ISTITUTO NAZIONALE DI FISICA NUCLEARE

CNAF

INFN/CCR-07/04
25 Maggio 2007

CCR-05/2006/P

OTTIMIZZAZIONI DEL PROTOCOLLO TCP PER CONNESSIONI LHCOPN A 10 GB/S

Marco Bencivenni

INFN-Sezione di Bologna, Viale Berti Pichat, 6/2, I-40127 Bologna, Italy

Abstract

L'obiettivo del lavoro è stato quello di verificare le prestazioni e le configurazioni di vari elementi di un sistema di comunicazione a 10 Gb/s (server, interfacce di rete per nodi terminali, linee locali e geografiche sperimentali) al fine di individuare una configurazione ottimale che permetta l'ottimizzazione delle prestazioni di un'applicazione basata sul protocollo di trasmissione TCP.

Sono stati quindi individuati una serie di parametri chiave sia di tipo hardware che software, la cui configurazione risulta particolarmente rilevante ai fini delle performance sperimentate.

Poster presentato al
Workshop sul Calcolo e Reti dell'INFN "Verso la sfida di LHC"
Otranto (Lecce), Hotel degli Haethey 6-9 Giugno 2006

*Published by **SIS–Pubblicazioni***
Laboratori Nazionali di Frascati

1 INTRODUZIONE

Molte applicazioni scientifiche moderne basano i propri risultati di ricerca sull'analisi di grandi volumi di dati distribuiti geograficamente. L'accesso a tali dati è reso possibile dall'utilizzo di infrastrutture di comunicazione a cui è richiesto di soddisfare svariati requisiti. Innanzi tutto, l'accesso e l'analisi distribuita tipicamente richiedono un elevato grado di affidabilità, il rispetto dell'integrità e della confidenzialità dei dati sensibili, e la generazione di grossi volumi di traffico. Là dove questi requisiti non sono in grado di essere soddisfatti da infrastrutture di reti condivise, si ricorre all'utilizzo di reti private virtuali, la cui implementazione è oggi resa possibile dall'offerta su scala europea di circuiti di livello di tipo SDH, come per esempio nel caso della rete di nuova generazione pan-europea per la ricerca e l'educazione, denominata GEANT2.

In questa ottica sono state verificate le prestazioni e le configurazioni di vari elementi di un sistema di comunicazione a 10 Gb/s (server, interfacce di rete per nodi terminali, linee locali e geografiche sperimentali) al fine di individuare una configurazione ottimale che permetta l'ottimizzazione delle prestazioni di un'applicazione basata sul protocollo di trasmissione TCP. Sono stati quindi individuati una serie di parametri chiave sia di tipo hardware che software, la cui configurazione risulta particolarmente rilevante. A livello hardware sono stati esaminati il bus interno PCI-X di un server, in particolare il parametro Maximum Memory Byte Count (mmbrc), inoltre è stato effettuato un confronto tra schede due modelli di interfacce di rete 10 Giga Ethernet, Intel e Chelsio. Sono stati comparati in termini di prestazione e di utilizzo della CPU, due diversi kernel Linux: le versioni 2.6 e 2.4. Sono stati individuati diversi parametri del kernel estremamente significativi al fine di massimizzare l'efficacia del processo trasmissivo a 10 Gb/s, in particolare: la dimensione delle code txqueuelen e backlog e la dimensione dei receive e transmit socket buffer. Inoltre, sono state confrontate le prestazioni di due diverse implementazioni del protocollo TCP: TCP Reno e TCP BIC. TCP BIC è una versione ottimizzata del protocollo, esplicitamente introdotta nei kernel Linux per migliorare le prestazioni di un flusso TCP in caso di scenari basati su cammini di rete a larghissima banda e altro ritardo trasmissivo. Particolare attenzione è stata posta alle caratteristiche di convergenza, dei due protocolli.

