UNIVERSITÀ DEGLI STUDI DI PAVIA
SEDE DISTACCATA DI MANTOVA
FACOLTÀ DI INGEGNERIA

Corso di Laurea in Ingegneria Informatica

Tesi di Laurea

Analisi delle Tecniche di Stima di Banda di una Rete a Pacchetto

Implementazione di un Software di Misurazione delle Grandezze Prestazionali di una Rete IP

Relatore Prof. Giuseppe Federico Rossi

Laureando Nicola Tassini

Anno Accademico 2003/2004

Ringraziamenti

Intendo ringraziare vivamente il Professor Giuseppe F. Rossi per la costante disponibilità offerta durante lo svolgimento dei progetti.

Ringrazio tutta la mia famiglia, mia prima fonte di sostegno e di riferimento.

Ringrazio infine tutti i miei amici e tutti coloro che hanno avuto la forza di sopportarmi durante questo periodo.

Indice Generale

Analisi delle Tecniche di Stima di Banda di una Rete a Pacchetto

Indice I° Progetto

1. INTRODUZIONE

Da quando le possibilità offerte da Internet sono cresciute, coinvolgendo attivamente sempre più persone ed aziende a livello mondiale, si sono sviluppate nuove esigenze, dovute alla complessità delle applicazioni generate.

In primo luogo, si è creata la necessità di poter gestire una banda d'accesso definita, o comunque entro un certo margine ben definito, basti pensare a tecnologie quali la videocomunicazione e videoconferenza, o la telefonia tramite protocolli Internet (VoIP: Voice over Internet Protocol).

Inoltre, molti ISP, (Internet Service Provider) propongono alla clientela connessioni a banda garantita e, per gli utenti più bisognosi, è necessario talvolta verificare se i limiti stabiliti durante l'abbonamento sono rispettati.

Infine, gli stessi ISP (Internet Service Provider) possono essere interessati alle proprie prestazioni, in modo da comprendere quali sono i punti deboli e come risolverli.

Per poter realizzare questo, è necessario studiare ed analizzare in modo dettagliato il traffico che viaggia attualmente in Internet, capendo quali migliorie si possono attuare, cercando quindi di ottimizzare l'intera gestione delle reti.

L'obbiettivo di questo trattato è, quindi, di studiare gli algoritmi nati nel corso dell'ultimo decennio, volti ad analizzare le prestazioni su reti basate

sul protocollo IP (Internet Protocol), cercando di capire quale tra questi risulta essere il più efficiente.
Si cercherà, infine, di analizzare tutti i problemi che si possono incontrare nella ricerca di una stima della banda offerta, considerando che l'eterogeneità e la complessità di Internet aiutano a rendere questo compito molto difficile.

Nei prossimi capitoli verranno introdotti, dapprima, gli strumenti base per una corretta comprensione delle metodologie di studio, quindi gli elementi che compongono una rete ed i parametri prestazionali d'interesse, per poi passare ad una completa trattazione di ogni algoritmo di stima.

2. COMPONENTI FONDAMENTALI

Occorre, inizialmente, chiarire quali siano gli elementi indispensabili per una corretta comprensione di quanto si affronterà più avanti. Nello specifico, bisogna far luce sulle diverse tipologie di collegamento che si possono trovare fra due host connessi tra loro. Questa distinzione, pertanto, è da effettuare tra i tre seguenti concetti:

- *Segment*
- *Hop*
- *End-to-End*

Un *Segment* corrisponde ad un collegamento a livello fisico tra due hosts, come ad esempio un point-to-point.

Un *Hop*, invece, è costituito da una sequenza di *Segment*, connessi tra loro tramite switch, bridge, hub o altri dispositivi di interconnessione che operano a livello 1 o 2.

Infine un percorso *End-to-End* da un IP host M (Mittente) ad un IP host D (Destinatario) è la sequenza di *Hop* che connette M a D.

3. PARAMETRI PRESTAZIONALI

In questa sezione vengono trattati i principali indicatori del livello prestazionale offerto da una rete. Verranno esaminati pertanto concetti quali la *Capacità* e la *Banda Disponibile*, generalmente definiti sia per singoli link che per percorsi End-to-End.

3.1 Capacità (C)

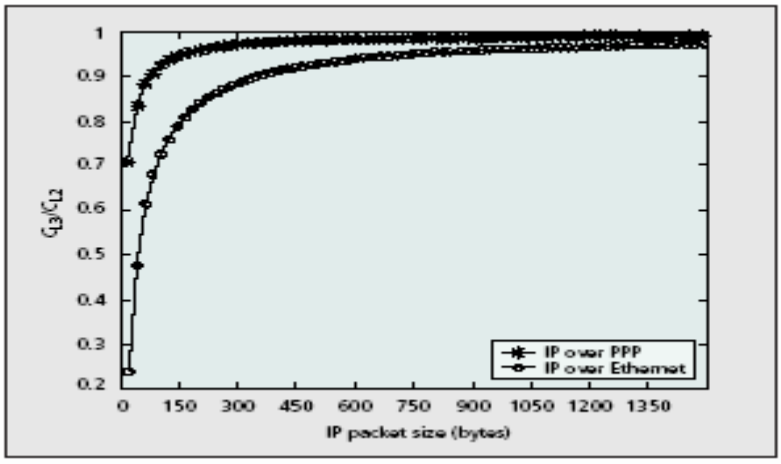
A livello 2 i link possono trasferire dati ad una velocità di trasmissione costante: un collegamento di tipo 10BaseT Ethernet, per esempio, è in grado di spedire pacchetti ad una velocità pari a 10Mb/s, mentre un segmento T1 trasmette a 1.544Mb/s. Queste velocità di trasmissione dipendono sostanzialmente dalla banda che il segnale può occupare, dalla tipologia e complessità dell'Hardware utilizzato come trasmettitore e ricevitore ed infine dal tipo di modulazione adottata.

Ad un livello superiore, l'IP (Internet Protocol), un hop può sfruttare una velocità inferiore rispetto a quella nominale, in quanto fenomeni quali l'incapsulamento e la frammentazione aumentano l'overhead dei pacchetti.

Questi fattori dipendono fortemente dal protocollo adottato, che può creare una notevole influenza sulle prestazioni finali. La figura seguente mostra, per esempio, il rapporto tra la capacità riscontrata a livello Network rispetto a quella nominale a livello DLC (Data Link Control) in

due casi diversi, sfruttando cioè un protocollo Point-to-Point(PPP) ed Ethernet:

Figura 1 - *Confronto Rendimento PPP e Ethernet*



Si può quindi definire la Capacità *C* di un hop *i* come la massima velocità di trasferimento dati a livello IP che quello specifico hop può raggiungere.

Estendendo questo concetto ad un intero percorso di rete, si può definire anche la Capacità End-to-End, vista come la velocità di trasmissione dati al di sopra del livello IP (Internet Protocol) che si può raggiungere tra un mittente ed un destinatario ai capi di una sequenza di Hop.

Questa velocità globale è determinata dal link più lento dell'intero percorso, che stabilisce il ritmo massimo della connessione, che risulta essere:

$$C=\min_{i=1\ldots H} C_i$$

dove C_i è la Capacità di tutti i singoli Hop e H è il numero di nodi che compongono il percorso.

Al giorno d'oggi alcune nuove tecnologie tendono a complicare la definizione di Capacità, come per esempio i link multi-canale (OC-3 Link), che sfruttano appunto diversi canali parallelamente per raggiungere prestazioni migliori. Un altro esempio in cui le precedenti definizioni possono non essere sempre formalmente valide è il protocollo IEEE 802.11b, che descrive il funzionamento delle reti Wireless LAN (Local Area Network). Queste reti infatti non trasmettono a velocità costanti, ma hanno un range che varia da 1Mb/s fino ad arrivare a 11Mb/s (non linearmente), scegliendo in base al tasso di errore che si rileva nell'ambiente di comunicazione.

È corretto quindi precisare che le definizioni suggerite in questo capitolo vanno applicate solo nei casi in cui la Capacità di un Hop, o End-to-End, rimane costante nell'intervallo di tempo in cui la si analizza.

Al quarto livello architetturale occorre effettuare una nuova distinzione, basata sul protocollo adottato. Nel caso in cui si stia valutando una connessione progettata a livello Transport con UDP (User Datagram Protocol), la Capacità del sistema sarà data da:

$$C_{SIS} = C\frac{P}{P + H_{TOT}}$$

Indicando con P il Payload utilizzato e con H_{TOT} l'Overheading totale, risultato di tutti gli incapsulamenti a partire da livello UDP in giù.

Nel caso invece si stia sfruttando una Connection TCP (Transfer Control Protocol) e si vogliono valutare le prestazioni offerte, bisogna appurare la dimensione della Window di trasmissione ed in generale considerare, nell'arco del Round Trip Time, quanti pacchetti invio con i relativi dati contenuti nel Payload, valutando quindi se il tempo che impiego per trasmettere le finestre di dati è inferiore o superiore del Round Trip Time.

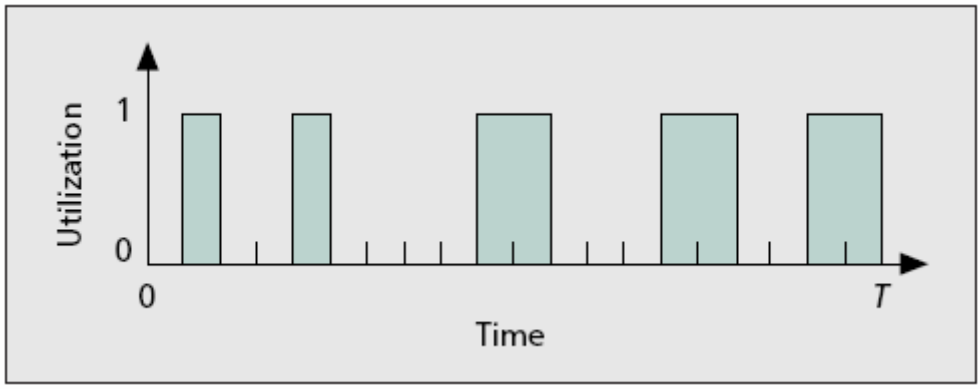
3.2 Banda Disponibile (AB)

Il secondo parametro di valutazione di una connessione è la Banda Disponibile, definibile anche stavolta sia su un singolo Hop che su un intero percorso End-to-End.

La banda disponibile rappresenta la quantità di banda inutilizzata rispetto all'intera capacità. In questo senso, mentre la capacità dipende unicamente dal sistema tecnologico di trasmissione adottato, la banda disponibile è funzione solo del carico che la rete supporta nell'istante di misurazione.

Per definire più in dettaglio questo parametro, si pensi al concetto di Utilizzo della rete e si immagini un caso in cui un host connesso ad un link compia due sole operazioni: o trasferisce i propri dati sfruttando tutta la capacità offerta dal canale trasmissivo, o non manda nulla. Questa situazione è schematizzata nel seguente grafico:

Figura 2 - ***Utilizzo di un Canale nel Tempo***



In questo caso il link è utilizzato solo otto volte sui venti intervalli di tempo considerati: l'utilizzo quindi è del 40%. La banda disponibile è data perciò dalla frazione seguente della capacità:

$$A_i = (1 - u_i)C_i$$

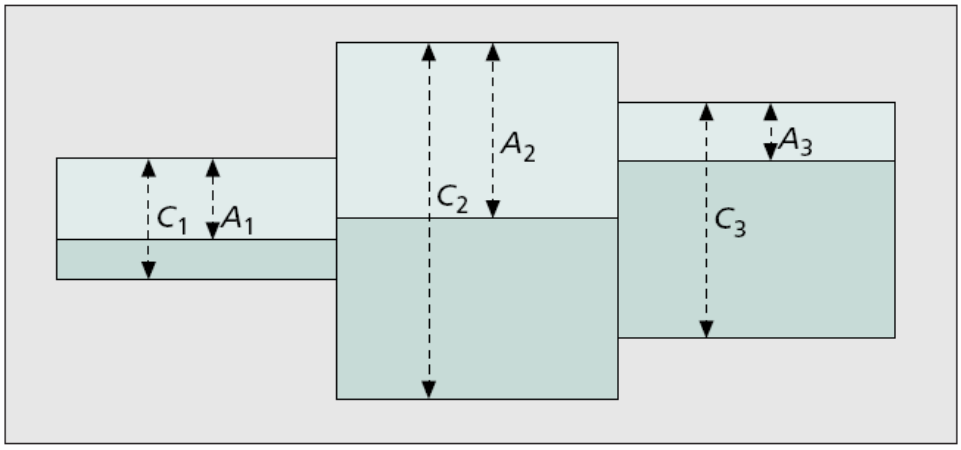
dove u_i rappresenta l'utilizzo del link i-esimo e C_i la capacità di quel tratto.

Estendendo l'idea da un singolo Hop ad un intero percorso, la banda disponibile End-to-End è risulta essere il minimo valore che si riscontra seguendo l'intero tragitto:

$$A = \min_{i=1\ldots H} A_i$$

Intuitivamente si potrebbe pensare che il collegamento con minor capacità, che determina quindi l'intera capacità del percorso, stabilisca anche la quantità di banda disponibile. Questa idea non è sempre corretta, come per esempio nel caso seguente:

Figura 3 - ***Traffico su 3 Hops adiacenti***



In questo particolare caso si può notare che la Capacità inferiore tra i tre diversi collegamenti è quella del primo canale, quindi $C = C_1$, mentre, per quanto riguarda la Banda Disponibile, quella inferiore è quella dell'ultimo tratto del percorso e si può affermare quindi che $A = A_3$.

Un secondo ed ultimo aspetto che è giusto sottolineare, soprattutto nel caso in cui si vogliano progettare dei sistemi di misurazione, è che la banda disponibile, dipendendo dal carico della rete, varia molto rapidamente e quindi gli intervalli di campionamento devono essere i più raffinati possibili, anche se non devono essere in grado di percepire i singoli istanti di invio e di ricezione.

4. INTRODUZIONE ALLE TECNICHE DI STIMA

Nei prossimi capitoli verranno trattate le principali tecniche di misurazione degli elementi prestazionali di una rete, come la capacità e la banda disponibile.

I principali argomenti analizzati saranno:

- Variable Packet Size
- Packet Pair Dispersion
- Packet Triplet Dispersion
- Packet Train Dispersion
- Trains of Packet Pairs
- Packet Tailgating

L'ipotesi comune a tutti gli algoritmi è l'invarianza del percorso e la stabilità del carico della rete durante le misurazioni.

Se queste condizioni non fossero rispettate, il risultato di qualsiasi metodo potrebbe differire notevolmente dalla realtà, producendo misure errate.

Rimane però da aggiungere che, nell'uso comune, le ipotesi necessarie non possono sempre essere mantenute vere e l'unico compromesso che si può raggiungere è adottare intervalli di tempo per la misurazione sufficientemente brevi, in modo da evitare o comunque da abbassare i disturbi legati alla dinamicità dell'utilizzo, anche se non devono essere

così piccoli da identificare momenti in cui non viene inviato nessun pacchetto.

5. VARIABLE PACKET SIZE

Questa tecnica è stata ideata nell'Aprile del 1997 da Jacobson e Bellovin presso il Mathematical Sciences Research Institute (MSRI).

Questo primo metodo si pone come obbiettivo la stima della Capacità di ogni Hop che attraversa sul percorso desiderato.

La strategia si fonda sull'impostazione del campo Time To Live (TTL), nella Header dei pacchetti IP, come numero massimo di nodi che si vogliono percorrere. Il campo TTL (Time to Live) ha il compito di stabilire infatti quanti Hop quel pacchetto può attraversare. Ad ogni passaggio quel valore viene decrementato di una unità e, nel caso in cui diventi nullo, l'intero pacchetto viene eliminato e viene mandato un messaggio di errore al mittente, come stabilisce il protocollo Internet Control Message Protocol (ICMP), del tipo "`Time-Exceeded Error Message`".

Dal lato mittente si sfrutta quindi questo meccanismo per determinare il Round Trip Time (RTT) dalla partenza del messaggio originale alla ricezione del messaggio d'errore ICMP (Internet Control Message Protocol), impostando semplicemente il numero di Hop che si vuole raggiungere.

Conoscendo perciò la dimensione del pacchetto di partenza, misurando l'RTT (Round Trip Time) e facendo poi opportuni confronti tra le misure si

può risalire al valore della Capacità di ogni Hop che compone tutto il percorso.

Le misurazioni che si compiono sono in funzione della dimensione del pacchetto originario, tenendo fissato il numero N di hop che si sta valutando, ed in questo senso prende significato il nome di questa tecnica, *Variable Packet Size*.

