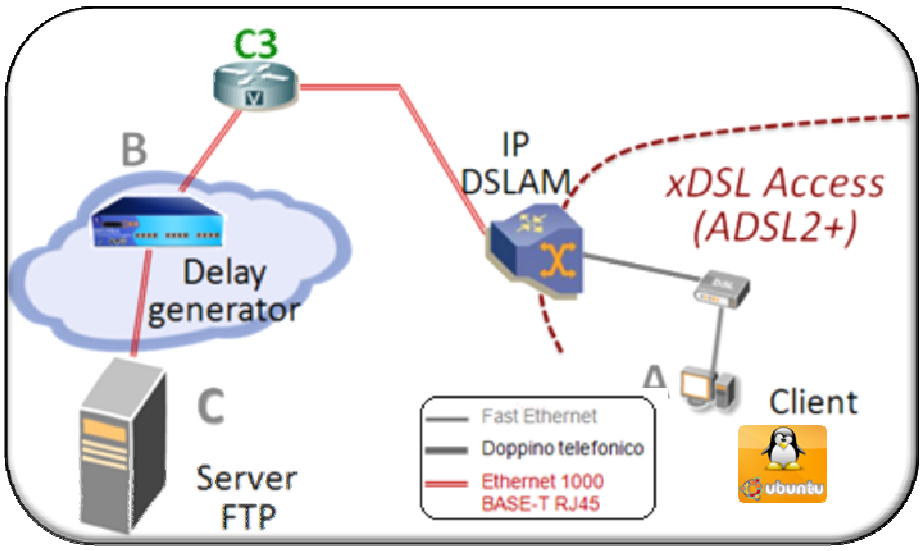


Figura 5.1: Test-bed design

La rete di accesso è la sezione più importante dell'intera topologia di rete dato che costituisce l'attuale collo di bottiglia in termini di banda ed è su di essa che vogliamo misurare le prestazioni. Con riferimento all'attuale panorama nazionale si è considerato un collegamento ADSL2+, una delle soluzioni più utilizzate per fornire accesso a larga banda in Italia. Dopo aver opportunamente configurato in modalità "bridged" il modem Speed-touch secondo la tecnologia ADSL2+, si sono realizzati i collegamenti, tramite cavo fast-ethernet, verso il client, mentre verso l'IPDSLAM attraverso un classico doppino telefonico. Per quanto riguarda il client, infine, è stato messo in comunicazione con il server aggiungendo la rotta IP (comprensiva di indirizzo, maschera e gateway) che porta a quest'ultimo.

Nella rete Core, invece, ho posizionato il server FTP e l'ho collegato, tramite connettore Gigabit Ethernet 1000BASE-T RJ45, al "delay generator", cioè

lo Shunra. Quest'ultimo, è stato collegato tramite cavo Ethernet al router Cisco 3 ed ha permesso di emulare le diverse latenze sul percorso di rete considerato.

5.3 Modalità di esecuzione

Per prima cosa si è proceduti a configurare il server FTP e a metterli in comunicazione con il client (definendo la rotta opportuna, come illustrato nella Figura 5.1).

Circa la scelta dei file da scambiare tra il server e il client, si è seguita la direttiva di ETSI EG 202 057-4 **[1]**, secondo la quale la dimensione di quest'ultimi dovrebbe essere di dimensioni tali da permettere alla finestra di ricezione di crescere e di andare a regime. Dunque, si sono creati dei file di dati non comprimibili attraverso la generazione di una sequenza di numeri random e con dimensione pari a dieci volte i profili di linea di volta in volta considerati.

Si sono considerati quattro profili di linea di utenti, ovvero quelli più utilizzati a livello commerciale, come illustrato nella tabella 2:

Tabella 2: Profili di download ADSL

ADSL2+ Download				
Profile	#1	#2	#3	#4
Bit Rate	7 Mbps	10 Mbps	12 Mbps	18 Mbps

Questo ha comportato l'utilizzo di quattro "file.rnd" (l'estensione rnd sta per random) con le seguenti dimensioni per ogni profilo:

Tabella 3: Dimensione file

FILE				
Profile	#1	#2	#3	#4
Dimensione	70 Mb	100 Mb	120 Mb	180 Mb

Tipicamente il ritardo in una rete non è uniforme, perciò, aggiungere un importo fisso di ritardo a tutti i pacchetti non sarebbe una considerazione dal tutto appropriata. Dunque, per emularla, occorre utilizzare una distribuzione normale per descrivere la statistica risultante di tale variazione.

La distribuzione normale o Gaussiana dipende da due parametri, il valore medio (μ) e la deviazione standard (σ). Di conseguenza, circa i RTT, ho considerato quelli medi di una rete nazionale con le seguenti deviazioni standard:

Tabella 4: RTT considerati

RTT					
Valore medio (μ)	10ms	20ms	40ms	60ms	100ms
Deviazione Standard (σ)	±5ms	±5ms	±5ms	±5ms	±10ms

Per far variare il BDP, quindi, sarà considerato (per ogni profilo di linea) i quattro valori di RTT sopra esposti in modo tale da avere delle limitazioni del throughput del protocollo.

Allora, per ogni scenario di rete avremo:

Tabella 5: Variazione del BDP (primo scenario)

RTT = 10ms				
Profile	#1	#2	#3	#4
Bit Rate	7Mbps	10Mbps	12Mbps	18Mbps
BDP	8750byte	12500byte	15000byte	22500byte

Tabella 6: Variazione del BDP (secondo scenario)

RTT = 20ms				
Profile	#1	#2	#3	#4
Bit Rate	7Mbps	10Mbps	12Mbps	18Mbps
BDP	17500byte	25000byte	30000byte	45000byte

Tabella 7: Variazione del BDP (terzo scenario)

RTT = 40ms				
Profile	#1	#2	#3	#4
Bit Rate	7Mbps	10Mbps	12Mbps	18Mbps
BDP	35000byte	50000byte	60000byte	90000byte

Tabella 8: Variazione del BDP (quarto scenario)

RTT = 60ms				
Profile	#1	#2	#3	#4
Bit Rate	7Mbps	10Mbps	12Mbps	18Mbps
BDP	52500byte	75000byte	90000byte	135000byte

Tabella 9: Variazione del BDP (quinto scenario)

RTT = 100ms				
Profile	#1	#2	#3	#4
Bit Rate	7Mbps	10Mbps	12Mbps	18Mbps
BDP	87500byte	125000byte	150000byte	225000byte

Affinché il pc client potesse eseguire il download dei file in modo automatizzato, si è sviluppato uno script in base al protocollo FTP. Essendo Linux il sistema operativo usato, ho creato un file "bash", ovvero un interprete di comandi che permette all'utente di comunicare con il sistema operativo attraverso una serie di funzioni predefiniti, o di eseguire programmi; inoltre, è in grado di eseguire comandi che gli vengono passati, utilizzando la re-direzione dell'input e dell'output per eseguire più programmi in cascata in una pipeline software, passando l'output del comando precedente come input del comando successivo.

La struttura di tale file è la seguente

```
#!/bin/bash

#numero prove
for i in {1..40} do

#credenziali
USER_FTP=nemesys
PASSWD_FTP=4gc0m244
HOST=192.168.138.2

ftp -nv <<FINE
open $HOST
quote user $USER_FTP
quote pass $PASSWD_FTP
binary
pass
cd download
get ADSL7M.rnd

FINE
sleep
done
```

Analizziamolo riga per riga:

for i in {1..40} do

Questo è l'inizio del ciclo in cui faccio eseguire lo script quaranta volte.

USER_FTP=nemesys

Attraverso tale riga vengono indicate le credenziali per l'accesso.

HOST=192.168.138.2

È l'indirizzo IP del server FTP.

ftp -nv <<FINE

Con la parola “ftp” indichiamo che stiamo tentando di instaurare una connessione del suddetto tipo; “-nv” è un’opzione in cui “n” impedisce a ftp di tentare “l'auto-login” alla connessione iniziale, mentre “v” (verbose) obbliga ftp a mostrare tutte le risposte dal server remoto, oltre a un rapporto sulle statistiche di trasferimento dati. La dicitura “<<FINE” indica all’interprete dei comandi di eseguire tutte le istruzioni che troverà da qui fino a quando non rincontrerà la stessa parola (“FINE”).

open $HOST

Con tale istruzione ci mettiamo in contatto con il server FTP aprendone la connessione; in pratica viene aperto il canale di controllo entro cui far transitare i comandi.

quote user $USER_FTP

Si inviano le credenziali di autenticazioni (dichiarate in precedenza) al server.

pass

Si imposta una connessione di tipo passiva in cui è il client ad aprire la connessione verso il server.

binary

Utilizzata per caratterizzare il formato dei dati scambiati; nello specifico saranno di tipo binario.

get ADSL7M.rnd

è il comando chiave per effettuare il download dei file attraverso il canale dei dati; in questo caso, osservando la dimensione del file e le considerazioni fatte precedentemente, si capisce che il profilo di linea utilizzato è 7Mbps; ogni volta che cambieremo profilo, indicheremo una dimensione diversa per il file da scaricare.

FINE

Indica all'interprete la fine delle istruzioni da eseguire.

Finalmente, si è proceduto a configurare la finestra iniziale di congestione (init_cwnd) su entrambi i lati (client e server FTP). Tale finestra iniziale è stata modificata usando l'opzione *initcwnd* attraverso il comando *ip route* presente in Linux. Quest'ultimo è utilizzato per il controllo del networking e del traffico di rete (sia IPv4 che IPv6) e consente di effettuare sia routing di base che avanzato, come politiche di instradamento.

Circa i valori di *initcwnd,* ho considerato **[6]**:

init_cwnd					
4	10	15	20	30	42

Tabella 10: Dimensione *initcwnd* (pkt)

Di seguito si trovano gli istruzioni utilizzati per la configurazione della finestra con *ip route*:

$ ip route help

```
nino@LabWan05:~$ ip route help
Usage: ip route { list | flush } SELECTOR
       ip route get ADDRESS [ from ADDRESS iif STRING ]
                            [ oif STRING ]  [ tos TOS ]
       ip route { add | del | change | append | replace | monitor } ROUTE
SELECTOR := [ root PREFIX ] [ match PREFIX ] [ exact PREFIX ]
            [ table TABLE_ID ] [ proto RTPROTO ]
            [ type TYPE ] [ scope SCOPE ]
ROUTE := NODE_SPEC [ INFO_SPEC ]
NODE_SPEC := [ TYPE ] PREFIX [ tos TOS ]
            [ table TABLE_ID ] [ proto RTPROTO ]
            [ scope SCOPE ] [ metric METRIC ]
INFO_SPEC := NH OPTIONS FLAGS [ nexthop NH ]...
NH := [ via ADDRESS ] [ dev STRING ] [ weight NUMBER ] NHFLAGS
OPTIONS := FLAGS [ mtu NUMBER ] [ advmss NUMBER ]
           [ rtt TIME ] [ rttvar TIME ] [reordering NUMBER ]
           [ window NUMBER] [ cwnd NUMBER ] [ initcwnd NUMBER ]
           [ ssthresh NUMBER ] [ realms REALM ] [ src ADDRESS ]
           [ rto_min TIME ] [ hoplimit NUMBER ]
TYPE := [ unicast | local | broadcast | multicast | throw |
          unreachable | prohibit | blackhole | nat ]
TABLE_ID := [ local | main | default | all | NUMBER ]
SCOPE := [ host | link | global | NUMBER ]
FLAGS := [ equalize ]
MP_ALGO := { rr | drr | random | wrandom }
NHFLAGS := [ onlink | pervasive ]
RTPROTO := [ kernel | boot | static | NUMBER ]
TIME := NUMBER[s|ms|us|ns|j]
nino@LabWan05:~$
```

Per visualizzare la tabella di routing di default:

$ ip route show

```
nino@LabWan05:~$ ip route show
192.168.131.0/24 dev eth0  proto kernel  scope link  src 192.168.131.136  metric 1
192.168.208.0/24 dev eth0  proto kernel  scope link  src 192.168.208.136
192.168.60.0/24 via 192.168.131.1 dev eth0  proto static
169.254.0.0/16 dev eth0  scope link  metric 1000
default via 192.168.208.1 dev eth0 onlink  initcwnd 4
nino@LabWan05:~$
```

Una volta riconosciuta la rotta in uscita (gateway) si prosegue a settare il numero di segmenti di partenza della finestra iniziale TCP. A tale scopo, viene eseguita la seguente istruzione:

$ ip route change default via <gateway> dev eth0 initcwnd <cwnd>

```
nino@LabWan05:~$ sudo ip route change via 192.168.208.1 dev eth0 onlink initcwnd 15
```

È possibile verificare la nuova tabella di routing con il comando:

$ ip route list

```
nino@LabWan05:~$ ip route list
192.168.131.0/24 dev eth0  proto kernel  scope link  src 192.168.131.136  metric 1
192.168.208.0/24 dev eth0  proto kernel  scope link  src 192.168.208.136
192.168.60.0/24 via 192.168.131.1 dev eth0  proto static
169.254.0.0/16 dev eth0  scope link  metric 1000
default via 192.168.208.1 dev eth0 onlink  initcwnd 15
nino@LabWan05:~$
```

Allora, per valutare la qualità di accesso ad Internet, ho considerato (per ogni profilo di linea) i valori di *initcwnd* sopra esposti in modo tale da avere delle limitazioni del throughput del protocollo, ossia della banda disponibile percepita dall'utente finale.