Il lavoro svolto si sviluppa all'interno di alcuni progetti che intendono fare ricerca e sperimentazione nell'ambito dello sviluppo di griglie computazionali per la condivisione di risorse distribuite in ambito geografico. In particolare, il lavoro svolto trova collocazione all'interno del progetto "LHC Computing Grid" (LCG), il cui intento è realizzare una rete privata virtuale per il collegamento di svariati centri di calcolo in Europa, America, Oceania e Asia. Tale infrastruttura renderà possibile l'analisi di una grande quantità di dati prodotti dal nuovo acceleratore di particelle LHC (Large Hadron Collider) che vedrà il via nel 2007. Ad oggi tale infrastruttura risulta già operativa in più di 100 siti distribuiti su 31 distinte nazioni.

2 INFRASTRUTTURA DI TESTO

L'obiettivo principale del nostro lavoro era testare la connessione 10 Giga Ethernet che collega il CERN al CNAF. La topologia di rete per i test a 10 Gb/s su WAN è costituita da due diverse organizzazioni: la rete GARR[1)] e la rete GEANT2[2)].

GEANT2 è la rete pan europea multi-gigabit per data communications più grande mai realizzata per la comunità accademica europea, ed è stata concepita ed adattata alle esigenze in continua evoluzione del mondo della ricerca. Sarà in grado di mettere a disposizione dei ricercatori capacità migliorate per gestire quantità enormi di dati generati da alcune iniziative

scientifiche all'avanguardia – che includono il nuovo Large Hadron Collider al CERN e i radio telescopi dislocati in varie località europee – senza compromettere l'alto livello qualitativo del servizio che sarà fornito alla comunità scientifica nel suo insieme. La rete GEANT2 servirà 3 milioni di ricercatori in 34 paesi. La topologia finale avrà un totale di 44 percorsi, usando un mix di dark fibre e di circuiti affittati: inizialmente 18 percorsi saranno connessi con dark fibre e 26 link useranno circuiti affittati. Diverse lunghezze d'onda a 10Gbps saranno utilizzate nel nucleo della rete. Ciascuna lunghezza è utilizzata per creare dei circuiti ottici privati virtuali (LHC optical private network - LHCOPN). Il percorso che abbiamo testato è quello evidenziato in giallo (connessione CERN - CNAF). In **Fig. 1** vengono messi in evidenza i vari LHCOPN che connettono il Tier-0 ai singoli Tier-1.