Vediamo ora in dettaglio come è composto l'RTT a partire ad esempio dal nodo (N-1)-esimo all'N-esimo nodo, considerando come N il numero di Hop che compongono l'intero percorso:

$$RTT = q_1 + (\tau + L/C_i) + q_2 + inoltro + q_3 + (\tau + error/C_i) + q_4$$

Tutti i tempi indicati con q_i rappresentano le code d'attesa all'ingresso o all'uscita dei router, mentre il valore di *inoltro* è dato dal sistema di Forwarding dell'elaboratore destinatario.

Questa prima espressione è molto rigorosa e vuole tener conto di tutti i fattori temporali che compongono l'intero Round Trip Time, ma a livello pratico è difficile considerarli e soprattutto misurarli tutti.

In questo senso quindi si semplifica la precedente soluzione, sostenendo per esempio che la dimensione dei pacchetti di errore (*error*) è molto piccola (56 byte) e quindi trascurabile, cosi come il tempo di inoltro e i tempi di attesa (*q*), quest'ultimi presupponendo però di compiere un elevato numero di tests.

Se queste condizioni sono verificate, la precedente formula diventa:

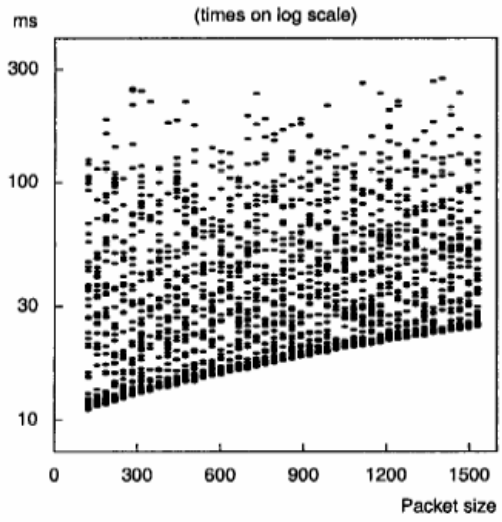
$$RTT = (\tau + L/C) + \tau = L/C + 2\tau$$

dove quindi il Round Trip Time è dato dal tempo di trasmissione del pacchetto più due volte il tempo di ritardo del canale (uno all'andata e uno al ritorno del messaggio ICMP d'errore); da notare infine la dipendenza direttamente proporzionale con la dimensione del pacchetto, proprietà che è alla base di tutto l'algoritmo.

D'ora in poi le misurazioni e i calcoli seguenti saranno impostati utilizzando questo modello di RTT (Round Trip Time), anche se sappiamo a priori che, essendo il frutto di numerose semplificazioni, non potrà rispecchiare al meglio la realtà, ma fornirci solo una stima ragionevolmente valida.

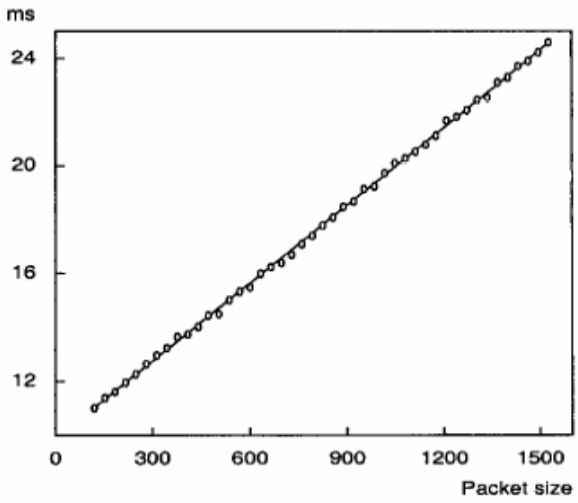
Detto questo, la tecnica prevede che, fissato il numero di Hop che sto valutando, misuri il tempo di ritorno dell'errore, variando la dimensione L del pacchetto originario. In questo modo sono in grado di determinare una serie di valori relativi all'RTT (Round Trip Time) di ogni dimensione stabilita. La figura seguente mostra il risultato di 2880 prove eseguite su un link, relative a dimensioni del pacchetto da 120 a 1528 bytes:

Figura 4 - *Round Trip Time di un Nodo*



La fase successiva è quella di selezionare, per ogni dimensione del pacchetto, il minimo RTT misurato, chiamato SORTT (Shortest Observed RTT), per poi identificare una retta di interpolazione, graficata nell'immagine seguente:

Figura 5 - *SORTT del Nodo precedente*



Questo risultato interpreta pienamente l'approssimazione eseguita sul valore di RTT (Round Trip Time), che risulta essere proporzionale alla

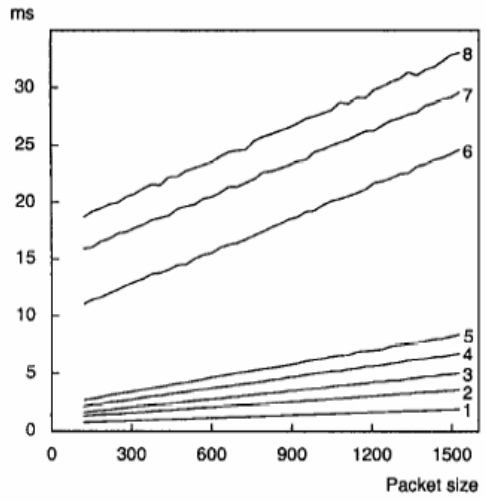
dimensione del pacchetto, generando una retta di pendenza pari all'inverso della Capacità e di intercetta pari al doppio del Ritardo del Canale (τ). Riassumendo ho che:

$$RTT = \frac{L}{C} + 2\tau$$

Identificando quindi una retta di interpolazione corretta per i nostri dati, si possono conoscere in modo abbastanza rapido i valori prestazionali appena elencati.

Questo lavoro è stato fatto per un solo link nell'arco dell'intero percorso, ma se il lavoro è iterato anche per gli altri Hops, si ottiene un grafico simile al seguente:

Figura 6 – *SORTT di tutti gli Hops*



Dopo aver prodotto questi risultati, non rimane che ricavare i parametri dei singoli canali trasmissivi, che possono essere calcolati per sottrazione: visto che gli elementi prestazionali sono cumulativi, i valori

dell'Hop numero N sono il risultato della differenza tra le misurazioni fatte fino ad N e fino a N-1.

Per esempio, per trovare il tempo di latenza del sesto canale, si sottrae il valore dell'intercetta della linea 5 a quello della linea 6, che nel grafico di figura sei equivale a fare 9.88ms – 2.22ms = 7.66ms. Il risultato appena trovato è uguale al doppio della latenza del sesto canale di trasmissione, che risulta quindi essere pari a 3.83ms.

Per quanto riguarda il calcolo della Capacità il discorso è analogo: bisogna sottrarre alla pendenza della linea sei la pendenza della linea cinque, effettuando poi l'inverso del risultato della sottrazione. Nell'esempio di figura sei, i conti sono i seguenti: 9.61μs/B – 4.02 μs/B = 5.6 μs/B, il cui inverso è pari a 1.43MB/s.

Per quanto riguarda il trattamento dei dati si utilizza solitamente il metodo dei minimi quadrati per individuare le caratteristiche della retta interpolatrice, applicando a monte un filtraggio di dimensione variabile, in base al tipo di accuratezza e di tolleranza decisa.

Tratterò ora i problemi che si possono riscontrare nelle misurazioni finali, dovuti alle semplificazioni fatte inizialmente sulla formula relativa al Round Trip Time.

Per quanto riguarda la latenza (τ), le approssimazioni legate al considerare la dimensione dei pacchetti d'errore nulla, così come il

tempo di inoltro, non creano grandi differenze nel risultato finale. Un effetto che potrebbe sentirsi maggiormente sull'intera misurazione è la possibile diversità dei due ritardi di canale, dovuta al diverso percorso che il pacchetto di ritorno potrebbe effettuare rispetto a quello d'andata, anche se questo fenomeno viola l'ipotesi iniziale relativa alla conservazione del percorso. Questi fattori in generale non influenzano molto la stima più importante, relativa cioè alla Capacità, in quanto spesso sono indipendenti dalla dimensione *L* e quindi non variano per piccoli o grandi pacchetti.

Semplificazioni invece che potrebbero interessare la Capacità risultante sono per esempio i link multi-canale, gli OC-3 in particolare, che vengono misurati non globalmente ma solo singolarmente e quindi la stima produce un valore non generato dal parallelismo degli n mezzi trasmissivi, ma da un solo canale.

Un secondo fenomeno che genera un valore molto differente dalla realtà è dovuto alla frammentazione, in quanto se il pacchetto spedito dal lato mittente incontra un nodo che non è in grado di inoltrarlo se non suddividendolo in ulteriori pacchetti, una volta che questi giungono al destinatario, egli genera l'errore ICMP (Internet Message Control Protocol) solitamente alla ricezione del primo frammento, causando una sottostima del Round Trip Time. Questo problema non si può risolvere semplicemente impostando la flag dedita alla frammentazione come

falsa, impedendo quindi ad eventuali nodi di suddividerla, in quanto essi bloccherebbero comunque il flusso di dati inviando un errore, l'ICMP "`Destination Unreachable (Fragmentation Required)`". La sola soluzione possibile sarebbe quella di mandare dei pacchetti "sentinella" all'inizio delle misurazioni, impostando come non possibile la frammentazione ed attendendo eventuali messaggi ICMP (Internet Message Control Protocol) d'errore. Se non si ricevono significa che tutti i nodi che i pacchetti futuri dovranno attraversare sono in grado di inoltrare i dati senza richiedere la frammentazione.

Un ultimo problema che rimane in sospeso è l'utilizzo di alcuni nodi interni al percorso che attuano del routing dinamico, che tende generalmente a variare il tragitto d'andata rispetto a quello di ritorno.

La soluzione più semplice ed immediata è quella di mandare ancora una volta una serie di pacchetti "sentinella", dediti stavolta a stabilire quale sarà il percorso da considerare per l'intera fase di test. Nel caso in cui, successivamente, alcuni pacchetti effettuino tragitti differenti, essi saranno esclusi dal conteggio finale.

Un ulteriore tematica che deve essere affrontata è relativa al tipo di approccio con cui le misure possono essere condotte, scegliendo cioè tra la possibilità di effettuare un basso numero di rilevamenti con un *range* elevato di dimensioni dei pacchetti o viceversa, cioè sviluppare molti

tentativi su poche tipologie di pacchetti. Alcuni studiosi hanno affrontato questa problematica e, confrontando i diversi risultati, si può affermare che le stime prodotte differiscono di molto poco, circa dello 0.3% in media.

Analizzando le misurazioni offerte da diversi ricercatori si può osservare che, a volte, si possono trovare misurazioni molto precise ma non accurate. La causa più probabile, potrebbe essere il non considerare il tempo necessario all'ultimo router del percorso per accorgersi che il TTL (Time to Live) è diventato nullo e generare di conseguenza il messaggio ICMP (Internet Message Control Protocol) d'errore.

Statisticamente, questi tempi si aggirano intorno a $100\mu s$,e tendono a traslare tutte le rette verso l'alto, inficiando le misurazioni.

Dopo aver analizzato questa prima tecnica di stima della Capacità di ogni singolo Hop attraverso lo studio di percorsi che si allungano linearmente, si può giungere a conclusioni interessanti. La prima riguarda il calcolo del tempo di latenza di un collegamento, che risulta essere relativamente semplice da stabilire in modo accurato, in quanto generalmente l'ordine di grandezza che lo caratterizza è molto maggiore rispetto a quello dell'errore, che fa si che le misure siano prodotte con un margine d'errore veramente valido. D'altro canto, però, la Capacità è difficilmente analizzabile, in quanto solitamente la pendenza delle rette

d’interpolazione è molto bassa e quindi la presenza di disturbi è molto più influente. Una possibile soluzione sarebbe quella di aumentare la dimensione dei pacchetti, aumentando però le probabilità di incontrare problematiche relative alla frammentazione.

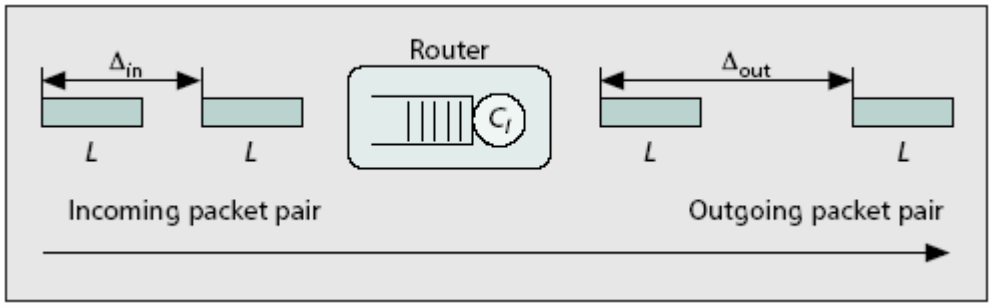
Molti ricercatori, infine, consigliano di effettuare un elevato numero di prove consecutivamente, in modo che i tempi di coda *q* tendano a decrescere continuamente, producendo quindi delle misurazioni più vicine alle approssimazioni teoriche relative al Round Trip Time. Questa soluzione è sicuramente migliore rispetto ad eventuali tentavi di stima dei tempi di coda, in quanto essi variano in maniera troppo rapida rispetto ad ogni previsione.

6. PACKET PAIR DISPERSION

Questo secondo metodo è stato ideato da una collaborazione tra Jacobson, Keshav e Bolot e si pone come obbiettivo la stima della Capacità End-to-End di una rete.

L'idea su cui si basa l'intero algoritmo è quella di trasmettere una coppia di pacchetti, adiacenti uno all'altro e di uguali dimensioni, e misurare il tempo che passa tra la ricezione del primo bit di ognuno dei due pacchetti dal lato ricevente.

Figura 7 - ***Funzionamento PPD tra due router intermedi***



Il fatto di misurare il tempo che intercorre tra le ricezioni dei primi bit dei due pacchetti rende la tecnica non molto robusta nel caso in cui alcuni router intermedi dovessero trovarsi costretti a frammentare i pacchetti per inoltrarli. In quest'ottica si preferisce misurare le complete ricezioni.

Da queste considerazioni si può quindi definire la Dispersione (Δ) come la distanza temporale tra la ricezione dell'ultimo bit del primo pacchetto e la ricezione dell'ultimo bit del secondo pacchetto.

Si può inoltre affermare che, secondo questa logica, alla fine dell'intero percorso avrò che:

$$\Delta_R = \max_{i=0...H}(\frac{L}{C_i}) = \frac{L}{\min_{i=0...H}(C_i)} = \frac{L}{C}$$

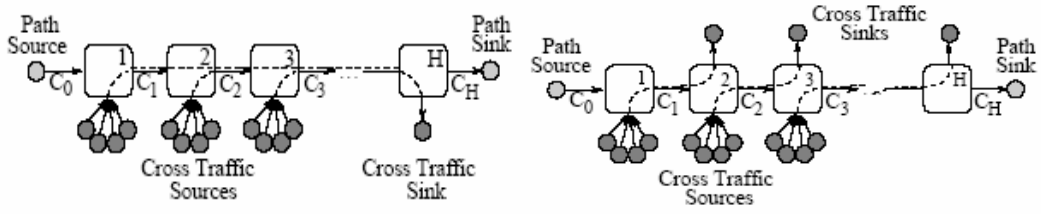
Misurando il ritardo totale all'ultimo host e conoscendo la dimensione *L* dei pacchetti, posso risalire all'intera Capacità *C* del percorso.

In generale, quando non è presente molto traffico, questa misura di Capacità tende alla Banda Disponibile. Quando invece l'utilizzo della rete all'interno del percorso scelto è alto, si risente maggiormente dei tempi di coda di ogni inoltro, generando tipicamente distribuzioni multimodali.

Solitamente, per analizzare casi di studio reali, si utilizzano delle strutture di reti "ad hoc", realizzate quindi in modo da simulare ogni possibile caso.

Un esempio di alcune situazioni ricostruite sono le seguenti:

Figura 8 – *Simulazione di Reti Reali*



Nel primo caso. sulla sinistra nel disegno, il traffico generato come disturbo parte da nodi successivi a quello trasmissivo e tende generalmente a percorrere lo stesso percorso, in quanto è destinato al ricevitore di questo algoritmo.

Nel secondo caso, invece, ho disturbi che impegnano i nodi intermedi verso l'inoltro a destinatari diversi da quello indicato dai pacchetti di stima.

I due casi tendono, nel complesso, a simulare ogni possibile situazione reale e vengono impiegati nelle fasi di testing dei software che implementano questa tecnica.