5.4 Considerazioni tecniche sulle prove

Prima di entrare nel pieno della fase sperimentale, si è effettuata una prova iniziale per analizzare le statistiche relative ad un singolo download e confrontarle con quanto realmente avviene in rete a livello protocollare. Utilizzando il profilo di linea 7Mbps, il client Linux e il server FTP, ho instaurato

la connessione con quest'ultimo mandando in esecuzione lo script bash precedentemente descritto (con l'unica differenza di eliminare il ciclo per poter scaricare il file una sola volta). Contemporaneamente si è catturato il traffico scambiato tra i due pc attraverso il software wireshark per poter analizzare nel dettaglio le statistiche ftp mostrate a video (standard output), illustrate qui di seguito:

```
Connected to 192.168.138.2.
220 (vsFTPd 2.0.6)
331 Please specify the password.
230 Login successful.
200 Switching to Binary mode.
Passive mode on.
250 Directory successfully changed.
local: ADSL7M.rnd remote: ADSL7M.rnd
227 Entering Passive Mode (192,168,138,2,127,29)
150 Opening BINARY mode data connection for ADSL7M.rnd (8960000 bytes).
226 File send OK.
8960000 bytes received in 11.88 secs (736.6 kB/s)
221 Goodbye.
```

Le prime nove righe caratterizzano l'apertura del canale comandi tra il server e il client FTP e comprendono sia l'autenticazione che la richiesta del file di 70Mbit da scaricare ("*local: ADSL7M.rnd remote: ADSL7M.rnd");* segue l'apertura del canale dati con la conferma da parte del server ("*150 Opening BINARY mode data connection for ADSL7M.rnd (8960000 bytes)"*) e l'invio del file. La penultima riga è quella più interessante, in quanto relativa alle statistiche del trasferimento:

8960000 bytes received in 11.88 secs (736.6 kB/s)

Sorge il problema di capire se il tempo stampato a video per il trasferimento (11.88 secs) comprende l'apertura del canale comandi oppure è relativo soltanto alla richiesta del file ("get" dello script bash).

A tale proposito, analizzando la cattura di wireshark e facendo una semplice differenza temporale tra il messaggio *"Request: RETR ADSL7M.rnd"* e il messaggio *"Response: 226 File Send OK"*, si ottiene il valore mostrato a video nello standard output. Dunque, le statistiche relative al trasferimento del file non tengono conto dell'apertura del canale controllo e si riferiscono al tempo che

intercorre tra la “get file” fino al completamento del download. Nello specifico si ha:

0.5 RTT per mandare la richiesta del file sul canale controllo

+

1 RTT per aprire il canale dati

+

0.5 RTT è il tempo che impiega il file per iniziare ad arrivare sul canale dati

+

Tempo trasferimento del file

=

---

2 RTT + Tempo trasferimento file = tempo necessario per i successivi file in FTP

CAPITOLO VI

RISULTATI

6.1 Definizioni

Prima di entrare nel merito della stima di banda ed evidenziarne la possibile dipendenza dalla dimensione della finestra iniziale TCP, è necessario dare delle definizioni:

- *Goodput* (Mbps): rappresenta il throughput a livello applicativo, ovvero il numero di bit utili per secondo raggiunti da un processo FTP.
- *Line Throughput* (Mbps): definito come il "data rate" medio (livello 2 della pila ISO/OSI) offerto dalla connessione xDSL.
- *Measurement Ratio*: è il parametro utilizzato per determinare l'accuratezza della stima di banda ed è descritto dalla seguente relazione:

  $$Measurement\ Ratio = Goodput / Line\ Throughput$$

  Alternativamente, dal punto di vista dell'utente finale, questo parametro rappresenta l'utilizzazione dell'applicazione d'utente della banda trasmissiva offerta dalla rete d'accesso.

Si capisce, quindi, come i test effettuati andranno a valutare proprio il ***goodput*** FTP tra un server e un client per i diversi valori della finestra iniziale di congestione (*initcwnd)* considerati; quest'ultimo aspetto sarà quello più importante perché metterà in evidenza come le eventuali differenze (in termini di ***goodput***) dipendano dalla dimensione della finestra iniziale di congestione.

I risultati che saranno presentati nel seguito saranno utili a valutare la capacità delle applicazioni di utente di sfruttare la larghezza di banda trasmissiva.

6.2 Risultati

Dopo aver eseguito gli script ftp sul sistema operativo Linux-Ubuntu (versione del Kernel 2.6.31, TCP CUBIC di default) e per ogni profilo di linea

considerato, RTT e *initcwnd* (come descritto nel capitolo cinque), sono stati raccolti i risultati ottenuti e poi elaborati; a titolo di esempio, si riporta lo standard output raccolto relativo al trasferimento di un file da 70Mbit tra un server e un client:

```
Connected to 192.168.138.2.
220 (vsFTPd 2.0.6)
331 Please specify the password.
230 Login successful.
200 Switching to Binary mode.
Passive mode on.
250 Directory successfully changed.
local: ADSL7M.rnd remote: ADSL7M.rnd
227 Entering Passive Mode (192,168,138,2,127,29)
150 Opening BINARY mode data connection for ADSL7M.rnd (8960000 bytes).
226 File send OK.
8960000 bytes received in 11.88 secs (736.6 kB/s)
221 Goodbye.
```

Si ricorda come tali script non facciano altro che automatizzare la ripetizione di trasferimenti FTP senza introdurre particolarità nell'esecuzione del protocollo, che risulta completamente gestito dal sistema operativo.

Come già introdotto nel precedente capitolo, i dati più interessanti ai fini della fase sperimentale sono le statistiche relative al trasferimento del file, ovvero la velocità di download. Per questo motivo sono state costruite delle tabelle, per ogni profilo di linea, che raccogliessero sia il ***goodput*** medio (rispetto alle quaranta prove) misurato per ogni scenario considerato, che il relativo "measurement ratio". Inoltre, per meglio caratterizzare la distribuzione dei valori di ***goodput*** misurati, si sono calcolati anche il quinto percentile (***goodput*** minimo) e il novantacinquesimo percentile (***goodput*** massimo).

Nelle seguenti tabelle vengono mostrate le statistiche relative ai diversi scenari di rete considerati:

Tabella 11(a): Statistiche FTP ADSL2+ 7Mbps

FTP ADSL2+ 7Mbps				
Rete Scarica		**Goodput FTP (Mbit/s)**	**Goodput FTP (Mbit/s)**	**Goodput FTP (Mbit/s)**
RTT	**(su 40 prove)**	**init_cwnd= 4pkt**	**init_cwnd= 10pkt**	**init_cwnd= 15pkt**
Media=10ms *Deviazione Standard= 5ms*	*Media*	6,037735849	6,143824462	6,228242253
	5° percentile	6,100425531	6,371555652	6,510445049
	95° percentile	6,008382229	5,946888371	5,953488372
	Ratio	0,862533692	0,877689208	0,889748893
Media=20ms *Deviazione Standard= 5ms*	*Media*	6,007123402	6,100425531	6,132208589
	5° percentile	6,064297803	6,266551961	6,469314079
	95° percentile	5,923966942	5,888304093	6,008382229
	Ratio	0,858160486	0,871489361	0,876029798
Media=40ms *Deviazione Standard= 5ms*	*Media*	5,881918516	6,019102531	6,055332629
	5° percentile	5,948547717	6,125238095	6,469314079
	95° percentile	5,496932515	5,938690961	5,968359701
	Ratio	0,840274073	0,859871791	0,865047518
Media=60ms *Deviazione Standard= 5ms*	*Media*	5,603064175	5,819598928	5,867714472
	5° percentile	5,968359702	5,943614257	6,003350083
	95° percentile	5,341281669	5,586905689	5,526599845
	Ratio	0,800437739	0,831371275	0,838244924
Media=100ms *Deviazione Standard= 10ms*	*Media*	5,560468543	5,786237461	5,827641099
	5° percentile	5,766693483	5,963394342	6,469314079
	95° percentile	5,209302325	5,430864197	5,711553784
	Ratio	0,794352649	0,826605351	0,832520157

Tabella 11(b): Statistiche FTP ADSL2+ 7Mbps

FTP ADSL2+ 7Mbps				
Rete Scarica		**Goodput FTP (Mbit/s)**	**Goodput FTP (Mbit/s)**	**Goodput FTP (Mbit/s)**
RTT	**(su 40 prove)**	**init_cwnd= 20pkt**	**init_cwnd= 30pkt**	**init_cwnd= 42pkt**
Media=10ms *Deviazione Standard= 5ms*	*Media*	6,171224462	6,062830409	5,945340687
	5^{o} percentile	6,341555655	6,179310344	6,008382229
	95^{o} percentile	5,983305509	5,799352751	5535135135
	Ratio	0,881603494	0,866118629	0,879334383
Media=20ms *Deviazione Standard= 5ms*	*Media*	6,104309586	5,964827041	5,866124266
	5^{o} percentile	6,360431651	6,371555556	6,184641932
	95^{o} percentile	6,085238045	5,943615257	5,693407461
	Ratio	0,872044226	0,852118148	0,852303466
Media=40ms *Deviazione Standard= 5ms*	*Media*	6,039602531	5,920541835	5,696327097
	5^{o} percentile	6,185238095	5,948566231	6,428699551
	95^{o} percentile	5,978690961	5,899352751	5,535135135
	Ratio	0,862800361	0,845791690	0,843761013
Media=60ms *Deviazione Standard= 5ms*	*Media*	5,832965418	5,762435542	5,577125072
	5^{o} percentile	5,968315262	5,928866812	5,983305509
	95^{o} percentile	5,741381669	5,743589743	5,393528969
	Ratio	0,833280774	0,823205077	0,796732153
Media=100ms *Deviazione Standard= 10ms*	*Media*	5,787181183	5,664378421	5,517414646
	5^{o} percentile	5,988304093	5,914191423	5,780645161
	95^{o} percentile	5,109090909	5,438543247	5,463414634
	Ratio	0,826740169	0,809196917	0,802444949

