ALMA MATER STUDIORUM – UNIVERSITÀ DI BOLOGNA

SCUOLA DI SCIENZE

Corso di Laurea in Informatica

# CRITERI DI VALUTAZIONE DEL PROTOCOLLO TCP

Relatore:

Prof. Vittorio Ghini

Presentata da:

Gessica Pazzaglia

Sessione I

Anno Accademico 2013/2014

# Indice

# Elenco delle figure

# 1. Introduzione

La diffusione di Internet e lo sviluppo di nuovi servizi applicativi nell'ambito delle reti di telecomunicazione ha portato alla diffusione di nuove tecnologie di trasmissione. Il protocollo di trasporto dati (TCP) non era stato progettato per operare in scenari diversi da quello della rete fissa o in ambiti in cui si ha necessità di trasmissione veloce dei dati. Con l'introduzione di nuovi scenari, si è notato che le prestazioni del TCP sono, in alcuni ambiti, peggiorate. È stato inoltre notato che in certi ambienti è preferibile l'utilizzo del protocollo UDP a scapito dell'affidabilità nella consegna dei pacchetti garantita dal protocollo TCP. Per questo motivo, il protocollo TCP ha subito delle modifiche negli anni e sono state introdotte alternative per migliorare le prestazioni di comunicazione. Le modifiche sono state fatte in base a criteri di valutazione del protocollo e sono state attuate per migliorare le prestazioni in ambienti cosiddetti ostili.

Questa tesi si occuperà di mostrare il protocollo TCP e di spiegarne brevemente il comportamento, approfondendo il concetto del meccanismo di controllo di congestione adottato da TCP, che sarà il fulcro dell'evoluzione del protocollo. Si proseguirà, quindi, vedendo come si è evoluto il protocollo TCP, al fine di essere sempre più efficiente nel soddisfare le nuove esigenze richieste dalla rete e mutate nel corso del tempo. Si andrà poi a vedere l'evoluzione degli algoritmi di controllo di congestione utilizzati dai vari TCP partendo dal primitivo TCP Berkeley e arrivando ai più recenti TCP SACK, TCP Westwood e TCP Westwood+, passando per i diffusi TCP Reno, TCP New Reno, TCP Vegas e TCP Veno. Verranno poi mostrate le interazioni tra questi TCP quando si trovano a condividere uno stesso collo di bottiglia. Saranno quindi confrontati tra loro in vari modi, al fine di capire in che scenari sia più conveniente utilizzarne uno rispetto ad un altro e quali benefici sono stati trovati in termini di stima della banda, al fine di garantire equità e di non congestionare la rete. Questi, equità e non congestione, sono i criteri su cui si basa la valutazione del protocollo TCP poiché, per essere efficiente, il protocollo TCP non deve congestionare la rete e deve quindi assegnare una giusta banda ai vari flussi, quando si presentano in maniera concorrente, e deve garantire che la latenza, cioè il tempo impiegato da un pacchetto per andare da un punto all'altro della rete, sia minima.

Dopo il discorso dell'evoluzione del TCP, vedremo come si comporta quest'ultimo quando interagisce con flussi di dati UDP e si trova a condividere con essi uno stesso collo di bottiglia. Questo è diventato un discorso fondamentale negli ultimi tempi, poiché nuove applicazioni, come ad esempio quelle che si avvalgono di meccanismi di telefonia di Voice Over IP (VoIP) o quelle di streaming audio/video, richiedono una banda minima garantita e preferiscono dunque un protocollo più "snello", senza cioè i ritardi causati dai sistemi di controllo di congestione, a scapito dell'affidabilità di consegna del pacchetto. Vedremo quindi vari scenari di interazione tra TCP e UDP e ne confronteremo i risultati per valutare il comportamento di TCP in questo ambito.

Parleremo poi del TCP-friendly, un meccanismo atto a inserire un controllo di congestione, simile a quello utilizzato nel protocollo TCP, alle applicazioni non basate su TCP, come ad esempio le applicazioni real-time. Il sempre più grande utilizzo di applicazioni che non si basano sul protocollo TCP ha reso fondamentale l'inserimento di questi algoritmi "TCP-friendly" poiché, in caso di congestione della rete, un protocollo senza un meccanismo di controllo di congestione, continuerebbe a inviare pacchetti come se la rete non fosse congestionata, causando la perdita dei pacchetti inviati e rischiando di far collassare la rete. Un flusso verrà considerato TCP-friendly se si comporterà come un flusso TCP in termini di controllo di congestione, se attuerà quindi un'equa distribuzione della larghezza di banda dove essa non è ancora presente. Ci occuperemo di classificare brevemente i protocolli TCP-friendly e di andarne a valutare alcuni, considerando degli scenari dove si trovano a condividere uno stesso collo di bottiglia con dei flussi TCP e vedremo come si comporta TCP quando interagisce con il protocollo TCP-friendly.

Come ultima cosa, andremo a vedere come si comporta il protocollo TCP quando sono i suoi stessi flussi a variare nella rete. Vedremo cioè come si comportano tra loro più flussi TCP quando si trovano a condividere tra loro uno stesso collo di bottiglia in una rete ad alta velocità. Vedremo come variano le prestazioni in una stessa rete quando il collo di bottiglia è condiviso da due o quattro flussi TCP.

Infine, mostreremo che conclusioni sono state tratte dal confronto tra i vari scenari.

# 2. Protocolli di trasporto

Nel modello ISO-OSI, al quarto livello, troviamo il livello di trasporto, il cui compito è quello di fornire un canale logico-affidabile di comunicazione end-to-end per pacchetti. Nello stack protocollare Internet, i protocolli di trasporto più utilizzati sono TCP e UDP. TCP è il più complicato fra i due e fornisce un servizio end-to-end orientato alla connessione e al byte, con verifica del corretto ordine di consegna, controllo di errore e di flusso. UDP, invece, è un protocollo più snello e fornisce un servizio a datagrammi, senza connessione, con un meccanismo di riduzione degli errori e con porte multiple, che non garantisce né il recapito né la sequenzialità dei datagrammi. Vedremo in questa sezione una breve descrizione dei due protocolli e andremo a vedere i più famosi algoritmi che hanno portato alla modifica nel tempo del protocollo TCP.

## 2.1. Il protocollo TCP

Il Transfer Control Protocol (TCP) è il protocollo di trasporto più utilizzato nelle reti e garantisce la consegna affidabile e in sequenza di un flusso di byte. È un protocollo full-duplex: per ciascuna connessione TCP fornisce una coppia di flussi di byte, una in ciascuna direzione. Il protocollo TCP prevede anche un meccanismo di controllo di flusso per ciascuno di questi flussi di byte, consentendo al ricevitore di limitare la quantità di dati che il mittente può inviare in un certo istante. Infine, il protocollo TCP fornisce il supporto ad un meccanismo di demultiplexing che consente a più programmi applicativi in un host di sostenere simultaneamente una conversazione con le loro pari entità. Il protocollo TCP realizza anche un meccanismo di controllo della congestione. L'idea su cui si basa questo meccanismo è quella di controllare la velocità con cui TCP invia i dati per evitare che il mittente sovraccarichi la rete.

### 2.1.1. Funzionamento di TCP

TCP necessita di un'esplicita fase di instaurazione della connessione, durante la quale le due parti coinvolte nella connessione si accordano per scambiarsi dati reciprocamente, e di un'esplicita fase di terminazione della connessione.

Una connessione TCP inizia quando un client effettua l'apertura attiva di un canale di comunicazione verso il server. Quando una delle due parti ha terminato di inviare i dati chiude la connessione nella direzione corrispondente, spingendo il protocollo TCP ad iniziare uno scambio di messaggi per la chiusura della connessione.

L'algoritmo usato dal protocollo TCP per instaurare e terminare una connessione viene chiamato three-way handshake ("stretta di mano a tre vie") e richiede lo scambio di tre messaggi fra mittente e destinatario:

1. il mittente deve inviare un segmento SYN al destinatario, contenente un numero di sequenza iniziale che serve per negoziare la dimensione massima dei pacchetti da trasmettere;
2. il destinatario risponde con un singolo segmento ACK che serve per confermare il numero di sequenza del mittente e un segmento SYN che serve per annunciare il proprio numero iniziale di sequenza;
3. il mittente invia nuovamente un segmento ACK per confermare il numero di sequenza del destinatario.

A questo punto la connessione è stabilita e può iniziare il trasferimento dati.

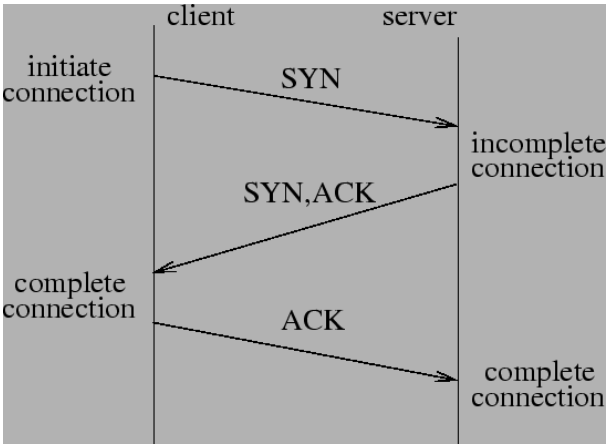


**Figura 1:** *Diagramma temporale per il three-way handshake*

Il TCP si aspetta di ricevere sempre i dati in maniera ordinata, altrimenti deduce che uno o più pacchetti siano andati persi o ritardati. Per questo motivo, il destinatario, per ogni segmento ricevuto, invia un ACK per confermare la corretta ricezione dei dati. Il mittente, ad ogni segmento inviato, avvia un Retrasmission Time Out (RTO) che indica il tempo

massimo di attesa per la ricezione dell'ACK da parte del destinatario. Se entro tale tempo, il mittente non riceve nessun ACK, il TCP assume che tutti i pacchetti trasmessi, a partire dal primo non riscontrato, siano andati persi e quindi li ritrasmette.

Il TCP utilizza un algoritmo di sliding window ("finestra scorrevole") per migliorare le performance di trasmissione. La dimensione della finestra viene impostata in fase di handshaking ma può assumere alcune variazioni durante la connessione. Tali variazioni possono essere attuate dal controllo di congestione che permette di limitare la quantità di dati trasmessi e non ancora riscontrati dal mittente, adattando il flusso di dati inviato alle eventuali condizioni di congestione della rete.

### 2.1.2. Formato del segmento

Il protocollo TCP non trasmette singoli byte ma memorizza un numero sufficiente di byte ricevuti dal processo applicativo sull'host mittente finché non ha riempito un pacchetto di dimensioni ragionevoli, dopodiché invia tale pacchetto al destinatario.

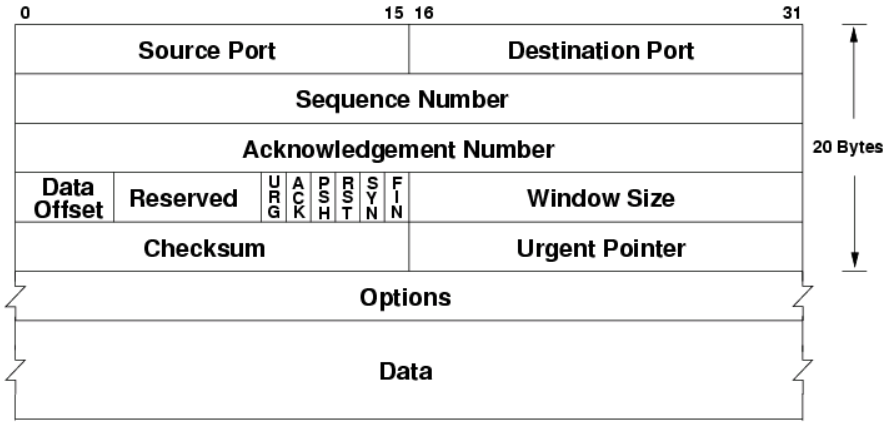


**Figura 2:** *Pacchetto TCP*

I pacchetti scambiati fra pari entità del protocollo TCP sono detti segmenti perché ognuno di essi trasporta un segmento del flusso di byte. Un pacchetto TCP è strutturato nel seguente modo:

- Source port (16 bit): identifica il numero di porta sull'host mittente associato alla connessione TCP
- Destination port (16 bit): identifica il numero di porta sull'host destinatario associato alla connessione TCP

- Sequence number (32 bit): indica il numero di sequenza del pacchetto che si sta inviando
- Acknowledgment number (32 bit): indica il numero del pacchetto che è stato ricevuto e di cui si sta comunicando l'acknowledgment
- Data offset (4 bit): indica la lunghezza dell'header del pacchetto TCP
- Reserved (4 bit): bit non utilizzati e predisposti per sviluppi futuri del protocollo
- Flags (6 bit): bit utilizzati per implementare il servizio di ritrasmissione in caso di perdita e per effettuare l'handshake iniziale. Ognuno dei 6 bit può essere visto come un campo a se stante:
  - URG indica che il segmento contiene dei pacchetti che il livello applicativo mittente ha marcato come urgenti
  - ACK, se settato a 1, indica che il pacchetto inviato è giunto correttamente a destinazione
  - PSH, se settato a 1, indica che i dati in arrivo non devono essere bufferizzati ma passati subito ai livelli superiori dell'applicazione
  - RTS, SYN, FIN vengono utilizzati per il processo di handshake e per la chiusura della connessione
- Window size (16 bit): indica la dimensione della finestra di ricezione dell'host mittente, cioè il numero di byte che il mittente è in grado di accettare a partire da quello specificato dall'Acknowledgment number
- Checksum (16 bit): utilizzato per la verifica della validità del segmento
- Urgent pointer (16 bit): puntatore a dato urgente, ha significato solo se il flag URG è settato a 1
- Options: opzioni (facoltative) per usi del protocollo avanzati.

### 2.1.3. Criteri di valutazione

Per misurare l'efficienza o l'inefficienza di un protocollo in certi ambiti, si vanno a guardare alcuni parametri, al fine di capire in che caso sia meglio utilizzare un protocollo piuttosto che un altro, o quali controlli eventualmente eseguire al fine di migliorare il protocollo. Due dei principali criteri di valutazione del protocollo TCP sono la latenza, cioè il tempo che un pacchetto impiega per raggiungere due punti distinti della rete, e il fairness, cioè l'equità

nell’assegnare la banda a diverse connessioni. Questi criteri sono stati precedentemente discussi e approfonditi in [1].

#### 2.1.3.1. Latenza e banda

Con latenza si indica il tempo che un pacchetto impiega per andare da un punto ad un altro della rete. A causa della variabilità della latenza il protocollo di trasporto ha difficoltà nel calcolare il tempo impiegato da un pacchetto per viaggiare da un host all’altro e tornare indietro, quindi la stima del tempo massimo di attesa di ricezione di un ACK prima della ritrasmissione dei dati risulta imprecisa causando ritrasmissioni ridondanti. Le principali cause di latenza sono:

- Ritardo di propagazione (propagation delay): è il tempo necessario al segnale fisico per propagarsi lungo la linea di trasmissione fino alla destinazione finale, dipende quindi dal mezzo fisico
- Serializzazione: è la conversione in byte dei dati memorizzati nella memoria di un computer in un flusso di bit seriale da trasmettere tramite mezzi di comunicazione
- Protocolli dati: gli handshake utilizzati dai protocolli di comunicazione per instaurare ed utilizzare il canale di collegamento tra mittente e destinatario richiedono un tempo di propagazione che va ad aggiungersi alla latenza
- Routing e switching: i pacchetti vengono inoltrati dalla sorgente alla destinazione attraverso una serie di routers o switches che aggiornano continuamente le loro decisioni su quale sia il router successivo migliore per instradare il pacchetto. Un’interruzione di circuito o una congestione su un collegamento lungo il percorso può modificare il percorso di instradamento che, a sua volta, influenza la latenza.
- Accodamento e buffering: un problema che può aggiungere ulteriore tempo alla latenza è il cosiddetto problema della latenza di coda ed è dovuto alla quantità di tempo che un pacchetto spende in attesa in una trasmissione di coda.

#### 2.1.3.2. Fairness e Stability

Con fairness s’intende l’equità nell’assegnare la banda a diverse connessioni TCP in uno stesso canale; è un obiettivo fondamentale nella progettazione di un controllo di congestione. Quando abbiamo più connessioni TCP in uno stesso canale ed una di queste ha un valore RTT (round-trip-time, cioè la migliore stima attuale del tempo di andata e ritorno

dalla destinazione con la quale si sta comunicando) elevato rispetto alle altre, questa viene penalizzata. In questo modo non viene garantita un'assegnazione equa delle risorse di rete infatti, in presenza di connessioni con RTT differenti, il fairness non può essere garantito poiché il tasso dell'incremento della finestra di congestione è inversamente proporzionale al valore di RTT. Viceversa, collegamenti con un valore di RTT piccolo aumentano rapidamente la loro finestra di congestione occupando la maggior parte della larghezza di banda disponibile.