In condizioni di rete carica, la distribuzione si presenta come *multimodale*, in quanto sono riscontrabili due diverse possibilità d'errore:

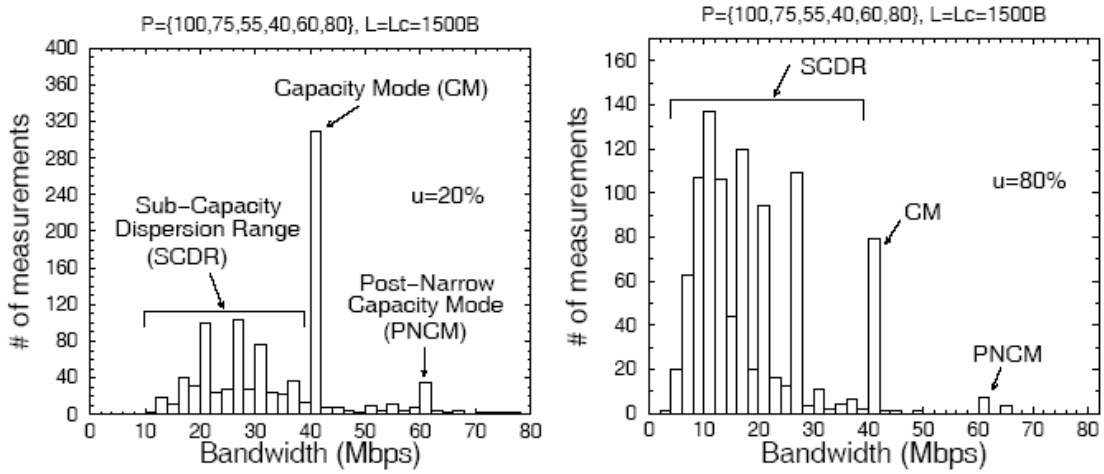
- *Sub-Capacity Dispersion Rate (SCDR)*
- *Post-Narrow Capacity Modes (PNCMs)*

Nel primo caso la coppia di pacchetti viene rallentata dal traffico intermedio, e generalmente il secondo più del primo. In questo modo la stima finale risente di un tempo Δ_R maggiore del previsto, presentando poi una stima inferiore rispetto al valore reale.

Nel secondo caso, invece, si ha una sovrastima della capacità End-to-End, in quanto il secondo pacchetto è ritardato molto meno rispetto al primo, il che provoca una misura di tempo inferiore.

I seguenti due istogrammi presentano i risultati di una misurazione, che evidenzia la presenza di una stima corretta, contornata però dai due casi appena descritti:

Figura 9 - *Distribuzione Multimodale nei Casi Reali*



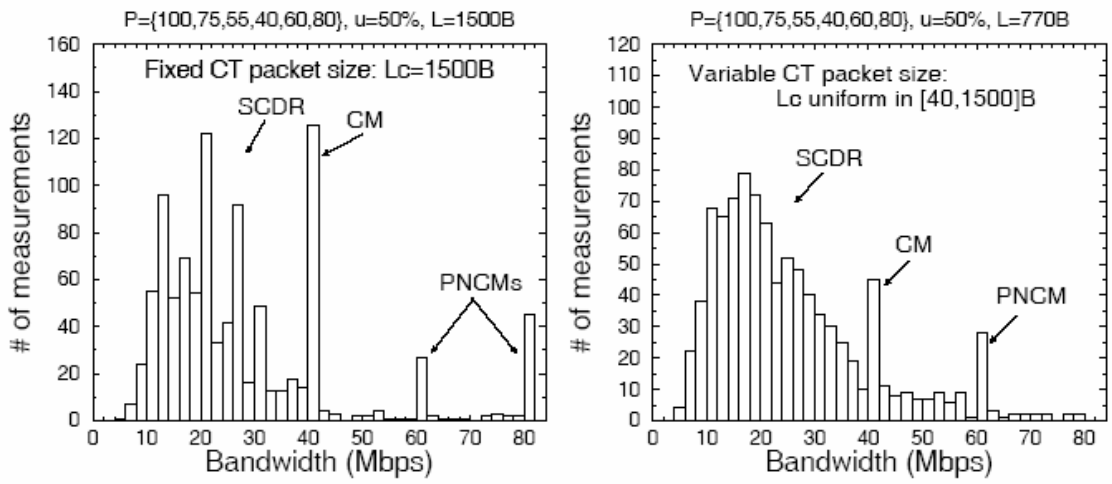
Questi due grafici rappresentano lo studio dello stesso percorso di rete in due casi differenti: il primo a sinistra riporta i valori di Capacità a rete scarica, mentre il secondo a rete carica.

Come si può facilmente notare nel secondo caso, quello che sicuramente si avvicina maggiormente ad una condizione reale, gli elementi di disturbo provocano una sottostima della capacità del percorso, in quanto la probabilità di riscontrare un ritardo superiore del normale tra la coppia di pacchetti è elevata.

Un aspetto interessante da trattare rimane l'approccio da seguire nei confronti della tipologia di pacchetti impiegati, se conviene cioè utilizzare sempre lo stessa dimensione o farla variare nell'arco delle misurazioni.

I diversi risultati sono riportati nei due istogrammi seguenti:

Figura 10 - *Dimensione Pacchetti Fissata o Variabile*



Nel primo caso si è utilizzata una misura fissata, pari a 1500 byte, mentre nella seconda la dimensione della coppia di pacchetti variava, all'interno di un intervallo tra 40 e 1500 byte.

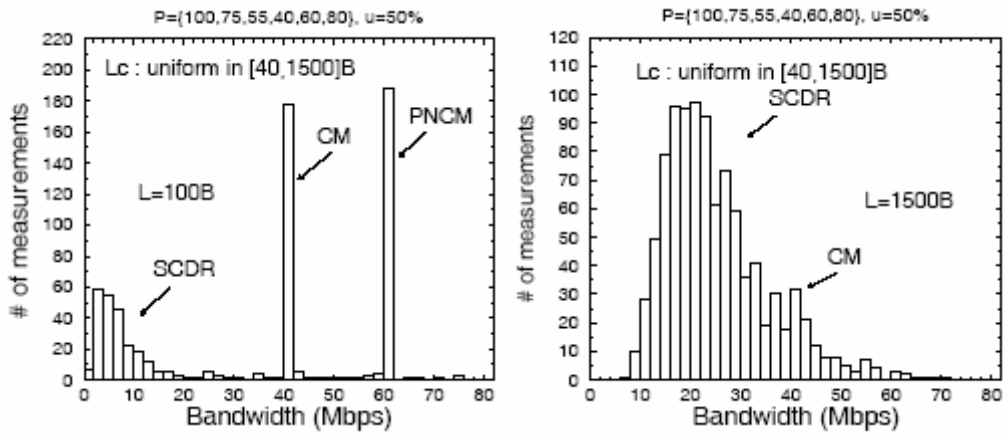
Dai risultati ottenuti si può notare che i valori errati di stima sono meglio distinguibili nel primo caso, a pacchetto fissato, e quindi sono meglio percepibili come sbagliati in una fase successiva di filtraggio dei dati.

Dopo aver realizzato che è meglio impiegare una sola dimensione per l'intera misurazione, rimane da considerare quale sia il valore di *L* più adatto e più robusto agli errori.

Intuitivamente si può pensare che sia meglio utilizzare la dimensione massima che il protocollo supporta senza dover frammentare, quindi generalmente 1500 byte, in quanto in questi casi i tempi di ritardo dovuti alle code e i difetti causati dalle granularità degli orologi si sentono meno sul risultato finale. Questa idea, però, non tiene conto degli effetti causati dal traffico di disturbo che possono modificare molto le aspettative iniziali.

I grafici seguenti mostrano un confronto dei diversi risultati ottenibili sfruttando nel primo caso pacchetti di dimensioni pari a 100 byte, mentre nel secondo pari a MTU (Maximum Transfer Unit) Ethernet, cioè 1500 byte:

Figura 11 - Confronto tra Dimensioni Fissate di Pacchetti



Il primo caso genera valori che tendono nel complesso a sovrastimare la capacità del percorso in esame, mentre il secondo grafico mostra che dimensioni alte dei pacchetti impiegati tendono maggiormente a sottostimare. Inoltre si può aggiungere che la dispersione dei risultati aumenta proporzionalmente all'aumentare della dimensione dei pacchetti.

Si può quindi concludere che la soluzione migliore da adottare è di impiegare come dimensione delle coppie di pacchetti valori intermedi, come per esempio 800 byte, che sono robusti al traffico intermedio dei vari canali e ben percepibili da qualsiasi destinatario.

7. PACKET TRIPLET DISPERSION

Questa tecnica propone una modifica all'algoritmo utilizzato precedentemente, sfruttando anziché coppie di pacchetti, delle triplette. La tecnica è stata proposta e studiata da ZiXuan, Sung, Peng e Jie, ricercatori presso l'Asia Pacific Advanced Network in Corea.

Come introdotto, in questo algoritmo si utilizzano tre pacchetti, di ugual dimensione, spediti in successione. Stavolta dal lato ricevente si misurano due dispersioni, tra l'arrivo del primo pacchetto e del secondo (Δ^1) e tra il secondo e il terzo(Δ^2): insieme esse vengono chiamate *Dispersion Pair*.

Analizzando queste misure si ricavano due stime per ogni tripletta inviata, calcolando quindi $C^1 = \frac{L}{\Delta^1}$ e $C^2 = \frac{L}{\Delta^2}$. Dalla correlazione tra i due valori ricavati si può comprendere la bontà delle misurazioni e l'eventuale impatto del rumore causato da altro traffico.

Teoricamente la differenza tra i due Δ^i dovrebbe essere nulla, mentre, a causa del traffico e delle conseguenti code d'attesa si ha che:

$$\delta_j = \Delta^1_j - \Delta^2_j \neq 0$$

In questo senso, si può applicare una semplice tecnica di filtraggio dati a monte dell'elaborazione del risultato, eliminando dal conteggio finale tutti i valori i cui δ superano una certa soglia prefissata, α, scelta in base

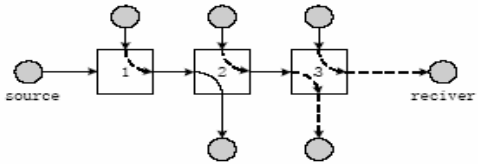
all'accuratezza desiderata (per esempio un valore $\alpha = 0.5Mbps$ può essere sensato).

Questa tecnica generalmente funziona bene, tranne nel caso in cui il rumore risentito dai due ritardi della Dispersion Pair siano uguali, situazione che nella realtà si riscontra molto raramente e che viene definita con la sigla *EN*, acronimo di *Equivalent Noise*.

Di seguito vengono riportati i risultati di un confronto tra la tecnica appena presentata e l'algoritmo *Packet Pair Dispersion*, descritto al capitolo precedente.

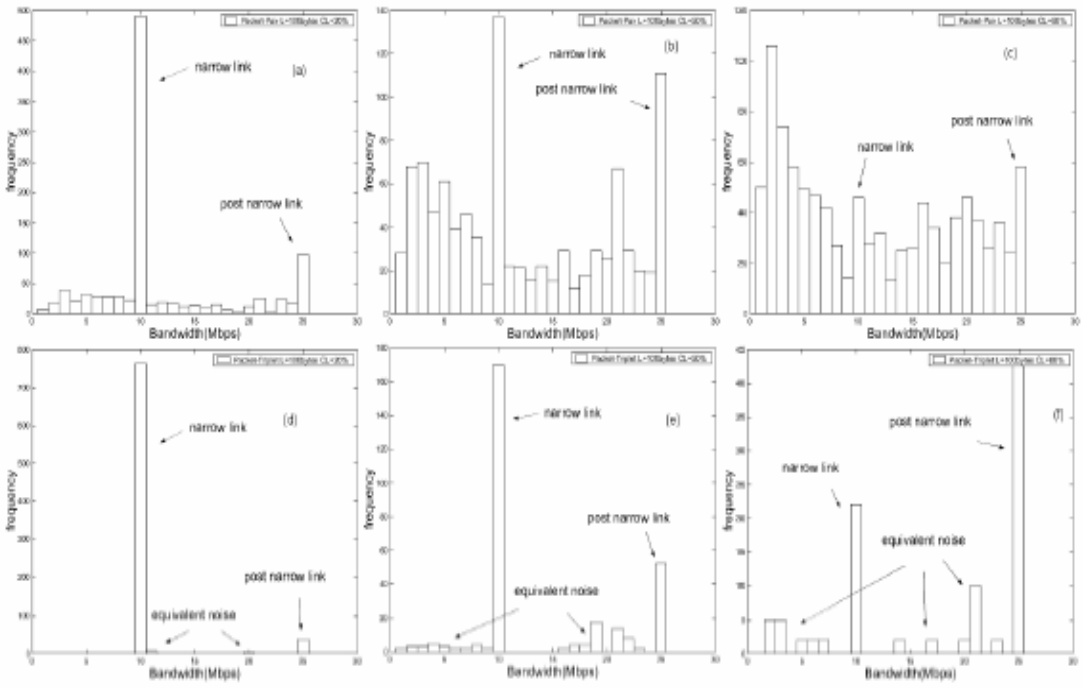
Per affrontare questa analisi è stata utilizzata una simulazione di rete, che corrisponde alla struttura seguente:

Figura 12 - *Struttura della Rete Utilizzata*



I risultati riscontrati sono riassunti nei seguenti istogrammi e sono relativi a circa 1000 prove, effettuate con una dimensione dei pacchetti *L* pari a 100 byte:

Figura 13 - ***Packet Pair vs. Packet Triplet (L=100byte)***



I tre istogrammi della prima riga si riferiscono alle stime ricavate tramite il *Packet Pair Dispersion*, mentre la seconda riga è relativa al *Packet Triplet Dispersion*.

La prima colonna riguarda una situazione in cui l'utilizzo medio della rete è circa pari al 20%, la seconda colonna circa al 50% mentre l'ultima al 80%: queste tre diverse condizioni mostrano le reazioni dei due diversi algoritmi al differente traffico imposto dalla rete.

Per quanto riguarda il *Packet Triplet* è stato utilizzato come valore di α quanto indicato precedentemente, quindi $0.5Mbps$ ed analizzando i risultati proposti dai grafici si può notare che il primo metodo di stima

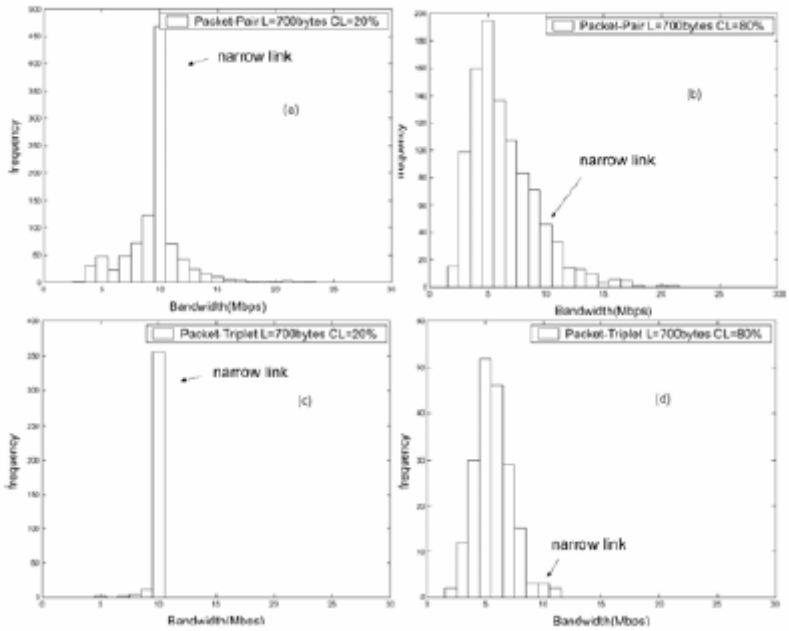
studiato supporta difficilmente carichi elevati di traffico, mentre il secondo in questo senso è ben più robusto.

È stata effettuato anche un altro paragone, sempre con la tecnica studiata al capitolo precedente, utilizzando stavolta una dimensione dei pacchetti più larga, pari cioè a 700 byte.

La struttura di rete impiegata per questo secondo confronto è identica alla seconda, impiegando traffico marginale di pacchetti con dimensioni variabili tra i 40 e i 1500 byte.