Tabella 12(a): Statistiche FTP ADSL2+ 10Mbps

FTP ADSL2+ 10Mbps				
Rete Scarica		**Goodput FTP (Mbit/s)**	**Goodput FTP (Mbit/s)**	**Goodput FTP (Mbit/s)**
RTT	**(su 40 prove)**	**init_cwnd= 4pkt**	**init_cwnd= 10pkt**	**init_cwnd= 15pkt**
Media=10ms *Deviazione Standard= 5ms*	*Media*	8,606886349	8,741305474	8,837377263
	5° percentile	8,685326547	9,183856502	9,258589511
	95° percentile	8,345558272	8,590604226	8,759623609
	Ratio	0,843812387	0,856990737	0,866409535
Media=20ms *Deviazione Standard= 5ms*	*Media*	8,557222245	8,615902397	8,793500790
	5° percentile	8,648648648	8,890604026	9,292196007
	95° percentile	8,258064516	8,235616960	8,597816960
	Ratio	0,838943357	0,844696313	0,861107920
Media=40ms *Deviazione Standard= 5ms*	*Media*	8,322158559	8,475447971	8,601427971
	5° percentile	8,497925311	8,712279226	9,183856502
	95° percentile	7,956487956	8,285451672	8,185451638
	Ratio	0,815897897	0,830926271	0,843277252
Media=60ms *Deviazione Standard= 5ms*	*Media*	7,912023191	8,106396412	8,270907016
	5° percentile	8,569037656	8,554720365	9,134701159
	95° percentile	7,236749116	7,981231167	7,901234567
	Ratio	0,775688548	0,794744746	0,810873236
Media=100ms *Deviazione Standard= 10ms*	*Media*	7,711133702	7,976320299	8,127212469
	5° percentile	8,231511254	8,278092531	8,519134775
	95° percentile	7,351040918	7,662776957	7,876923076
	Ratio	0,755993501	0,781992186	0,796785536

Tabella 12(b): Statistiche FTP ADSL2+ 10Mbps

FTP ADSL2+ 10Mbps				
Rete Scarica		**Goodput FTP (Mbit/s)**	**Goodput FTP (Mbit/s)**	**Goodput FTP (Mbit/s)**
RTT	**(su 40 prove)**	**init_cwnd= 20pkt**	**init_cwnd= 30pkt**	**init_cwnd= 42pkt**
Media=10ms *Deviazione Standard= 5ms*	*Media*	8,800633091	8,699968107	8,696012591
	5° percentile	8,842832469	8,707482993	9,309090902
	95° percentile	8,590604026	8,252365415	8,325203252
	Ratio	0,862807165	0,852938049	0,852550254
Media=20ms *Deviazione Standard= 5ms*	*Media*	8,650979960	8,582683764	8,560245346
	5° percentile	8,759623609	8,707482993	9,159212880
	95° percentile	8,462809917	8,547579298	8,345558272
	Ratio	0,848135291	0,841439582	0,839926014
Media=40ms *Deviazione Standard= 5ms*	*Media*	8,555400663	8,437705998	8,388285889
	5° percentile	8,692279226	8,590604027	9,053934571
	95° percentile	8,198217169	8,103990461	7,895142636
	Ratio	0,838764771	0,827226078	0,822380969
Media=60ms *Deviazione Standard= 5ms*	*Media*	8,240184417	8,196278118	8,169431957
	5° percentile	8,788888889	8,533333354	8,483844241
	95° percentile	7,962674961	8,102891373	8,146380270
	Ratio	0,807861213	0,803556678	0,800924701
Media=100ms *Deviazione Standard= 10ms*	*Media*	8,069981874	7,998125439	7,761397657
	5° percentile	8,519134775	8,258064516	8,379705401
	95° percentile	7,913446676	7,485538011	7,125956854
	Ratio	0,791174693	0,784129945	0,760921338

Tabella 13(a): Statistiche FTP ADSL2+ 12Mbps

FTP ADSL2+ 12Mbps				
Rete Scarica		**Goodput FTP (Mbit/s)**	**Goodput FTP (Mbit/s)**	**Goodput FTP (Mbit/s)**
RTT	**(su 40 prove)**	**init_cwnd= 4pkt**	**init_cwnd= 10pkt**	**init_cwnd= 15pkt**
Media=10ms *Deviazione Standard= 5ms*	*Media*	10,342999032	10,491645828	10,640758712
	5° percentile	10,404741744	10,520947945	11,263061411
	95° percentile	10,072131147	10,163882063	10,378378378
	Ratio	0,861916586	0,8743038192	0,886729892
Media=20ms *Deviazione Standard= 5ms*	*Media*	10,261806338	10,378378378	10,580113334
	5° percentile	10,369620253	10,422391857	11,150635208
	95° percentile	9,9497975709	10,080613091	10,172185430
	Ratio	0,855150528	0,8648648648	0,881676111
Media=40ms *Deviazione Standard= 5ms*	*Media*	10,088669950	10,225247198	10,353456628
	5° percentile	10,248540450	10,434735071	10,893617021
	95° percentile	9,7369255151	9,7601270842	9,9740259742
	Ratio	0,840722495	0,8521039336	0,862788052
Media=60ms *Deviazione Standard= 5ms*	*Media*	9,8335467352	10,120174751	10,302099382
	5° percentile	10,121911037	10,987652198	11,849566055
	95° percentile	9,1428571421	9,8146257891	10,240000142
	Ratio	0,819462227	0,8433478959	0,858508281
Media=100ms *Deviazione Standard= 10ms*	*Media*	9,4603125721	9,8060180919	10,031090774
	5° percentile	9,7912350593	9,9840259741	10,274247491
	95° percentile	8,6596194502	9,0686346862	9,6832151305
	Ratio	0,788359381	0,8171681743	0,835924228

Tabella 13(b): Statistiche FTP ADSL2+ 12Mbps

FTP ADSL2+ 12Mbps				
Rete Scarica		**Goodput FTP (Mbit/s)**	**Goodput FTP (Mbit/s)**	**Goodput FTP (Mbit/s)**
RTT	**(su 40 prove)**	**init_cwnd= 20pkt**	**init_cwnd= 30pkt**	**init_cwnd= 42pkt**
Media=10ms *Deviazione Standard= 5ms*	*Media*	10,589838743	10,454887476	10,296200092
	5° percentile	11,110307414	11,020627802	11,110307414
	95° percentile	10,100008382	9,9993726612	10,014669926
	Ratio	0,8824865619	0,8712406233	0,858016674
Media=20ms *Deviazione Standard= 5ms*	*Media*	10,487627089	10,325980066	10,265235370
	5° percentile	10,845542806	10,574870912	10,845542806
	95° percentile	9,4985003942	9,8304000125	9,9016921831
	Ratio	0,8739689248	0,8604983338	0,855436280
Media=40ms *Deviazione Standard= 5ms*	*Media*	10,251177199	10,178926441	10,113580246
	5° percentile	10,374247491	10,574870912	10,991055456
	95° percentile	9,7821283512	9,6376470587	9,6908517353
	Ratio	0,8542647665	0,8482438701	0,842798353
Media=60ms *Deviazione Standard= 5ms*	*Media*	10,197756847	10,109215686	10,018344136
	5° percentile	10,250255321	10,246985962	10,214463840
	95° percentile	9,7382706162	9,8304025316	9,6075058641
	Ratio	0,8498130705	0,8424346405	0,834862011
Media=100ms *Deviazione Standard= 10ms*	*Media*	9,8832574452	9,6414280125	9,2488333585
	5° percentile	10,039562686	9,8304002152	10,138613861
	95° percentile	8,8022922631	8,9043478261	8,5511482259
	Ratio	0,8236047871	0,8034523343	0,770736113

Tabella 14(a): Statistiche FTP ADSL2+ 18Mbps

FTP ADSL2+ 18Mbps				
Rete Scarica		**Goodput FTP (Mbit/s)**	**Goodput FTP (Mbit/s)**	**Goodput FTP (Mbit/s)**
RTT	**(su 40 prove)**	**init_cwnd= 4pkt**	**init_cwnd= 10pkt**	**init_cwnd= 15pkt**
Media=10ms *Deviazione Standard= 5ms*	*Media*	15,256383727	15,676952293	15,944636678
	5° percentile	15,450125733	16,985621325	16,530941704
	95° percentile	14,733812949	14,952538276	15,233057851
	Ratio	0,832226910	0,8709417945	0,869770711
Media=20ms *Deviazione Standard= 5ms*	*Media*	15,113151853	15,337210657	15,624298829
	5° percentile	15,360000102	15,455642737	16,530941704
	95° percentile	14,593824228	15,058823529	15,195383347
	Ratio	0,839719547	0,8520672587	0,852296466
Media=40ms *Deviazione Standard= 5ms*	*Media*	14,766864284	15,034257748	15,763951250
	5° percentile	15,120590648	15,195383347	16,725952813
	95° percentile	14,310559006	14,593824582	15,058823529
	Ratio	0,805523908	0,8352365415	0,848914425
Media=60ms *Deviazione Standard= 5ms*	*Media*	13,742916790	14,466862364	15,047759007
	5° percentile	15,022004890	15,034257748	15,360200010
	95° percentile	13,175125089	13,592925312	12,387096774
	Ratio	0,749668164	0,8037144575	0,820846552
Media=100ms *Deviazione Standard= 10ms*	*Media*	13,069096323	13,860394317	14,510019023
	5° percentile	14,343968871	14,456470588	15,360200010
	95° percentile	11,961064244	12,930097087	13,184549356
	Ratio	0,712911647	0,7700219065	0,792897214

Tabella 14(b): Statistiche FTP ADSL2+ 18Mbps

FTP ADSL2+ 18Mbps				
Rete Scarica		**Goodput FTP (Mbit/s)**	**Goodput FTP (Mbit/s)**	**Goodput FTP (Mbit/s)**
RTT	**(su 40 prove)**	**init_cwnd= 20pkt**	**init_cwnd= 30pkt**	**init_cwnd= 42pkt**
Media=10ms *Deviazione Standard= 5ms*	*Media*	15,810936227	15,499148936	15,294361697
	5^{o} percentile	16,369444932	15,599937121	15,700170357
	95^{o} percentile	14,582278321	15,123004826	14,804819277
	Ratio	0,8783853459	0,8055082742	0,834298587
Media=20ms *Deviazione Standard= 5ms*	*Media*	15,462560426	15,344642737	15,216081231
	5^{o} percentile	15,673469387	15,425645344	15,450125733
	95^{o} percentile	14,675159235	15,021136549	14,189376443
	Ratio	0,8590311347	0,8524801525	0,830028443
Media=40ms *Deviazione Standard= 5ms*	*Media*	15,220780702	15,100257386	14,877714101
	5^{o} percentile	15,420590625	15,334592595	15,270919635
	95^{o} percentile	14,548659006	14,554059286	14,445141065
	Ratio	0,8455989282	0,8389031881	0,811570702
Media=60ms *Deviazione Standard= 5ms*	*Media*	14,899372683	14,455636836	14,282282747
	5^{o} percentile	15,022004876	14,997689912	14,997558991
	95^{o} percentile	14,178304415	12,387052387	13,395348837
	Ratio	0,8277429268	0,8030909353	0,779090265
Media=100ms *Deviazione Standard= 10ms*	*Media*	14,274562572	13,439844235	12,802667223
	5^{o} percentile	14,459868871	14,479183032	14,593824228
	95^{o} percentile	13,395348835	12,541640576	10,472727272
	Ratio	0,7930312541	0,7466580135	0,698378094

Da queste tabelle si sono ottenuti dei grafici che mostrano le prestazioni, in termini di goodput medio rispetto alle quaranta prove indipendenti, ottenute nelle diverse configurazioni di scenari di accesso. Per semplicità espositiva, nei grafici si riportano i tre valori più significativi della finestra iniziale di congestione:

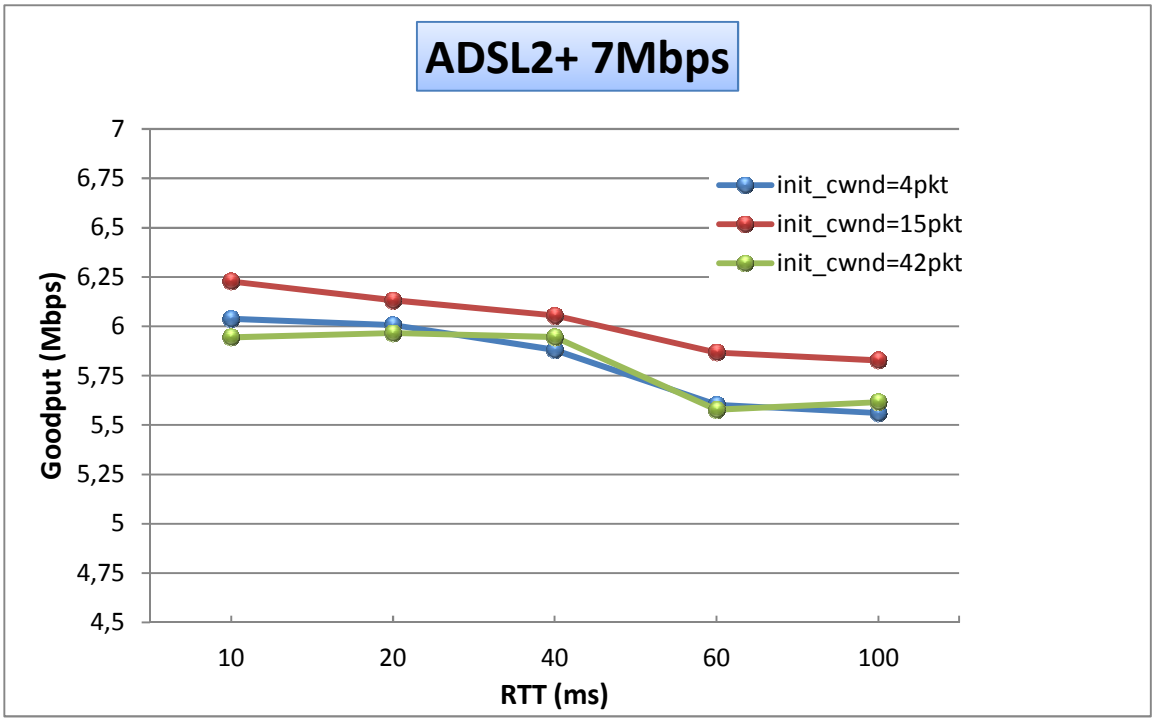


Figura 6.1: Measured Goodput FTP ADSL2+ 7Mbps

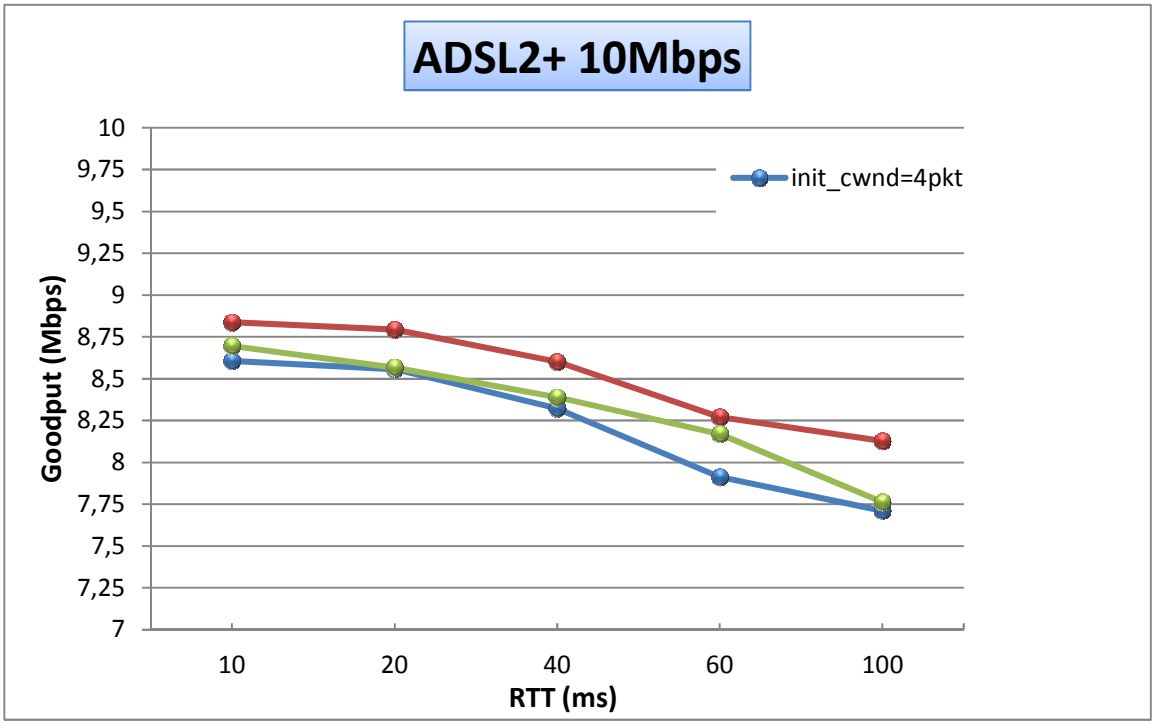


Figura 6.2: Measured Goodput FTP ADSL2+ 10Mbps

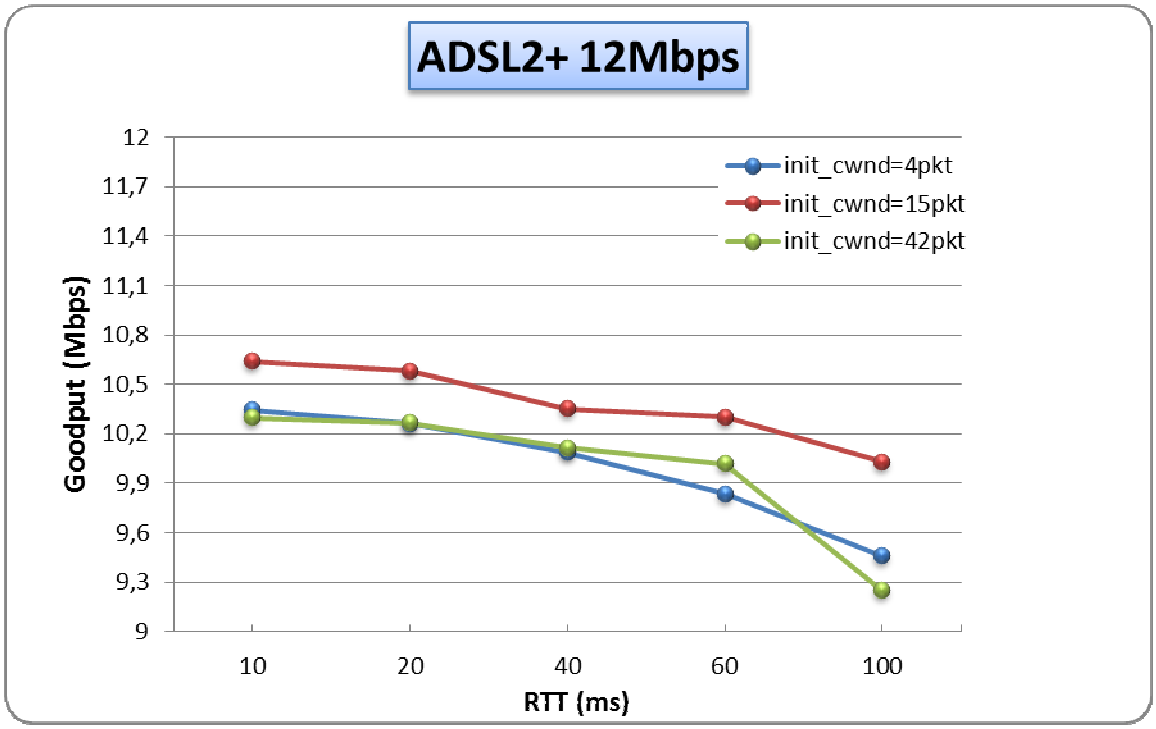


Figura 6.3: Measured Goodput FTP ADSL2+ 12Mbps

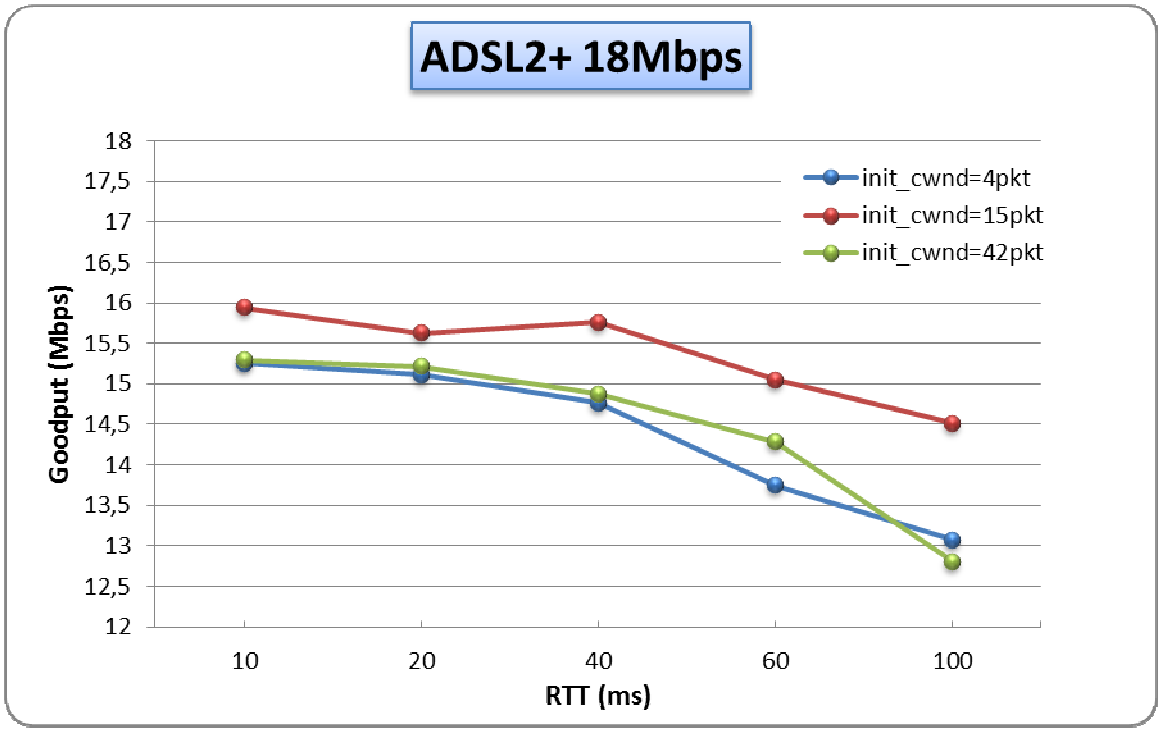


Figura 6.4: Measured Goodput FTP ADSL2+ 18Mbps

D'ora in poi si parlerà di ***goodput*** intendendo il parametro mediato sulle quaranta prove.

Tutti i precedenti grafici mettono subito in risalto come, per gli scenari considerati (rappresentativi delle configurazioni di rete effettivamente impiegate sul mercato), la dimensione della finestra iniziale di congestione possa avere un impatto considerevole sulle prestazioni FTP. In particolare, si nota come con una finestra iniziale di congestione pari a 15 segmenti[1] si ottengono prestazioni significativamente migliori, mentre con *initcwnd*=4 (default) e *initcwnd*=42 il ***goodput*** misurato è inferiore portando di conseguenza ad una inefficienza di banda, che di fatto, decresce significativamente per valori di RTT elevati. Tale comportamento è riconducibile principalmente ad un aspetto già introdotto nel terzo capitolo: allargamento della finestra iniziale di congestione. Precisamente, siccome il rate dei nostri collegamenti è elevato e i dati devono viaggiare velocemente, si vede come 4 segmenti non siano sufficienti per sfruttare

[1] dove la dimensione massima di un segmento (MSS) solitamente viene calcolata automaticamente a partire dei valori di MTU. Poiché l'intestazioni (ACKs) sono in genere di 40byte, MSS è di solito 40byte inferiore di un MTU, ovvero 1460byte.

efficientemente la capacità del link. D'altra parte, all'aumentare della finestra iniziale di congestione fino ad un valore pari a 15 segmenti, il tempo di trasmissione viene ridotto dell'ordine delle decine di millisecondi, rendendo più rapido lo scambio dati con conseguente aumento delle prestazioni e quindi un miglioramento della qualità complessiva di accesso ad Internet.

Nello specifico, avevamo già intuito come il punto debole di una finestra iniziale di congestione troppo elevata è l'aumento del numero di ritrasmissioni nel momento in cui sia presente un errore nella comunicazione. Infatti, negli scenari congestionati una finestra allargata può comportare la perdita di alcuni segmenti dati dovuto alla incapacità di gestione dei router di transito della rete. Ecco allora perché con *initcwnd*=42 abbiamo delle prestazioni basse, e in alcuni casi valori di ***goodput*** al di sotto di quelli ottenuti con *initcwnd*=4.