### 2.1.4. Controllo di congestione

L'idea del controllo di congestione in TCP affida ad ogni sorgente la responsabilità di determinare quanta capacità sia disponibile nella rete, in modo da sapere quanti pacchetti in transito sia possibile avere senza provocare problemi. Quando una sorgente sa di avere in transito un tale numero di pacchetti, usa l'arrivo di un ACK come segnalazione del fatto che uno dei propri pacchetti è uscito dalla rete, per cui ne può inserire in rete un altro senza timore di aumentare il livello di congestione. Determinare la capacità disponibile non è comunque un compito facile, basti pensare che la banda disponibile cambia nel tempo a causa di altre connessioni che vengono instaurate e terminate. Ciò significa che una sorgente deve essere in grado di modificare il numero dei propri pacchetti in transito nella rete e per fare ciò si avvale di alcuni algoritmi:

- AIMD: il protocollo TCP gestisce per ogni connessione una nuova variabile di stato che viene usata dalla sorgente per limitare la quantità di propri dati in transito nella rete in un certo istante. Il protocollo TCP viene modificato in modo che il numero massimo consentito di byte di dati non confermati sia il minimo fra la finestra di congestione e la finestra annunciata. In questo modo una sorgente TCP non può spedire più velocemente di quanto sia accettabile per il componente più lento.
- Slow start: viene usato per aumentare rapidamente la dimensione della finestra di congestione all'inizio della connessione: lo slow start, a differenza di AIMD, aumenta la finestra di congestione esponenzialmente, anziché linearmente. In pratica, TCP raddoppia il numero di pacchetti in transito ad ogni intervallo di tempo uguale al RTT. Esistono fondamentalmente due situazioni in cui viene usata la partenza lenta: all'inizio di una connessione, nel momento in cui la sorgente non ha alcuna

idea in merito a quanti pacchetti si possa permettere di mantenere in transito nella rete, e quando la connessione si interrompe mentre la sorgente è in attesa della scadenza di una temporizzazione.

- Fast retransmit: poiché TCP portava a lunghi periodi di stagnazione durante i quali la connessione era ferma in attesa che scadesse il timeout, fu inserito questo algoritmo la cui idea è quella di far rispondere il ricevente con un ACK, anche se con un numero di sequenza già confermato, ogni volta che un pacchetto di dati arriva a destinazione. In questo modo, quando arriva un pacchetto fuori sequenza il protocollo rispedisce la stessa conferma spedita la volta precedente, generando un ACK duplicato. Quando il mittente vede un ACK duplicato, sa che il ricevente ha ricevuto un pacchetto fuori sequenza; il pacchetto potrebbe essere stato perduto o ritardato dalla rete, per questo motivo il mittente, prima di ritrasmettere il pacchetto mancante, aspetta di ricevere un certo numero di ACK duplicati (solitamente ne attende 3). A volte questo approccio può causare la ritrasmissione di un pacchetto perduto prima che scada il timeout.

## 2.2. Il protocollo UDP

L'User Datagram Protocol (UDP) è il più semplice protocollo di trasporto e prevede che i processi si identifichino l'un l'altro indirettamente usando un localizzatore astratto: la porta. Un processo sorgente invia un messaggio ad una porta e un processo destinatario riceve un messaggio da una porta. In pratica, un processo con funzioni di mittente inizia uno scambio di messaggi con un processo destinatario. Una volta che il mittente ha contattato il destinatario, quest'ultimo conosce la porta del mittente. Una porta viene implementata mediante una coda di messaggi e quando arriva un messaggio, il protocollo accoda il messaggio stesso al termine della coda e se la coda è piena il messaggio viene ignorato. Quando un processo applicativo vuole ricevere un messaggio, ne viene prelevato uno dal fronte della coda. Anche se UDP non implementa il controllo di flusso, né la consegna affidabile o in sequenza corretta, assicura la correttezza del messaggio per mezzo di una somma di controllo.

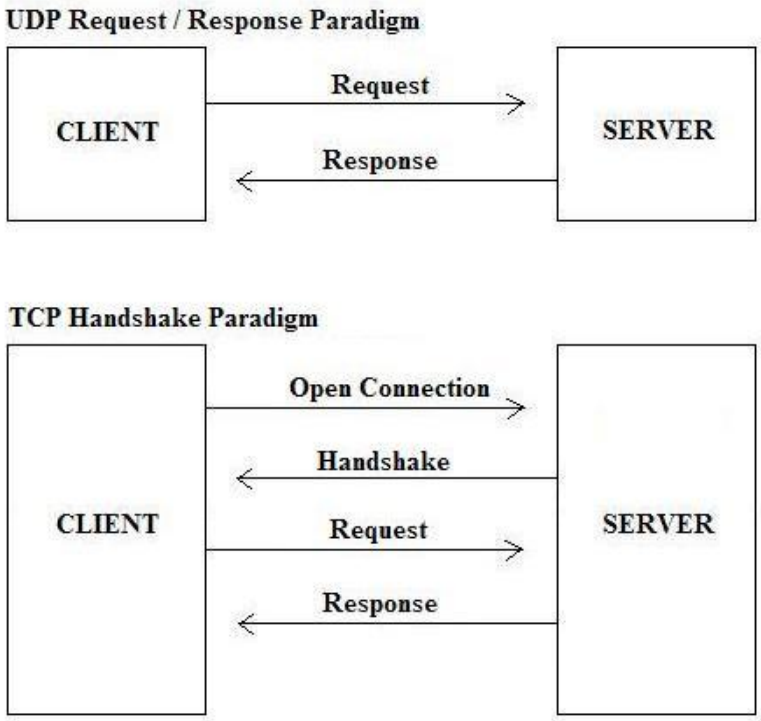


**Figura 3:** *UDP e TCP a confronto*

## 2.3. Evoluzione del protocollo TCP

Prima del 1986, non venivano adottati algoritmi di controllo di congestione. Dal 1986 in poi, invece, TCP ha iniziato ad adottare alcuni algoritmi atti a controllare la congestione della rete. In questa data, infatti, venne introdotto TCP Berkeley che faceva uso di slow start e congestion avoidance. Due anni più tardi venne introdotto TCP Tahoe che aggiunge, agli algoritmi di slow start e congestion avoidance, il fast retransmit. Nel 1990, venne introdotto il TCP Reno che inseriva, oltre agli algoritmi già presenti, il fast recovery. Qualche anno più tardi, TCP Reno venne ulteriormente migliorato sviluppando così il protocollo TCP New Reno (1995). Nel 1993 venne introdotto TCP Vegas che si basa su criteri di controllo di congestione diversi da quelli dei precedenti protocolli sviluppati e che cerca di prevenire la congestione e di rallentare la velocità d'invio dei dati prima di un'eventuale perdita. Vennero poi introdotte altre versioni di TCP, quali TCP Westwood e Veno, che permettono al mittente di limitare il ritmo con cui inserire i pacchetti dati nella sua coda di trasmissione in funzione della congestione di rete percepita, o SACK che aggiunge un meccanismo di Selective Acknowledgment a TCP. Di seguito vedremo il funzionamento e lo sviluppo nel

tempo di queste versioni di TCP, con particolare interesse per TCP Westwood e TCP SACK. Approfondimenti su queste versioni TCP si possono trovare in [2], [3], [4], [5], [6].

### 2.3.1. TCP Berkeley

TCP Berkeley è stato il primo a cercare di evitare la congestione di rete con un eccessivo traffico. Tuttavia, non prevedeva di cercare di capire lo stato della rete per poi regolarsi di conseguenza, ma modificava il ritmo di invio dei dati quando questo raggiungeva una certa soglia. In pratica, quando la connessione viene avviata, il valore che regola il ritmo di spedizione è inizializzato a una quantità molto piccola. Dato che, molto probabilmente, la banda disponibile è molto maggiore, nella prima parte della connessione, tale valore viene aumentato esponenzialmente ogni RTT e ogni volta che il mittente riceve un ACK, il valore viene incrementato. Quindi, il mittente invia il primo pacchetto, riceve il rispettivo ACK e incrementa di uno il valore della finestra. Vengono ora inviati, quindi, due pacchetti che, una volta riscontrati entrambi, fanno salire la finestra di congestione a 4. In questo modo la quantità di dati inviata raddoppia ogni RTT (questa è la fase di slow start). Quando la finestra raggiunge una soglia massima, la connessione entra nella fase di Congestion Avoidance: riduce il ritmo di trasmissione incrementando la finestra di congestione solo di 1 a ogni RTT, ottenendo così una crescita lineare e non più esponenziale della quantità dei dati spediti. Inizialmente, la soglia è impostata ad un valore molto alto, quindi nella prima parte della connessione si rimane nella fase di slow start a lungo. Quando si verifica il primo evento di perdita, cioè quando scade un timeout, il valore della soglia viene impostato alla metà del valore della finestra di congestione raggiunto in quel momento e la finestra di congestione viene riportata a 1. In questo modo la connessione riparte dalla fase di slowstart e vi rimane finché la finestra di congestione non raggiunge la nuova soglia, dove si ripete la situazione vista precedentemente.

### 2.3.2. TCP Tahoe

Tahoe introduce l'algoritmo di Fast Retransmit. In pratica, aggiunge un nuovo tipo di evento di perdita: la ricezione da parte del mittente di un certo numero di ACK identici tra loro. Ciò è stato fatto poiché il tempo di timeout risultava essere troppo lungo; l'algoritmo di Fast Retransmit si propone di capire in tempo minore se un pacchetto è andato perso o se è stato ricevuto con degli errori. Analizzando il TCP è stato intuito che un nuovo modo per

far capire al mittente che c'è stata una perdita è la ricezione di ACK duplicati. Supponiamo che al destinatario arrivi un pacchetto corretto e nel giusto ordine, egli invierà al mittente un ACK. Se il segmento successivo viene ricevuto con degli errori, il destinatario invierà un ACK che è identico a quello mandato precedentemente. Stessa cosa succederà se un pacchetto viene perso del tutto. Se il mittente, dunque, riceve diversi ACK duplicati, capisce che qualcosa non è andato a buon fine e può prendere provvedimenti. Per quanto riguarda la dimensione della finestra di congestione e la soglia, Tahoe si comporta come Berkeley e aggiorna le due variabili o ogni volta che scade un timeout o ogni volta che il mittente riceve 3 ACK duplicati.

### 2.3.3. TCP Reno

TCP Reno introduce l'algoritmo di Fast Recovery che comporta una differente reazione del mittente a seconda che si verifichi la ricezione di 3 ACK duplicati o lo scadere del timeout. Questa distinzione è stata introdotta poiché è facile pensare che la ricezione di 3 ACK duplicati non sia necessariamente sintomo di congestione della rete, poiché i pacchetti in tutti i modi arrivano al destinatario ma possono essere errati o fuori ordine, mentre lo scadere del timeout indica che il destinatario non ha ricevuto pacchetti per un lungo periodo di tempo. Quindi, alla ricezione di 3 ACK duplicati, TCP Reno porta il valore della soglia alla metà del valore della finestra di congestione raggiunto in quel momento, ritrasmette il pacchetto e imposta la finestra di congestione alla metà del valore che aveva fino a quel momento + 3. Da ora in poi la finestra di congestione viene aumentata di 1 per ogni ACK duplicato ricevuto fino a che non arriva l'ACK non duplicato che riscontra tutti i dati spediti successivamente al pacchetto perso. A questo punto il valore della finestra di congestione viene nuovamente abbassato a quello della soglia e il ritmo viene rallentato portandolo al tipico incremento lineare di Congestion Avoidance. Il sistema di ritrasmissione di TCP Reno risulta particolarmente valido nei casi in cui viene perso un singolo pacchetto ma può essere particolarmente inefficiente nel caso in cui si perdano o arrivino corrotti due o più segmenti della stessa finestra di trasmissione.

### 2.3.4. TCP New Reno

TCP New Reno introduce alcune modifiche a TCP Reno al fine di migliorarne le prestazioni. Tali modifiche riguardano l'impostazione del valore di soglia e il

miglioramento della reazione del protocollo nel caso in cui vengano persi due o più segmenti all'interno della stessa finestra. Per quanto riguarda il valore di soglia, questo viene calcolato quando viene instaurata una nuova connessione utilizzando l'equivalente in byte del prodotto ritardo-ampiezza della banda dove, per valutare l'ampiezza di banda disponibile, viene utilizzato un algoritmo di Packet Pair che si basa sugli intertempi di arrivo al mittente degli ACK. Per risolvere il problema dei due segmenti persi o corrotti nella stessa finestra riscontrato da TCP Reno, TCP New Reno introduce il concetto di "ACK parziale", cioè un ACK che riscontra solo alcuni pacchetti inviati prima di entrare in fase di Fast Recovery. Il TCP New Reno si comporta esattamente come il TCP Reno quando riceve esattamente 3 ACK duplicati e successivamente si comporta come il TCP Reno se il primo ACK non duplicato che riceve riscontra tutti i pacchetti inviati prima di entrare in Fast Recovery. Altrimenti, se riceve un ACK parziale che riscontra solo alcuni pacchetti inviati prima di entrare in Fast Recovery, il TCP New Reno interpreta questo fatto come un'ulteriore perdita e ritrasmette immediatamente il pacchetto successivo a quello riscontrato dall'ACK parziale, rimanendo nella fase di Fast Recovery. Il TCP New Reno entrerà in Congestion Avoidance solo quando riceverà l'ACK che riscontrerà tutti i pacchetti inviati prima di entrare in Fast Recovery.

### 2.3.5. TCP Vegas

La versione Vegas di TCP si propone di rallentare il ritmo di invio dei dati prima che si verifichi un evento di perdita. In pratica, cerca di evitare che si crei una congestione della rete piuttosto che prendere provvedimenti quando la congestione si è già verificata. È dunque necessario che il mittente conosca in ogni istante lo stato della rete e, per fare questo, osserva i cambiamenti degli RTT dei segmenti già inviati. Se il mittente si accorge che l'RTT diventa ampio, deduce che la rete si stia per congestionare e diminuisce il ritmo d'invio. Viceversa, se l'RTT risulta breve vuol dire che la rete non ha problemi e la finestra viene aumentata. Vegas non ha comunque riscosso molto successo, soprattutto a causa del suo comportamento in presenza di altri terminali che utilizzano TCP Reno o New Reno poiché, in questo caso, Vegas viene soffocato dalle altre versioni più aggressive che finiscono con l'occupare tutta la banda disponibile, infatti Vegas si sacrifica per non congestionare la rete e riduce continuamente il suo ritmo di invio, mentre Reno e New Reno tardano a ridurre il ritmo di invio approfittando della maggior banda disponibile.

### 2.3.6. TCP Veno

Tutti i TCP visti fin'ora sono stati progettati per reti cablate, mentre TCP Veno è progettato per reti con un alto tasso di pacchetti persi, quali le reti wireless, e le modifiche che porta coinvolgono solo il mittente. L'idea di base è quella di tenere traccia delle condizioni della rete distinguendo tra stato di congestione o meno e, in base a questa distinzione, vengono effettuate differenti operazioni al manifestarsi di un evento di perdita. Per capire se la rete è congestionata, vengono fissati due tassi: quello atteso e quello attuale. Quando la rete è congestionata i nodi creano delle code dove mettono in attesa i pacchetti che arrivano; si può, attraverso vari calcoli, ricavare una stima del numero di pacchetti presenti in una coda. L'idea base del TCP Veno è che se tale numero è minore di una certa variabile nel momento in cui si verifica un evento di perdita, allora il mittente considererà la perdita come un errore dovuto alla rumorosità del canale e non ridurrà il ritmo di invio in maniera molto elevata. Viceversa, se la stima del numero di pacchetti presenti in una coda è maggiore di una certa variabile, allora si assumerà che la rete si trova in uno stato di congestione e la dimensione della finestra verrà adattata utilizzando lo stesso schema del TCP Reno. In pratica, si può dire che TCP Veno nasca dall'unione di due protocolli: TCP Reno e TCP Vegas.

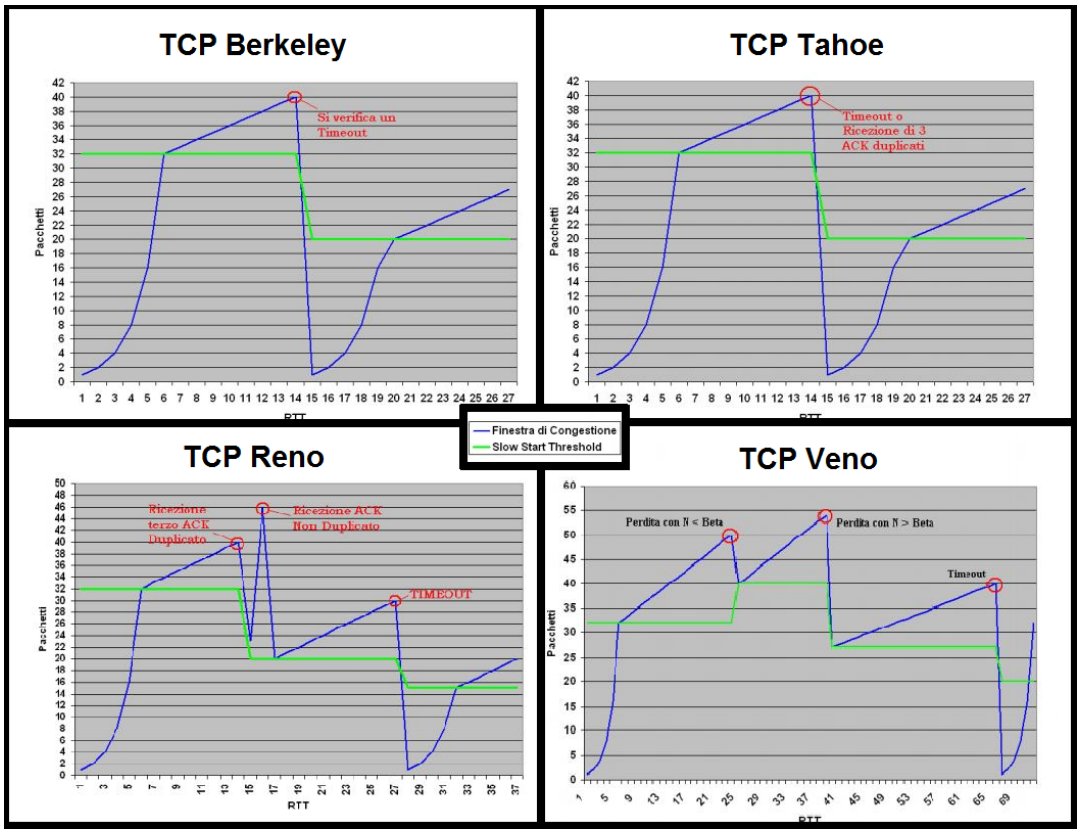


**Figura 4:** *TCP Berkeley, TCP Tahoe, TCP Reno e TCP Veno a confronto*

Abbiamo descritto fin'ora l'evoluzione avuta dal protocollo TCP nel corso degli anni. Esistono, in realtà, numerose versioni del TCP, più o meno recenti, nate per migliorarne le prestazioni sotto vari aspetti, tra le quali sono di particolare interesse il TCP SACK e il TCP Westwood.