La seguente figura riassume i dati ottenuti, disponendo i grafici come nel caso precedente, dove sulle due righe ho le diverse tecniche e sulle due colonne ho i diversi carichi di traffico, pari a 20% e 80%:

Figura 14 - Packet Pair vs. Packet Triplet (L=700byte)



Anche in queste condizioni a basso traffico di disturbo entrambe le tecniche generano stime molto vicine alla realtà, mentre stavolta ad alti livelli di utilizzo di rete neanche l'algoritmo *Packet Triplet* fornisce valori corretti, in quanto la distribuzione dei dati è molto ampia e non permette di fornire un risultato preciso ed accurato.

Da queste osservazioni è giusto quindi osservare che la dimensione dei pacchetti risulta essere fondamentale per preservare stime corrette.

8. PACKET TRAIN DISPERSION

L'ultima variante del metodo *Packet Pair Dispersion* è *Packet Train Dispersion*, proposta dagli stessi autori di quella originale.

In questo nuovo algoritmo dal lato mittente non vengono spediti due pacchetti, ma un numero N superiore, che determina la lunghezza del treno.

Il destinatario misura la dispersione totale $\Delta(N)$ dei pacchetti, dal primo all'ultimo, in modo da calcolare poi la Capacità dell'intero percorso nel seguente modo:

$$C = \frac{(N-1)L}{\Delta(N)}$$

La tecnica analizzata nel capitolo precedente non deve essere considerata come un caso particolare di quella appena descritta, in quanto questo algoritmo produce una sola stima per ogni treno inviato, mentre *Packet Triplet* due ogni invio della tripletta di pacchetti, con la conseguente possibilità di correlare i dati ricavati.

Nel caso in cui non c'è molto traffico nel percorso in esame, il valore stimato coincide con la Capacità reale, come per l'algoritmo *Packet Pair*.

Quando si vogliono analizzare le prestazioni relative ad un percorso contenente multi-canali, è necessario utilizzare Packet Train come metodo di studio della Capacità, in quanto è in grado di determinare un valore corretto per quella particolare tipologia di canali trasmissivi. Un

mezzo di quel tipo infatti, formato da *k* canali paralleli e con una capacità totale pari a *C*, è misurabile da un treno di lunghezza $N = k + 1$.

Di seguito verranno analizzate le misurazioni prodotte da questa tecnica, valutando gli effetti delle diverse lunghezze del treno sulla stima finale della capacità.

L'esperimento è stato condotto attraverso un percorso Internet che collegava una stazione mittente a jhana (San Diego, CA) e un destinatario a ren (Newark, DE) durante Giugno 2000 e l'utilizzo medio riscontrato era alto, pari a circa 80%.

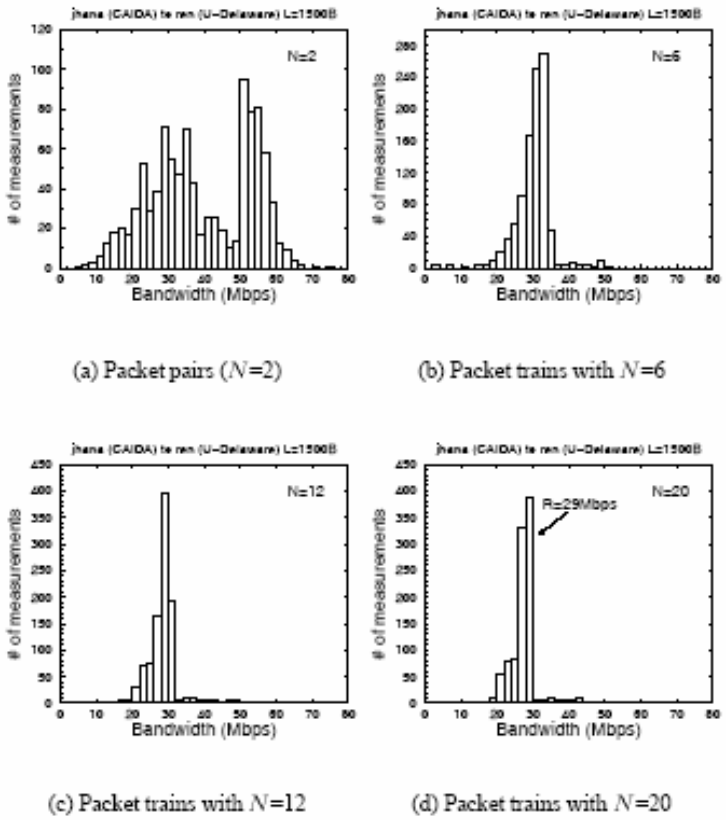
Il percorso era formato da link con le seguenti Capacità: 100, 75, 55, 40, 60, 80 Mbps.

Dalla figura numero 15 si può notare che, come N cresce, la stima della Capacità media e fenomeni quali il PCNMs tendono a scomparire, mentre prendono il sopravvento valori appartenenti alla famiglia SCDR.

La ragione di questo fenomeno è che all'aumentare della lunghezza *N* del treno, è più facile che molti pacchetti incontrino del ritardo causato dalle code e che questo ritardo aumenti nel tempo.

Si può quindi affermare che, se si vuole evitare al massimo l'influenza di ulteriori ritardi rumorosi, si dive ricorrere ad un treno di lunghezza pari a due, quindi ad una coppia di pacchetti, *Packet Pair Dispersion*.

Figura 15 - *Packet Train in funzione di N*



(a) Packet pairs (N=2) (b) Packet trains with N=6

(c) Packet trains with N=12 (d) Packet trains with N=20

Una seconda osservazione che si può notare è che più la lunghezza del treno aumenta, più la distribuzione dei dati diventa unimodale.

Unendo quindi le due precedenti affermazioni si può prevedere che per *N* molto alti, i valori stimati sono molto pochi e molto differenti dalla realtà, in quanto tendono a rispecchiare solo l'influenza del traffico di rete.

Il centro delle stime unimodali, quindi per lunghezze del treno alte, è indipendente da N e viene chiamato *ADR* (*Asymptotic Dispersion Rate*).

Questo parametro dipende unicamente dalle Capacità intermedie e dallo stato di utilizzo di questi link, fornendo un'idea dei rallentamenti possibili derivanti dal traffico sulla rete.

Questa tecnica, come le due precedenti, prevede l'impiego di un host mittente ed uno destinatario in ascolto, ma può essere implementata anche in modalità differenti. Un esempio potrebbe essere l'utilizzo di messaggi d'errore derivanti dal protocollo ICMP (Internet Message Control Protocol) o di pacchetti speciali di chiusura di sessione, quali il TCP-Fin o TCP-Rst.

In queste diverse realizzazioni, però, le misurazioni potrebbero risentire maggiormente di eventuali code lungo il percorso, in quanto deve essere attraversato due volte nelle due diverse direzioni, e fornire quindi una stima finale non molto accurata.

Per questo motivo, anche se lo svincolarsi dal nodo ricevente rende l'impiego di questi algoritmi generalmente più flessibile, i software che implementano queste tecniche di stima prevedono comunque l'utilizzo di entrambe i nodi previsti, lato mittente e destinatario, posti ai due diversi capi del percorso da analizzare.

9. TRAINS OF PACKET PAIRS

Questo metodo è stato inventato da Melander e propone una stima della Banda Disponibile su un percorso voluto.

L'algoritmo di questa tecnica prevede l'invio di molte coppie di pacchetti dal mittente al destinatario, che nel loro insieme formano un treno, a velocità via via maggiori.

L'idea fondamentale si basa sul confronto tra le velocità di invio e di ricezione di ogni coppia trasmessa.

Dal lato mittente si spedisce un paio di pacchetti ad una velocità fissata, determinata dal calcolo della dispersione Δ_S, tale che $R_0 = \frac{L}{\Delta_S}$.

Successivamente, poi, il destinatario calcola le prestazioni offerte da quella coppia di pacchetti, misurando $R_M = \frac{L}{\Delta_M}$.

Infine si confrontano i valori R_0 ed R_M e si valutano i seguenti due casi:

- Se $R_0 \cong R_M$, significa che i due pacchetti sono stati ricevuti con la stessa dispersione con cui sono stati mandati e quindi si può supporre che la velocità di invio della coppia sia inferiore alla Banda Disponibile di quel percorso nel momento di misurazione, ossia che $R_0 < AB$. Si ripetono quindi le misurazioni con una velocità di invio dati maggiore.
- Se $R_0 > R_M$, significa che la velocità di trasmissione dati è maggiore rispetto a quanto la rete possa offrirmi in quell'istante, in altre parole

posso affermare che $R_0 > AB$ e che ho quindi superato il limite di banda disponibile.

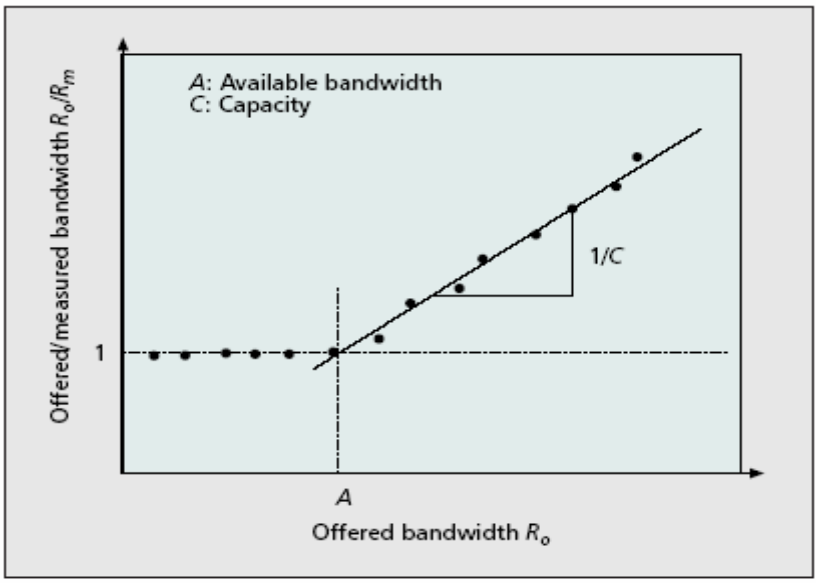
Il procedimento corretto è quindi quello di iniziare con una velocità di invio dati relativamente bassa, per poi incrementarla di un valore scelto in base alla risoluzione finale desiderata.

Nel momento in cui si ha il passaggio da una situazione descritta al primo punto verso una del secondo tipo, il valore R_0 è una stima della Banda Disponibile della rete utilizzata in quell'analisi.

Con questo metodo si può anche analizzare la capacità del percorso, estendendo le misurazioni anche dopo il punto di transizione tra i due stati diversi, come mostra la Figura numero 16.

In questo grafico è rappresentato il rapporto tra le due velocità rilevate sul destinatario e mittente in funzione della trasmissione imposta all'inizio:

Figura 16 - *Rilevamento AB con TOPP*



Nell'esempio illustrato dalla Figura 16 si è considerato un percorso molto semplice, costituito da un solo link, con Capacità *C*, Banda Disponibile *A* e con una Banda di Utilizzo $R_C = C - A$.

L'algoritmo *Trains Of Packet Pairs* invia coppie di pacchetti con una velocità crescente pari a R_0.

Finché non arrivo al momento del cambiamento, quando cioè $R_0 = A$, si può facilmente constatare che $R_M = R_0$. Nel momento di transizione, però, ho che:

$$\textit{Se } R_0 = A \Rightarrow R_M = \frac{R_0}{R_0 + R_C} C = \frac{A}{A + R_C} C = \frac{A}{A + C - A} C = A \Rightarrow R_M = A$$

Quando R_0 diventa maggiore di *A*, la velocità rilevata dal destinatario è pari a quella d'invio rapportata all'utilizzo del canale:

$$R_M = \frac{R_0}{R_0 + R_C} C \quad \text{cioè} \quad \frac{R_0}{R_M} = \frac{R_0 + R_C}{C}$$

Dalla seconda formula, si può inoltre ricavare il valore della Capacità del percorso, identificando la pendenza della retta data dal rapporto tra le velocità in funzione della trasmissione impostata dal mittente, come rappresentato nella Figura 16.

10. PACKET TAILGATING TECHNIQUE

Questo metodo, realizzato da Kevin Lai e Mary Baker presso il Department of Computer Science all'Università di Stanford, misura la capacità di un percorso analizzato.

Secondo gli autori di questo algoritmo, le tipologie analizzate precedentemente presentano alcuni difetti che le rendono talvolta inutilizzabili. Nello specifico, la prima soluzione alle problematiche prestazionale descritte in questo trattato ed in generale tutti i modelli che si appoggiano ad un solo pacchetto (*"One Packet Model"*), impongono la presenza di soli Router in modalità Store-Forward, il che nel mondo di Internet non crea problemi, link a canale singolo e che ci sia poco traffico o che comunque ci sia un basso ritardo causato dalle code.

La seconda tipologia affrontata, "Packet Pair/Train Model", suppone che i due pacchetti stiano in coda insieme e vicini, il che non è sempre vero, e si è visto quali errori possono nascere. Inoltre queste tecniche non producono stime molto affidabili in presenza di multi-canali e considerano che la schedulazione adottata dai router sia di tipo FIFO (First In First Out), mentre talvolta se ne possono incontrare alcuni che seguono logiche diverse, quali SJF (*"Shortest Job First"*), che modificando i tempi di coda, generano valori di stima incorretti.

Secondo gli autori, in base ai risultati sperimentali registrati, quest'ultima tecnica tende a generare stime più affidabili degli algoritmi menzionati precedentemente, in quanto risente molto poco delle problematiche appena descritte.

Il funzionamento dell'algoritmo *"Packet Tailgating"* si può suddividere in due fasi:

1. Fase Sigma, dove si misurano le caratteristiche di tutto il percorso;
2. Fase TailGating, dove si risale ai parametri di ogni link.

Nella prima fase si fornisce una stima della Capacità End-to-End, sfruttando algoritmi simili a quelli analizzati nei capitoli precedenti e solitamente realizzati impiegando singoli pacchetti di dimensioni differenti ed applicando una regressione lineare sui tempi minimi di trasmissione-ricezione rilevati, con una tolleranza bassissima, dell'ordine dell'unità percentuale. A differenza dall'algoritmo *Variable Packet Size*, in questo caso, non si applica il metodo descritto per tutti i link del percorso, ma solo tra il mittente ed il destinatario ai due diversi capi del percorso.

La seconda fase di *Packet Tailgating* produce invece una stima dei parametri di ogni singolo tratto del percorso generale. Il funzionamento è basato sull'invio di due pacchetti, eliminando ogni coda possibile attraverso il primo, ed un secondo che gli rimane accodato dietro fino ad un host scelto.

Nello specifico, si manda un primo pacchetto di dimensioni il più grandi possibili, detto Tail Gated, con il campo IP TTL (Internet Protocol Time To Live) impostato ad un valore desiderato, e successivamente un pacchetto molto piccolo, detto Tail Gater, con tempo di invio quindi nettamente inferiore al precedente.

Il pacchetto più "leggero" segue continuamente quello più "pesante", fin quando quest'ultimo non viene eliminato da un server, in quanto il TTL (Time To Live) raggiunge lo zero.

Dal momento in cui il pacchetto più grande sparisce, quello più piccolo può continuare il percorso, assumendo che non incontri altri ritardi.

Analogamente alla Fase Sigma, nella fase di raccolta dati si ricerca il minor tempo misurato per ogni link analizzato, applicando poi una regressione lineare.

Packet Tailgating risulta essere una tecnica più veloce e robusta rispetto alle precedenti, in quanto non richiede di effettuare un elevato numero di invii per ogni link, rendendo anche l'algoritmo meno intrusivo.

In teoria è anche in grado di determinare link multi-canale, basta concepire non un solo Tail Gater ogni Tail Gated, ma più pacchetti piccoli ogni grande, in modo che seguano i diversi canali a disposizione. Supponendo poi che i singoli mezzi trasmissivi disposti parallelamente abbiano tutte le stesse caratteristiche, basta indagare sulla singola

capacità ed analizzare il numero di canali per determinare la prestazione globale dell'intero tratto.

A differenza di altri algoritmi, *Packet Tailgating*, non utilizza messaggi ICMP "`Time-Exceeded`" per misurare il tempo di invio dati e questo rende la tecnica generalmente ancora più veloce e flessibile, in quanto molti router assegnano all'invio di pacchetti come quelli appena indicati priorità molto basse, che dal punto di vista temporale significa un ulteriore accumulo di ritardi.

Anche con questa tecnica però possono nascere alcuni problemi, soprattutto quando nel percorso ho in sequenza un canale molto lento e subito dopo uno molto veloce. In questo caso infatti potrebbe accadere che il pacchetto più grande sia già spedito attraverso il canale veloce, mentre quello più piccolo è ancora in fase di trasmissione nel link precedente.