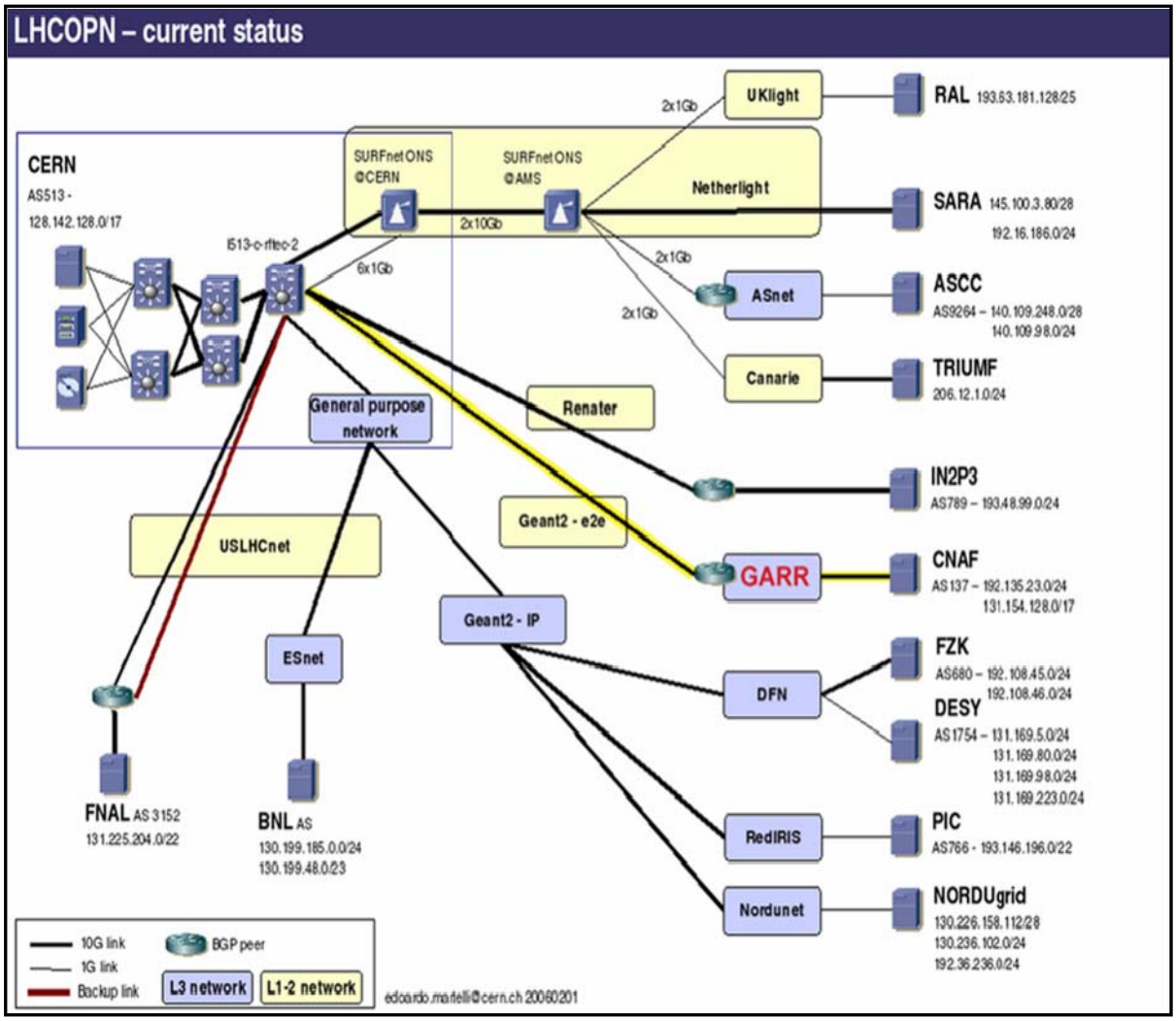

Fig. 1: LHC Optical Private Network.

In particolare l'intera infrastruttura di rete era così costituita (linea gialla di **Fig. 2**): la LAN a 10 GE del CNAF è interfacciata sulla rete GARR attraverso un router M320. Il POP di Bologna (RT1.BO1) è connesso al PoP di Milano (RT.MI1) mediante l'utilizzo di connessioni affittate da operatori di servizio nazionali e illuminate da una specifica lunghezza d'onda e switch STM-64 che utilizzano tecnologia SDH. Il POP di Milano si interfaccia alla rete europea per la ricerca, GEANT2, in particolare è connesso al POP GEANT2-IT tramite dark

fiber e switch SDH. Per connettere il POP GEANT2-IT con il POP GEANT2-CH si utilizza tecnologia DWDM. Infine, tramite dark fiber e switch SDH si giunge nella LAN del CERN.

Assegnando una predefinita lunghezza d'onda ad ogni percorso tra il CERN e i Tier-2 → Tier-1 si creano dei circuiti ottici privati (la lunghezza d'onda è usata solo per quel percorso) e virtuali (riconfigurando gli switch coinvolti i percorsi possono cambiare). In questo modo le connessioni tra Tier-0 e Tier-1 sono equiparabili a connessioni punto-punto.