### 2.3.7. TCP SACK

Abbiamo visto che TCP usa un sistema di riconoscimento cumulativo che costringe il mittente ad attendere il tempo di RTT per scoprire se l'invio del pacchetto non è andato a buon fine e quindi reinviare il pacchetto. Questa attesa riduce i throughput complessivo. La conferma selettiva è una strategia che corregge questo comportamento. Con i riconoscimenti selettivi, il destinatario può informare il mittente su tutti i segmenti che sono arrivati con successo, in modo tale che quest'ultimo ritrasmetta solo i segmenti effettivamente persi. In pratica nel TCP SACK, ogni ACK riscontra un solo pacchetto e non tutti quelli ricevuti precedentemente, permettendo così al mittente di avere un'idea più chiara di quali pacchetti sono stati ricevuti e quali sono ancora all'interno della rete o sono andati perduti. SACK utilizza la parte opzionale del pacchetto TCP e viene utilizzato solo se entrambe le parti lo supportano. L'uso di SACK è comunque diffuso poiché tutti i più popolari stack TCP lo supportano. Il comportamento di TCP SACK è identico a quello di TCP Reno quando al massimo un pacchetto viene scartato da una finestra di congestione. Mentre quando più pacchetti vengono eliminati da una finestra di congestione, il comportamento di TCP SACK e simile a quello di TCP Reno quando un solo pacchetto è stato scartato dalla finestra. In pratica, TCP SACK non penalizza inutilmente la connessione TCP quando più pacchetti vengono eliminati da una singola finestra. In tutti gli altri aspetti, SACK si comporta esattamente come TCP Reno. In pratica, dopo un timeout di ritrasmissione, TCP SACK utilizza lo stesso procedimento di slow start di Reno o Tahoe e quindi ha esattamente la stessa probabilità di ritrasmettere inutilmente pacchetti. Possiamo dire che, con SACK, il destinatario può informare il mittente circa tutti i segmenti che sono arrivati in modo corretto, in modo tale che quest'ultimo trasmetta solo i segmenti che sono stati effettivamente persi. SACK usa due nuove opzioni TCP:

1. SACK-permitted option: è un'opzione di abilitazione, grande 2 byte, che può essere inviata in un segmento di comunicazione preliminare rispetto alla connessione per

indicare le opzioni SACK che verranno usate una volta che la connessione sarà stabilita.

2. SACK option: è l'opzione SACK stessa che può essere inviata in una connessione una volta che è stata abilitata dalla SACK-permitted option. La SACK option viene spedita dal destinatario per informare il mittente che verranno scambiati blocchi non contigui di dati che devono essere ricevuti e accodati. Il destinatario aspetta la ricezione di dati per riempire i vuoti nello spazio della sequenza tra i vari blocchi ricevuti.

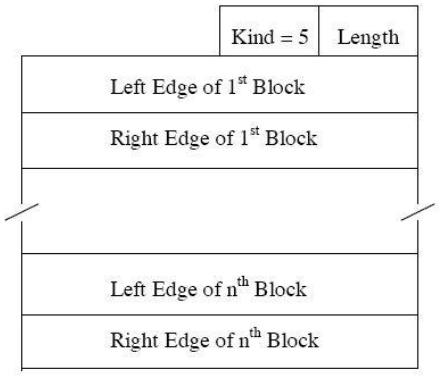


**Figura 5:** *SACK option*

Kind e Length sono le permitted option. Le opzioni di SACK saranno incluse in tutti gli ACK che non riconoscono il numero di sequenza più alto nella coda del destinatario. In questa situazione, la rete ha dati persi o disordinati quindi il destinatario ha blocchi non contigui nella sua coda di dati. Ogni blocco non contiguo di dati nella coda è definito nelle opzioni SACK da due interi senza segno a 32 bit. "Left Edge of a Block" è il primo numero di sequenza di questo blocco mentre "Right Edge of a Block" è il numero di sequenza immediatamente successivo all'ultimo numero di sequenza di questo blocco.

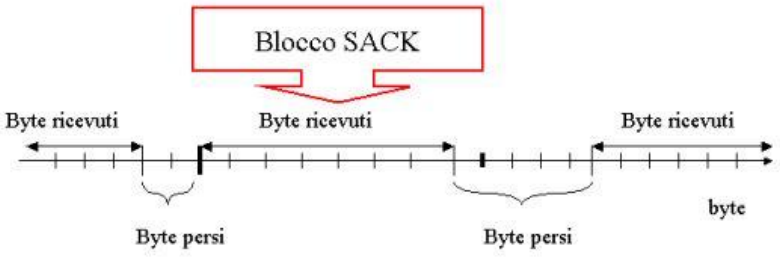


**Figura 6:** *Blocco SACK*

L’uso delle opzioni SACK non affronta il problema del riconoscimento di ACK duplicati quindi, per quanto riguarda il riconoscimento di un segmento duplicato, si usa il termine D-SACK (duplicate SACK). Il blocco D-SACK fornisce informazioni sul segmento duplicato che ha attivato l’ACK. Se presente, sarà il primo blocco delle opzioni SACK. Un blocco D-SACK è usato solamente per segnalare la ricezione di una sequenza contigua duplicata di dati da parte del destinatario nel pacchetto più recente. Ogni sequenza contigua duplicata di dati è riportata in un solo blocco D-SACK, infatti il destinatario invia due blocchi D-SACK identici in pacchetti consecutivi solo se riceve due segmenti duplicati. L’uso di D-SACK non richiede una negoziazione separata tra mittente e destinatario che abbiano già negoziato SACK. Questo significa che il destinatario può spedire blocchi D-SACK anche quando il mittente non è in grado di capire questa estensione di SACK. In questo caso, il mittente semplicemente scarterà ogni blocco D-SACK e processerà tutti gli altri blocchi SACK come di consueto. L’uso dell’opzione D-SACK permette di prevenire alcune situazioni di errore:

- False ritrasmissioni dovute al riordino dei pacchetti: se i pacchetti vengono riordinati in rete in modo che un segmento arrivi oltre i terzo pacchetto fuori ordine, l’algoritmo di Fast Retransmit ritrasmetterà il pacchetto fuori ordine. L’uso di D-SACK permetterà al mittente di capire quando un Fast Retransmit è dovuto al riordino di pacchetti invece che ad una perdita. Inoltre, se il mittente determina una riduzione non necessaria della finestra di congestione, esso ha l’opzione di non portare la soglia al valore della vecchia finestra e ripartire con uno slow start finché la finestra di congestione non raggiunge questo punto.
- Timeout di ritrasmissione dovuto a una perdita di ACK: se viene persa un’intera finestra di ACK, entra in gioco il timeout. Questa condizione può essere identificata in quanto il primo ACK ricevuto che segue il timeout trasporta un blocco D-SACK che indica il dato ricevuto duplicato. Senza l’uso di D-SACK il mittente sarebbe incapace di decidere se in realtà i suoi dati non sono stati persi.
- Timeout di ritrasmissione anticipato: in questo caso, il primo pacchetto dopo il timeout viene ritrasmesso, ma la finestra originale dei pacchetti arriva al destinatario e ciò porta ad ACK per questi segmenti. In seguito arrivano i pacchetti ritrasmessi che portano negli ACK i blocchi SACK che identificano i segmenti duplicati. Se non venisse usato D-SACK e uno degli ACK duplicati venisse incorporato in un

pacchetto, il mittente non sarebbe in grado di sapere quanti pacchetti duplicati sono stati ricevuti.

### 2.3.8. TCP Westwood

Il TCP Westwood è una modifica sender-side-only del protocollo TCP Reno che ha lo scopo di gestire meglio i ritardi di banda prodotti dai percorsi con potenziale perdita di pacchetti causata dalla trasmissione o da altri errori e con carico dinamico. Westwood migliora le prestazioni di Reno sia per le reti cablate che per le reti wireless. Una caratteristica importante di TCP Westwood rispetto alle precedenti "estensioni" di TCP per reti wireless è che non richiede l'ispezione e/o l'intercettazione dei pacchetti TCP ai nodi intermedi. L'idea innovativa su cui si basa Westwood è quella di far misurare continuamente al mittente il tasso di ritorno di un ACK e di utilizzare poi questa stima per calcolare la finestra di congestione e la soglia di avvio lento dopo un episodio di congestione (triplice ACK duplicato o scadenza del timeout). Invece che procedere come TCP Reno, che "alla cieca" dimezza la finestra di congestione dopo 3 ACK duplicati, TCP Westwood cerca di selezionare una soglia di slow start e una finestra di congestione che sono coerenti con la banda effettivamente utilizzata al momento della congestione. Questo meccanismo è particolarmente efficace su collegamenti wireless dove le perdite dovute a problemi del canale radio vengono interpretate, in maniera errata, come un sintomo di congestione TCP e quindi portano a un riduzione inutile della finestra di congestione. In pratica, possiamo dire che Westwood è simile al TCP Veno, con la differenza che il fattore di riduzione della finestra di congestione dopo una perdita viene calcolato dinamicamente di volta in volta in base alla percezione che il mittente ha sullo stato della rete. Poiché però l'algoritmo di stima della banda di Westwood non funzionava bene in presenza di traffico di ritorno per la compressione degli ACK, è stato introdotta un'evoluzione di questo protocollo: TCP Westwood+.

### 2.3.9. TCP Westwood+

TCP Westwood+ è un'evoluzione di TCP Westwood in cui è stato modificato l'algoritmo di rilevamento della banda al fine di ridurre l'aggressività di Westwood in presenza di compressione degli ACK. La compressione degli ACK è un fenomeno di Internet in cui gli ACK arrivano a una distanza molto più vicina rispetto a quella che avevano quando sono

stati generati a causa di un ritardo di accodamento della rete. Poiché Westwood attua una stima della larghezza di banda basata sugli intertempi di arrivo degli ACK, la compressione degli ACK porta Westwood a sovrastimare la banda con conseguenti problemi di equità in presenza di molteplici flussi TCP. Per alleviare l'effetto della compressione degli ACK, Westwood+ modifica il meccanismo di stima della banda effettuando il calcolo a ogni RTT e non a ogni ricezione di un ACK. Il risultato è quello di una misura della banda più accurata che assicura prestazioni migliori quando si confronta Westwood+ con Reno e New Reno.

## 2.4. Valutazione delle performance e confronto di alcuni algoritmi TCP

Ci occuperemo, in questa sezione, di confrontare tra loro alcuni dei TCP visti precedentemente, al fine di valutarne le prestazioni per capire, attraverso alcuni scenari, quali siano le migliori condizioni d'uso. Ci concentreremo, inizialmente, sul confronto tra TCP Westwood e TCP Westwood+ per poi confrontare questi recenti protocolli con TCP meno recenti, quali TCP New Reno, TCP Reno, TCP Tahoe, TCP Vegas e TCP SACK.

### 2.4.1. Westwood e Westwood+

Andremo ora a confrontare le prestazioni, in termini di bandwidth con compressione degli ACK e bandwidth senza la compressione degli ACK, di Westwood+ e Westwood. Facciamo questo confronto al fine di comprendere in che modo il nuovo algoritmo di stima della banda, utilizzato da Westwood+, sia efficace rispetto a quello vecchio, utilizzato da Westwood. I dati e lo scenario proposti fanno riferimento a [7].

Lo scenario proposto è quello di un ambiente emulato WAN e da Internet in tempo reale. Sono stati valutati il goodput e la correttezza dell'algoritmo di stima della larghezza di banda. La topologia della rete è caratterizzata da due segmenti Ethernet collegati da un router che emula ritardi attraverso l'uso di Dummynet e da due PC collegati con il primo segmento Ethernet e un PC collegato col secondo segmento Ethernet.

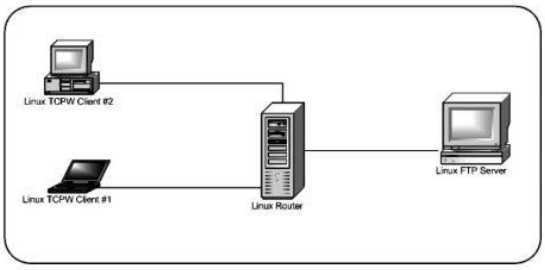


**Figura 7:** *Scenario della simulazione*

Al fine di testare l'algoritmo di stima della larghezza di banda, Dummynet è stato configurato in modo da fornire un collegamento a 500 Kbps tra i due segmenti Ethernet con 100 ms di RTT. Un trasferimento FTP dal PC 1 al server FTP dimostra che sia Westwood che Westwood+ forniscono una buona stima del collegamento in assenza di compressione degli ACK, come mostra la figura 8.

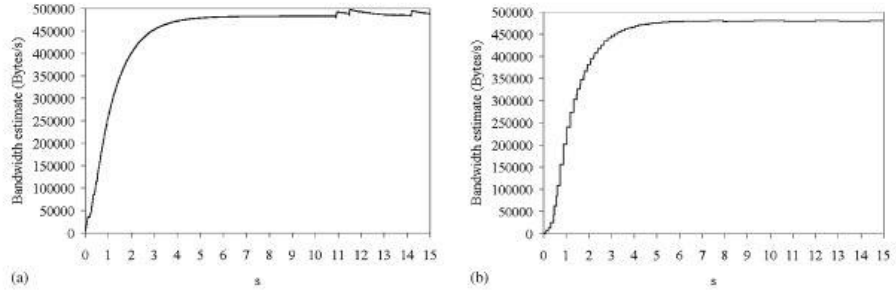


**Figura 8:** *Stima della banda in assenza di compressione ACK. (a) Westwood, (b) Westwood+*

Per valutare Westwood e Westwood+ in presenza di compressione ACK, un altro trasferimento FTP è stato avviato lungo il percorso inverso, andando cioè dal server FTP al PC 2. L'algoritmo di stima della banda utilizzato da Westwood sovrastima notevolmente la larghezza di banda disponibile, fino al 300%, mentre Westwood+ risulta migliore. Ciò può essere ben visto dalla figura 9.

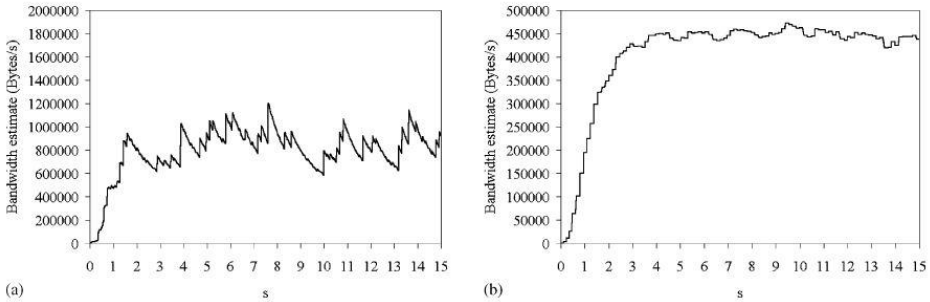


**Figura 9:** *Stima della banda in presenza di compressione ACK. (a) Westwood, (b) Westwood+*

Possiamo perciò affermare che il nuovo algoritmo di stima della banda utilizzato da Westwood+ fornisce una buona stima della banda disponibile, mentre l'algoritmo utilizzato da Westwood sovrastima la banda disponibile in presenza di compressione degli ACK. Si può anche affermare che Westwood+ migliora significativamente l'equità nella ripartizione della larghezza di banda. In pratica, Westwood+ è migliore per gli scenari congestionati in cui la compressione degli ACK è diffusa e dove l'utilizzo di Westwood potrebbe congestionare ulteriormente la rete.