Questo tipo di problema, utilizzando le due dimensioni estreme, ossia di 40 byte e 1500 byte, si riscontra solo quando il rapporto tra la capacità del canale veloce rispetto a quello lento supera 1500/40 = 37,5.

Una seconda problematica che va considerata, è la difficoltà di produrre buone stime nei confronti di link molto lontani, in quanto l'accumulo d'errore aumenta proporzionalmente al numero di tratti che sto affrontando.

Quest'ultima tecnica risulta essere quindi frutto dell'esperienza maturata attraverso tutte le precedenti metodologie realizzate, e tende a produrre stime più realistiche, in maniera più veloce, robusta e soprattutto meno intrusiva.

11. TOOLS DI ANALISI PRESTAZIONALE

In questo paragrafo verranno trattati i più diffusi software di analisi prestazionale, presentandoli in funzione del parametro stimato.

La prima tipologia riguarda la misurazione della Capacità dei singoli Hop, realizzata attraverso la tecnica *Variable Packet Size* (VPS).

Pathchar è stato il primo tool ad implementare quell'algoritmo, scritto dallo stesso Van Jacobson e rilasciato nel 1997. Il codice sorgente non è disponibile pubblicamente.

Un altro software che sfrutta lo stesso metodo è **Clink**, di natura Open-Source e scritto inizialmente solo per ambiente Linux. *Clink*, a differenza di *PathChar*, utilizza delle tecniche di analisi dei dati più raffinate, tra cui la possibilità di riconoscere eventuali instabilità da parte di Routers appartenenti al percorso in esame e l'eventuale filtraggio delle relative misurazioni.

L'ultimo tool progettato sulla base del VPS è **Pchar**, anch'esso Open-Source. La particolarità di questo programma è data dall'uso di una libreria, *Libpcap*, che migliora la precisione delle misurazioni, in quanto le operazioni di Time-Stamping vengono eseguite a livello Kernel. *Pchar,* inoltre, è dotato di tre differenti algoritmi di regressione lineare del SORTT, fornendo ulteriore accuratezza.

La seconda classe di Tools è relativa alla stima della Capacità all'interno di un percorso completo, formato da più Hop.

Bprobe è un software che implementa l'algoritmo Packet Pairs, con la possibilità di analizzare ed eliminare in fase di filtraggio dati i pacchetti affetti da ritardo. Inoltre *Bprobe* mette a disposizione dell'utente la scelta della dimensione delle coppie di pacchetti, in modo da testare ogni possibile combinazione. *Bprobe* infine non richiede la partecipazione attiva dell'Host destinatario, in quanto invia pacchetti di tipo ICMP-Echo e attende la ricezione automatica di ICMP-Replies. Questa soluzione però a volte porta a cattive misurazioni, in quanto alcuni Router applicano basse priorità a processi di risposta ICMP, in modo da prevenire tipologie di attacchi DOS.

Nettimer è multifunzionale, in quanto ha la possibilità di funzionare secondo *VPS* o *Packet Pairs*, anche se viene impiegato maggiormente per questa seconda classe di algoritmi. Questo software implementa una tecnica statistica molto sofisticata, chiamata *"Kernel Density Estimation"*, in grado di identificare la moda dominante all'interno di una distribuzione.

Pathrate è un tool che colleziona molte misurazioni di coppie di pacchetti, utilizzando varie dimensioni. Analizzando la distribuzione dei dati si riscontrano tipicamente varie mode, una delle quali rappresenta il valore della Capacità del percorso. Successivamente *Pathrate* utilizza lunghi treni di pacchetti per determinare l'ADR e, considerando che

questo valore è sempre inferiore della Capacità reale, per confronto con i risultati ottenuti in una prima fase con le coppie di pacchetti, si risale alla stima della Capacità.

Sprobe è un software di analisi della Capacità molto veloce, che opera solo dal lato mittente del percorso, sfruttando coppie di pacchetti del tipo TCP-SYN. Il destinatario tipicamente risponde a queste interrogazioni con l'invio nel senso opposto di pacchetti del tipo TCP-RST, permettendo cosi la misurazione del tempo di ritorno.

Una terza tipologia di Tools riguarda la stima della Banda Disponibile. Il primo software implementato è stato **Cprobe**, in grado di misurare la dispersione di un treno di pacchetti di otto dimensioni diverse. Altri programmi, molto simili a *Cprobe*, sono **IGI** e **pathChirp**.

Di seguito riporto una tabella riassuntiva dei Software di Benchmark appena analizzati, dando cosi la possibilità di eventuali approfondimenti e ricerche:

Tabella 1- ***Software di analisi prestazionale***

Nome	Autore	Parametro Stimato	Metodologia
PathChar	Jacobson	Capacità Hop	VPS
Clink	Downey	Capacità Hop	VPS
Pchar	Mah	Capacità Hop	VPS
Bprobe	Carter	Capacità Percorso	Packet Pairs
Nettimer	Lai	Capacità Percorso	Packet Pairs
Pathrate	Dovrolis–Prasad	Capacità Percorso	Racket Pairs-Train
Sprobe	Saroiu	Capacità Percorso	Packet Pairs
Cprobe	Carter	AB Percorso	Packet Train
Pathload	Jain – Dovrolis	AB Percorso	SLoPS
IGI	Hu	AB Percorso	SLoPS
pathChirp	Ribeiro	AB Percorso	SLoPS

12. BIBLIOGRAFIA

- *"Bandwidth Estimation: Metrics, Measurement Techniques and Tools"*, Ravi Prasad Costantinos Dovrolis Georgia Institute Tecnology, Murray Claffy CAIDA, IEEE Network Magazine, November/December 2003
- *"Using Pathchar to estimate Internet Link Characteristics"*, Allen B. Downey, Colby College, IEEE Network
- *"Packet Triplet: An Enhanced Packet Pair Probing for Path Capacity Estimation"*, Zou ZiXuan, Lee Bu Sung, Fu Cheng Peng, Song Jie, Asia Pacific Advanced Network, Network Research Group Korea, August 2003
- *"What Do Packet Dispersion Techniques Measure?"*, Constantinos Dovrolis, Parameswaran Ramanathan, David Moore, University of Wisconsin e CAIDA
- *"Measuring Link Bandwidths Using a Deterministic Model of Packet Delay"*, Kevin Lai, Mary Baker, Department of Computer Science of Stanford University, September 2000
- *"Internet Path Characterization Using Common Internet Tools"*, Dulal C. Kar, Department of Computer Science of Texas A&M University
- BEst (Bandwidth Estimation Workshop) at La Jolla, CA, on December 2003:

- *"Bandwidth Measurements from a Consumer Prospective in Swede"*, Andreas Johnsson, Malardalen University
- *"Some Application of Bandwidth Estimation"*, Andrew Odlyzko, University of Minnesota
- *"On Evaluating a new class of Bandwidth Methods"*, P.Haga, D.Veitch, A.Pasztor, ERICSSON San Diego CA
- *"Bandwidth Estimation Metrics and Terminology"*, C.Dovrolis, College of Computing Georgia Tech
- *"Tomografy with Available Bandwidth"*, Shriram, Kaur, University of North Carolina at Chapel Hill
- *"Segmentation of Internet Paths for Capacity Estimation"*, Luckie, McGregor, WAND Group University of Waikato
- *"The CAIDA Bandwidth Estimation Testbed"*, Margaret Murray
- *"Accuracy and Expressiveness in Adaptive Bandwidth Measurements"*, Marc Pucci for Telcordia Technologies
- *"Bandwidth Estimation for Test Traffic Measurements Projects"*, Uijterwaal, Santcroos, Ripe NCC
- *"Towards Tunable Measurement Techniques for Available Bandwidth"*, Hu, Steenkiste, Carnegie
- *"Evaluating PathRate and PathLoad with realistic cross traffic"*, Ravi Prased, Manish Jain, Costantinos Dovrolis, Georgia Institute of Technology

- *“Spatio-Temporal Available Bandwidth Estimation”*, Ribeiro at Rice University
- *“Available Bandwidth Estimation in IEEE802.11-based Wireless Network”*, Shah, Chen, Nahrstedt, University of Illinois

Implementazione di un Software di misurazione delle grandezze prestazionali di una rete IP: Variable Train

Indice II° Progetto

1. INTRODUZIONE

Questo progetto nasce con l'intento di realizzare un software che implementi un algoritmo di analisi prestazionale di una rete.

La tecnica adottata non risulta essere una semplice riproduzione di una già esistente e già ben collaudata, ma nasce come maturazione e riflessione sulle metodologie precedentemente analizzate, ponendo nuove problematiche e nuove vie di sviluppo.

Pertanto l'algoritmo che ho ideato è il risultato di un'attenta osservazione su ciò che le tecniche precedenti offrivano, cercando di cogliere i punti forti di ognuna di esse. Nasce quindi una nuova tecnica di stima, che ho chiamato *"Variable Train"* e che ha la particolarità di essere trasversalmente valida per ogni tipologia di rete, dalle più lente alle più veloci.

Nei prossimi capitoli cercherò dapprima di introdurre l'idea di base, passando poi alla trattazione di alcune problematiche di origine pratica riscontrate in fase di implementazione che hanno modificato e complicato la teoria iniziale, per poi concludere con alcuni esempi di utilizzo ed alcune prove effettuate.

2. L'ALGORITMO

La stima prodotta dall'algoritmo *Variable Train* è la Capacità che effettivamente il sistema offre al di sopra del livello Transport, adottando in particolare il protocollo UDP (User Datagram Protocol).

Questa Capacità, definita come Capacità del Sistema, teoricamente è il risultato dell'impatto dell'overhead dei pacchetti sulla quantità di dati che effettivamente sfrutto nella connessione. Genericamente essa vale:

$$C_{SIS} = \frac{P_{UDP}}{P_{UDP} + H_{UDP}} C$$

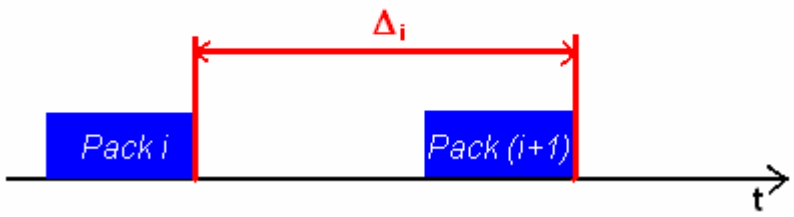
Indicando appunto con P_{UDP} il Payload visto all'interno del protocollo UDP (User Datagram Protocol) e quindi i dati effettivi che posso inserire a quel livello, H_{UDP} l'Header inserita dal protocollo UDP in giù ed infine C come Capacità fisica che il livello offre.

Come già annunciato precedentemente, la tecnica "*Variable Train*" nasce dalla fusione delle diverse potenzialità offerte dalle precedenti metodologie.

Riassumendo, gli algoritmi coinvolti sono i seguenti:

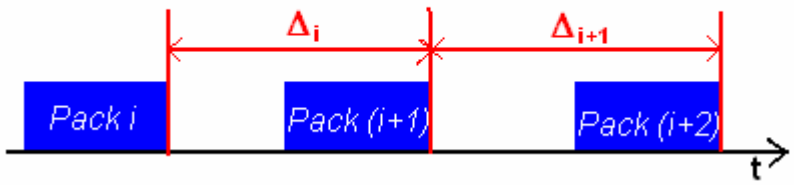
- Packet Pair Dispersion: questa tecnica funziona misurando la dispersione tra la ricezione completa di due pacchetti di dimensione nota. La situazione è quella descritta nella figura seguente:

Figura 17 - Packet Pair Dispersion



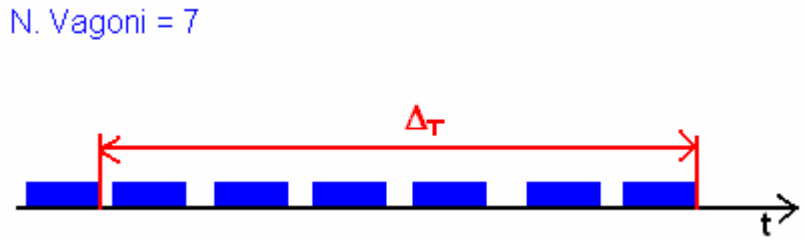
- Packet Triplet Dispersion: questa seconda metodologia si sviluppa attraverso un funzionamento simile a quella precedente ma introduce l'idea di correlare due misurazioni consecutive per determinare l'effettiva bontà raggiunta. Graficamente ho che:

Figura 18 - Packet Triplet Dispersion



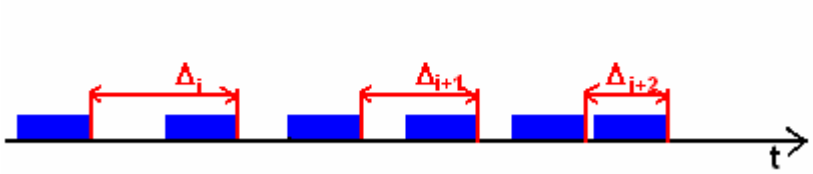
- Packet Train Dispersion: quest'algoritmo è stato rivoluzionario ed ha introdotto il concetto di treno di pacchetti e la relativa adozione come tecnica di stima. L'idea si basa sull'invio di un numero prefissato di pacchetti che nel loro insieme costituiscono un treno. Viene poi misurato il tempo che intercorre tra la ricezione del primo e dell'ultimo pacchetto e, conoscendone il numero e la singola dimensione, viene effettuata una stima di Capacità. In questo caso la situazione è la seguente:

Figura 19 - Packet Train Dispersion



- Trains of Packet Pairs: questa ultima tecnica da cui ho preso spunto è la chiave di collegamento fra le metodologie che sfruttano singoli pacchetti e le famiglie che adottano i treni, in quanto funziona inviando un treno composto da continue coppie di pacchetti e valutando i tempi offerti da ogni singola coppia. Graficamente questo algoritmo si presenta nel modo seguente:

Figura 20 - Trains of Packet Pairs



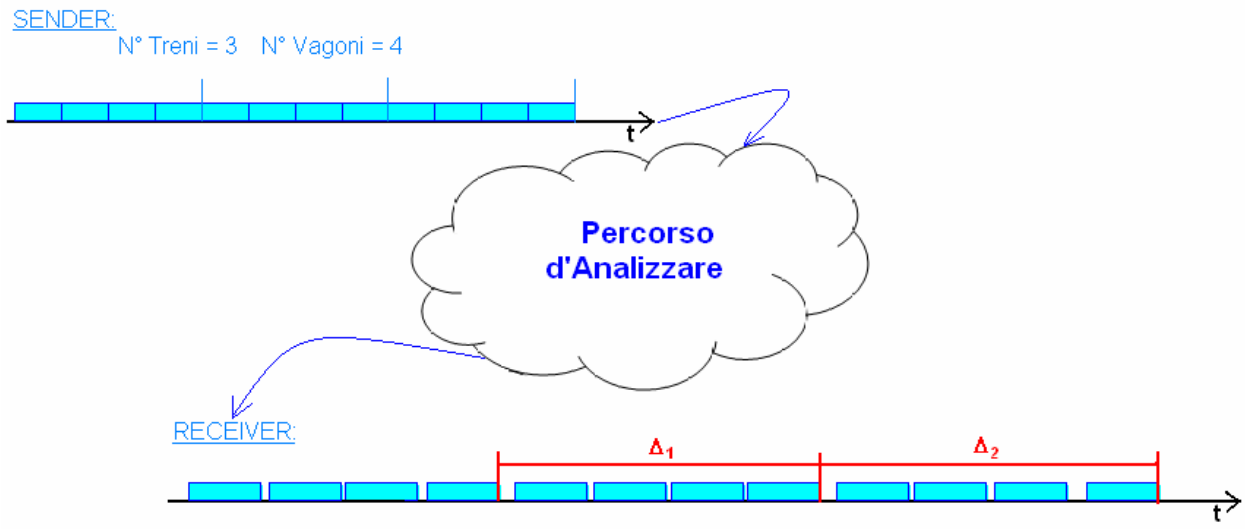
Variable Train nasce quindi come combinazione di varie idee, come la correlazione di diverse misurazioni successive tipica del Packet Triplet, l'utilizzo di treni tipica del Packet Train e il susseguirsi di diverse misurazioni in modo completamente automatico, originaria del Trains of Packet Pair.

Da questa miscelazione nasce l'idea di base, il cuore di tutto l'algoritmo, che si prefigge di analizzare un numero variabile di treni, costituiti da una quantità variabile di pacchetti, denominati vagoni, di dimensione variabile.