In definitiva, i risultati ottenuti confermano e mostrano come il ***goodput*** dell'applicazione per i diversi scenari di rete dipende significativamente dalla dimensione della finestra iniziale di congestione. Nel caso specifico delle prove effettuate, l'effetto è ancora più evidente considerando gli scenari di accesso con un elevato bit-rate ed elevata latenza: nel profilo di linea 18Mbps, ad esempio, per valori di RTT circa a 100ms il ***goodput*** misurato arriva ad avere valori addirittura superiori a 15Mbps, ossia più del 80% della velocità nominale di linea.

Per rappresentare l'efficienza del sistema di misura utilizzato ed evidenziare come la stima di banda si basi esclusivamente sulle prestazioni, è necessario considerare l'altro parametro definito all'inizio del capitolo: il Measurement Ratio. Abbiamo già visto, in effetti, come quest'ultimo quantifichi la differenza tra la stima di throughput e la massima banda fornita dalla rete di accesso. Ecco allora che, dopo averlo calcolato per ogni configurazione di scenario considerato (come riportato nel capitolo cinque), si sono elaborati dei grafici che rappresentassero l'efficienza del sistema di misura ipotizzato in funzione del BDP, che costituisce l'altro indice determinante per la valutazione a priori delle prestazioni. In particolare, sono stati considerati i valori di Measurement Ratio per tre casi di *initcwnd* ottenuti nelle diverse configurazioni di profili di linea e latenza; attraverso la variazione di questi ultimi, d'altronde, è

stato possibile calcolare ogni volta il BDP (in byte) e valutare l'efficienza del sistema di misura in funzione di essa.

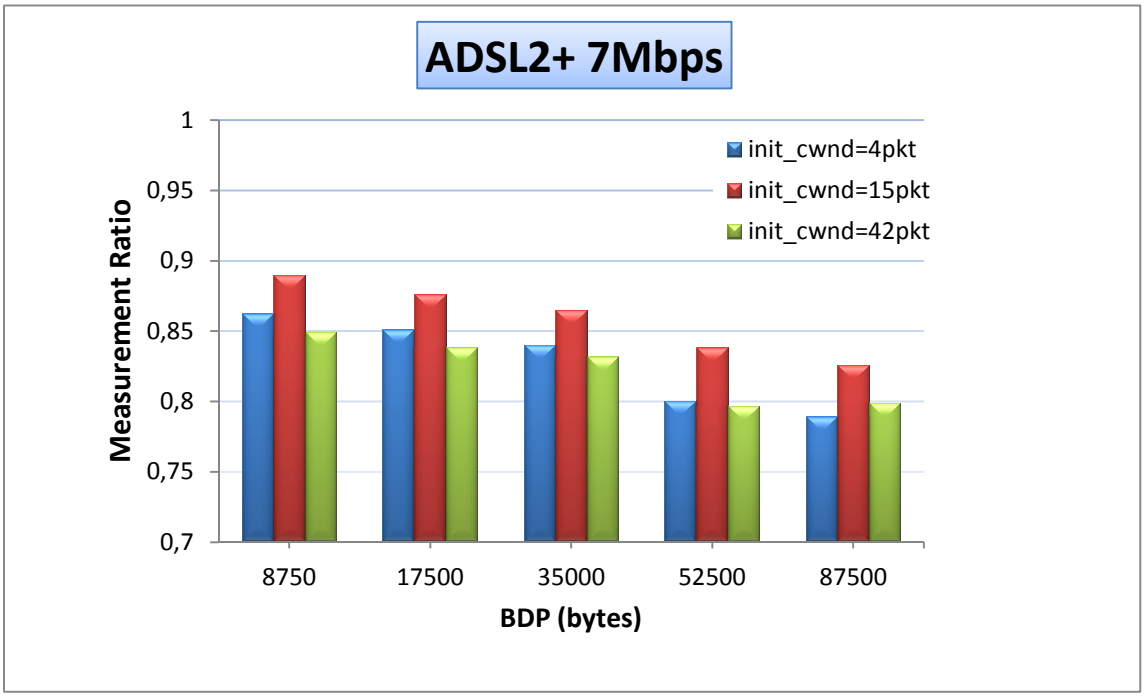


Figura 6.5: Impatto del BDP sulla misura di efficienza (primo scenario)

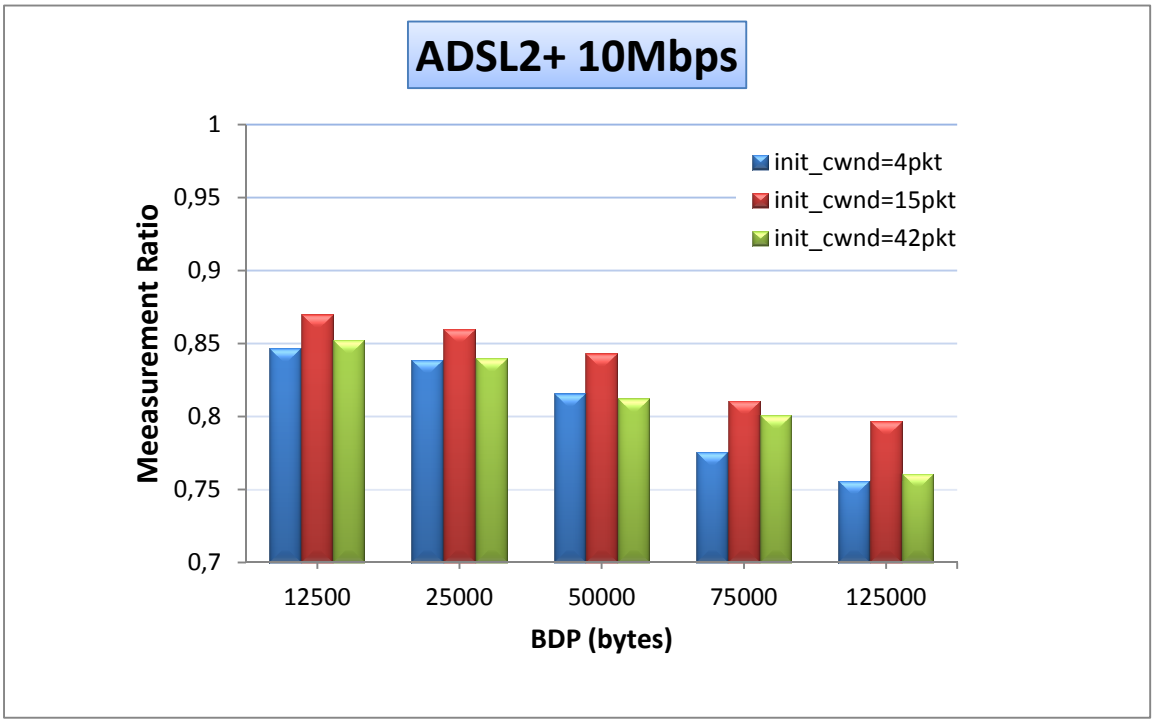


Figura 6.6: Impatto del BDP sulla misura di efficienza (secondo scenario)

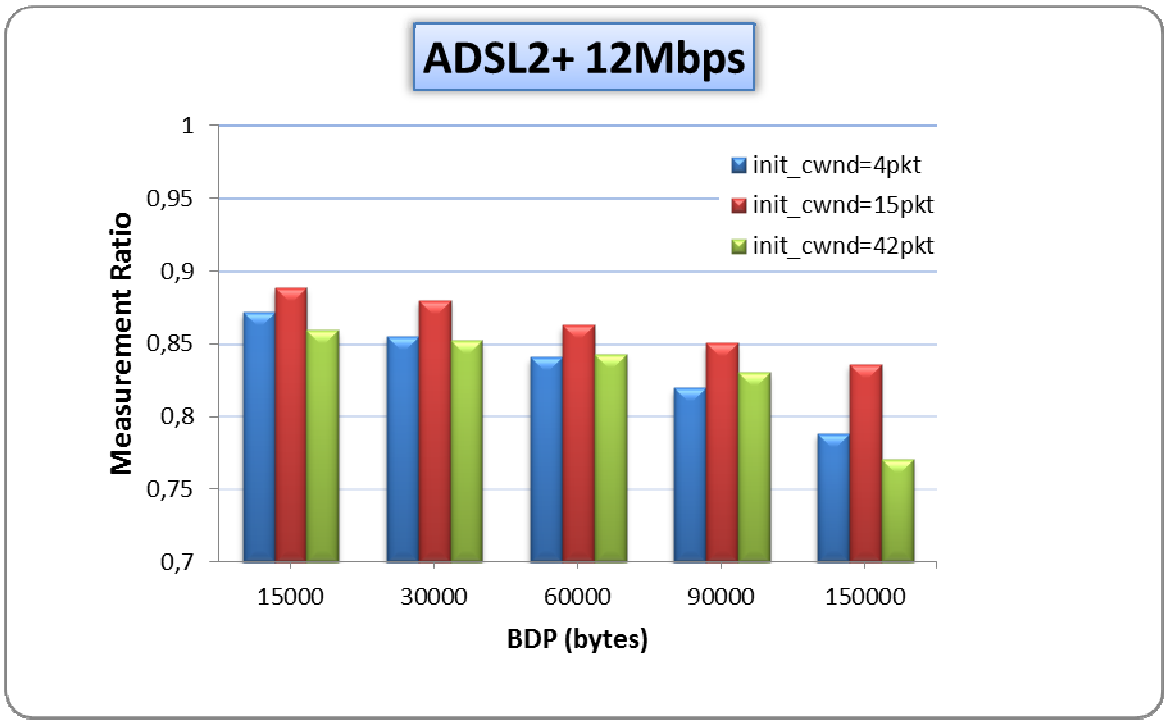


Figura 6.7: Impatto del BDP sulla misura di efficienza (terzo scenario)

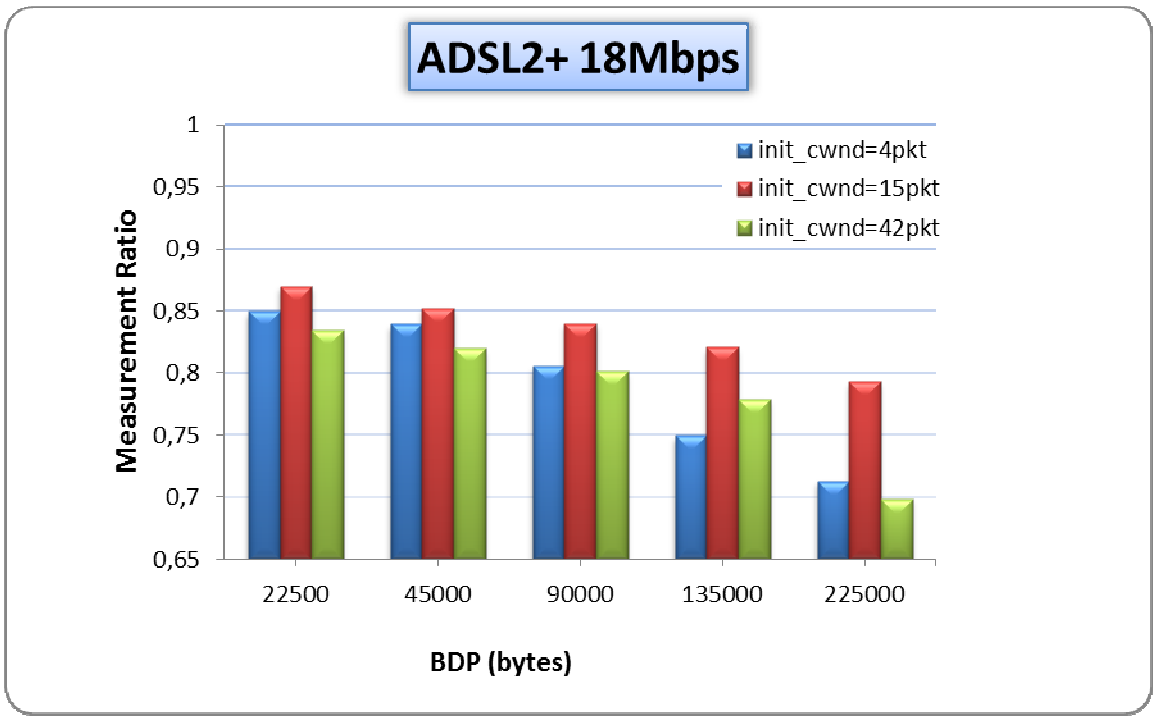


Figura 6.8: Impatto del BDP sulla misura di efficienza (quarto scenario)

Questi ulteriori grafici non fanno altro che confermare le considerazioni fatte in precedenza; infatti si nota come per *initcwnd*=4 e *initcwnd*=42, all'aumentare dei valori di BDP (ottenuti dall'aumento del RTT e/o del bit-rate di linea), le prestazioni diminuiscano più notevolmente con rispetto al caso *initcwnd*=15. Considerando, ad esempio, un valore di BDP pari a 225000byte (corrispondente ad una linea 18Mbps con latenza pari a 100ms) si vede come la banda stimata per un pc con un valore di finestra iniziale di congestione pari a 15 segmenti può raggiungere approssimativamente un'efficienza del 80%. Per *initcwnd*=4 e *initcwnd*=42, al contrario, i valori di efficienza sono pressoché del 70% e ciò sottolinea come all'aumentare del BDP l'impatto della finestra iniziale TCP sia più significativo sulla velocità effettiva e le prestazioni del sistema.