### 2.4.2. Westwood, Westwood+, New Reno, Reno e Tahoe

Lo scenario di questa simulazione è lo stesso descritto dalla topologia di rete del paragrafo precedente. Abbiamo quindi un flusso di dati che va dal PC 1 al server, e un flusso di dati che va dal server al PC 2, una larghezza di banda di 1 Mbps e una larghezza del collo di bottiglia di 500 Kbps. L'unica differenza è che il secondo flusso è fatto partire 2 secondi dopo il primo flusso. I vari parametri di simulazione sono descritti nella figura 10. L'implementazione relativa a questi dati è descritta in [8].

| Parameter | Values |
|---|---|
| Access link bandwidth | 10 Mb/s |
| Bottleneck link bandwidth | 2 Mb/s |
| Access link propagation delay | 45 ms |
| Bottleneck link propagation delay | 0.01 ms |
| Packet MTU size | 400 B |
| Delayed ACK count | 2 segments |
| Delayed ACK timeout | 200 ms |
| Error model | Uniform error model |
| Error rate | 0.005 |
| Application type | Bulk send application |
| Simulation time | 600 s |

**Figura 10:** *Parametri di simulazione della topologia*

Consideriamo il Packet Error Rate (PER) e lo facciamo variare da 0,0001 a 0,05. Il throughput medio è tracciato per vari livelli del PER e per i vari TCP presi in considerazione ed è mostrato nella figura 11.

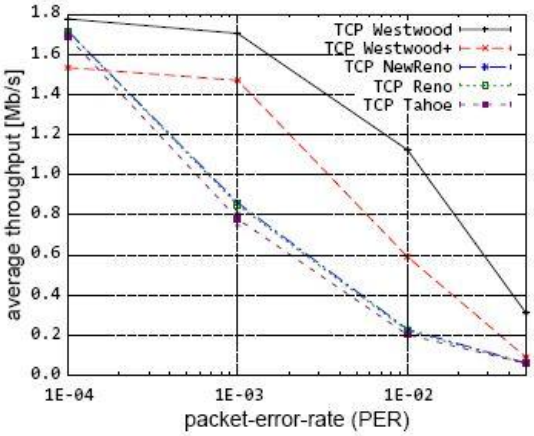


**Figura 11:** *Throughput vs PER*

Dal grafico possiamo notare un enorme miglioramento delle prestazioni di TCP Westwood sul resto dei TCP. Questo miglioramento è attribuito al metodo adottato da Westwood che imposta la finestra di congestione e la soglia di slow start in funzione della larghezza di banda stimata, invece che a un valore statico, quando incorre una perdita. Il secondo miglior protocollo performante è Westwood+ che campiona la larghezza di banda a ogni RTT, a differenza di Westwood che la campiona a ogni ACK ricevuto. Dato che l'intervallo di campionamento è superiore, Westwood+ richiede più tempo per stabilizzare la corretta lunghezza della banda rispetto a Westwood e questo, in presenza di errori, risulta essere un comportamento peggiore poiché, se gli errori incorrono prima che Westwood+ stabilizzi il parametro di banda, viene utilizzato un parametro basso di stima della larghezza di banda per determinare la finestra di congestione e ciò causa ritardi di invio. Si può quindi dire che maggiore è il tasso di errore e maggiore è la probabilità di sottovalutare la banda. Ciò è dimostrato nella figura 11, dove si può notare che le prestazioni di Westwood+ diminuiscono notevolmente a partire dal tasso di errore 0,001. Possiamo quindi dire che, in un intervallo di campionamento più breve, Westwood ha un feedback più aggiornato rispetto a Westwood+. Un'altra osservazione interessante è data dalla somiglianza, in termini di prestazioni, del TCP Reno con il TCP New Reno. Questo perché TCP New Reno migliora le prestazioni del TCP Reno solo in caso di più errori che si verificano nella stessa finestra si trasmissione.

Andiamo ora a studiare gli effetti della variazione del ritardo di propagazione nella rete. Ipotizziamo un PER costante di 0,005 che si verifica sul collegamento del collo di bottiglia.

Variamo il ritardo di propagazione da 0,01 a 250 ms nella simulazione al fine di confrontare le velocità raggiunte dai vari TCP.

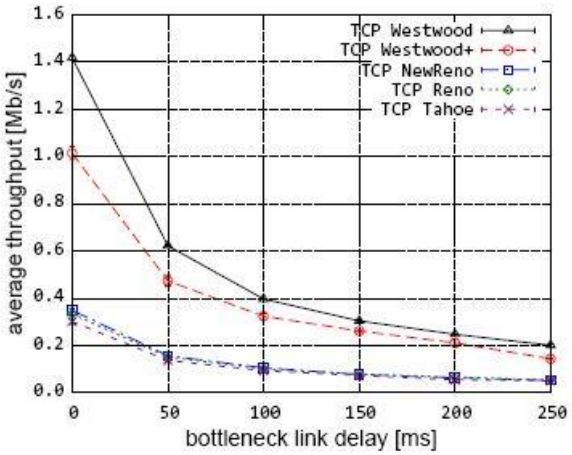


**Figura 12:** *Throughput vs bottleneck link delay*

Nella figura 12 possiamo notare che, variando il ritardo nella rete, le prestazioni di Westwood e Westwood+ sono ancora superiori rispetto a quelle degli altri protocolli. Ciò è di nuovo dovuto al fatto che, mentre gli altri TCP eseguono un dimezzamento statico della finestra di congestione, Westwood e Westwood+ utilizzano l'algoritmo dinamico di stima della banda, che è indipendente dal livello di corruzione del collegamento. Possiamo inoltre notare che Reno, New Reno e Tahoe si comportano in modo pressoché identico. Ciò perché, come detto, usano un algoritmo statico di modifica della finestra di congestione in caso di perdita.

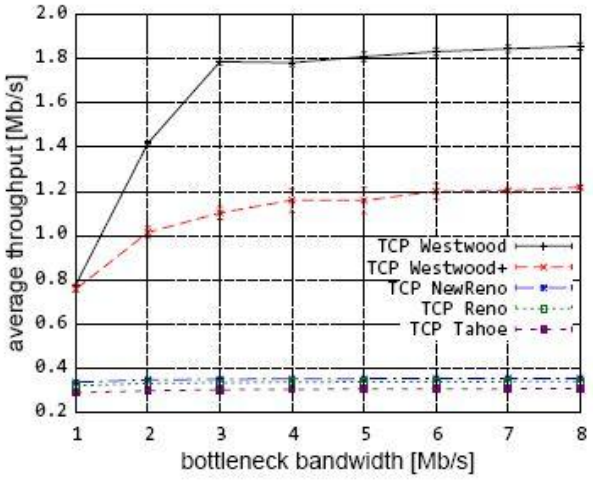


**Figura 13:** *Throughput vs bottleneck bandwidth*

Nella figura 13 viene mostrato il throughput conseguito per diversi valori di larghezza di banda nel collo di bottiglia. La larghezza di banda varia da 1 Mbps a 8 Mbps e un tasso di errore di pacchetto constante di 0,005 è impostato sul collegamento del collo di bottiglia. Possiamo notare che il protocollo TCP Westwood, così come il protocollo TCP Westwood+, fornisce una performance migliore rispetto a quella delle altre varianti TCP che, possiamo dire, si equivalgono. Le migliori prestazioni di Westwood e Westwood+ sono, ancora una volta, da attribuire all'impostazione della finestra di congestione e della soglia di slow start in funzione della larghezza di banda stimata dopo una perdita. Notiamo inoltre che, in questo caso, TCP Westwood supera di gran lunga le prestazioni di TCP Westwood+. Si nota, infine, che le prestazioni di TCP Tahoe, TCP Reno e TCP New Reno non sono influenzate dal cambiamento di larghezza di banda e ciò è dato dal fatto che il tasso di invio di tali protocolli è indipendente dalla larghezza di banda del collegamento.

Abbiamo quindi osservato il comportamento di TCP Westwood, TCP Westwood+, TCP New Reno, TCP Reno e TCP Tahoe in termini di congestione. Possiamo affermare che TCP Westwood potrebbe essere utilizzato nei casi in cui non c'è congestione e le perdite occorrono solo per causa della corruzione.

### 2.4.3. Westwood, Reno e SACK

Lo scenario che si sta per mostrare fa riferimento a uno studio descritto in [9]. La topologia di rete per questo confronto è data da rete mista, con una porzione di collegamento cablata di 10 Mbps tra un nodo mittente e una stazione base. Il tempo di propagazione attraverso il collegamento via cavo viene inizialmente considerato uguale a 45 ms. La parte wireless è data da un collegamento di 2 Mbps con un tempo di propagazione di 0,01 ms. Si assume che il collegamento wireless fornisca la connessione tra la stazione base e un terminale mobile di destinazione. Si presume che nel collegamento wireless gli errori si verifichino in entrambe le direzioni.

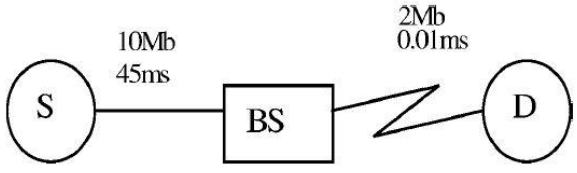


**Figura 14:** *Scenario della simulazione*

Si confronteranno la velocità di TCP Westwood con quella di TCP Reno e di TCP SACK assumendo che vi sia dallo 0% al 5% di probabilità di perdita di pacchetti.

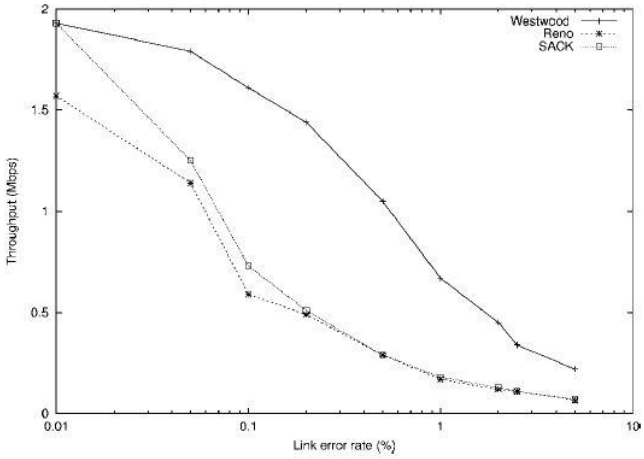


**Figura 15:** *Throughput vs Error rate del collegamento wireless*

La figura 15 mostra che TCP Westwood guadagna fino al 394% rispetto a TCP Reno o a TCP SACK. Questo guadagno avviene ad una realistica probabilità di errore di pacchetto pari all'1%.

Per valutare il guadagno del TCP Westwood in relazione al throughput e il suo rapporto con il tempo di propagazione end-to-end, sono state eseguite simulazioni con tempo di propagazione, per la parte cablata, variabile da 0 a 250 ms. I risultati della figura 16 mostrano un beneficio per TCP Westwood pari al 567% in un tempo di propagazione di 100 ms. Si nota, inoltre, che quando il tempo di propagazione è piccolo, tutti i protocolli sono ugualmente efficaci. TCP Westwood raggiunge il massimo miglioramento rispetto a TCP Reno e a TCP SACK quando il tempo di propagazione è pari a circa 100 ms.

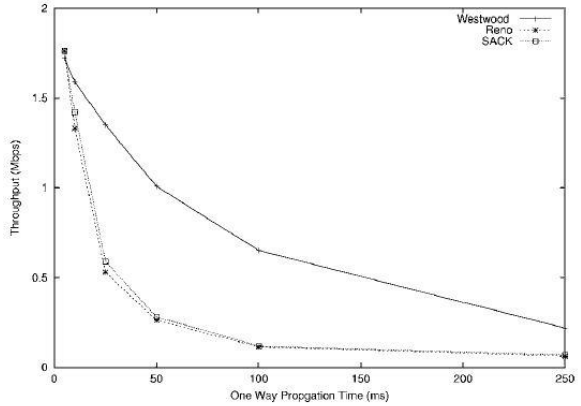


**Figura 16:** *Throughput vs ritardo di propagazione one-way*

Possiamo sostanzialmente affermare che uno dei principali vantaggi di TCP Westwood è che modifica solo il mittente al contrario di altre varianti, come ad esempio TCP SACK, che richiedono anche modifiche al destinatario. Notiamo, infine, che in entrambe le simulazioni (cablata e wireless) SACK e Reno hanno un comportamento molto simile.

### 2.4.4. Westwood+, New Reno e Vegas

Lo scenario che andremo a trattare è composto da M mittenti TCP con diversi RTT che condividono uno stesso collo di bottiglia di 10 Mbps. Per ottenere la compressione degli ACK, 10 mittenti di flussi TCP New Reno inviano traffico lungo il percorso ACK delle M connessioni. Le M connessioni vanno da 10 a 200. Lo scenario è stato valutato in [10].

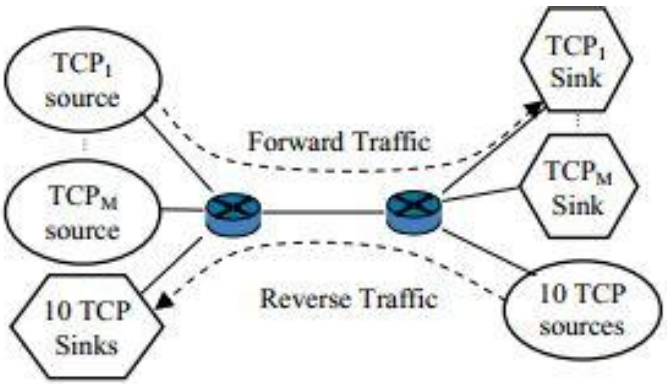


**Figura 17:** *Topologia dello scenario*

Il goodput totale è definito come la somma dei goodput di tutte le M connessioni. Dalla figura 18, si può notare che, quando M è maggiore di 40, il goodput totale si avvicina alla capacità massima del collo di bottiglia, mentre, quando M è minore di 40, Vegas fornisce un bassissimo goodput totale e ciò è dovuto al traffico di ritorno, che ha un impatto significativo su TCP Vegas.

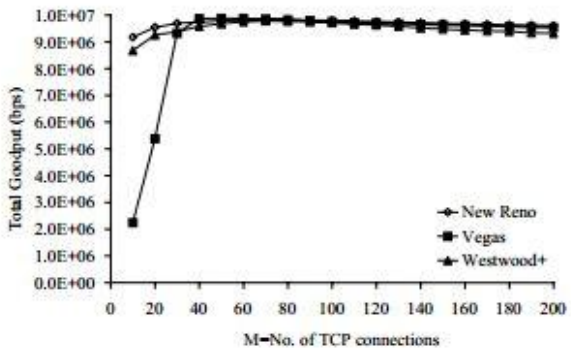


**Figura 18:** *Goodput totale delle M connessioni TCP*

Nella figura 19, invece viene mostrato il fairness e si può notare che Westwood+ migliora l'equità nella condivisione di banda, rispetto a TCP New Reno, quando M è minore di 60. Ciò è dato dal fatto che i tempi di RTT sono ripartiti uniformemente sull'intervallo in modo tale che, per M piccolo, i RTT sono più distanti. Si può notare, inoltre, che TCP Vegas presenta il miglior indice di equità, pur avendo un goodput basso.

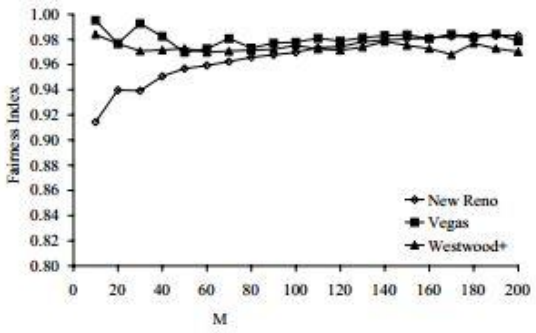


**Figura 19:** *Fairness in un collo di bottiglia di 10 Mbps*

Guardando i risultati ottenuti, possiamo quindi dire che sia Westwood+ che TCP New Reno raggiungono la piena utilizzazione del canale con Westwood+ che fornisce una maggiore equità rispetto a TCP New Reno e che TCP Vegas è equo ma non è in grado di sfruttare appieno la larghezza di banda quando coesiste con TCP Reno o in presenza di traffico di ritorno, ciò a causa del suo meccanismo di controllo di congestione basato sul RTT.

# 3. Interazione tra TCP e UDP

Negli ultimi anni sono state sviluppate nuove tecnologie che si avvalgono dell'utilizzo della rete, come ad esempio quelle per la trasmissione di voce, e il protocollo TCP non è adatto per queste applicazioni a causa della sua complessa ritrasmissione. Per questo motivo tali applicazioni attualmente trasmettono utilizzando il protocollo UDP, a scapito dell'affidabilità della consegna del pacchetto e rischiando la saturazione della rete, poiché UDP non dispone di un sistema di controllo della congestione della rete. Quando i flussi TCP e i flussi UDP condividono lo stesso collo di bottiglia in una rete, c'è una complessa interazione tra i protocolli di trasporto. Infatti, quando lo stesso collo di bottiglia è condiviso tra i flussi TCP e UDP, in presenza di un elevato carico di rete e quindi di congestione, il flusso TCP riduce il traffico utilizzando i metodi di controllo di congestione visti precedentemente, mentre il flusso UDP continua a trasmettere ignorando il fatto che la rete sia sovraccarica. In questo modo il flusso UDP catturerà la maggior parte della banda penalizzando il flusso TCP. In questo capitolo andremo ad analizzare l'impatto di questi fattori sul flusso effettivo (throughput) di TCP e UDP e a mostrare i risultati delle prestazioni di TCP in presenza di flussi UDP nell'ambito di diversi scenari. Analizzeremo scenari basati su reti wireless e, tranne nell'ultimo caso in cui parleremo esplicitamente di TCP Westwood, il TCP analizzato avrà un comportamento dove il mittente adatterà la velocità di invio dei pacchetti in base alla congestione della rete; comportamento molto simile a quello di TCP Veno.