Dall'ampio grado di libertà offerto ho deciso pertanto di chiamare la tecnica "*Variable Train*".

Successivamente poi si analizza il tempo che intercorre tra la completa ricezione del primo vagone dell'i-esimo treno e del primo vagone dell'(i+1)-esimo treno, generando una differenza iterata a coppie.

Graficamente la situazione che si presenta è la seguente:



In questo primo momento vengono misurate, con la precisione del millisecondo, tante differenze quanti sono i treni, meno uno, in quanto sono calcolate a coppie.

In una fase successiva i dati appena ricavati vengono filtrati, attraverso una tecnica dove vengono confrontate tutte le differenze: dapprima si calcola una media generale e, dopo aver stabilito una tolleranza percentuale, si eliminano tutti i dati che non rientrano in questo intervallo. Questo procedimento viene eseguito tante volte quanto necessario, purché alla fine ogni differenza rimanga entro la media di quel passo.

Riassumendo quindi i passaggi del filtraggio dati sono i seguenti:

$$M_i = \frac{\sum_{j=1}^{N_i} \Delta_j}{N_i}$$

Dove Δ_j rappresenta le varie differenze temporali, N_i il numero di differenze che si sta considerando al passo i-esimo e M_i la media di quel passo.

Successivamente valuto per ogni Δ_j il valore della seguente differenza:

$$\left|\Delta_j - M_i\right| \leq (\alpha \cdot M_i)$$

Dove il parametro α rappresenta la tolleranza percentuale impostata. Se questa disuguaglianza si verifica il dato viene conservato; diversamente esso viene scartato.

Dopo aver ripulito eventuali dati errati che rischierebbero di inficiare l'intera stima, si effettua una media finale, che verrà adottata come misura definitiva del tempo che intercorre tra due treni.

Il calcolo della Capacità risulta quindi essere il seguente:

$$C_{SIS} = \frac{N_{VAGONI} \cdot Dim_{VAGONE}}{\Delta_T}$$

Che nasce appunto dal prodotto fra il numero di vagoni che costituiscono un treno e la dimensione di un singolo pacchetto, rapportati alla media precedentemente filtrata.

In questo modo l'algoritmo Variable Train genera una stima della Capacità di Sistema effettivamente offerta al di sopra del livello UDP (User Datagram Protocol).

3. PROBLEMATICHE DI ORIGINE PRATICA

In teoria la differenza fra teoria e pratica è nulla, in pratica questo non accade mai.

Dopo aver riflettuto su questo concetto, vorrei affrontare in questo capitolo le problematiche maggiormente rilevanti che ho incontrato durante lo sviluppo del software di Variable Train, che hanno leggermente modificato, complicato ed irrobustito l'algoritmo rispetto a cosi come era stato ideato inizialmente.

La difficoltà riscontrata praticamente sin dalla creazione dello scheletro generale è stata la parziale perdita dei pacchetti sul lato ricevente, per vari motivi, tra cui probabilmente la risoluzione ARP (Address Resolution Protocol) degli indirizzi di livello MAC o la difficoltà di bufferizzazione su reti molto veloci.

Questo problema è molto forte su tratti di rete in cui le prestazioni sono molto elevate, ed è un fenomeno che si incontra almeno una volta in ogni sessione di analisi, quindi mi sono trovato costretto ad ideare una soluzione e a risolverlo in modo brillante.

Il sistema che ho realizzato è un sistema basato su una sorta di "Wait – Reset", nel senso che il Sender, cioè l'Host a cui è affidato il compito di inviare i treni, dopo aver eseguito il suo compito principale, attende un numero di secondi prefissato, definiti come "TimeBeforeReset" e, se

nell'arco di questo tempo non riceve un messaggio da parte del Receiver di completa ricezione, provvede a mandare un messaggio speciale, definito come Messaggio di Reset, in cui comunica al ricevitore di azzerare tutti i suoi conti relativi a quella singola sessione di analisi in quanto sono andati persi dei pacchetti e che quindi sta per ricominciare ad inviare tutti i treni dall'inizio.

Questa procedura viene continuamente ripetuta finché effettivamente il Receiver non ha ricevuto tutti i vagoni di tutti i treni e quindi non spedisce al Sender il messaggio di completa ricezione.

Un secondo accorgimento che intendo affrontare in questo paragrafo è l'automatizzazione del Sender.

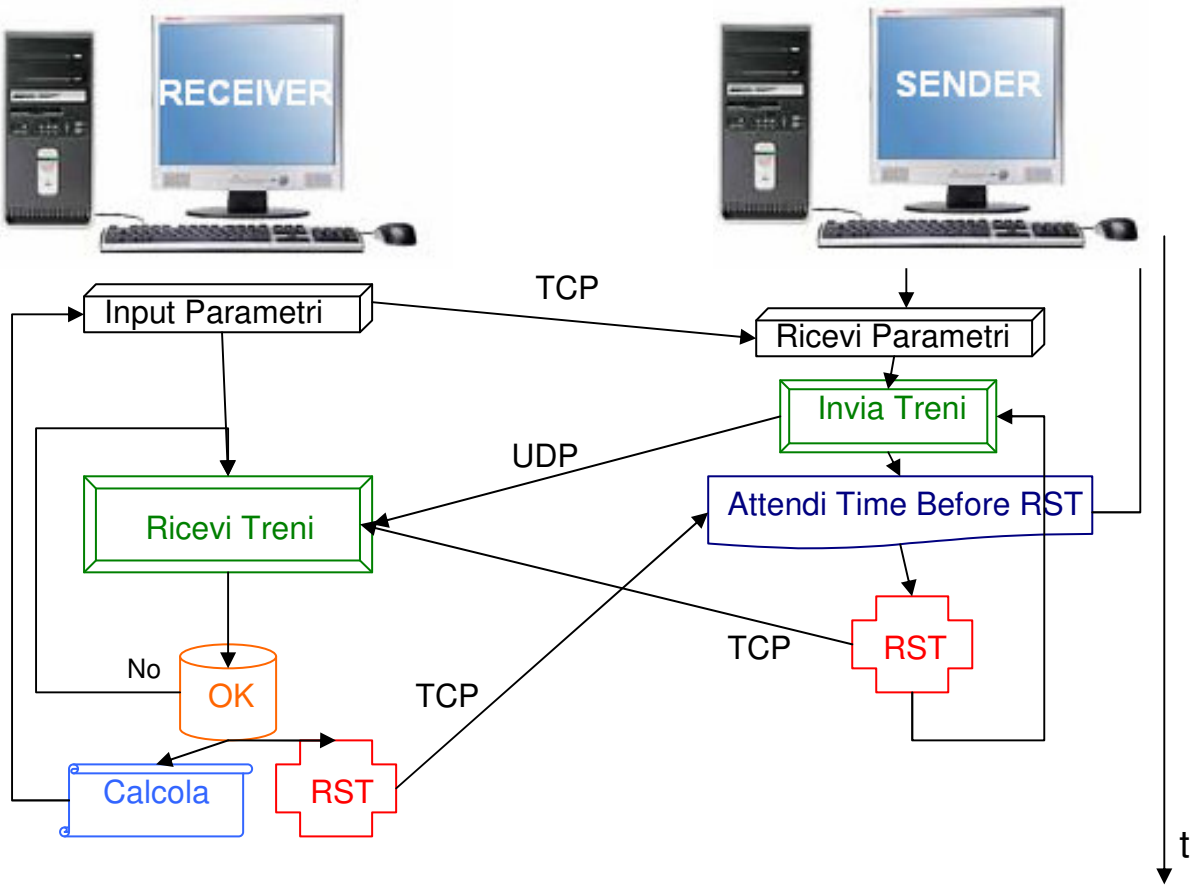
Inizialmente se si voleva configurare una sessione di analisi lo si doveva fare doppiamente, sia sul Ricevitore sia sull'host che inviava i treni. Utilizzando invece una "Connessione di Servizio" ho fatto in modo che i parametri, una volta acquisiti tramite interfaccia grafica dal lato ricevente, vengano trasmessi automaticamente al Sender, che li imposta all'interno della sua struttura dati e procede all'invio dei treni.

In questo modo l'architettura del lato emittente è costituita da due passaggi fondamentali: acquisizione dati e relativa impostazione, invio treni. Una volta avviato il Sender effettua sempre e solo questi due

processi in modo perfettamente automatico, permettendone l'avviamento anche in remoto.

Un'ultima modifica che espongo è relativa alla modalità di studio permessa all'interno di Variable Train. Le possibilità offerte sono sia uno studio singolo, impostando la dimensione dei vagoni, sia un'analisi iterata, programmando la misurazione della Capacità di Sistema con diverse sessioni, diversificate dalla dimensione dei vagoni, a partire da un valore iniziale fino ad uno finale, con un incremento prestabilito.

Il diagramma delle sequenze complessivo di questa tecnica risulta essere il seguente:



4. IMPLEMENTAZIONE

In questo capitolo sarà affrontata l'architettura generale in termine di implementazione e quindi di traduzione in linguaggio di programmazione delle classi più importanti.

Il linguaggio utilizzato per la realizzazione di un primo prototipo dell'algoritmo Variable Train è stato interamente Java, nella sua versione JVM 1.4.2_04, installato su un due PC dalle seguenti caratteristiche: Intel Celeron 2,4GHz con 256MB RAM, 3 schede Fast-Ethernet 10/100 RealTek RTL8139 e OS Linux nella sua distribuzione Red Hat 9 (Kernel 2.4.20-8).

L'architetture generale è da scomporre in due settori: il lato ricevitore, chiamato Receiver ed il lato emittente, denominato Sender.

Il lato ricevente dell'algoritmo è formato dalle seguenti componenti:

- EngineReceiver: è il motore del Ricevitore, colui che si preoccupa di ogni suo azionamento e dello svolgimento generale delle sue mansioni.

```
public class EngineReceiver extends Thread {
  public void run() {
       boolean last = false;
for (int dim = startDim, i = 0; dim <= endDim; dim += gap,
i++) {
 /*Attendo finchè il Thread di Misurazione precedente non ha
 */finito
     while (!parameter.s) {
       try {
         Thread.sleep(1500);
       }
```

```
        catch (InterruptedException ex) {
          System.err.println("Errore    nell'attesa:    "    +
ex.toString());
        }
      }
      parameter.s = false;

      if((dim + gap) > endDim) {
        last = true;
      }

/*Imposto la dimensione del vagone per questa misurazione e
*/la faccio partire
      parameter.setDimWagon(dim);
      OneLoopReceiver   r   =   new   OneLoopReceiver(parameter,
result, area);
    }
  }
```

- OneLoopReceiver: è dedito allo sviluppo di una singola sessione di studio, con la misurazione quindi di un solo invio di treni con un'unica dimensione dei vagoni.

```
public class OneLoopReceiver {
  public OneLoopReceiver(Parameter parameter,
ResultContainer res, JTextArea workArea) {
    this.parameter = parameter;
    this.result = res;
    this.area = workArea;

    //Mando i parametri al Sender
    sendParameter();

    //Avvio la singola misurazione
      rec = new Receiver(parameter, result, area);
  }
```

- Receiver: si occupa nello specifico di intercettare i singoli vagoni, contarli e memorizzare la quantità di treni. È il componente che riconosce la ricezione completa di tutti i pacchetti in base al valore atteso.

```
public class Receiver extends Thread {

  public Receiver(…) {
```

```
   //Inizializzo la misurazione
    prepareMeasure();

    //Creo il descrittore dei treni
    td = new TrainDescriptor();

    //Implemento in un Thread a parte la misurazione
    Thread rec = new Thread(this);

    //Creo il Thread di Controllo verso il Sender
    recCtrl = new ReceiverController(conn, td, parameter,
area);

    rec.start();
    recCtrl.start();
  }

  public void run() {
/*Ricevo i pacchetti e procedo con la misurazione del tempo
*/intermedio di ricezione
      while (td.getTrain() != NUM_TRAIN) {
        try {
          //Ricevo il vagone
          dSock.receive(dPacket);
          //Incremento il numero di Vagoni
          td.moreWagon();

/*Se il numero di vagoni è uguale a numero impostato di
*Vagoni da cui un treno è formato,incrementa il numero di
*/Treni e memorizza tempo
          if (td.getWagon() == NUM_WAGON) {
            td.addTime(new Long(new GregorianCalendar().get(
                GregorianCalendar.MILLISECOND)));
            td.setWagon(0);
            td.moreTrain();
          }
        }
        catch (IOException e) {
          System.err.println("Errore Ricezione Pacchetto: "
+ e.toString());
        }
/*Fermo il Sender attraverso il canale di Controllo se il
*/numero di treni corretto
        if (td.getTrain() == NUM_TRAIN) {
          conn.sendMessage(messageRst);
          recCtrl.stop();
          dSock.close();

          //Calcolo la capacità e la memorizzo
          showData();
          parameter.s = true;
        }
      }
  }
```

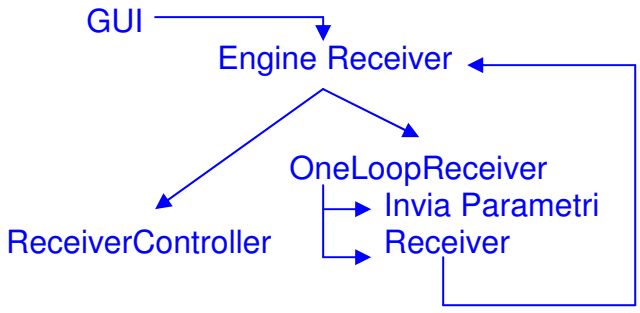
- ReceiverController: si mette in ascolto di eventuali messaggi di reset da parte del Sender inoltrati sulla porta di servizio.

```
public class ReceiverController extends Thread {
  public void run() {
    while (semaphore &&
connection.listenMessage(parameter.getReset())) {
     /* se ricevo un reset cancello ogni cosa della
      * misurazione che non è andata a buon fine
      */
      td.setTrain(0);
      td.setWagon(0);
      td.cleanTime();

      if (td.getTrain() == parameter.getNumTrain()) {
        semaphore = false;
      }
    }
  }
```

Queste quattro procedure nel loro insieme costituiscono il funzionamento del Ricevitore, che ha il compito di acquisire i parametri da interfaccia grafica, inviarli al Sender ed ascoltare e conteggiare i treni in arrivo. Nel caso in cui arrivino tutti i vagoni di tutti i treni, viene inviato un messaggio di avvenuta ricezione all'altro lato in modo da bloccarlo e si procede con il calcolo di una nuova misurazione.

Nel loro insieme esse vengono sfruttate come segue:



Queste erano le quattro classi fondamentali che implementano il concetto di Ricevitore nell'algoritmo VariableTrain.