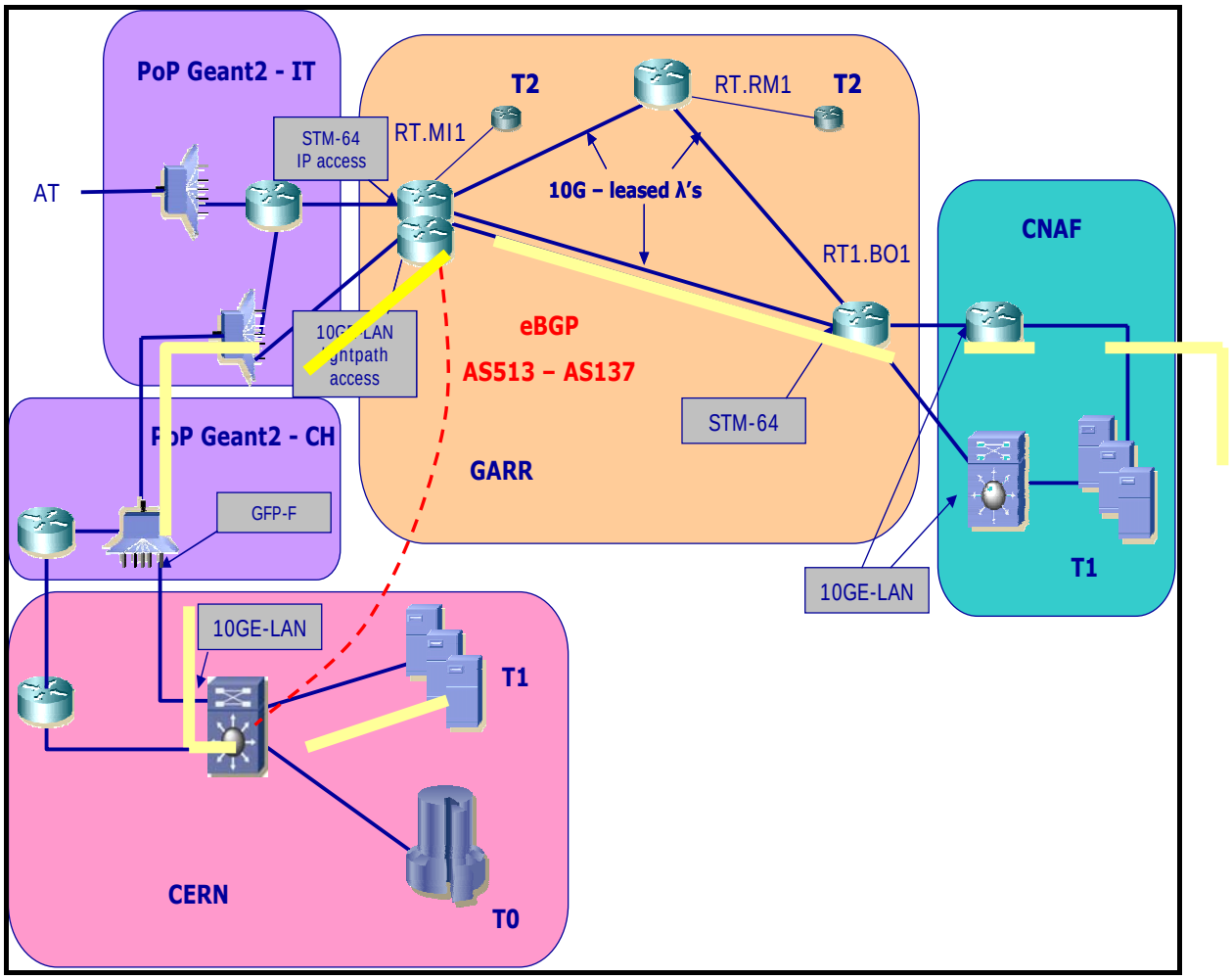

Fig. 2: Topologia rete WAN 10 Giga Ethernet di test.

3 PARAMETRI DI CONFIGURAZIONE

3.1 Hardware

Il bus PCI-X, così come quello PCI, permette all'utente di configurare una serie di parametri in modo da rendere il sistema il più adatto possibile alle proprie esigenze. In particolare daremo una descrizione di un parametro denominato Maximum Memory Read Byte Count (mmbrc)[3]. Il registro Maximum Memory Read Byte Count definisce il massimo numero di byte che possono essere trasferiti su un BUS PCI-X dalla memoria della scheda di rete, per ogni sequenza di lettura/scrittura di dati, in seguito ad un comando burst memory

read. Al software del sistema di configurazione è permesso sovrascrivere il valore di questo registro in ogni istante. Il valore di default è di 512 B.

Un dispositivo leggendo o scrivendo un grande blocco di dati può usare transizioni di ogni dimensione compresa tra 1 e 4096 byte. In generale, però, le prestazioni sono migliori se il dispositivo usa come dimensione del blocco la più grande che è permessa. In questo modo la mole dell'overhead diventa trascurabile rispetto a quella dei dati effettivi. Il valore più recente del registro è usato ogni volta che il dispositivo inizia una sequenza. In alcuni casi, se il dispositivo aveva già preparato qualche sequenza con il precedente valore di mmbrc, ma non le aveva ancora iniziate, queste inizieranno con il nuovo valore impostato.