## 3.1. Scenario 1: Rete wireless ad hoc

Lo scenario è composto da una topologia di rete a griglia costante con costante numero di nodi. Nello scenario viene mantenuto lo stesso protocollo di routing e viene variata solo la quantità di traffico inviato sulla rete. Si trasmette in Constant Bit Rate (CBR) per UDP e in File Transfer Protocol (FTP) per TCP. Le simulazioni sono state effettuate sulla base delle specifiche indicate nella figura 20 e il protocollo di routing utilizzato è On-Demand Distance Vector (AODV). I dati a cui si fa riferimento sono descritti in [11].

| Simulation Parameters | |
|---|---|
| Routing Protocol | AODV |
| MAC Protocol | IEEE 802.11 |
| Traffic Type | FTP for TCP<br>CBR for UDP |
| Simulation Time | 1800 seconds |
| Terrain Size | 3000m x 3000m |
| Number of Nodes | 49 |
| Node Placement | Grid |
| Grid Size | 7x7 |
| Distance between nodes | 250m |
| Mobility Model | None |
| Packet Size | 1024 Bytes |
| Data Transmission Rate | 100 packets per second |
| Number of Data Flows | 2 parallel flows, 2x2, 3x3, 4x4 and 7x7 (evenly spaced) |

**Figura 20:** *Parametri di rete per la simulazione dello scenario 1*

In questo scenario, aumentare i flussi porta all'aumento del traffico di rete e flussi sia orizzontali che verticali sulla griglia portano congestione di rete nei nodi comuni. Nel caso dello scenario con 7x7 flussi di dati, tutti nodi terminali sono sorgente e destinazione e tutti i nodi intermedi gestiscono il traffico da entrambi i collegamenti verticali ed orizzontali.

Verranno di seguito discussi i risultati relativi a casi dello scenario con 2 flussi di dati paralleli e 2x2, 3x3, 4x4 e 7x7 flussi di dati e andremo ad analizzare il modo in cui TCP e UDP reagiscono all'aumentare del traffico di dati e della congestione.

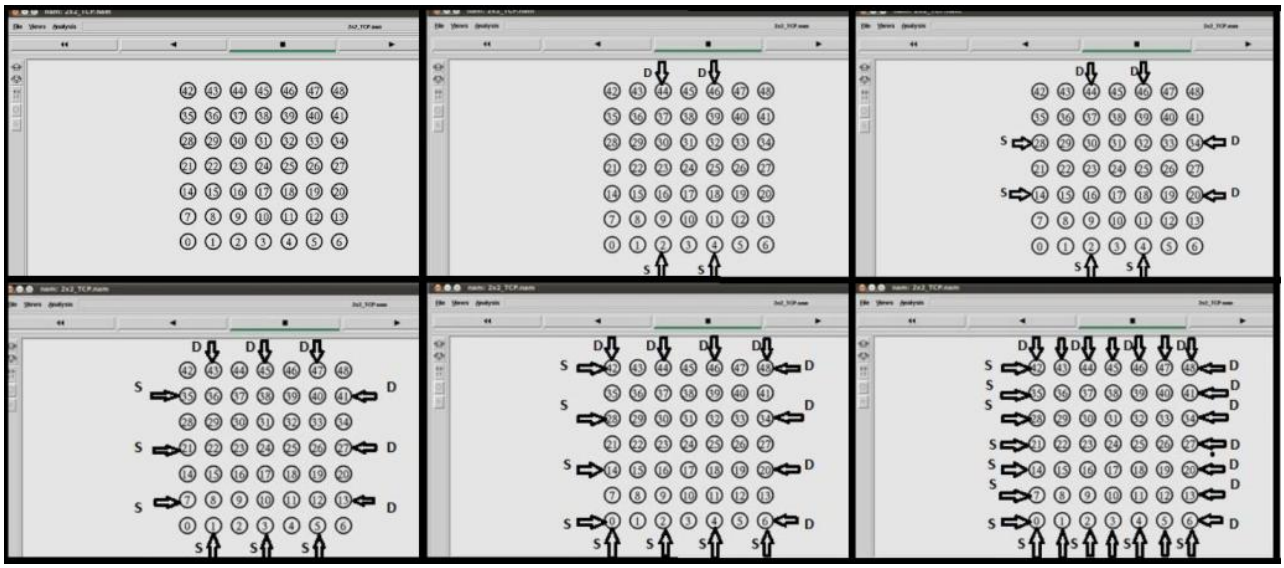

**Figura 21:** *Caso base e casi dello scenario discussi*

In tutti gli scenari, i dati TCP e quelli UDP vengono scambiati separatamente e la performance è misurata in termini di packet delivery, throughput e latenza.

### 3.1.1. Packet Delivery Ratio (PDR)

Il PDR è la misura dell'abilità che ha un protocollo di spedire pacchetti da una sorgente a una destinazione.

$$PDR = \frac{\sum_i^n CBR\_Recived}{\sum_i^m CBR\_Sent} \times 100$$

dove *n* è il numero dei pacchetti ricevuti e *m* è il numero di quelli inviati. Il valore di PDR varia da 0 a *n*.

### 3.1.2. Throughput

Il throughput è il numero di pacchetti trasferiti in una rete in un tempo di osservazione ed è espresso in bit al secondo.

$$Throughput = \frac{CBR\_Sent \times 8 \times 1024}{SimTime}$$

dove *CBR_Sent* è il numero di pacchetti inviati, 1024 è la dimensione del pacchetto in byte (moltiplicato per 8 per ottenere il numero di bit) e *SimTime* è la durata della simulazione.

### 3.1.3. Latenza

La latenza è il tempo impiegato da un pacchetto per raggiungere la destinazione.

$$Latenza = \sum_i^n (CBR\_ST - CBR\_RT)$$

dove *n* è il numero di pacchetti ricevuti, *CBR_ST* è CBR al tempo di invio e *CBR_RT* è il CBR al tempo di ricezione. Il valore della latenza varia da 0 a *n*.

### 3.1.4. Risultati ottenuti

Il valore PDR è alto per il TCP poiché, essendo un protocollo affidabile, garantisce la consegna dei pacchetti e il ricevitore conferma ogni pacchetto ricevuto. Nel caso di UDP, essendo un protocollo inaffidabile, il numero dei pacchetti persi è alto. I nodi destinatari ricevono più pacchetti da TCP rispetto che da UDP. Aumentando il numero di flussi di dati

si può notare che il PDR di TCP deteriora. Le prestazioni complessive di TCP rimangono comunque migliori rispetto a quelle di UDP pertanto, in termini di PDR, possiamo dire che il protocollo TCP è migliore del protocollo UDP. Andando a vedere la variazione del throughput, si può concludere che nel caso del protocollo UDP il rendimento è alto; si vede cioè che il throughput diminuisce con l'aumento dei flussi a causa dell'aumento del traffico di rete, mentre nel caso del protocollo TCP rimane costante poiché un nuovo pacchetto viene inviato solo dopo aver ricevuto conferma, da parte del destinatario, della ricezione del precedente pacchetto inviato. Per quanto riguarda la latenza, si nota che UDP richiede più tempo rispetto a TCP e ciò può essere a causa del numero di pacchetti reinviati. In conclusione, in base allo scenario considerato, si può notare che TCP si comporta meglio di UDP per quanto riguarda PDR e latenza mentre UDP fornisce un throughput migliore.

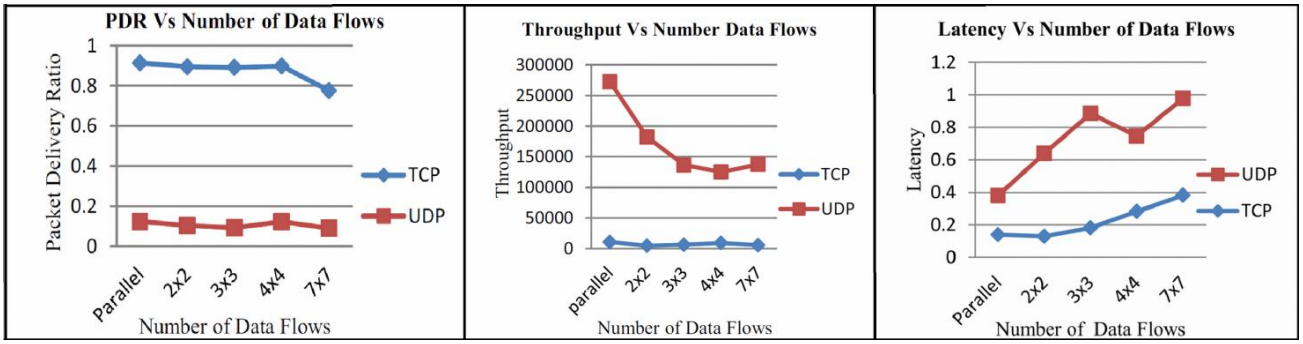


**Figura 22:** *PDR, throughput e latenza di TCP e UDP nello scenario a conforto*

## 3.2. Scenario 2: Rete wireless multi-hop ad hoc

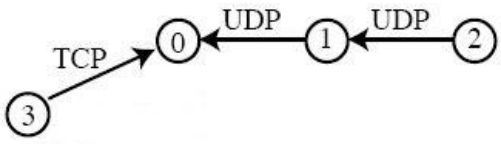


**Figura 23:** *Scenario 2*

I dati trattati sono stati studiati facendo riferimento a [12]. Lo scenario è caratterizzato da 4 nodi: un flusso UDP va dal nodo 2 al nodo 0 (passando quindi per il nodo 1), mentre il nodo 3 invia un flusso TCP al nodo 0. Inoltre, sono presenti i fattori di routing dinamico e mobilità. Sono state fatte sperimentazioni all'interno dello scenario e sono stati valutati gli andamenti dei flussi TCP e UDP in 3 differenti step. Inizialmente è stato testato lo scenario

staticamente e senza la presenza del nodo 3, e quindi del flusso TCP. È stato successivamente inserito il nodo TCP, per arrivare poi, nello step finale, allo scenario completo.

Il nodo 3 interferisce con UDP provocando una perdita di pacchetti. Il fatto che la rete sia mobile e che sia presente il routing dinamico provoca una perdita media di pacchetti pari al 21,6% sul flusso UDP. Sia TCP che UDP hanno un calo del rendimento. La misurazione del throughput mostra che la velocità di UDP è superiore alla velocità di TCP. Aumentando il numero di passaggi da un nodo all'altro, la velocità diminuisce in base al numero di passaggi; in caso di TCP si arriva a una diminuzione pari a circa 1/n, dove n è il numero di passaggi. Si noti anche che c'è una notevole differenza nelle velocità di passaggio tra nodi adiacenti (short hop) e nodi che sono situati nel raggio di trasmissioni di altri nodi. L'analisi di TCP mostra una variazione superiore rispetto al tempo di andata e ritorno tra nodi separati. TCP è più colpito rispetto a UDP a causa dei suoi sistemi di controllo di congestione che non sono adatti ad ambienti wireless dinamici.

| [Mbit/s] | short hop (10 cm) | 1 hop (ca. 20 m) | 2 hop | 3 hop |
|---|---|---|---|---|
| UDP | 5.64 | 4.85 | 3.60 | 2.01 |
| TCP | 3.71 | 1.68 | 0.78 | 0.62 |

**Figura 24:** *Massimo throughput nei vari casi dello scenario*

Quando i flussi TCP e UDP non condividono canali comuni si osserva un incremento dell'interspacing dei pacchetti nel flusso UDP, causato dalla perdita dei pacchetti e si ha un'instabilità del TCP in forma di brevi stalli della rete. Quando invece i flussi TCP e UDP condividono un canale comune, si hanno un aumento della perdita dei pacchetti UDP e delle interruzioni di TCP. Il fattore di routing dinamico aggiunge un'ulteriore instabilità poiché i messaggi di controllo vengono persi a causa del traffico dati; il flusso UDP risulta avere in media il 21,6% di pacchetti persi.

| Scenario | UDP Delivery Ratio | TCP Throughput |
|---|---|---|
| Static multi-hop UDP | 99.2 % | - |
| Multi-hop UDP with TCP flow | 97.4 % | 1.34 Mbit/s |
| Roaming node | 78.4 % | 0.93 Mbit/s |

**Figura 25:** *PDR e throughput in 3 casi possibili all'interno dello scenario*

TCP soffre in un ambiente wireless poiché mal interpreta la perdita dei pacchetti. Sia le prestazioni di TCP che quelle di UDP sono comunque ridotte in un ambiente wireless multi-hop a causa delle interferenze radio presenti tra i vari nodi. Un altro fattore che può compromettere le prestazioni è dato dai frequenti aggiornamenti di routing poiché, nelle reti ad hoc con nodi mobili, è probabile che il protocollo TCP perda i pacchetti quando si cambia percorso. Infine, UDP risulta iniquo nei confronti di TCP quando arrivano a condividere un canale comune, in quanto TCP cerca di trovare una velocità di invio che si adatti a UDP e questa regolazione può portare a ritardi.

## 3.3. Scenario 3: Rete wireless multi-hop ad hoc e livello MAC

Lo scenario è simile al primo ed è composto da una topologia di rete a griglia costante con 13x13 nodi. Si trasmette in Constant Bit Rate (CBR) per UDP e in File Transfer Protocol (FTP) per TCP. Le simulazioni sono state effettuate sulla base delle specifiche indicate nella figura 26 e il protocollo di routing utilizzato è On-Demand Distance Vector (AODV). Si fa riferimento allo studio [13].

| Protocol | Parameter/Mode | Range Observed | Optimal Settings |
|---|---|---|---|
| TCP (FTP) | Max. segment size | 200-1460 bytes | 1460 bytes |
| UDP(CBR) | Data generation rate | 50-800 KB/sec | 800 KB/sec |
| | Packet size | 200-2920 bytes | 1460 bytes |
| | Num of application | 1-2 | 2 |
| | Time of start | +/-20 sec w.r.t. TCP start time | 12 seconds after TCP |
| | Flow hop length | 2-10 | 8 |
| AODV | Local link maintenance mode | Link layer feedback or Hello Messages | Hello Messages |
| | Messages interval | 0- | 9 Seconds |
| | Route repair wait duration | 0- | 6 Seconds |
| 802.11 | Retry limits | (7,4) to (21,12) | 21,12 |

**Figura 26:** *Parametri di rete per la simulazione dello scenario 3*

### 3.3.1. Instradamento stabile

È stato dimostrato che in presenza di flussi UDP, la quantità di dati utili trasferiti nell'unità di tempo sul canale fisico di comunicazione tra due nodi (goodput) dei client TCP è ridotta

del 10% rispetto al goodput ottenuto in assenza di flussi UDP. Il throughput ottenuto dipende dai percorsi seguiti dai flussi UDP. Analizziamo due diversi casi:

1. Il percorso seguito dai flussi UDP isola il server TCP: una volta che i flussi UDP stabiliscono una connessione, il server TCP è isolato dai client e, anche se i pacchetti TCP vengono consegnati al server, gli ACK inviati dal server possono essere persi a causa della congestione. In questo caso tutti i client TCP soffrono.
2. Il percorso seguito dai flussi UDP non isola il server TCP: in questo caso alcuni client TCP possono raggiungere un normale throughput.

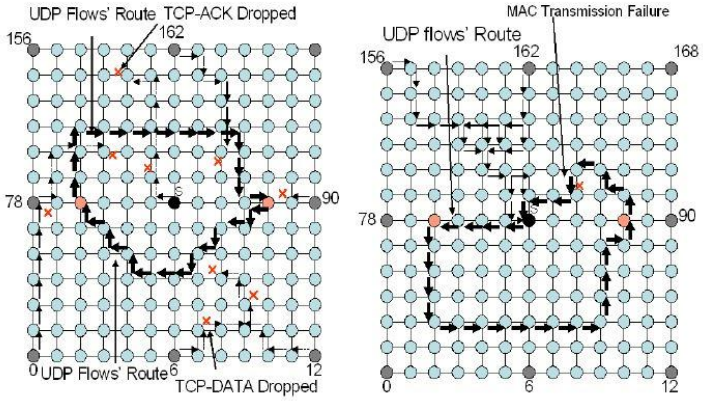


**Figura 27:** *Caso di isolamento e di non isolamento del server TCP*

Da ciò si evince che il flusso UDP sovraccarica i nodi sul percorso ed è difficile per i flussi TCP attraversare il percorso congestionato.

### 3.3.2. Instradamento stabile con aumento della persistenza per le trasmissioni MAC

Un elevato numero di errori di trasmissione MAC compromette le prestazioni TCP. Se si aumentano la persistenza del protocollo MAC e il parametro relativo al limite di tentativi, si può osservare che il goodput dei client TCP migliora significativamente. In particolare, in presenza di flussi UDP, il goodput dei client è di circa il 33% del goodput iniziale, in contrapposizione al 10% osservato nel paragrafo precedente. Aumentare il limite di tentativi non è comunque una soluzione accettabile.