Dal lato mittente la situazione è formata dai seguenti componenti:

- EngineSender: similmente al ricevitore, questo processo è dedita al funzionamento generale ed esegue in un loop infinito due passaggi, pari all'acquisizione dei parametri relativi alla misurazione e all'invio dei treni. Essendo un ciclo infinito, al termine di ogni misurazione il Sender si rende automaticamente disponibile ad una nuova sessione di analisi.

```
public class EngineSender {
  public EngineSender() {
    parameter = new Parameter();
    while (true) {
      //Ricevo i parametri dal Receiver
      receiveParameter();

      //Inizio L'esecuzione
      new Sender(parameter);
    }
  }
```

- Sender: implementa direttamente l'invio dei pacchetti; successivamente attende un tempo pari a TimeBeforeReset, entro il quale se dall'altro lato dell'analisi i pacchetti sono stati ricevuti tutti, dovrebbe ricevere un messaggio di avvenuta ricezione. Se questo non accade è lui ad inviare un Reset al Receiver provocando l'azzeramento delle variabili di conteggio e la conseguente ripetizione della misura dall'inizio. Questo viene iterato automaticamente finchè non viene prodotta una misurazione completa.

```
public class Sender extends Thread {
  public Sender(Parameter para) {
    this.parameter = para;

    //Ricevo le impostazioni desiderate dal Ricevitore
```

```
    IP_RECEIVER = parameter.getIPReceiver();
    PORT_RECEIVER = parameter.getPort();
    NUM_TRAIN = parameter.getNumTrain();
    NUM_WAGON = parameter.getNumWagon();
    DIM_WAGON = parameter.getDimWagon();
    SERVICE_PORT = parameter.getServicePort();

    //Apro la connessione di controllo con il Receiver
    conn = new CtrlConnection(IP_RECEIVER, SERVICE_PORT);

    //Inizializzo il Sender
    prepareSending();

    //Faccio partire il sender e la connessione di controllo
    send = new Thread(this);
    sendCtrl = new SenderController(conn, send, parameter);

    sendCtrl.start();
    send.start();
  }

  public void run() {
    while (true) {
      //Invio i pacchetti
      for (int i = 1; i <= NUM_WAGON * NUM_TRAIN; i++) {
        try {
          dSock.send(dPacket);
        }
        catch (IOException e) {
          System.err.println("Errore nell'invio: " +
e.toString());
        }
      }
      //Attendo qualche secondo
      try {
        Thread.sleep(parameter.getTimeBeforeRST() * 1000);
      }
      catch (InterruptedException ex) {
        send.stop();
        System.err.println(
            "Errore nell'attesa tra i diversi tentativi di
invio: " +
            ex.toString());
      }

       /*Invio un Reset Delle Variabili al Ricevitore
       */tramite il canale di Controllo
      conn.sendMessage(parameter.getReset());

      //Attendo qualche secondo per poi ricominciare
      try {
        Thread.sleep((int)WAITING_SEC * NUM_TRAIN * 1000);
      }
      catch (InterruptedException ex) {
        System.err.println(
            "Errore nell'attesa tra i diversi tentativi di
invio: " +
            ex.toString());
      }
```

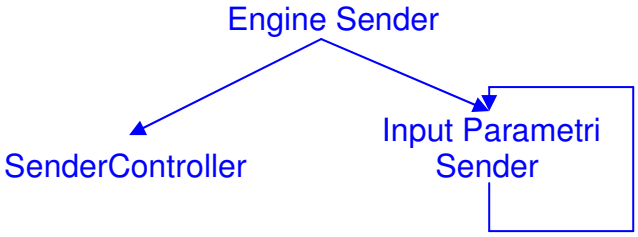
```
        }
      }
```

- SenderController: dedito all'ascolto sulla porta di servizio di un eventuale messaggio di completa ricezione da parte del Receiver.

```
public class SenderController extends Thread {
  public void run() {
    while (stopSem && connection.listenMessage(parameter.getReset()))
{
      //Se ricevo un Reset fermo il Sender
      thread.stop();
      stopSem = false;
      //Mando un messaggio di avvenuto stop
      connection.sendMessage(parameter.getReset());
    }
  }
```

Queste tre classi nel loro insieme implementano il concetto di Sender, che ha il compito fondamentale di acquisire dati, inviare treni ed eventualmente far ripartire la misurazione.

Nel loro insieme questi tre processi funzionano nel seguente modo:



Esistono poi altre classi che vengono utilizzate sia dal package relativo al Receiver che da quello del Sender, che servono per la gestione dei parametri della misurazione, alla creazione del canale di connessione TCP per i messaggi di Reset, al conteggio dei treni ricevuti o al riempimento delle coppie dimensione vagone – capacità misurata valide come risultato finale.

Intendo esporre infine una classe il cui funzionamento è già stato descritto precedentemente, quella relativa al filtraggio dati. Il codice è il seguente:

```
public class DataFilter {
  public DataFilter(ArrayList data, int packetDim, int
numWagon, double alfa) {
    this.data = data;
    this.alfa = alfa;

    //Filtra i dati e calcola la media
    calculateMediaTime();

    //Calcolo della capacità in Byte al Secondo
    capacity = ( (numWagon * packetDim) / (mediaTime /
1000));
  }

  private void filterData() {
    long m = 0;
    int i, element = 0, initialSize;
    double tollerance;
    boolean semaphore = true;

    while (semaphore && data.size() != 0) {
      m = 0;
      initialSize = data.size();

 //Calcolo la media a questo passo,riempiendo la variabile m
      for (i = 0; i < data.size(); i++) {
        try {
          m += Integer.parseInt(data.get(i).toString());
        }
        catch (NumberFormatException ex) {
          System.err.println("Errore nella conversione dei
Dati Temporali: " +
                             ex.toString());
        }
      }
      if (data.size() != 0) {
        m /= data.size();
      }

/*Ricavo ogni singolo elemento dai dati passati e valuto la
*/diseguaglianza
      for (i = 0; i < data.size(); i++) {
        try {
          element =
Integer.parseInt(data.get(i).toString());
        }
        catch (NumberFormatException ex) {
          System.err.println("Errore nella conversione dei
Dati Temporali: " + ex.toString());
        }
```

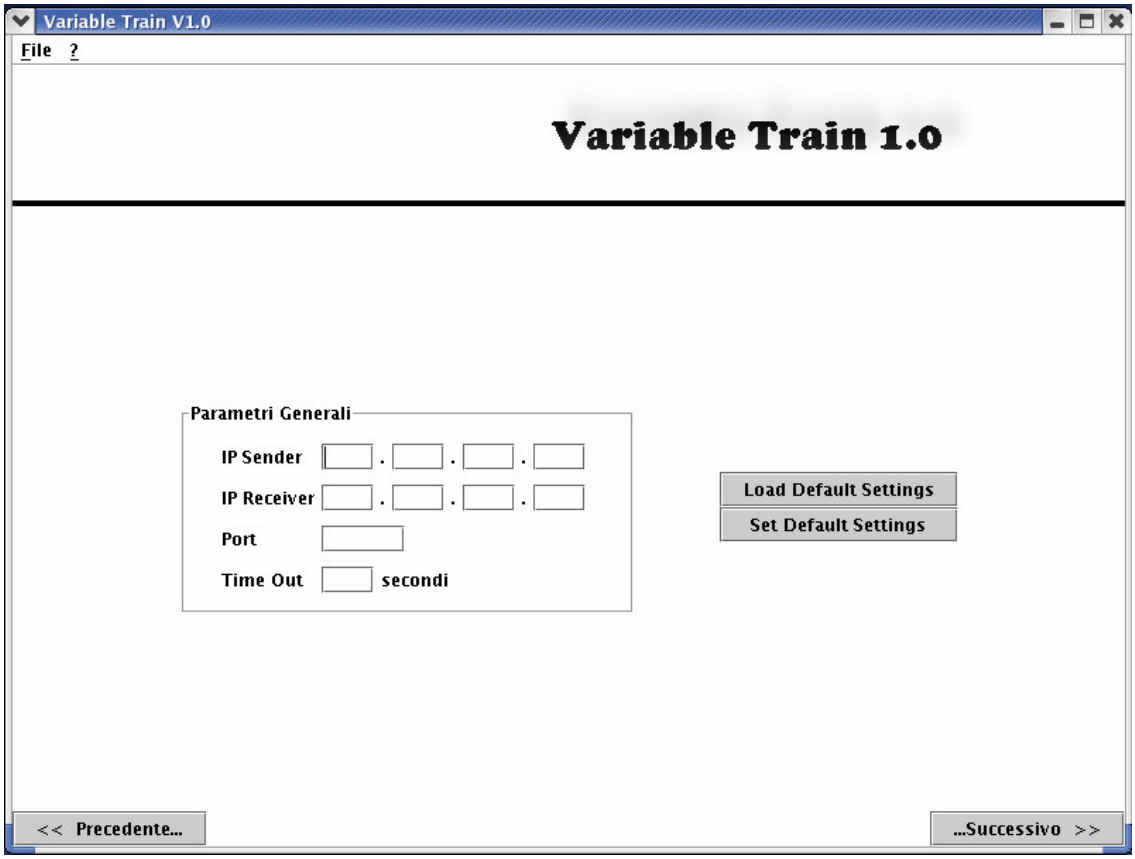
```
        tollerance = alfa * m;
  /*Se la differenza supera la tolleranza prefissata da alfa,
  */scarto il dato
  if ( (Math.abs(element - m)) > tollerance) {
          data.remove(i);
        }
      }
//Se tutti I dati rientrano nella tolleranza esco
      if (data.size() == initialSize) {
        semaphore = false;
      }
    }
  }

  //Calcolo la media dopo che ho filtrato tutte le differenze
  private void calculateMediaTime() {
    filterData();
    if (data.size() != 0) {
      for (int i = 0; i < data.size(); i++) {
        try {
          mediaTime +=
Integer.parseInt(data.get(i).toString());
        }
        catch (NumberFormatException ex) {
          System.err.println("Errore nella conversione dei
Dati Temporali: " +
                             ex.toString());
        }
      }
      mediaTime /= data.size();
    }
  }
```

5. COME UTILIZZARE VARIABLE TRAIN V1.0

Questo capitolo cercherà di essere una sorta di guida all'utilizzo del software che implementa l'algoritmo Variable Train.

All'avvio il programma presenta la seguente interfaccia:

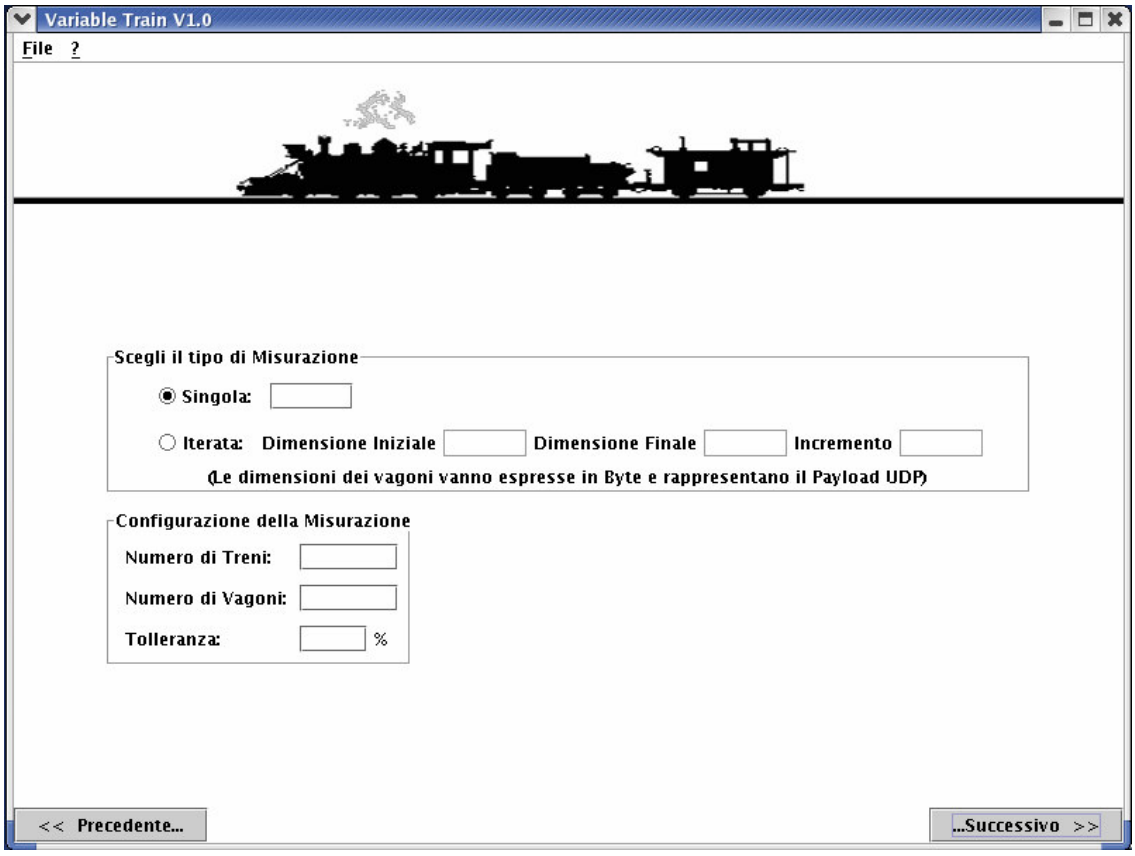


Dove negli otto singoli campi dedicati all'IP vanno inseriti gli indirizzi del ricevitore e dell'host che invia i treni. Nel campo Port invece bisogna un numero valido di porta su cui effettuare l'invio dei pacchetti che compongono i vari treni.

Nell'ultima riga di input, il Time Out, va inserito il tempo massimo in secondi che il Sender può attendere prima che lanci un messaggio di Reset al Receiver, considerando quindi la perdita di alcuni vagoni.

Questi quattro parametri sono di natura generale, quindi tra una sessione di analisi ed un'eventuale successiva possono essere i medesimi e per questo motivo ho riservato la possibilità all'utente di poterli compilare una sola volta, salvandoli poi in un file XML, che si può riutilizzare nelle varie misurazioni successive.

Una volta stabiliti questi primi parametri, si passa alla fase successiva, dove vengono immessi i dati specifici di questa sessione. L'interfaccia è la seguente:



All’interno di questo pannello bisogna innanzitutto valutare il tipo di studio che si vuole fare, pensando quindi se si intende sfruttare una singola sessione o diverse successive.

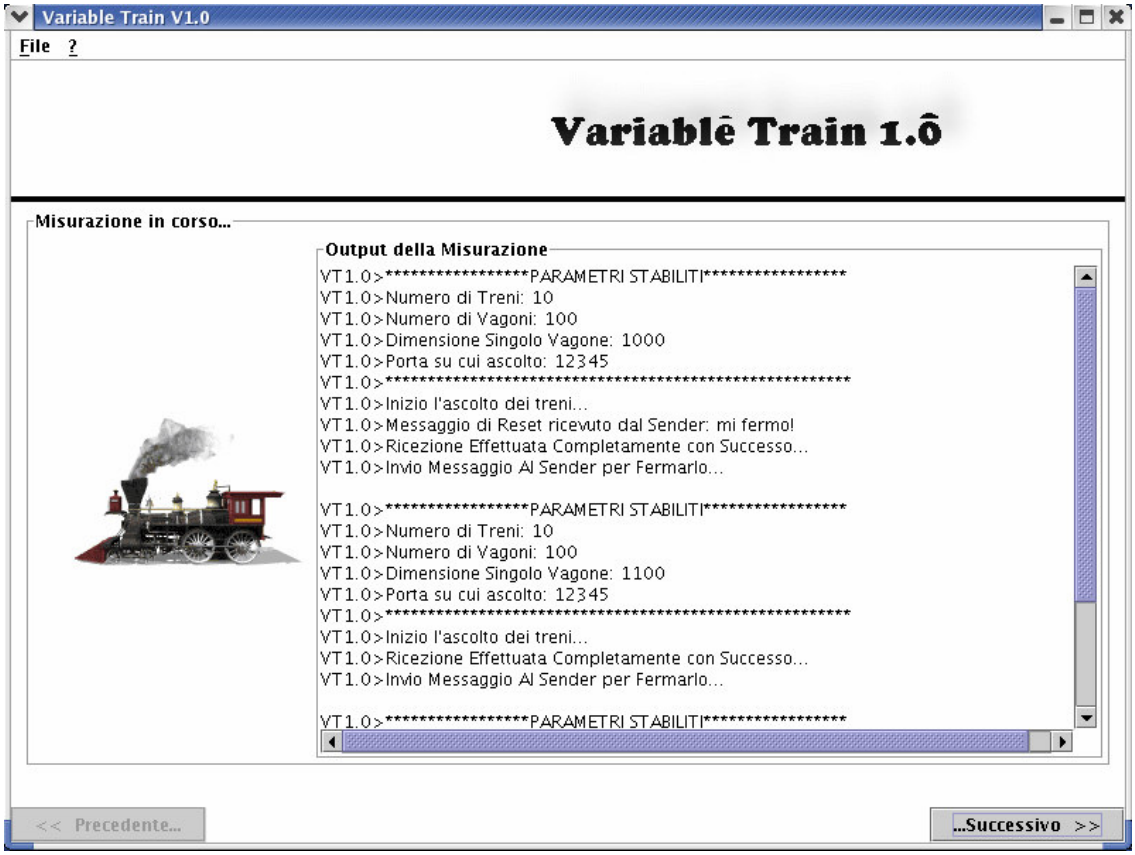
Nel caso di una misurazione singola vengono inviati un numero di treni stabiliti, formato da un numero di vagoni di dimensione scelta.

Nel caso invece si scelga una sessione iterata vengono inviati dapprima i treni stabiliti con una dimensione dei vagoni iniziale, per calcolare quindi una prima stima. Una volta valutato questo primo valore, si passa all’invio di un altro giro di treni formati da vagoni la cui dimensione vale stavolta quella iniziale, più un incremento stabilito: in questo modo produco un secondo valore di capacità. Tutto questo viene ripetuto ciclicamente finché la dimensione dei vagoni non supera una soglia finale prestabilita.

In questa tipologia di studio alla fine non avrò un solo valore stimato, ma avrò diverse coppie di risultati, che considerano la dimensione dei singoli vagoni e la capacità del sistema rilevata in quel caso.

Un ultimo valore da inserire è la Tolleranza, espressa in percentuale, in un intervallo da 0 a 100, che sarà adottata in una successiva fase di filtraggio dati.