3.2 Applicazione

A livello applicativo è possibile configurare la dimensione dei dati che l'applicazione legge/scrive dalla/nella memoria del kernel. Ad esempio il funzionamento del generatore di traffico iperf[4)] consiste nello scrivere in memoria degli array di dati di una certa dimensione. In iperf il valore di tale dimensione può essere configurato esplicitamente mediante l'opzione –l.

Dal punto di vista sperimentale abbiamo testato l'impatto di questi due parametri sul massimo valore di throughput ottenibile. Il grafico (Fig. 3) illustra l'andamento del throughput al variare del parametro mmbrc mittente e della dimensione del buffer di lettura/scrittura su RAM da parte dell'applicazione (application read/write block size). I dati ottenuti dai test evidenziano (Fig. 3) come un valore troppo basso della dimensione del blocco di dati scritto dall'applicazione in memoria e/o del parametro mmbrc causa un sottoutilizzo della CPU a causa di un elevato numero di stati di inattività della CPU con un suo conseguente sottoutilizzo e un abbassamento delle performance in generale. La saturazione dal punto di vista grafico è espressa dall'andamento a "ginocchio" delle curve: il ginocchio si forma laddove la CPU è utilizzata al 100%; oltre a tale punto le prestazioni non possono più aumentare e si mantengono quindi costanti così come è evidenziato dall'andamento piatto delle curve dopo il ginocchio. Ne consegue che per raggiungere le prestazioni massime sia necessario utilizzare un valore di mmbrc pari a 4096 B e un valore di application read/write block pari a 5000 B.

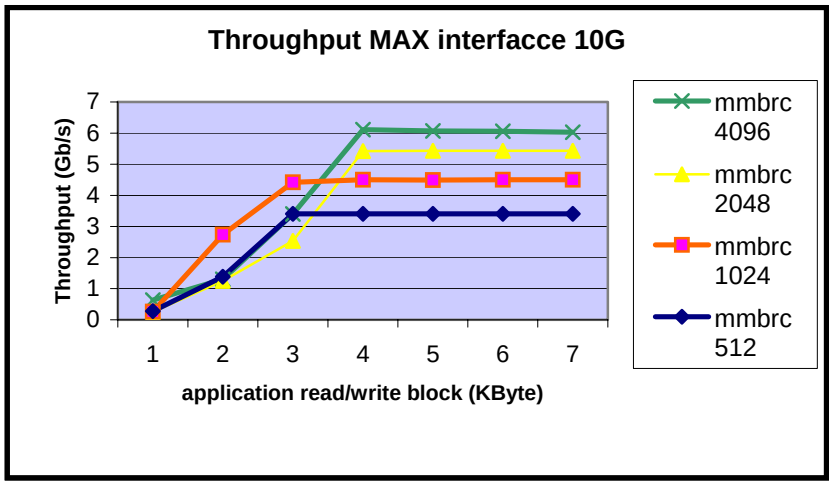

Fig. 3: Relazione tra il throughput e il parametro hardware mmbrc.

3.3 Kernel

Sono stati individuati diversi parametri del kernel estremamente significativi al fine di massimizzare l'efficacia del processo trasmissivo a 10 Gb/s, in particolare la dimensione delle code txqueuelen e backlog, la dimensione dei receive e transmit socket buffer.

3.3.1 Socket buffer

Definiamo due aree di memoria RAM destinate al protocollo TCP al fine di memorizzare dati che sono stati ricevuti o da trasmettere[5)]:

- Send buffer: area di memoria in cui TCP pone i messaggi in attesa di trasmissione; il send buffer è anche detto "send socket buffer".
- Receive buffer: area di memoria del sistema operativo in cui TCP pone i messaggi ricevuti; il receive buffer è anche detto "receive socket buffer".