### 3.3.3. Equità di MAC e del Network Layer

Si ipotizzino questi due scenari:

1. Ipotizziamo di sostituire lo standard IEEE 802.11 con un protocollo MAC equo: uno schema di Time Division Multiple Access è stato implementato per una griglia 5x5 dove 4 client sono stati posizionati agli angoli di tale griglia, mentre il server è stato posizionato al centro. Lo slot di tempo è stato scelto in modo tale che un frame MAC possa trasportare un singolo pacchetto TCP in un unico slot di tempo. Questo schema non è realistico, ma mostra che il protocollo TCP è ancora inefficace a causa dell'uso di code FIFO, poiché i flussi UDP riempiono le code nei vari nodi.
2. Ipotizziamo di implementare un sistema di accodamento equo al protocollo MAC sopra descritto: sono stati studiati quattro schemi di classificazione dei pacchetti per far rispettare l'equità basati rispettivamente su IP sorgente, IP destinazione, next-hop e previous-hop. Da qui è stato dimostrato che next-hop e previous-hop non sono sufficienti a prevenire la diminuzione delle prestazioni TCP in presenza di flussi UDP ma che i sistemi basati su IP sorgente e IP destinazione sono in grado di prevenire tale diminuzione delle prestazioni. Questo perché se un flusso TCP si trova a condividere un percorso con un flusso UDP, da quel momento in poi i pacchetti del flusso TCP non sarebbero distinti dai pacchetti dei flussi UDP.

### 3.3.4. Accodamento equo

Nello scenario simulato i client TCP inviano pacchetti di dati al server. L'accodamento basato su IP destinazione è quello che fornisce i risultati migliori poiché gli ACK provenienti dal server sono collocati su code separate rispetto ai nodi del percorso di ritorno ai vari client. In questo modo l'ACK può ricevere fino all'80% della banda. Questo riduce i tempi di RTT delle varie connessioni TCP, aumentando quindi il loro goodput. Se invece il server sta inviando pacchetti di dati, l'accodamento basato su IP sorgente risulta migliore. Sono state fatte le seguenti osservazioni:

1. Nelle condizioni iniziali, senza traffico UDP, con l'accodamento equo la performance dei client TCP migliora del 5%.
2. In presenza di flussi UDP, i client TCP possono, con l'accodamento equo, ottenere circa il 64% del goodput iniziale mentre prima, con l'aumento della persistenza per le trasmissioni MAC, questo valore era di circa il 33%.

### 3.3.5. Risultati ottenuti

Attraverso la casistica dello scenario si è visto che provvedimenti semplici come l'instradamento stabile e l'aumento della persistenza per le trasmissioni MAC aiutano a raggiungere il 10%-33% nel goodput del flusso TCP in presenza del flusso UDP. Inoltre, il protocollo MAC equo non è sufficiente a migliorare le prestazioni del TCP in presenza di flussi UDP, ma un accodamento equo migliora le prestazioni di TCP in modo significativo.

## 3.4. Scenario 4: TCP Westwood nell'interazione con flussi UDP

Come ultima cosa, testiamo l'efficacia dell'algoritmo di stima della larghezza di banda di TCP Westwood (considerato precedentemente in [9]) quando un suo flusso si trova a condividere un collo di bottiglia con le connessioni UDP. La configurazione della simulazione dispone di un collegamento con collo di bottiglia di 5 Mbps con un ritardo di propagazione di sola andata di 30 ms. Una connessione TCP condivide lo stesso collo di bottiglia con due connessioni UDP di tipo ON-OFF e sia i pacchetti TCP che quelli UDP sono assegnati con la stessa priorità. Ogni connessione UDP trasmette a un bit rate constante di 1 Mbps quando è ON. Entrambe le connessioni UDP iniziano nello stato OFF; dopo 25 secondi il primo collegamento è attivato, mentre il secondo collegamento si attiva dopo 50 secondi. La seconda connessione segue un modello di OFF-ON-OFF con tempi di 75, 125 e 175 secondi. Quando si raggiungono i 200 secondi, anche la prima connessione viene messa ad OFF. Dopodiché, i collegamenti UDP rimangono OFF fino alla fine della simulazione, mentre la connessione TCP Westwood invia i dati durante la simulazione. Il comportamento del processo di stima della larghezza di banda è mostrato nella figura 28.

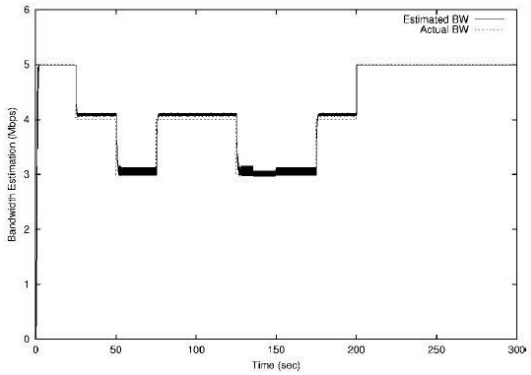


**Figura 28:** *Stima della larghezza di banda di TCP Westwood con traffico UDP*

# 4. Il protocollo TCP-friendly

Come abbiamo detto precedentemente, ci sono applicazioni che prediligono l'utilizzo del protocollo TCP, mentre altre prediligono l'utilizzo del protocollo UDP. Per quanto riguarda Internet, va sottolineata l'importanza del controllo di congestione della rete poiché, se tale controllo non ci fosse, in caso di rete congestionata le applicazioni ritrasmetterebbero nuovamente i pacchetti persi, perdendo anche questi ultimi e aumentando così il traffico di rete, rischiando il collasso della rete stessa, situazione in cui la banda è occupata quasi esclusivamente da pacchetti che vengono scartati a causa della congestione prima che arrivino a destinazione. Abbiamo inoltre visto, nei capitoli precedenti, che in caso di condivisione della banda tra flussi TCP e flussi UDP, questi ultimi si appropriano di tutta la banda a scapito dei flussi TCP che diminuiscono il proprio bit rate.

Fin'ora, l'effetto indesiderato di appropriazione della rete da parte di applicazioni che non si basano sul protocollo TCP non ha avuto un impatto pesante su Internet, poiché la maggior parte del traffico utilizza protocolli basati su TCP. Tuttavia, il numero di applicazioni "non-TCP" come, ad esempio, le applicazioni audio/video in streaming, quali lettori audio, telefonia IP, videoconferenza e simili (applicazioni real-time, che hanno cioè bisogno di un basso ritardo e di banda minima garantita) è in costante crescita e si teme, col tempo, un aumento della percentuale di traffico non-TCP. Poiché queste applicazioni non hanno meccanismi di controllo di congestione, come UDP, anche se riscontrano una congestione di rete, continuano a inviare pacchetti rischiando di far collassare la rete. Per questo motivo vengono definiti opportuni meccanismi per il traffico non-TCP che sono compatibili col traffico TCP. Questi meccanismi sono detti TCP-friendly e servono a portare un'equa distribuzione della larghezza di banda dove essa non è ancora presente.

Un flusso è considerato TCP-friendly se, in caso di congestione, si comporta come un flusso TCP. Nell'ambito del controllo di congestione attuato dal protocollo TCP-friendly, possiamo distinguere fra tecniche di controllo di congestione window-based, dove il sender può inviare pacchetti solo se la finestra non è totalmente piena, e tecniche rate-based, dove si ha direttamente e istantaneamente un valore massimo di bit rate accettabile. Esempi di

metodi window-based sono RAP (Rate Audio Protocol) e LDA+ (Loss-Delay Based Adaption Algorithm), mentre un metodo rate-based è TFRC (TCP-Friendly Rate Control). Esiste anche un algoritmo ibrido che combina aspetti window-based con aspetti rate-based ed è TEAR (TCP Emulation at Receiver). Approfondimenti su TCP-friendly sono in [14].

## 4.1. Classificazione dei meccanismi di controllo di congestione

Di seguito andremo a classificare i diversi meccanismi di controllo di congestione, in modo da capirne le varie caratteristiche e riuscire ad avere una classificazione dei protocolli TCP-friendly per poi vedere alcuni scenari in cui, tali protocolli, interagiscono con i flussi TCP.

### 4.1.1. Window-Based vs Rate-Based

Gli algoritmi window-based utilizzano una finestra di congestione per garantire la TCP-friendliness. Similmente a TCP, ogni pacchetto trasmesso consuma uno slot nella finestra di congestione e ogni pacchetto ricevuto o l'ACK di un pacchetto ricevuto libera uno slot. Il mittente può trasmettere i pacchetti solo quando c'è uno slot disponibile. La finestra viene aumentata se non c'è congestione e viene diminuita se c'è congestione.

Gli algoritmi rate-based garantiscono la TCP-friendliness adattando dinamicamente la velocità di trasmissione attraverso un meccanismo di feedback che indica la congestione di rete e che si avvale di tecniche AIMD o model-based. Le tecniche AIMD sono però inadatte per quanto riguarda i flussi multimediali continui, poiché hanno un andamento a "dente di sega". Le tecniche model-based, invece, utilizzano un modello TCP che, adattando il tasso di invio di throughput, sono in grado di produrre delle variazioni sul tasso di invio molto più regolari e più adatte all'invio di flussi multimediali continui.

### 4.1.2. Unicast vs Multicast

Un flusso unicast è considerato TCP-friendly se riduce il throughput di un flusso TCP coesistente al massimo al pari di quanto lo ridurrebbe un altro flusso TCP che condivide lo stesso percorso.

Un flusso multicast è considerato TCP-friendly quando, per ogni coppia di mittente-destinatario, il flusso multicast risulta essere un flusso unicast TCP-friendly.

### 4.1.3. Single-Rate vs Multi-Rate

Questa classificazione è solo per i protocolli multicast TCP-friendly, poiché i protocolli unicast sono limitati a sistemi single-rate. Nei regimi single-rate, i dati vengono inviati a tutti i destinatari alla stessa velocità, limitando la scalabilità. Nei regimi multi-rate, invece, si ha un'allocazione più flessibile di banda lungo i diversi percorsi di rete.

### 4.1.4. End-to-End vs Router-Supported

Il controllo di congestione end-to-end è il più diffuso in Internet e può essere suddiviso in approcci che si basano sul destinatario, dove i destinatari utilizzano un approccio di controllo della congestione a strati, o approcci che si basano sul mittente, dove il mittente utilizza informazioni sulla congestione di rete in base alle quali regola il rate o la dimensione della finestra per poter raggiungere la TCP-friendliness.

I protocolli router-supported sono, invece, quelli che si basano su funzionalità aggiuntive nella rete, come ad esempio l'aggregazione di feedback, le misure di gerarchia RTT, la gestione di sottogruppi di ricevitori, la modifica delle strategie di accodamento dei router, ecc.

## 4.2. Classificazione dei protocolli TCP-friendly

Ora che abbiamo classificato i metodi di controllo di congestione, possiamo proseguire vedendo, nella figura 29, la classificazione dei protocolli TCP-friendly.

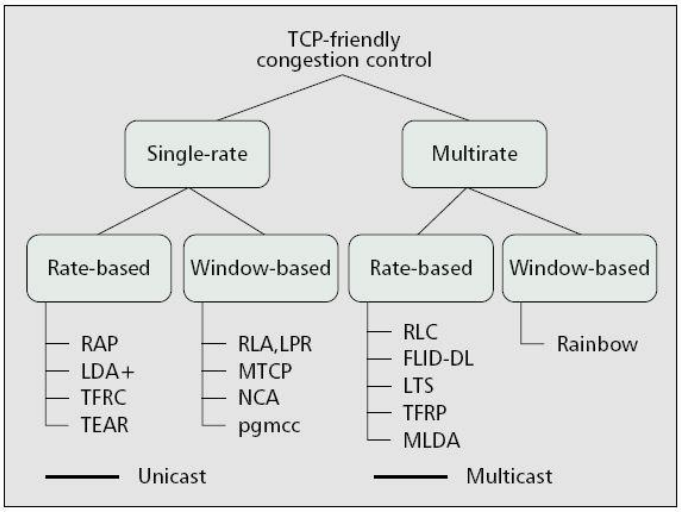


**Figura 29:** *Classificazione dei protocolli TCP-friendly*

## 4.3. Interazione tra flussi TCP e flussi TCP-friendly

Mostreremo di seguito cosa succede quando flussi TCP si trovano a condividere lo stesso collo di bottiglia con flussi, cosiddetti, "non-TCP" che attuano un meccanismo di controllo di congestione simile a quello utilizzato dal protocollo TCP. Tutti gli scenari considerati, si basano su un TCP che adotta l'algoritmo di AIMD per il controllo di congestione, dove cioè il mittente si avvale di una costante per limitare la quantità di propri dati in transito nella rete in un certo istante.

### 4.3.1. Scenario 1: Controllo di congestione equation-based – TFRC

Lo scenario è descritto in [15]. TFRC regola il suo tasso di invio, e quindi il suo controllo di congestione, in base alla seguente equazione di throughput:

$$T(t_{RTT}, t_{RTO}, s, p) = \min\left(\frac{W_m \times s}{t_{RTT}}, \frac{s}{t_{RTT}\sqrt{\frac{2bp}{3}} + t_{RTO}\min\left(1, 3\sqrt{\frac{3bp}{8}}\right)p(1+32\,p^2)}\right)$$

dove $t_{RTT}$ è il parametro di round-trip-time, $t_{RTO}$ è il valore di retransmission time-out, $s$ è la dimensione del segmento, $p$ è il numero di pacchetti persi, $b$ è il numero di pacchetti di ACK e $W_m$ è la dimensione massima della finestra di congestione. Il tasso di perdita viene misurato in termini di intervalli di perdita e viene calcolato come l'inverso della dimensione media dell'intervallo di perdita.

TFRC funziona nel seguente modo: subito dopo l'invio di un messaggio, il mittente entra in una fase di slow start simile a quella di TCP. Lo slow start di TFRC termina con il primo evento di perdita. A ogni RTT, il destinatario aggiorna i suoi parametri e invia un rapporto sullo stato al mittente che calcola, quindi, un nuovo tasso equo al fine di regolare i suoi tassi di invio.

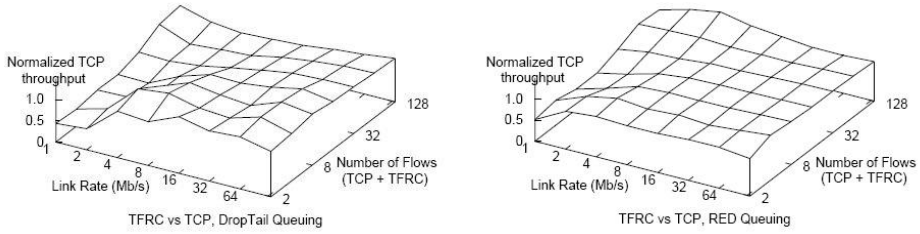


**Figura 30:** *Tasso di invio del flusso TCP quando coesiste con TFRC*

Simuliamo uno scenario in cui n flussi TCP e n flussi TFRC condividono lo stesso collo di bottiglia e andiamo a variare il numero di flussi, la larghezza della banda nel collo di bottiglia e a scalare la dimensione della coda. Per le valutazioni sono state viste le implementazioni di TCP Reno e TCP Tahoe. La figura 31 mostra la media di throughput TCP negli ultimi 60 secondi della simulazione. Si può notare che l'utilizzo della rete è sempre superiore al 90% e spesso è addirittura superiore al 99% quindi quasi tutta la banda rimanente viene utilizzata dai flussi TFRC. Questi dati dimostrano che TFRC e TCP possono coesistere in una vasta gamma di condizioni di rete e che il throughput TCP è simile a quello che sarebbe stato se il traffico concorrente fosse stato TCP invece che TFRC. Esistono comunque alcuni casi, in genere quando la finestra di TCP è molto piccola, in cui TCP subisce delle "ingiustizie". Inoltre, anche se la velocità media dei due protocolli è piuttosto simile, la varianza può essere molto elevata poiché la varianza di trasmissione tra i flussi dipende dal tasso di perdita. La figura 31 mostra il coefficiente di variazione tra i flussi (CoV) contro il tasso di perdita in simulazioni con 32 flussi TCP e 32 flussi TFRC. Sono mostrati i risultati di 10 simulazioni per ogni insieme di parametri. Dai risultati ottenuti si può concludere che in tempi medi e con tassi di perdita tipici (circa meno del 9%), l'equità di TFRC risulta essere migliore di quella di TCP ma, in condizioni di forte sovraccarico della rete, i flussi TFRC mostrano una maggiore varianza tra i loro throughput rispetto a TCP.

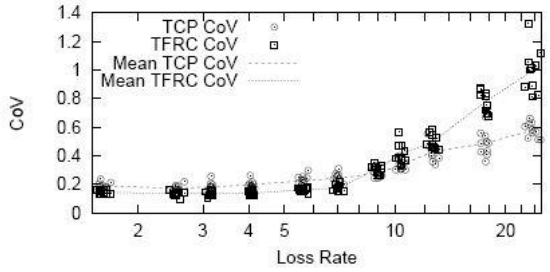


**Figura 31***: CoV di throughput tra i fluss*

#### 4.3.1.1. Lunga durata e traffico di background

Consideriamo ora uno scenario con 16 flussi TCP SACK e 16 flussi TFRC, con larghezza di banda di 15 Mbps nel collo di bottiglia e una coda RED.

La figura 32 mostra i rapporti di equivalenza di TCP e TFRC in funzione di una scala temporale di misurazione. Le curve mostrano il rapporto medio di equivalenza tra coppie di

flussi TCP, coppie di flussi TFRC e coppie di flussi diversi tra loro. Le misurazioni per le coppie TFRC e per le coppie TCP mostrano che i flussi TFRC sono “equivalenti” tra loro su una gamma più lunga di tempo rispetto ai flussi TCP.