Una volta inseriti questi parametri ben più specifici rispetto ai precedenti, si può iniziare la misurazione, che mentre si evolve presenta in output le seguenti informazioni:

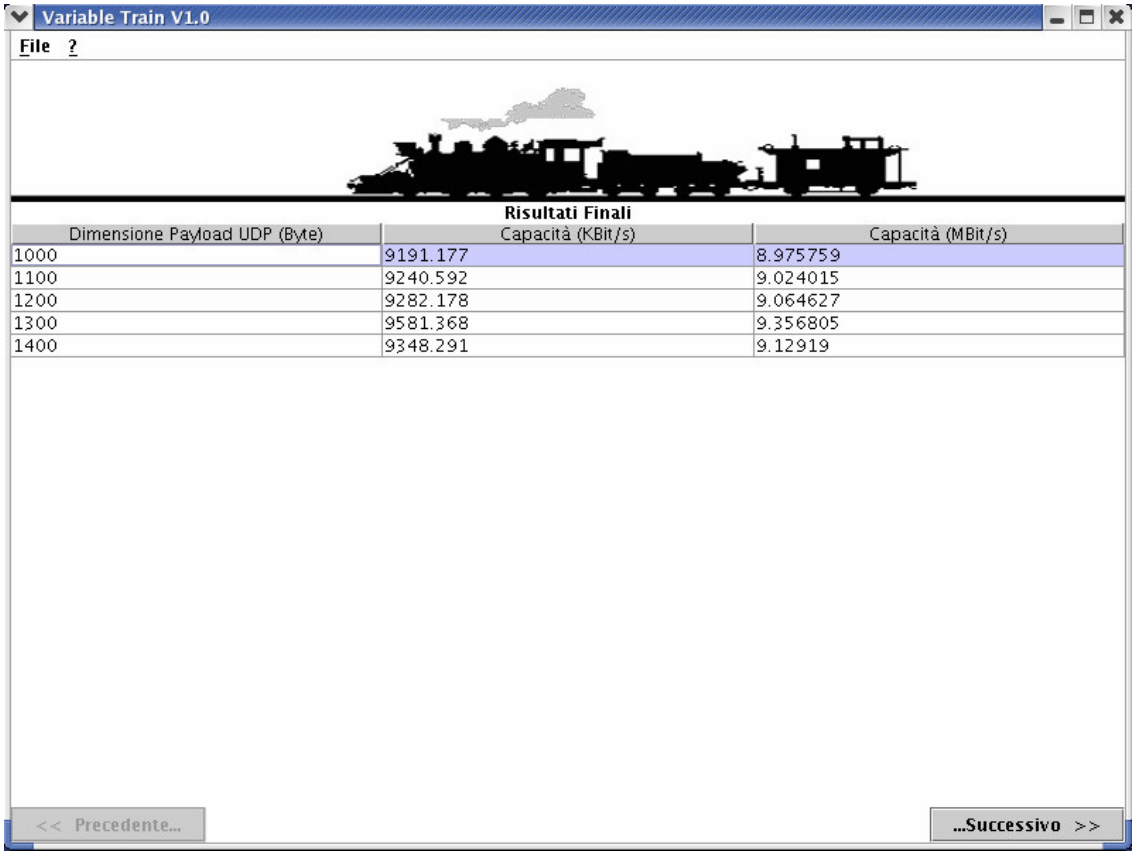


Dopo aver atteso il completo sviluppo di tutte le misurazioni che si è scelto di fare, si può passare all'ultima fase, dove vengono riassunti in una tabella tutti i dati misurati, rappresentati quindi dalla coppia dimensione vagone – capacità di sistema.

In questa tabella sono appunto riassunti sulla prima colonna le dimensioni del Payload a livello UDP (User Datagram Protocol) che sono impostate per quanto riguarda i vagoni, sulla seconda colonna la capacità misurata in kilobit al secondo mentre sull'ultima in megabit al secondo.

In un’unica tabella sono quindi riportati tutti i valori, sia per quanto riguarda la singola misurazione, che per quelle composte da diverse sessioni.

Una tipica schermata è la seguente:

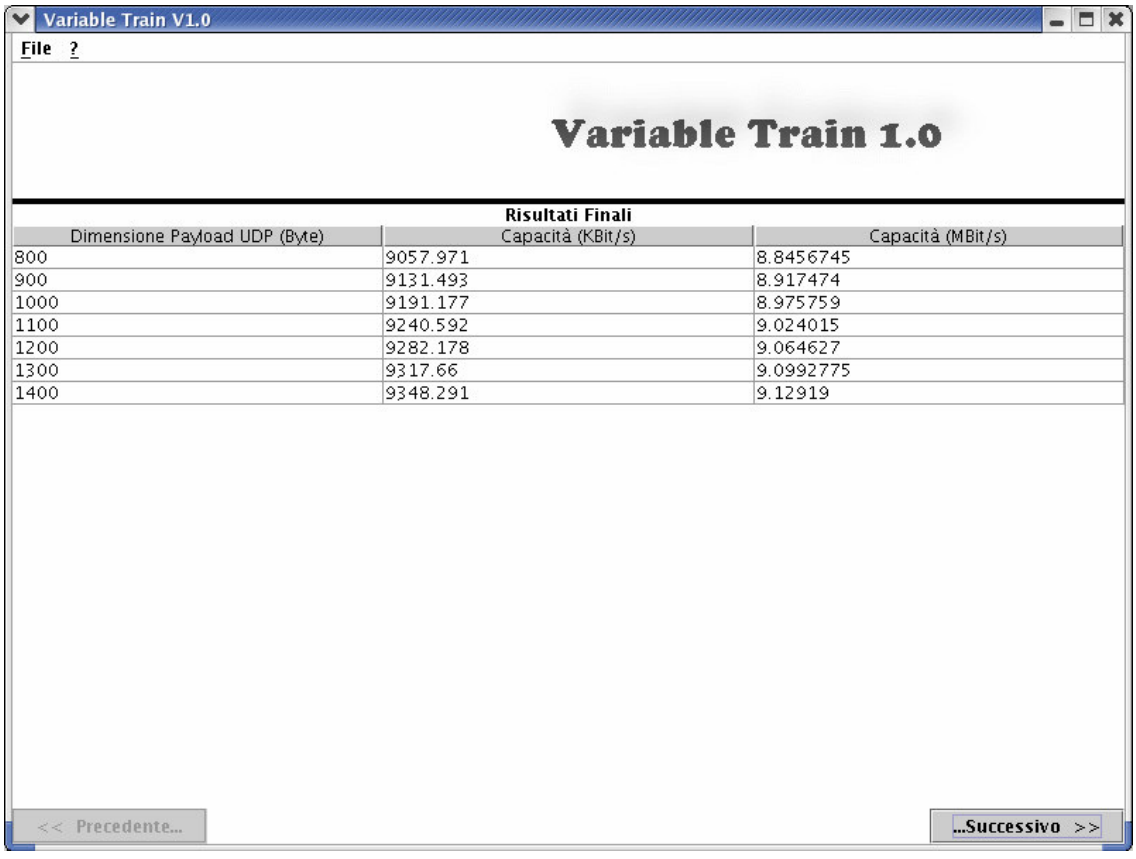


6. CASI DI TEST

In questo capitolo verranno presentati alcuni casi di test effettuati in laboratorio.

Un primo test rappresenta una connessione punto a punto effettuata su due pc caratterizzati da processore Intel Celeron 2.4GHz con 256MB RAM, 3 schede Fast-Ethernet 10/100 RealTek RTL8139 e OS Linux nella sua distribuzione Red Hat 9 (Kernel 2.4.20-8) collegati fra loro da un Hub 3COM Modello OfficeConnect 8 porte gestito con agenti SNMP e programmato per funzionare ad una velocità massima di 10Mbit/s.

I risultati ottenuti sono stati i seguenti:



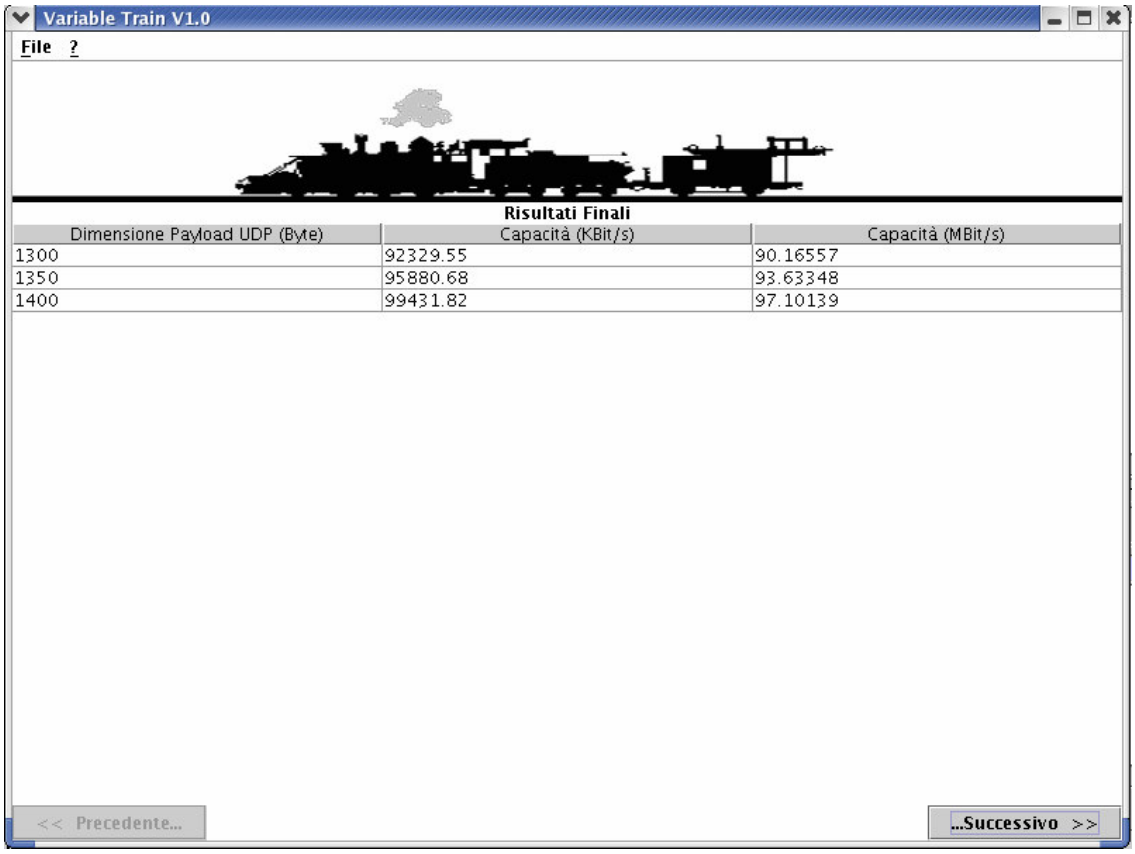
Dimensione Payload UDP (Byte)	Capacità (KBit/s)	Capacità (MBit/s)
800	9057.971	8.8456745
900	9131.493	8.917474
1000	9191.177	8.975759
1100	9240.592	9.024015
1200	9282.178	9.064627
1300	9317.66	9.0992775
1400	9348.291	9.12919

Che confrontati con i calcoli teorici risulta essere:

Dimensione Payload UDP (byte)	Teorici	Misurati	Differenza Percentuale
800	9,26	8,85	-4,42
900	9,34	8,92	-4,46
1000	9,40	8,98	-4,45
1100	9,45	9,02	-4,55
1200	9,49	9,07	-4,46
1300	9,53	9,1	-4,52
1400	9,56	9,13	-4,53

Come si può notare si ha un errore praticamente costante che si aggira intorno ad un 4.5% in difetto. La costanza di questa differenza è da spiegare unicamente con un fattore di ritardo che si riscontra in tutte le misurazioni. Questo elemento è facilmente ipotizzabile in un problema di acquisizione veloce del tempo.

Un secondo test coinvolge la stessa architettura, connessa a 100Mbit/s:



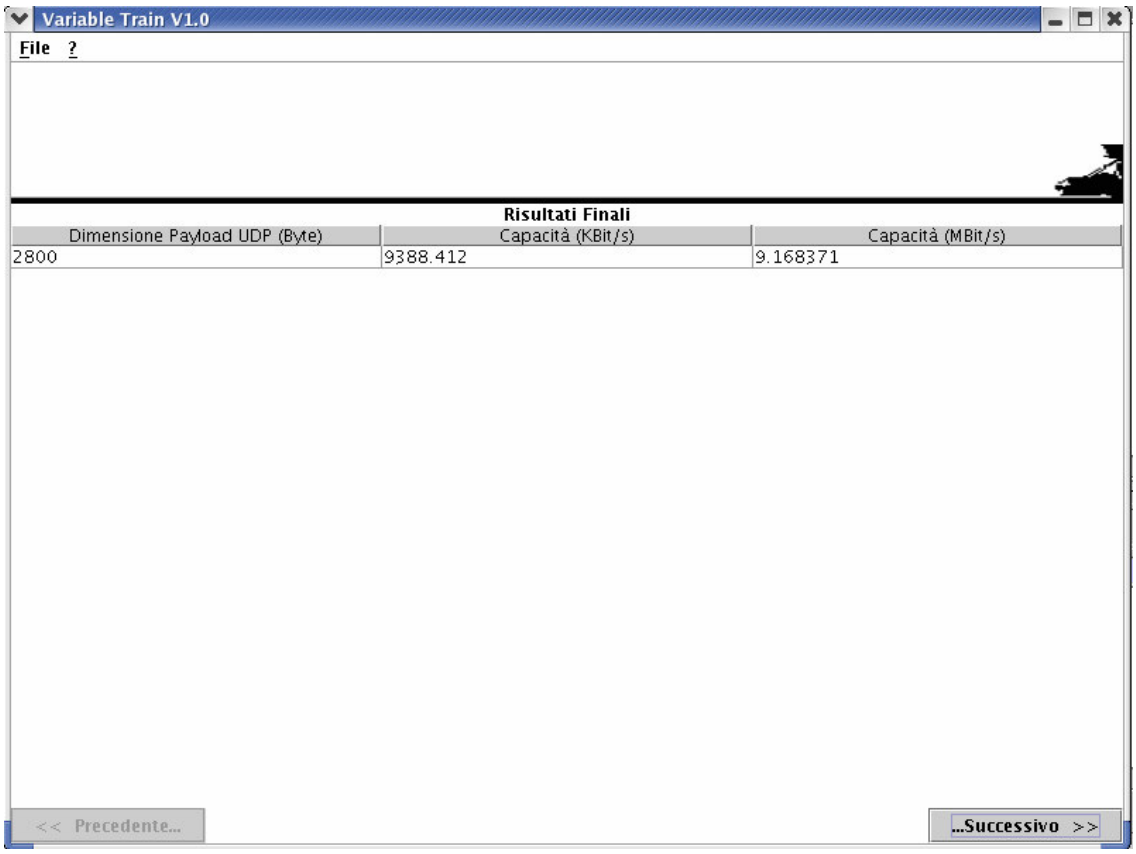
Il confronto teorico-misurato è il seguente:

Dimensione Payload UDP (byte)	Teorici	Misurati	Differenza Percentuale
1300	95,31	90,17	-5,39
1350	95,47	93,63	-1,93
1400	95,63	97,10	1,54

In questo caso la connessione è dieci volte più veloce e come si può notare capita anche che la Capacità venga sovrastimata, anche se solo del 1.5%. Negli altri due casi le misure risultano essere più che soddisfacenti.

Un ultimo caso di test riguarda la frammentazione, effettuata con un Payload di dimensione pari a 2800 byte, che devono essere divise in due pacchetti di dimensioni pari a circa la metà ognuno.

I risultati sono i seguenti:



7. CONCLUSIONI E SVILUPPI FUTURI

Variable Train nella sua prima implementazione ha dato ottimi risultati, che si possono migliorare solo introducendo una misurazione del tempo ad un livello più vicino possibile a quello macchina.

Una soluzione estrema in questo senso potrebbe essere quella di produrre un sistema elettronico che anticipi la scheda di rete, dotato di un Clock Real Time (RTC) in grado di memorizzare i tempi in cui i treni sono completamente ricevuti.

Variable Train ha rappresentato un'ottima opportunità per comprendere, affrontare e superare problematiche di tipo innovativo e di ricerca.

In questo senso Variable Train mi ha incuriosito e mi ha dato stimoli nuovi, che mi spingeranno a continuare questo lavoro anche oltre questo progetto di Tesi, cercando di coltivarlo e di migliorarlo sempre di più.