Per ogni nuova connessione vengono allocati una nuova coppia di send e receive buffer.
Il valore ottimale del socket buffer size[6)] (B) è due volte il prodotto tra la capacità del link (banwidth B/s) e One-way Delay (s):

$$\text{buffer size} = 2 * \text{bandwidth} * \text{One-way Delay} \qquad (1)$$

Il valore di default della dimensione dei socket buffer, sia di ricezione che di trasmissione, nel kernel di Linux è di 64 KB. La dimensione dei socket buffer può essere configurata dell'applicazione, tuttavia tale dimensione non può eccedere le dimensioni massime definite nella fase di configurazione del kernel. In particolare in Linux 2.4 e 2.6 il kernel assegna ai buffer una dimensione pari al doppio del valore definito dall'applicazione, oppure assegna il valore di default memorizzato nel kernel se l'applicazione non ha possibilità di agire sulla dimensione di tale buffer.

Dal punto di vista sperimentale abbiamo verificato la dimensione minima dei socket di trasmissione e ricezione necessaria per ottimizzare il throughput di un singolo flusso TCP. Variando la dimensione della window, mediante il comando iperf –w, è possibile verificare a quale valore si raggiunge il massimo valore di throughput (Fig. 4).

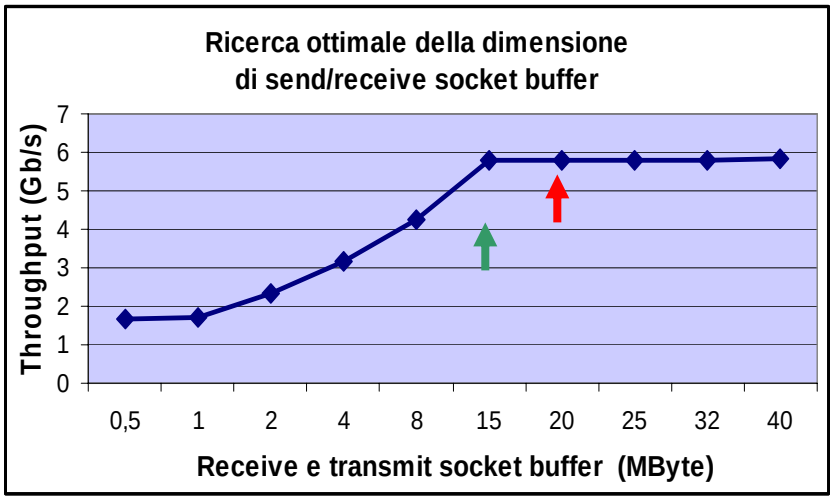

Fig. 4: Ricerca della dimensione ottimale di send/receive socket buffer.

I risultati di **Fig. 4** si riferiscono ad un test in cui si è utilizzato iperf come generatore di traffico, un solo flusso, una MTU pari a 9000 B e txqueuelen e coda backlog di lunghezza pari a 10000 pacchetti. Abbiamo quindi verificato tali risultati ottenuti con i valori indicati dalla consueta formula ed è stato notato che è necessario considerare il valore di RTT a carico, cioè quando il link è pienamente utilizzato, al fine di un corretto dimensionamento dei buffer. Di seguito si riportano i conti rispettivamente con RTT in assenza e in presenza di traffico.

RTT = 12 ms (in condizione di assenza di traffico)
W = (BandaMax * RTT)/8 = (6 Gb/s * 12 ms)/8 = 9 MB (2)

RTT = 20 ms (in condizione di presenza di traffico)
W = (BandaMax * RTT)/8 = (6 Gb/s * 20 ms)/8 = 15 MB (3)

Come risulta evidente dai dati riportati nella **Fig. 4**, freccia verde per indicare il valore di throughput ottenibile con buffer di dimensioni calcolate con RTT in assenza di traffico (2) e freccia rossa con RTT in presenza di traffico (3), è confermato il fatto che per un corretto dimensionamento dei socket buffer è indispensabile considerare esclusivamente il parametro RTT quando misurato in condizioni di carico della rete.

3.3.2 Transmission queue length

La dimensione della coda di trasmissione (transmition queue length[7]) determina la massima quantità di pacchetti che possono essere memorizzati all'interno del kernel. Quando un'applicazione necessita di inviare una serie di dati e quando ciò è permesso dalla dimensione del send socket buffer, i dati vengono accodati nella RAM dell'interfaccia di rete, per poi essere spediti in rete ad una velocità che dipende dal tipo di interfaccia utilizzata.