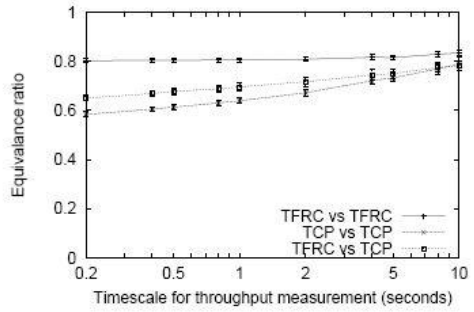


**Figura 32:** *Equivalenza tra TCP e TFRC*

Nella figura 33 notiamo invece che il tasso di trasmissione di TFRC è meno variabile di quello di TCP in un intervallo di tempo ampio.

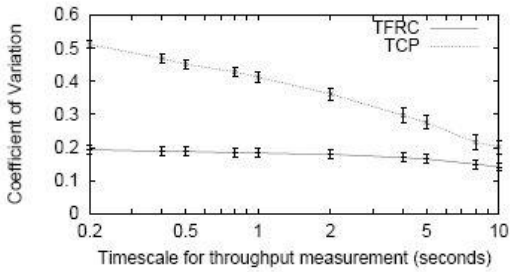


**Figura 33:** *CoV di TCP e TFRC*

Grazie a questi grafici possiamo dire che in questo ambiente, con basse perdite e lunga durata, la velocità di trasmissione di TFRC è paragonabile a quella di TCP ed è meno variabile di un flusso TCP equivalente.

#### 4.3.1.2. Flussi ON-OFF di traffico di background

In questo scenario la media di tempo di ON è di un secondo, mentre quella di OFF è di due secondi. Ogni sorgente invia a 500 Kbps durante un tempo di attivazione. Il numero di connessioni simultanee varia tra 50 e 150 e la simulazione è eseguita per un totale di 5000 secondi. Vengono monitorati due collegamenti: una connessione TCP a lunga durata e una connessione TFRC a lunga durata.

Nella figura 34 viene mostrata l'equivalenza tra TCP e TFRC in questo scenario e possiamo notare che non c'è molta differenza rispetto al caso precedente, anche se ore abbiamo un flusso ON-OFF di traffico  di background.

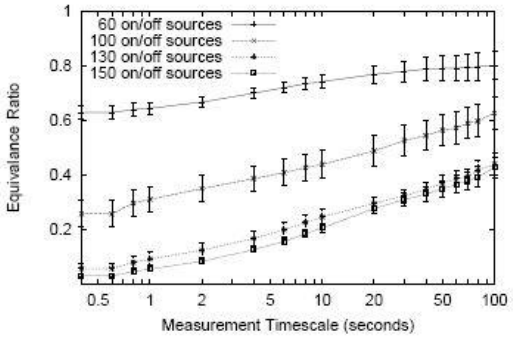


**Figura 34:** *Equivalenza tra TCP e TFRC, con flussi ON-OFF di traffico di background*

La figura 35 mostra che il tasso di trasmissione di TFRC è meno variabile rispetto al tasso di trasmissione di TCP, soprattutto quando il tasso di perdita è alto. Si noti che il CoV, per entrambi i flussi, è molto più alto, rispetto ai valori visti nella figura 33, in tempi comparabili.

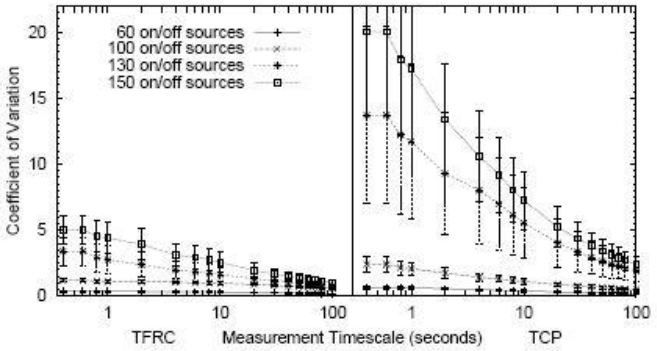


**Figura 35:** *CoV di TFRC e TCP con flussi ON-OFF di traffico di background*

#### 4.3.1.3. Risultati ottenuti

È stato confrontato TFRC quando condivide uno stesso collo di bottiglia con TCP ed è stato visto che TFRC può coesistere con TCP (quindi TFRC risulta essere TCP-friendly). TCP presenta, in tutti gli scenari, degli "svantaggi" e, per questo motivo, TFRC è risultato essere preferibile rispetto a TCP per le applicazioni che richiedono un controllo della congestione e che preferiscono un tasso di invio di dati agevole. Possiamo dire che l'emergere di meccanismi di controllo della congestione, come quelli introdotti da TFRC, per il traffico

unicast può svolgere un ruolo chiave nel prevenire la degradazione della rete, fornendo una buona gestione della banda e un giusto assegnamento di essa ai vari flussi, quando questa si trova a essere condivisa da flussi TFRC e flussi TCP, che risultano essere inadatti a tali applicazioni.

### 4.3.2. Scenario 2: Adattamento basato sul mittente - LDA

Facciamo riferimento allo studio [16]. LDA è un sistema di controllo di congestione AIMD che utilizza alcuni elementi aggiuntivi: i fattori di aumento e diminuzione per AIMD vengono regolati dinamicamente rispetto alle condizioni della rete. Una stima della larghezza di banda del collo di bottiglia è ottenuta utilizzando coppie di pacchetti. In pratica, l'algoritmo LDA è progettato in modo simile a TCP e impedisce la starvation delle connessioni ma permette un comportamento di trasmissione stabile.

LDA funziona nel seguente modo: sulla base delle informazioni sulle perdite da parte dei destinatari, il mittente aumenta o diminuisce la sua velocità di trasmissione e durante i periodi senza perdite aumenta la velocità di trasmissione di un tasso di incremento additivo (AIR) che è stimato utilizzando i fattori di perdita, di ritardo e di larghezza di banda nei colli di bottiglia.

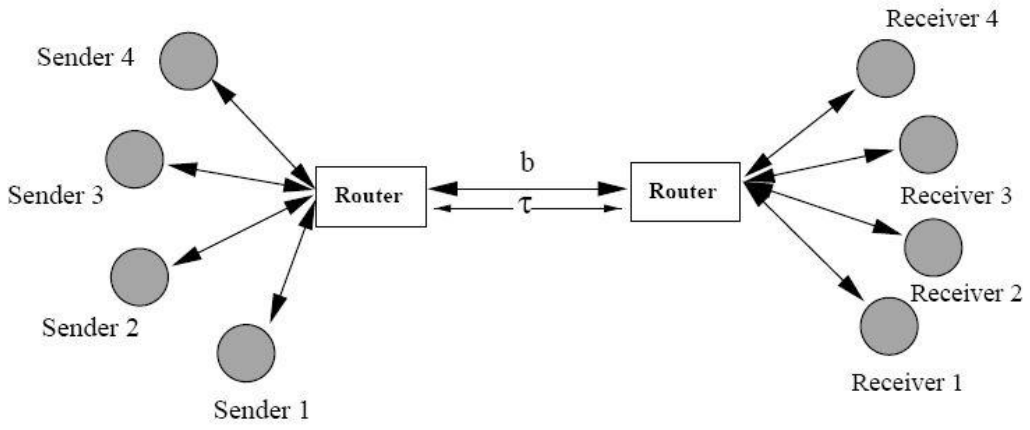


**Figura 36:** *Topologia per LDA*

La topologia della simulazione è quella della figura 36, dove uno dei mittenti invia flussi TCP. Viene presentato un caso con invio di 10 Mbps nel collo di bottiglia e diversi tempi di round-trip-time.

I risultati della distribuzione di banda, rappresentati nella figura 37, mostrano che:

- caso in cui τ = 1 ms: la connessione TCP riceve la stessa quota di larghezza di banda della connessione LDA
- caso in cui τ = 100 ms: la connessione TCP riceve quasi 1,4 Mbps, che è circa la metà della sua quota che, in questo caso, ci aspettiamo che sia 2,5 Mbps

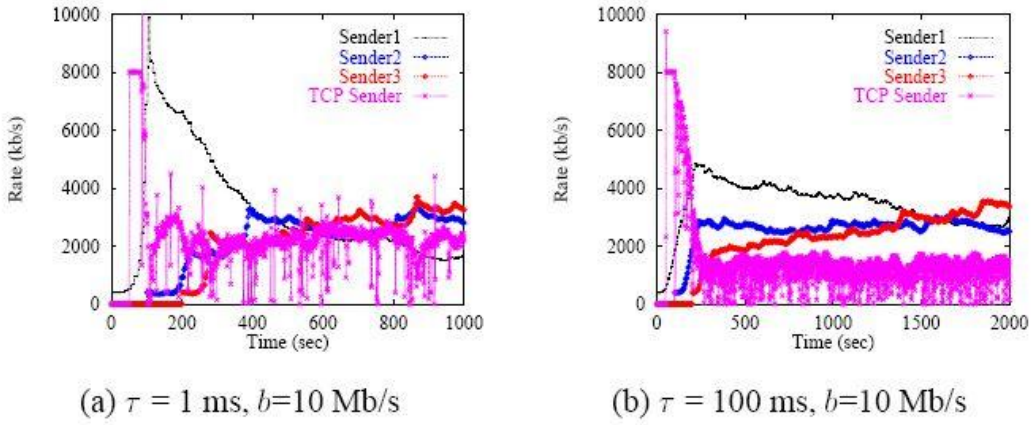


**Figura 37:** *Bandwidth della topologia*

#### 4.3.2.1. Invio di dati tra due workstation

Vengono connesse due workstation attraverso un router di capacità limitata a 1 Mbps. Vengono inviati due video JPEG dalla prima workstation alla seconda e, in parallelo, viene iniziato un trasferimento dati di tipo FTP.

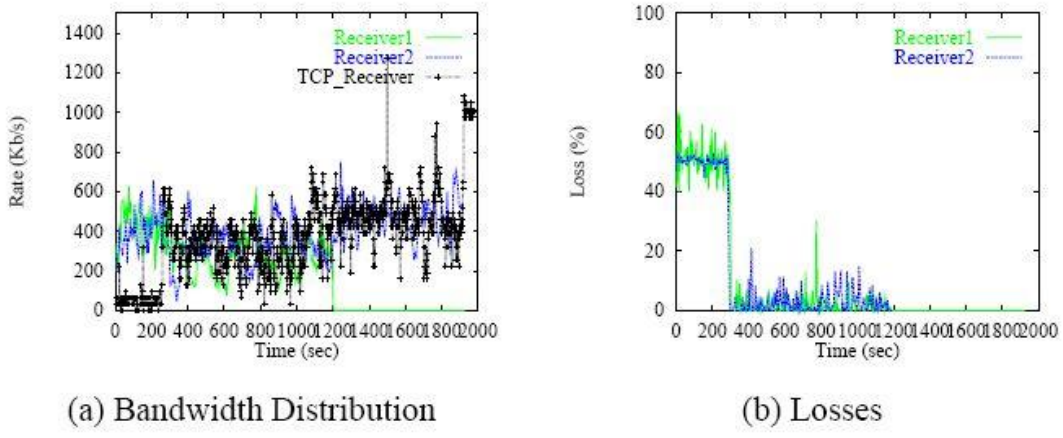


**Figura 38:** *Distribuzione della banda e perdite nell'interazione tra TCP e LDA*

Possiamo notare, dalla figura 38, che i primi 300 secondi della misurazione assomigliano a una situazione tipica di Internet, dove entrambi i flussi video sono stati inviati senza utilizzare alcuno schema di adattamento e provocano starvation tra le connessioni TCP

attive e dove le connessioni UDP soffrono di perdite elevare (circa il 50%). Nei 900 secondi successivi, entrambe le connessioni UDP adattano la loro velocità di trasmissione al loro livello di perdita portando, quindi, a una distribuzione di banda equa di circa 300 Kbps per entrambi i collegamenti attivi e per la connessione TCP e facendo scendere le perdite delle connessioni UDP del 7%. Quando si arriva al punto in cui una connessione UDP interrompe l'invio dei dati, si può notare che sia l'altra connessione UDP che la connessione TCP aumentano la loro quota di banda fino a circa 500 Kbps.

### 4.3.3. Regolazione diretta – DDA

Lo studio approfondito riguardante questa sezione è in [17]. DDA si basa sui meccanismi di controllo della congestione di TCP e sul protocollo di trasporto per sistemi real-time (RTP) per le informazioni di feedback. Ci troviamo in uno scenario identico a quello della sezione precedente ed illustrato nella figura 36.

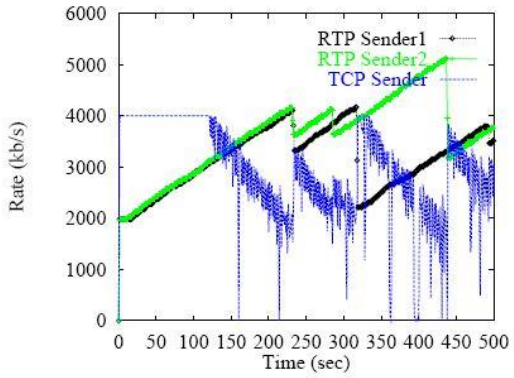


**Figura 39:** *Distribuzione della banda con DDA e TCP*

Nella figura 39 si mostra che la connessione TCP ottiene circa il 30% di banda e le due connessioni DDA ne ottengono il 35% di ciascuna. Poiché i mittenti del DDA aggiornano il loro tasso di trasmissione ogni pochi secondi, potrebbero continuare a inviare con un altro tasso durante il periodo di congestione finché un messaggio di controllo non indica che ci sono delle perdite. In questo periodo, la connessione TCP riduce la sua finestra di trasmissione portando a una riduzione della congestione.

# 5. Variazioni dei flussi TCP

Come ultima cosa, vedremo come si comporta il protocollo TCP quando sono i suoi stessi flussi a variare nella rete. Facciamo riferimento alla documentazione [18].

## 5.1. Rete ad alta velocità

Andremo ora a vedere che, in reti ad alta velocità, l'assegnamento della banda a diversi flussi TCP, quando tra loro condividono uno stesso collo di bottiglia, è ingiusto.

Quando vari flussi TCP con diversi tempi di RTT condividono lo stesso collo di bottiglia, essi non avranno accesso alla stessa quota di banda disponibile. Questo si può evincere dalla seguente equazione:

$$Thru = \frac{c}{RTT\sqrt{p}}$$

che riguarda il throughput dei flussi *Thru*, la probabilità *p* di un pacchetto di essere perso in un flusso e l'RTT e dove *c* è una costante fissa. In seguito a questa formula, ci si aspetta che quando due flussi *i* e *j* condividono lo stesso collo di bottiglia e sono soggetti alla stessa probabilità *p*, il rapporto tra i loro throughput sia inversamente proporzionale al rapporto dei loro RTT. Ciò è dato dalla seguente formula:

$$FR_{i,j} := \frac{Thru_i}{Thru_j} = \frac{RTT_j}{RTT_i}$$

dove $FR_{i,j}$ rappresenta il rapporto di correttezza dei due flussi.

Questo unfairness è causato dal controllo di congestione TCP attuato attraverso il meccanismo AIMD, visto precedentemente.

Lo scenario in cui ci troviamo è quello in cui vengono eseguite due ns-2 simulazioni di 4 flussi TCP in competizione per lo stesso collo di bottiglia da 500 Mbps con diversi ritardi di propagazione. In entrambe le simulazioni, i 4 flussi TCP condividono il collo di bottiglia

con 50 Mbps (10%) di traffico UDP di background con periodi di ON-OFF la cui durata è, in entrambi i casi, distribuita con una media di 0,5 secondi. Le due simulazioni differiscono dal fatto che la prima (figura 40 (a)) usa solo 10 sorgenti UDP per generare il traffico di background, mentre la seconda (figura 40 (b)) usa 1000 sorgenti UDP per generare la stessa quantità di traffico di background.