D'altra parte, per rappresentare l'impatto della dimensione della finestra iniziale di congestione ed evidenziare la sua influenza sulle prestazioni (in termini di ***googdput*** medio), si sono ottenuti ulteriori grafici che mostrano l'andamento della finestra per ogni valore di ritardo in rete, ottenute nelle diverse configurazioni di scenari di accesso:

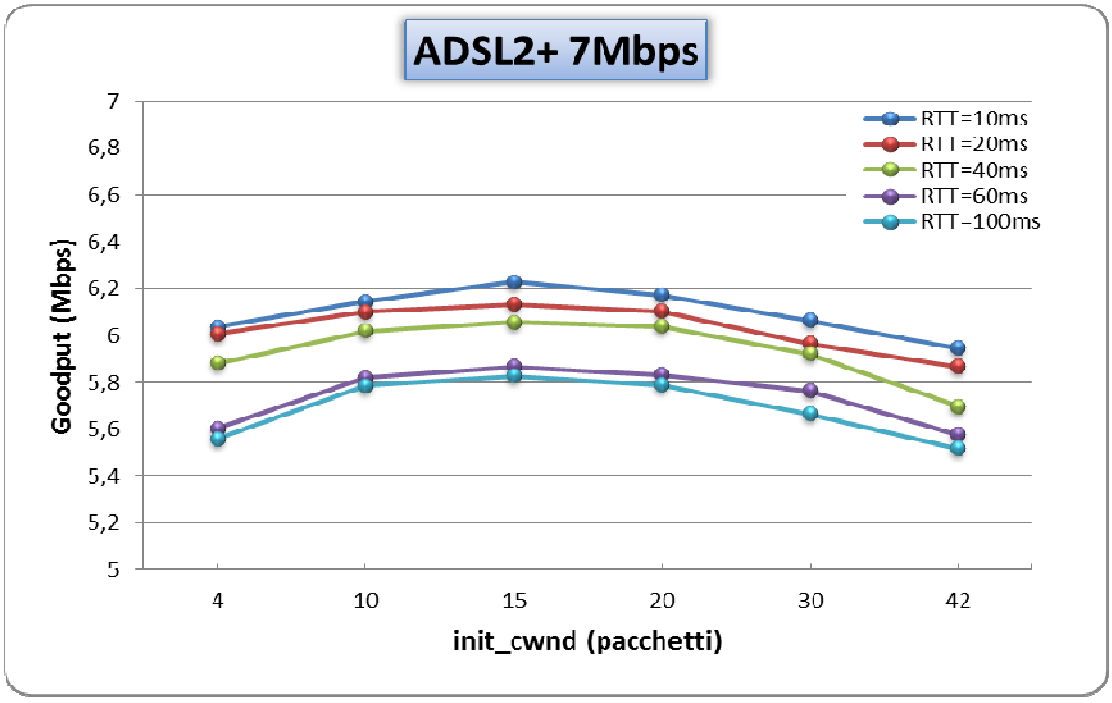


Figura 6.9: Impatto della finestra iniziale di congestione sulle prestazioni in rete

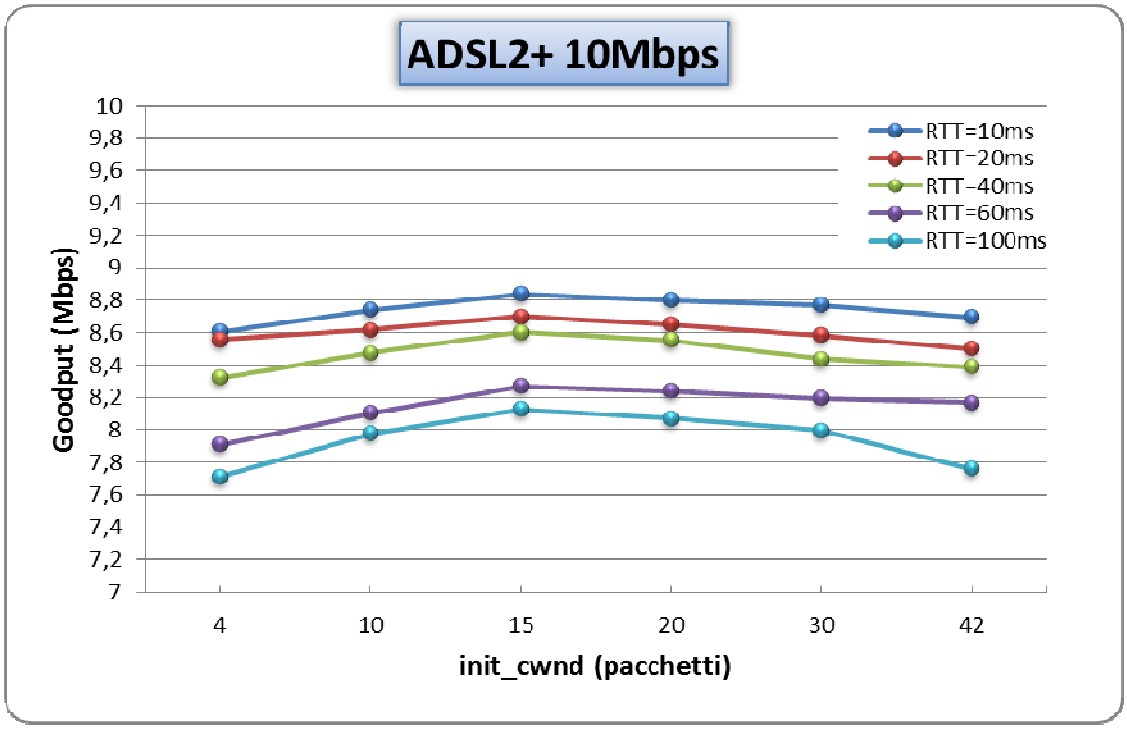


Figura 6.10: Impatto della finestra iniziale di congestione sulle prestazioni in rete

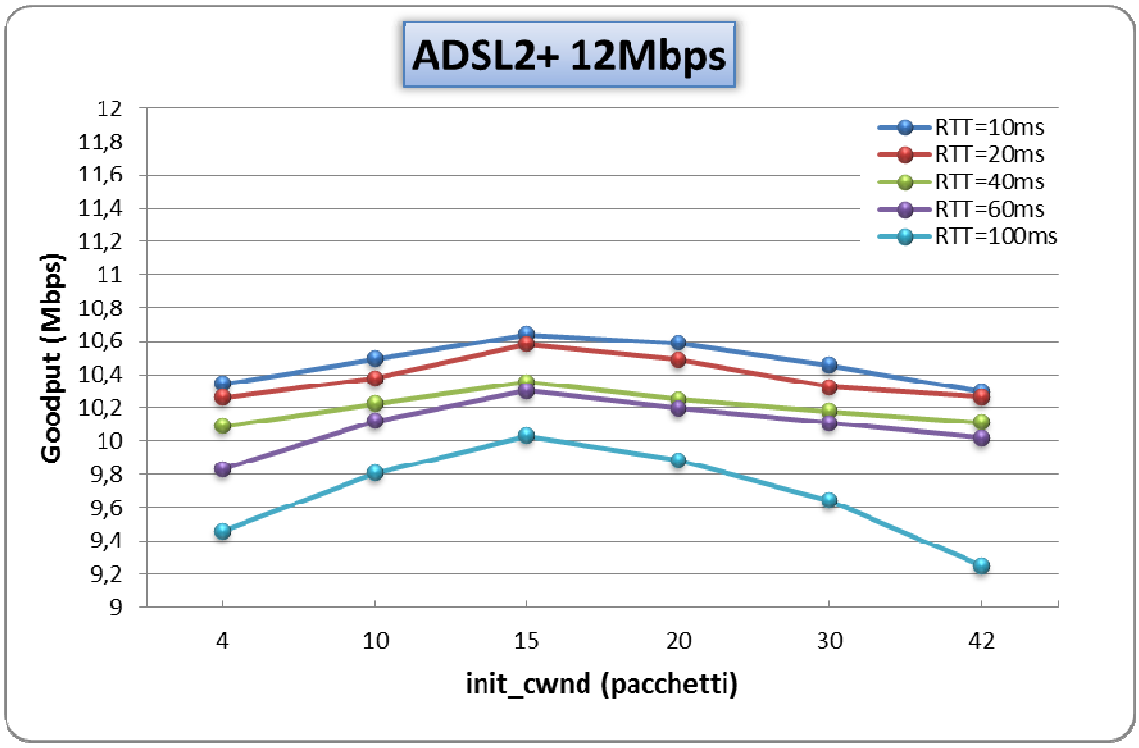


Figura 6.11: Impatto della finestra iniziale di congestione sulle prestazioni in rete

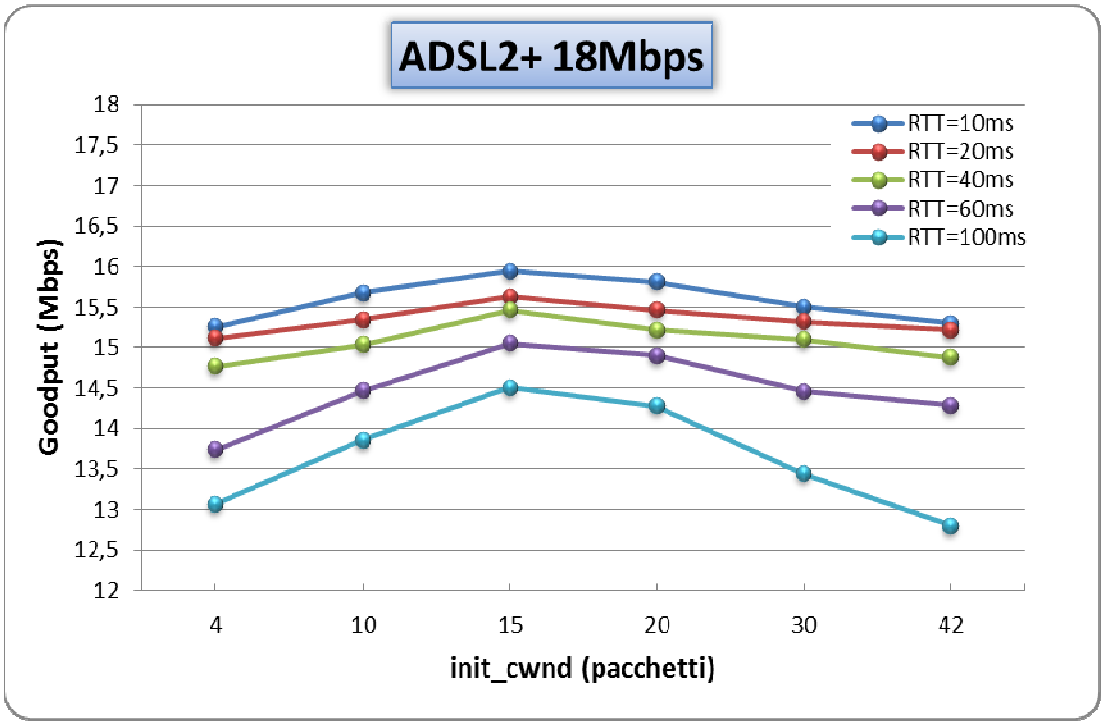


Figura 6.12: Impatto della finestra iniziale di congestione sulle prestazioni in rete