Più è grande la coda, più pacchetti possono essere memorizzati e minori sono quelli perduti. Nel protocollo TCP un eventuale riempimento di questa coda causa perdita di pacchetti, conseguentemente TCP attiverà i propri meccanismi controllo della congestione. Specialmente nelle connessioni a grande distanza questo parametro deve essere ben configurato in modo da permettere all'applicazione di utilizzare pienamente la banda disponibile sul link di collegamento tra sorgente e destinazione.

3.3.3 Backlog queue

Questo parametro è molto simile al precedente, ma si riferisce al meccanismo di ricezioni dei pacchetti.

La dimensione di default della backlog queue[7] è 300 pacchetti, ma tale parametro è configurabile, per esempio, nel file sysctl.conf.

Le statistiche della coda backlog sono disponibili in /proc/net/softnet_stat. Il formato delle statistiche è definito in net/core/dev.c. C'è una linea per ogni CPU in quanto ad ogni CPU viene assegnata una distinta coda backlog, ad esempio:

015e4a05 00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000
00001feb 00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000

I risultati ottenuti, riportati nella **Fig. 5**, si riferiscono ad un test in LAN in cui si è utilizzato iperf come generatore di traffico, un solo flusso, una dimensione di socket buffer pari a 4 MB, una MTU pari a 1500/9000 B e txqueuelen di lunghezza pari a 1000 pacchetti,

la dimensione della coda backlog è stata invece fatta variare al fine di individuarne il valore ottimale.

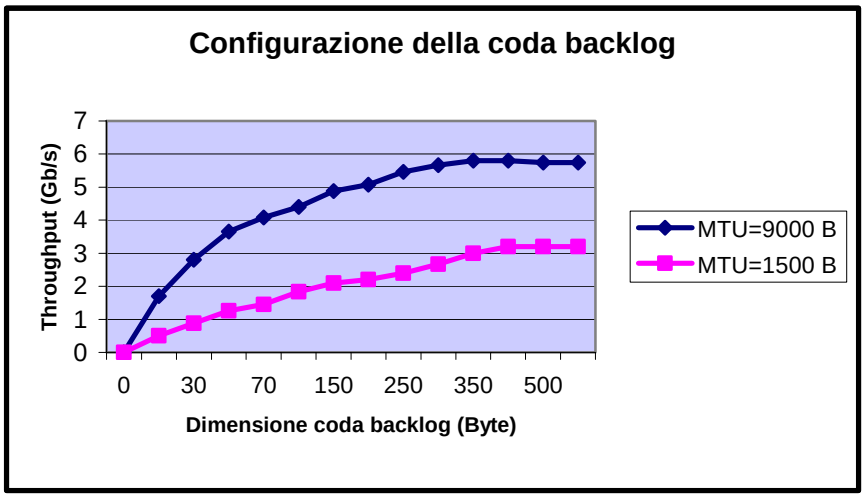

Fig. 5: Configurazione della coda backlog in LAN.

Dall'andamento del grafico si evidenzia che il valore di default non è sufficiente per ottenere il massimo throughput. La dimensione della coda per ottenere le prestazioni migliori è nell'intorno di 500 pacchetti. E' stato anche notato che la dipendenza delle performance dalla dimensione della coda backlog e txqueuelen sia molto più evidente nel kernel 2.4 rispetto alla versione 2.6. Si ipotizza che nella versione 2.6 siano stati implementati meccanismi di autoconfigurazione delle code.

4 CONFRONTO TRA INTERFACCE 10 GIGA ETHERNET

Nel lavoro svolto sono state utilizzate e comparate due differenti interfacce di rete 10 Giga Ethernet, Intel e Chelsio, che implementano diversi meccanismi per alleggerire il lavoro svolto dalla CPU di cui di seguito ne verrà data una breve descrizione.

4.1 INTEL PRO10Gbe – SR

Nelle schede Intel[8)] testate era implementato il meccanismo TSO (TCP Segmentation Offload) che consiste nella segmentazione di grandi pacchetti TCP e del checksum TCP in ricezione e trasmissione a livello di scheda si rete, sgravando la CPU da queste mansioni.