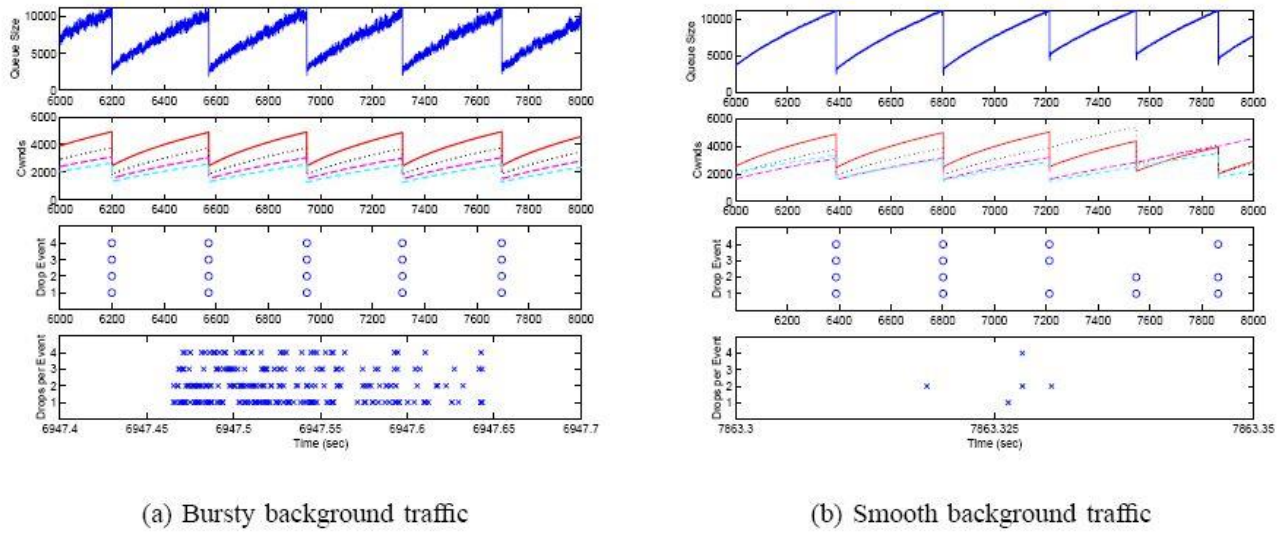


**Figura 40:** *Dimensione della coda, finestra di congestione, eventi di derivazione e perdite di pacchetti nello scenario*

Si può notare che, nella simulazione mostrata nella figura 40 (a), in media 3,8 connessioni TCP su 4 riscontrano una perdita di pacchetti, durante la sincronizzazione, su ogni evento di congestione. Ciò va a contrastarsi con la simulazione della figura 40 (b) dove, in media, solo 2,6 connessioni TCP su 4 riscontrano perdite.

### 5.1.1. Topologia con due flussi TCP

Per capire come la sincronizzazione influisce sull'equità, consideriamo la topologia mostrata nella figura 41, dove le sorgenti hanno sempre pacchetti da inviare e dove abbiamo due flussi TCP con diversi ritardi di propagazione. Partiamo dal presupposto che tutti i flussi sono sincronizzati (accodamento drop-tail).

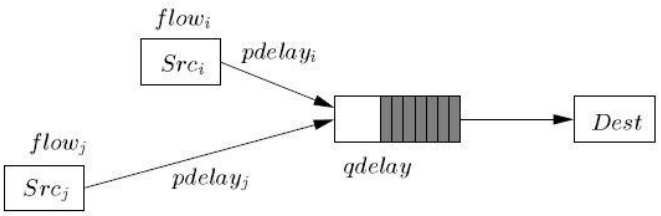


**Figura 41:** *Topologia a 2 flussi TCP*

Nella figura 42 viene mostrata l'evoluzione delle finestre di congestione di TCP quando due flussi sono sincronizzati (dove per semplicità non è stata considerata la partenza lenta). Si può notare che, poiché la finestra di congestione aumenta di un pacchetto per ogni RTT, il flusso con RTT minore mostra un aumento più rapido nella dimensione della sua finestra di congestione. Nella figura viene denotata con $W_i$ la finestra dell'*i*-esimo flusso al tempo *T*.

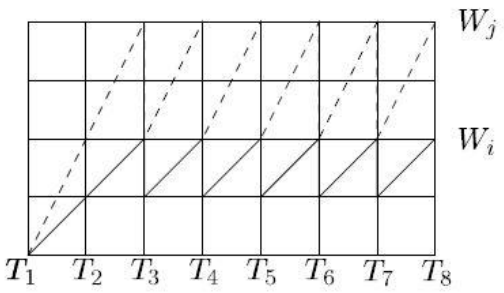


**Figura 42:** *Finestre di congestione di due flussi TCP*

È stato da qui dimostrato che il rapporto del fairness tra due flussi TCP sincronizzati è dato da:

$$FR_{i,j} := \frac{Thru_i}{Thru_j} = \left(\frac{RTT_j}{RTT_i}\right)^2$$

che si può notare essere il quadrato del risultato ottenuto precedentemente.

### 5.1.2. Topologia con 4 flussi TCP

Come mostrato nella figura 43, quattro connessioni TCP vengono simulate con ritardi di propagazione di 45 ms ($Src_1$), 90 ms ($Src_2$), 135 ms ($Src_3$) e 180 ms ($Src_4$). Per generare il burstiness del traffico di background vengono inviati diversi tipi di traffico di background nel router *R2*.

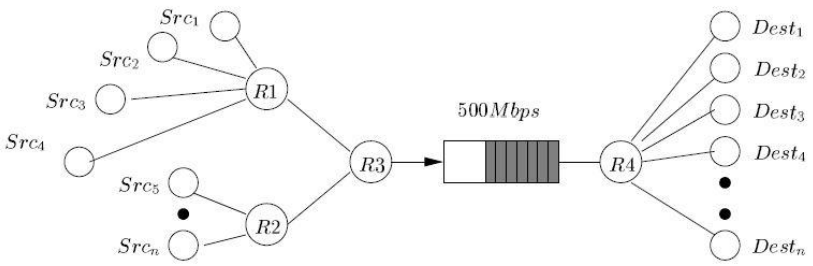


**Figura 43:** *Topologia a 4 flussi TCP*

Vengono definite tre diverse metriche per misurare la burstiness del traffico in background:

1. Burst Density Packet Drop (BPDD): misura il numero medio di perdite di pacchetti per ogni evento di invio e include le perdite di pacchetti sia in "primo piano" che in background
2. Percentuale del numero medio di flussi che perdono pacchetti (P): mostra il numero medio di flussi in cui si verifica almeno la perdita di un pacchetto per ogni evento di perdita
3. Coefficiente di variazione (CoV): è il rapporto tra la deviazione standard e la media del numero di pacchetti che arrivano nel collo di bottiglia di un router.

Nella figura 44, possiamo notare i vari risultati ottenuti da queste metriche.

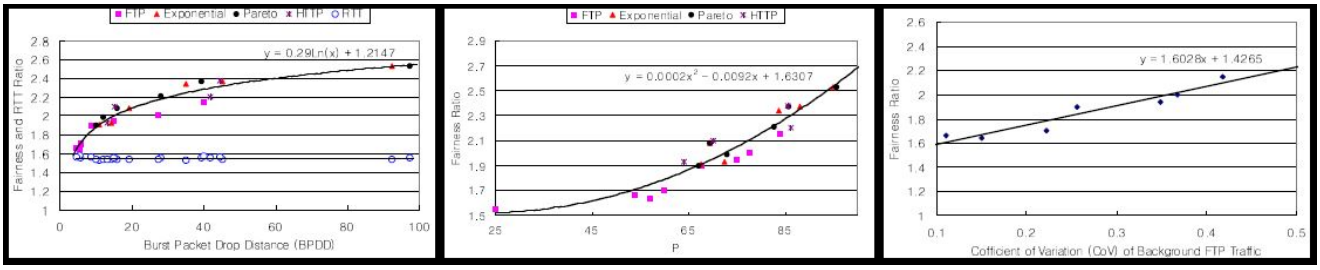


**Figura 44:** *BPDD, P e CoV dello scenario*

Si può notare che BPDD ha una forte relazione con il fairness: se un traffico basso provoca perdite di pacchetti più alte nella coda di un collo di bottiglia, allora il fairness tra due flussi diventa più grande. Inoltre, se P aumenta, allora aumenta anche il fairness. Stessa cosa avviene per la misurazione del CoV a breve termine, cioè, se CoV aumenta allora aumenta anche il fairness. Da questi risultati si evincono le scarse prestazioni di TCP nelle reti ad alta velocità.

# 6. Conclusioni

In questa tesi ci siamo occupati di valutare il protocollo TCP attraverso vari scenari. Abbiamo visto come si comporta il protocollo TCP quando interagisce con il protocollo UDP e con i protocolli TCP-friendly, protocolli cosiddetti "non-TCP" che fanno uso, a differenza del più semplice protocollo UDP, di meccanismi di controllo della congestione simili a quelli di TCP. Infine abbiamo dato uno sguardo al comportamento del protocollo TCP quando, su una stessa rete, sono i suoi flussi a interagire tra loro.

Abbiamo iniziato la tesi mostrando il funzionamento dei protocolli di trasporto più comuni, quali TCP e UDP, e notando che il protocollo TCP risulta essere più complesso del protocollo UDP a causa dei suoi meccanismi di controllo di congestione, che causano dei ritardi di trasmissione rispetto a UDP. Dall'altra parte, però, abbiamo visto che UDP non attua tali meccanismi e ciò va a scapito della consegna affidabile dei pacchetti, e può provocare la congestione della rete o addirittura il collasso, cioè una situazione in cui il protocollo UDP continua a inviare pacchetti anche se il canale è saturo, causando la perdita dei pacchetti inviati e il successivo collasso. Ci siamo quindi soffermati a discutere tali meccanismi di controllo di congestione, il cui funzionamento si basa sulla riduzione della finestra di congestione al riscontro di una perdita di pacchetto. È stato notato, attraverso gli scenari, che TCP non fa distinzione tra pacchetti persi a causa di congestione o pacchetti persi a causa della tipologia di rete utilizzata, infatti in reti wireless una perdita può essere causata da fattori diversi dalla congestione, come ad esempio lo spostamento dell'utente o il traffico di rete, e quindi TCP, indipendentemente dal tipo di perdita, riduce la finestra di congestione, ottenendo in questo caso prestazioni inferiori e non utilizzando appieno la capacità massima di banda ottenibile.

Abbiamo poi parlato dell'evoluzione di TCP nel corso degli anni, visto che inizialmente TCP era stato ideato solamente per reti cablate e ha dovuto subire modifiche per riuscire a rendere le comunicazioni ottimali anche per le reti wireless e per vari applicativi che stanno prendendo piede nello sviluppo della rete degli ultimi anni. Siamo quindi andati a vedere come si è passati da TCP Berkeley, primo meccanismo TCP atto a cercare di evitare la

congestione di rete con un eccessivo traffico, fino ai più moderni TCP Westwood e TCP Westwood+, che hanno apportato modifiche sender-side-only al più famoso TCP Reno con lo scopo di migliorare la gestione dei ritardi di banda prodotti dai percorsi con potenziale perdita di pacchetti causata dalla trasmissione o da altri errori e con carico dinamico. Lungo il percorso di evoluzione di TCP abbiamo anche visto il funzionamento di TCP Tahoe, TCP Reno, TCP New Reno, TCP Vegas e TCP Veno, ognuno dei quali apporta delle modifiche al protocollo precedente al fine di soddisfare al meglio le varie richieste di rete che si sono succedute nel corso degli anni. Ci siamo inoltre soffermati su TCP SACK che, rispetto agli altri TCP, non si basa su un riscontro cumulativo degli ACK; il mittente, cioè, non attende tutto il tempo di un RTT per scoprire se l'invio del pacchetto è andato o meno a buon fine. SACK, acronimo di Selective Acknowledgment, attua un meccanismo di controllo selettivo, dove cioè un ACK si riferisce unicamente a un pacchetto, e non a tutti quelli ricevuti precedentemente, in modo tale da permettere al mittente di capire quali siano effettivamente i pacchetti ricevuti dal destinatario e quali siano ancora all'interno della rete o siano andati perduti. Abbiamo inoltre visto che SACK non affronta il problema del riconoscimento di ACK duplicati quindi, per questo motivo, è stato introdotto D-SACK (Duplicate SACK) che permette di prevenire alcune situazioni di errore.

Avendo ottenuto una buona panoramica sui vari TCP evolutisi nel tempo, li abbiamo confrontati attraverso vari scenari ed è emerso che, in presenza di compressione degli ACK, Westwood+ fornisce una buona stima della banda, a differenza di Westwood che la sovrastima e possiamo dire che Westwood+ è migliore per gli scenari congestionati in cui la compressione degli ACK è diffusa e che, in tali scenari Westwood potrebbe portare a una maggiore congestione di rete. Inoltre notiamo che, in termini di prestazioni, TCP Reno, TCP New Reno e TCP Tahoe mostrano un comportamento molto simile. TCP New Reno migliora le prestazioni di TCP Reno solo nel caso in cui più errori si verifichino nella stessa finestra di trasmissione. Inoltre possiamo dire che Westwood offre il vantaggio di modificare solo il mittente, al contrario di altre varianti, come ad esempio TCP SACK, che richiedono anche modifiche al destinatario. Notiamo infine che, confrontando TCP New Reno, TCP Vegas e TCP Westwood+ quando condividono un stesso collo di bottiglia da 10 Mbps e interagiscono con altri flussi che creano traffico di ritorno, Westwood+ e TCP New Reno raggiungono la piena utilizzazione della banda con Westwood+ che risulta più equo di

TCP New Reno, mentre TCP Vegas risulta non essere in grado di sfruttare appieno la banda.

Nella tesi ci si è poi occupati dell'interazione tra il protocollo TCP e il protocollo UDP. Sono quindi stati mostrati degli scenari in cui questi due protocolli hanno dei flussi che si trovano ad interagire nello stesso collo di bottiglia. Da qui è emerso che i flussi TCP vengono penalizzati quando condividono lo stesso collo di bottiglia con i flussi UDP, poiché i flussi TCP attuano meccanismi di controllo di congestione, mentre quelli UDP continuano a inviare pacchetti anche se la rete è congestionata, appropriandosi così di tutta la banda disponibile.

Poiché sempre più applicazioni prediligono l'uso di protocolli cosiddetti "non-TCP", cioè che non attuano un meccanismo di controllo di congestione rischiando di saturare e far collassare la rete, sono stati introdotti i cosiddetti protocolli TCP-friendly, cioè dei protocolli non-TCP che però attuano dei meccanismi di controllo di congestione. Sono quindi stati mostrati alcuni scenari di coesistenza tra flussi TCP e flussi TCP-friendly ed è emerso che TCP mostra ancora alcune problematiche prestazionali ma che comunque la situazione è meno "pericolosa" rispetto a quando i flussi TCP interagivano con flussi non-TCP che non facevano uso di meccanismi di controllo di congestione, come ad esempio UDP. Ci troviamo infatti in casi in cui i meccanismi di controllo di congestione per i flussi non-TCP si comportano in maniera pressoché equivalente rispetto ai meccanismi del controllo di congestione di TCP.

Infine, la tesi si è occupata di vedere cosa succede quando sono proprio gli stessi flussi TCP a condividere una stessa rete. Si è infatti guardato cosa succede quando in una rete ad alta velocità sono prima due e poi quattro flussi TCP a condividere lo stesso collo di bottiglia. Da qui si può notare che c'è ingiustizia nell'assegnamento della banda tra i diversi flussi TCP e questo fa capire che i flussi TCP non sono adatti per le reti ad alta velocità.

# 7. Riferimenti bibliografici


[1] Rosa Divina Di Fiore. *Criteri di Valutazione del Protocollo TCP*. 2012/2013, Università degli Studi di Bologna.

[2] Riccardo Zambon. *Ottimizzazione del TCP su reti di accesso IEEE 802.11b/e e valutazioni delle prestazioni di QoS*. 2005/2006, Università degli Studi di Siena.

[3] Sally Floyd. *Issues of TCP with SACK*.

[4] Rajkumar Kettimuthu, William Allcock. *Improved Selective Acknowledgment Scheme for TCP*.

[5] M. Mathis, J. Mahdavi, S. Floyd, A. Romanow. *TCP Selective Acknowledgment Option*. Ottobre 1996.

[6] Serena Alessandrini. *Prestazioni del protocollo TCP su reti geografiche e ottimizzazione basata su ECN (Explicit Congestion Notification)*.

[7] S. Mascolo, L. A. Grieco, R. Ferorelli, P. Camarda, G. Piscitelli. *Performance evaluation of Westwood+ TCP congestion control*.

[8] Siddharth Gangadhar, Truc Anh N. Nguyen, Greeshma Umapathi, James P. G. Sterbenz. *TCP Westwood(+) Protocol Implementation in ns-3*.

[9] Claudio Cassetti, Mario Gerla, Saverio Mascolo, M. Y. Sanadidi e Ren Wang. *TCP Westwood End-to-End Congestion Control for Wired/Wireless Networks*.

[10] Luigi A. Grieco e Saverio Mascolo. *Performance Evaluation and Comparison of Westwood+, New Reno and Vegas TCP Congestion Control*.

[11] T. Gopinath, A.S. Rathan Kumar, Rinki Sharma. *Performance Evaluation of TCP and UDP over Wireless Ad-hoc Networks with Varying Traffic Latency*.

[12] Christian Rohner, Erik Nordstorm, Per Gunninberg, Christian Tschudin. *Interactions between TCP, UDP and Routing Protocols in wireless Multi-hop Ad hoc Networks*.

[13] Vikram Gupta, Srikanth V. Krishnamurthy, Michalis Faloutsos. *Improving the performance of TCP in the presence of interacting UDP flows in ad hoc networks*.

[14] Joerg Widmer, Robert Denda, Martin Mauve. *A Survey on TCP-Friendly Congestion Control*.

[15] Sally Floyd, Mark Handley, Jitendra Padhye, Jorg Widmer. *Equation-Based Congestion Control for Unicast Applications*.

[16] Dorgham Sisalem, Hanning Schulzrinne. *The Loss-Delay Based Adjustment Algorithm: A TCP-Friendly Adaptation Scheme*.

[17] Dorgham Sisalem, Hanning Schulzrinne. *The Direct Adjustment Algorithm: A TCP-Friendly Adaptation Scheme*.

[18] Junsoo Lee, Stephan Bohacek, Joao P. Hespanha, Katia Obraczka. *A Study of TCP Fairness in High-Speed Networks*.