In particolare, si nota come nei diversi scenari considerati esiste un valore ottimo della finestra iniziale di congestione, dove si ottengono prestazioni significativamente migliori in termini di ***goodput*** medio. Per capire meglio questo comportamento abbiamo analizzato il traffico in rete scambiato sulle interfacce tramite uno dei principali software utilizzati per il monitoraggio del traffico di dati (Wireshark). L'idea è quella di stimare l'andamento della finestra di congestione. Per ragioni di semplicità e brevità espositiva si farà riferimento all'ultimo scenario di rete di accesso e saranno considerati tre valori di ritardo nel sistema:

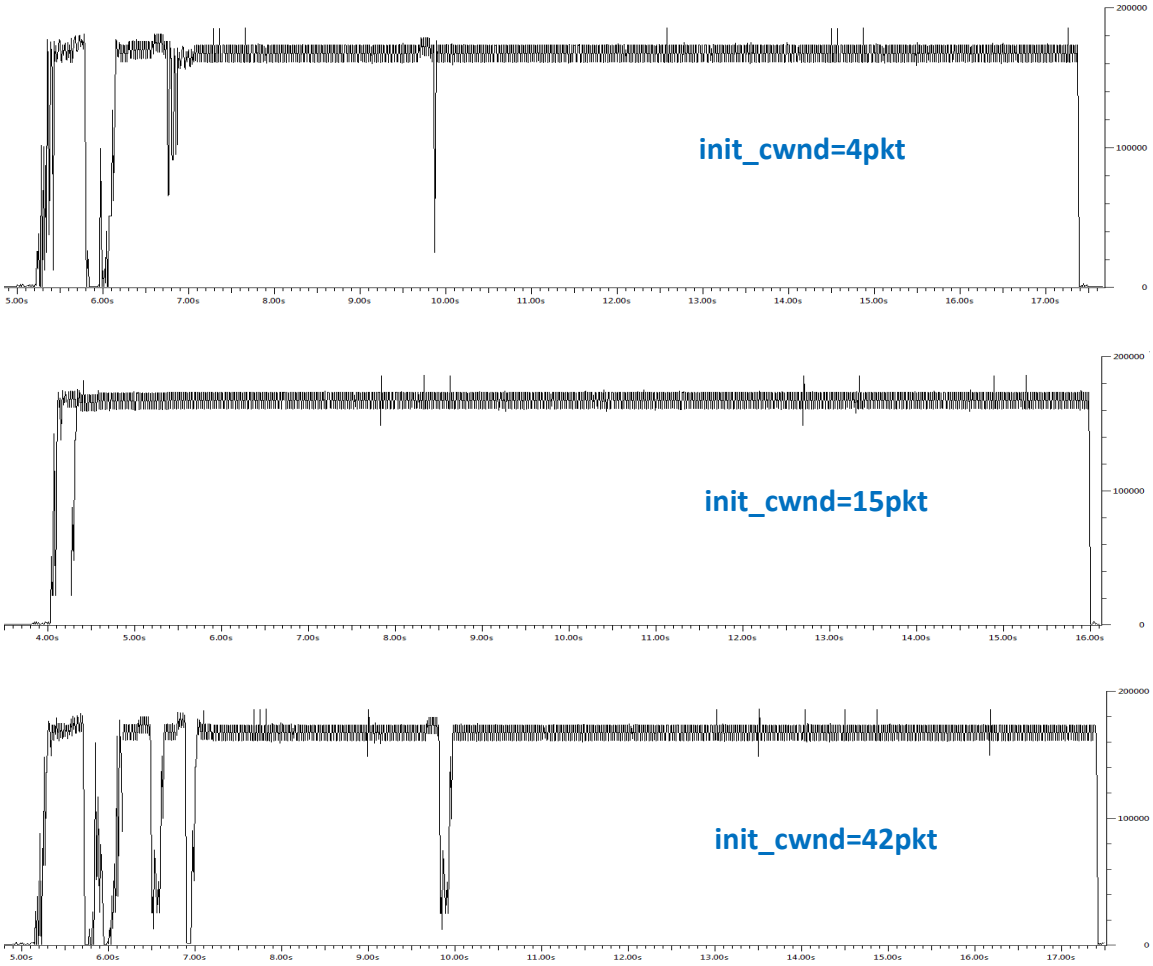
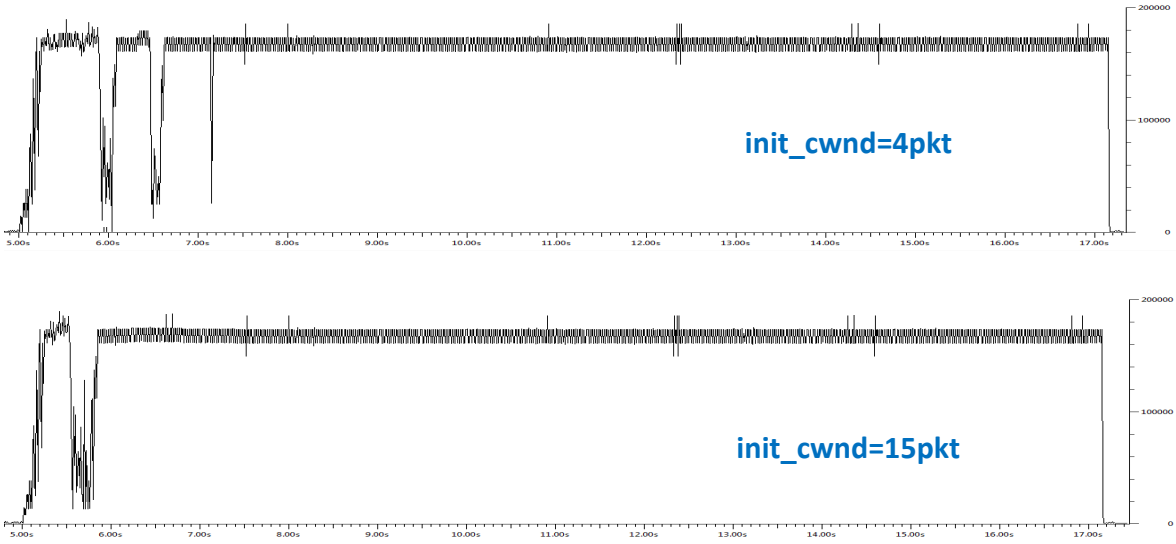


Figura 6.13: ADSL2+ 18Mbps Outsending data (RTT=10ms)



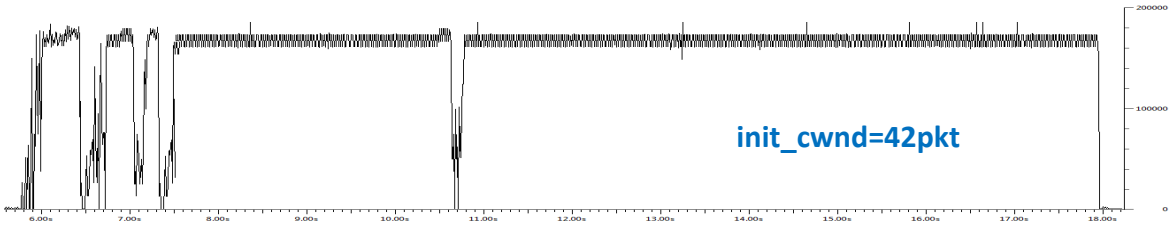


Figura 6.14: ADSL2+ 18Mbps Outsending data (RTT=20ms)

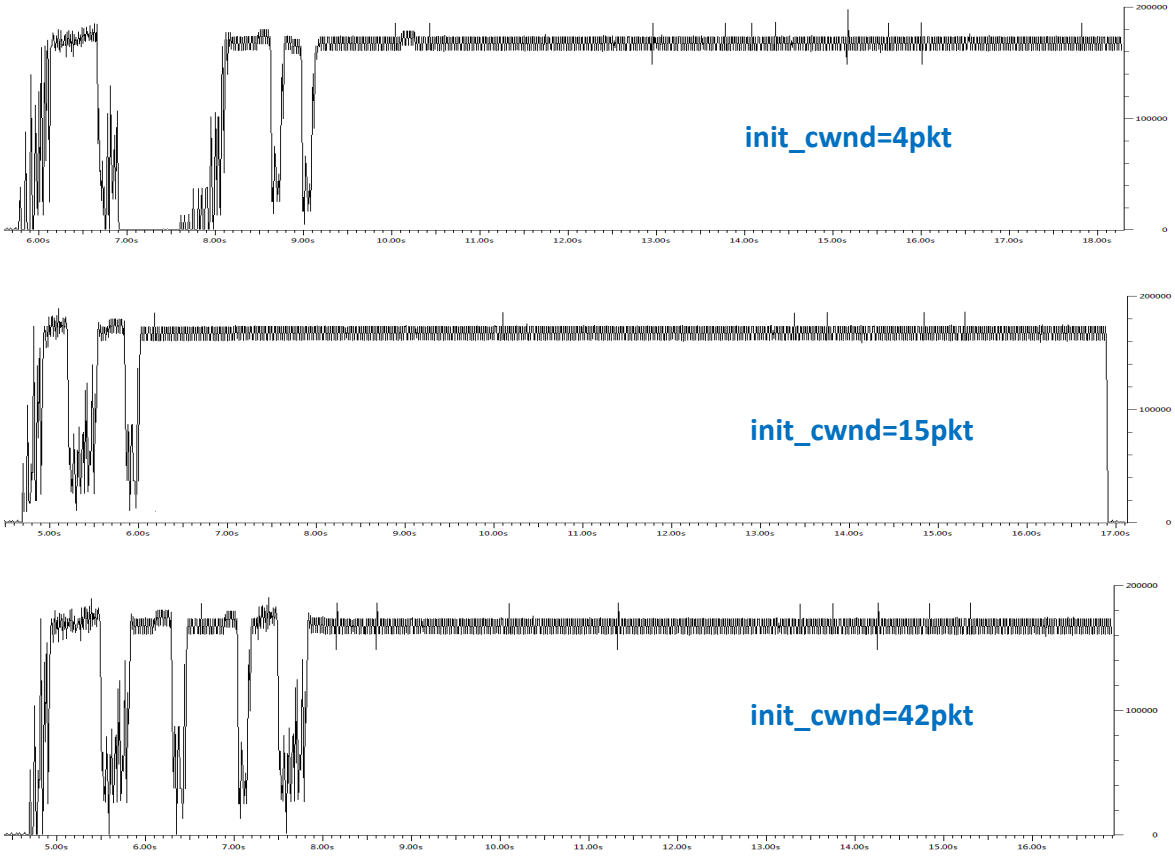


Figura 6.15: ADSL2+ 18Mbps Outsending data (RTT=40ms)

Questi ulteriori grafici non fanno altro che confermare quello visto in precedenza da un'altro punto di vista.

Quando viene creata una connessione, si sceglie una "dimensione di finestra accettabile". Il ricevente può specificare una finestra basata sulla dimensione del proprio buffer; se il mittente accetta questa dimensione di finestra,

non vi saranno problemi causati da esaurimento del buffer del ricevente; i problemi potrebbero però essere causati dalla congestione interna della rete. Come visto nel capitolo 1, la soluzione adottata consiste allora nel tenere conto dell'esistenza di due problemi potenziali (capacità del ricevente e capacità della rete) ed affrontarli separatamente.

Ogni mittente mantiene due finestre: una relativa al ricevente (*RcvWindow*) ed una relativa alla congestione (*CongWindow*). Il numero di byte che può essere spedito è il valore minimo tra le due finestre, il che significa che la finestra effettiva corrisponde al minimo tra ciò che è ritenuto adatto dal mittente e ciò che è ritenuto adatto dal ricevente. Dunque, le prestazioni TCP vengono fortemente limitate dalla finestra di ricezione e dalla finestra di congestione:

$$Throughput = min(CongWindow, RcvWindow) / RTT$$

D'altra parte, la versione 2.6.31 di Ubuntu utilizza normalmente il seguente settaggio di default per la finestra di ricezione:

$$tcp\_rmem = 4096 \;\; 87380 \;\; 3457024$$

Considerando i precedenti settaggi base del tcp_rmem, la *RcvWindow* oscillerà tra il valore minimo (pari a $RcvWindow_{min} = 4096 - 4096/4 = 3072$ byte) ed un determinato valore massimo, come descritto in seguito, in base alla presenza o meno della funzione di windows scaling:

- $RcvWindow_{max} = (87380 - 87380/4) = 65535$ byte con la funzione windows scaling disabilitata;
- $RcvWindow_{max} = (3457024 - 3457024/4) = 2592768$ byte con la funzione windows scaling abilitata (di default in Linux).

Dovuto al fatto che TCP può inviare solo il minimo tra la finestra di congestione e la finestra di ricezione, quest'ultima può limitare significativamente i miglioramenti delle prestazioni all'aumentare la dimensione della finestra di congestione. Infatti, questo fenomeno spiega ciò che accade nelle diverse configurazioni di scenario di accesso considerati.

Quando la dimensione della finestra di congestione è bassa (per esempio circa 4 segmenti), la finestra effettiva TCP è dominata nella maggior parte del tempo dalla *CongWindow;* dato che questo valore è più piccolo rispetto alla quantità di dati che il client può ricevere (*RcvWindow*). Tuttavia, le prestazioni non sono troppo elevate poichè 4 segmenti non sono sufficienti per sfruttare pienamente la capacità di link.

Per una *CongWindow* pari a 15 segmenti, la finestra effettiva TCP è determinata "equilibratamente" sia per la finestra di congestione sia per la finestra di ricezione durante tutto il trasferimento. Utilizzare 15 segmenti:

- permette di sfruttare più efficientemente la risorsa di rete con un conseguente aumento delle prestazioni;
- evita la saturazione del link riducendo la possibilità di congestione e degradazioni delle performance della rete.

Invece, per una finestra di congestione troppo elevata (per esempio 42 segmenti), la finestra effettiva nella maggior parte del tempo è fondamentalmente limitata dalla *RcvWindow* e non dalla *CongWindow,* questo per il fatto che il client non ha sufficiente memoria per allocare queste grosse quantità di dati. Perciò per evitare l'overflow del buffer ed una situazione di imminente congestione, la finestra di ricezione limita l'effetto della *CongWindow*. Ecco allora perché si vede un peggioramento delle prestazioni in termini di goodput medio.

D'altra parte, dal punto di vista di rete, se la finestra effettiva è troppo lunga questa può aumentare il traffico prima che la rete abbia ripristinato del tutto le condizioni precedenti di congestione, quindi in caso di una nuova congestione, le dimensioni della finestra diminuiranno rapidamente. Questi aumenti e diminuzioni alternati causano una oscillazione frenetica della dimensione della finestra ed una diminuzione dell'efficienza in rete.

Inoltre, dal punto di vista della connessione, quando i router sono troppo sotto pressione presentano problemi di gestione in periodi di congestione e

tendono a perdere pacchetti. Avere un buffer di dimensioni elevate non basta: anche con buffer infiniti, può scadere il TO per i pacchetti che si trovano in coda da troppo tempo e vengono inviati i duplicati, incrementando così il traffico complessivo sulla rete e peggiorando la situazione di congestione.

6.3 Considerazioni finali

Dopo aver illustrato la metodologia di misura utilizzata e i risultati ottenuti, si è giunti alle seguenti conclusioni:

- Realizzare un sistema di misura che si basi esclusivamente su misurazioni effettuate sul pc dell'utente finale, implica una stretta dipendenza con le caratteristiche del protocollo di trasporto sul sistema operativo utilizzato.
- Ottimizzare lo sfruttamento della banda e analizzare le prestazioni indicate dal protocollo lanciato da riga di comando, ha permesso di evidenziare l'impatto della finestra iniziale di congestione sulla valutazione della qualità di accesso ad Internet.
- La conseguenza principale derivante dai primi due punti consente nel superare alcune delle limitazioni del TCP, potenziando le prestazioni (in termini di throughput raggiungibile dall'applicazione) e migliorando il funzionamento dei protocolli sulle reti. In particolare, si è sottolineato come tali prestazioni peggiorino sensibilmente per *initcwnd*=4 e *initcwnd*=42, soprattutto nelle reti di accesso con elevata banda o latenza.

In definitiva, è evidente l'importanza della dimensione della finestra iniziale di congestione nella valutazione delle prestazioni attraverso un sistema di misura implementato direttamente sul pc dell'utente finale.

CONCLUSIONI

Nel caso delle Telecomunicazioni, e in particolare nelle tecnologie utilizzate nelle reti di accesso, ciascun operatore dovrebbe sempre indicare le caratteristiche del servizio offerto per poter mettere il cliente al riparo da eventuali differenze tra le proprie aspettative e ciò che si riceve realmente. In questo senso, pubblicizzare un prodotto illustrandone sia i fattori che lo valorizzano che quelli che lo limitano, rappresenta un passo fondamentale per rispondere alle nuove esigenze dell'utente finale e soprattutto se si mira a far sostenere parte dei costi delle nuove installazioni a chi fa uso del servizio.

Il presente lavoro ipotizza lo sviluppo di un sistema di valutazione delle prestazioni della rete d'accesso, basato su misure effettuate direttamente dalla postazione dell'utente finale, con l'intento di rappresentare il più fedelmente possibile l'esperienza del consumatore. In particolare, si è adottato un sistema di valutazione delle prestazioni in grado di stimare la banda effettivamente raggiunta mediante connessioni FTP.

Per queste ragioni si è approfondito un aspetto fondamentale del protocollo finora mai considerato nei meccanismi di stima della banda: l'impatto della dimensione della finestra iniziale di congestione sulla valutazione della qualità di accesso a Internet da postazione fissa. Come da letteratura, il valore di tale finestra determina il limite superiore del numero di dati che un'entità mittente può spedire in rete prima di ricevere un pacchetto di riscontro, avendo un impatto significativo nello sfruttamento della capacità di un link.

Nel dettaglio, la parte preliminare si è concentrata sullo studio del protocollo TCP, attraverso un'attenta analisi, che ha messo in evidenza come la finestra iniziale di congestione intervenga pesantemente sulle prestazioni, intendendo con prestazioni i valori di throughput effettivamente raggiungibili. Vista l'importanza della dimensione della finestra di congestione, si è proceduto

ad una fase di sperimentazione al fine di poter valutare le differenze prestazionali nella stima di banda dalla postazione di utente.

I test effettuati, basati su misure di throughput delle applicazioni, e attraverso il trasferimento di file da un server al client, hanno messo in evidenza una significativa differenza nelle prestazioni all'aumentare la dimensione della finestra iniziale di congestione. In particolare, i risultati ottenuti hanno dimostrato che un incremento opportuno della finestra porta a notevoli vantaggi:

- Gestire con efficienza la crescita delle dimensioni delle pagine web andando incontro alle diverse esigenze del mercato attuale.
- Velocizzare e completare con successo lo scambio di dati, con perdite di pacchetto gestite correttamente e senza perdite eccessive e sprechi di banda.
- Gestire più efficientemente la congestione di rete, rendendo più rapido il recupero di pacchetti persi.

Tutto ciò determina un sostanziale miglioramento sulla qualità di accesso ad Internet in termini di banda effettivamente raggiunta dall'utente finale e dimostra come una piccola, ma sostanziale modifica della dimensione di tale finestra può avere un notevole impatto sulle prestazioni del sistema.

Un successivo lavoro potrebbe riguardare la possibilità di analizzare se la scelta di una determinata dimensione della finestra iniziale di congestione ha un impatto sulle prestazioni raggiungibili dall'utente anche utilizzando protocolli differenti come ad esempio HTTP. Altrimenti, potrebbe essere interessante studiare l'impatto che ha trasferire altri tipi di file, che necessitano comunque di una trasmissione fedele e affidabile, come ad esempio video o file audio.

BIBLIOGRAFIA

[1] ETSI EG 202 057-4 "User related QoS parameter definitions and measurement", October 2005.

[2] E. Baruch, "An Experimental Investigation of TCP Performance in High Bandwith-Delay Product Paths", National University of Ireland, February 2007.

[3] R. Stevens, "TCP/IP Illustrated", Volume 1, 1994.

[4] W. Xiucho, M. Choon Chan, A. Ananda and C. Ganjihal, "Sync-TCP: A New Approach to High Seep Congestion Control", School of Computing, National University of Singapore,2009.

[5] M. Jain, R.S. Prasad, C. Drovolis, "The TCP Bandwith-Delay Product revisited: Network Buffering, Crosstraffic and Socket buffer Auto-Sizing", Technical Report, 2003.

[6] D. Kakadia, "Understanding Tuning TCP", March 2004.

[7] J.Postel and J.Reynolds, "File Transfer Protocol", STD 9, RFC 959, October 1985.

[8] J.Postel, "Transmission Control Protocol", RFC 793, September 1981.

[9] M. Allman, V. Paxon and W. Stevens, "TCP Congestion Control", RFC 2581, April 1999.

[10] A. Afanasyev, N. Tilley, P. Reiher and L. Kleinrock, "Host-to-Host Congestion Control for TCP", 2010.

[11] Yee-Ting Li, D.Leith and R.N. Shorten, "Experimental evaluation of TCP protocols for High-Speed Networks", 2010.

[12] N. Blefari - Melazzi, "Internet: Architettura, Principali protocolli e line evolutive", Versione 3.0, Luglio 2000.

[13] I. Rhee, L. Xu, "CUBIC: a new TCP-Friendly High-Speed TCP Variant"

[14] G. Turner, "Tuning Linux hosts for increased TCP performance", aprile 2008.

[15] Politecnico di Milano, "I protocolli applicativi", formato pdf, 2003.

[16] N. Dukkipati, T. Refice, Y. Cheng, J. Chu, N. Sutin, A. Agarwal, T. Herbert, A. Jain, "An Argument for increasing TCP's Initial Congestion Window", 2010.

[17] V. Paxon, M. Allman, S. Dawson, W. Fenner, J. Griner, I. Heavens, K. Lahey, J. Semke and B. Volz, "Known TCP Implementation Problems", RFC 2525, March 1999.

[18] P. Valerio, "Valutazione della Qualità di accesso ad Internet da postazione fissa in Reti Ottiche Passive", Tesi di Laurea Sperimentale, 2010.

[19] L. Angelo Maria, "Impatto dei sistemi operativi sulla valutazione della qualità di accesso ad Internet da postazione fissa", Tesi Sperimentale in commutazione, 2010.

[20] M. Allman, S. Floyd, and C. Partridge, "Increasing TCP's Initial Window", RFC 3390, 2002.

[21] M. Allman, V. Paxon, and J. Chu, "Computing TCP's Retransmission Timer, Internet-draft-paxson-tcpm-rfc2988bis-00", work in progress, 2010-

[22] M. Allman, V. Paxson, and E. Blanton, "TCP Congestion Control", RFC 5681, 2009.

[23] N. Cardwell, S. Savage, and T. Anderson, "Modeling TCP Latency", In Proceedings of IEEE Infocom, 2000.

[24] P. Benko and G. Malicsko and A. Veres, "A large-scale, passive analysis of End-to-End TCP performance over GPRS", 2004.

[25] S. Floyd, and K. Fall. "Promoting the Use of End-to-End Congestion Control in the Internet", *In IEEE/ACM Transaction on Networking*, 1999.

[26] S. Ramachandran, and A. Jain, “Web page stats: size and number of resources”, 2010.