Al fine di ottenere le massime prestazioni è stato necessario utilizzare un valore di MTU pari a 9000 B. I risultati ottenuti sono stati raggruppati in **Tabella 1** e classificati in base alla topologia della rete e al numero di flussi in parallelo attivati per ottenere quel particolare risultato:

Topologia rete	1 flusso	2 flussi
LAN	5,8 Gb/s	6,8 Gb/s
WAN	6,2 Gb/s	6,5 Gb/s

Tabella 1: Prestazioni ottenute con schede Intel.

4.2 CHELSIO T210 – SR T110 – SR

Nelle schede Chelsio[9)] testate era implementato il meccanismo TOE (TCP Offload Engine) che consiste nel processare tutto il traffico TCP/IP a livello hardware nella scheda di rete, consentendo quindi un forte risparmio di risorse in termini di utilizzo di CPU.

Vi è da precisare che nei test in WAN (CNAF e CERN) ricevente e mittente non disponevano dello stesso modello di interfacce Chelsio, in particolare al CNAF erano disponibili schede denominate T210-SR, mentre al CERN erano disponibili schede denominate T110-SR che garantiscono prestazione leggermente inferiori rispetto alle prime (7,1 Gb/s a lato CERN contro 7,4 Gb/s a lato CNAF). Al fine di ottenere le massime prestazioni è stato sufficiente un valore di MTU pari a 1500 B.

I risultati ottenuti sono stati raggruppati in **Tabella 2** e classificati in base alla topologia della rete e al numero di flussi in parallelo attivati per ottenere quel particolare risultato:

Topologia rete	1 flusso	2 flussi
LAN	7,4 Gb/s	7,4 Gb/s
WAN	6,2 Gb/s	6,9 Gb/s

Tabella 2: Prestazioni ottenute con schede Chelsio.

Al fine di raggiungere i risultati citati con le schede Chelsio riportiamo di seguito i comandi utilizzati, i parametri che sono stati configurati e i relativi valori:

```
toe.toe0_tom.override_tx_wscale = 0
toe.toe0_tom.tx_lease_time = 9
toe.toe0_tom.soft_backlog_limit = 0
toe.toe0_tom.max_conn = -1
toe.toe0_tom.delayed_ack = 1
```

toe.toe0_tom.mss = 12272
toe.toe0_tom.auto_passv_open = 0
toe.toe0_tom.min_newconn_tx_pages = 2
toe.toe0_tom.rx_credit_thres = 4096
toe.toe0_tom.max_tx_pages = 168
toe.toe0_tom.tx_hold_thres = 0
toe.toe0_tom.max_host_sndbuf = 32768

Il buffer di trasmissione associato ad una connessione è diviso in 2 parti, una risiede nella memoria della interfaccia di rete e la restante parte nella RAM dell'host:

- max_tx_pages è il parametro che indica la dimensione della memoria che risiede sulla scheda ed è espressa in pagine di memoria (64 KB). Il minimo valore possibile è pari a 2 pagine ed è anche il valore di default.
- max_host_sndbuf è il parametro che indica la dimensione della memoria che risiede nella RAM dell'host. Il valore di default è pari a 32 KB.

register 0x4a4 – controlla la rate alla quale la scheda invia i dati – valore utilizzato 0x150

register 0x3040 – controlla il "Inter-Packet Gap" (IPG)- valore utilizzato 0x3e703e70

more /proc/net/toe/toe0/snmp - per visualizzare le statistiche SNMP (Simple Network Management Protocol14)) in modo da verificare la presenza di eventuali "TCP Retransmitted Segment "

ethtool –S eth0 - a lato ricevente per verificare, tra l'altro, se sono state inviate delle richieste di pausa al trasmittente (TX pause frames) il chè indicherebbe che il trasmittente invia frame ad una rate troppo elevata per il ricevente.

Il grafico seguente (**Fig. 6**) dà una comparazione dell'utilizzo della CPU a seconda della scheda di rete utilizzata in funzione del valore di mmbrc.

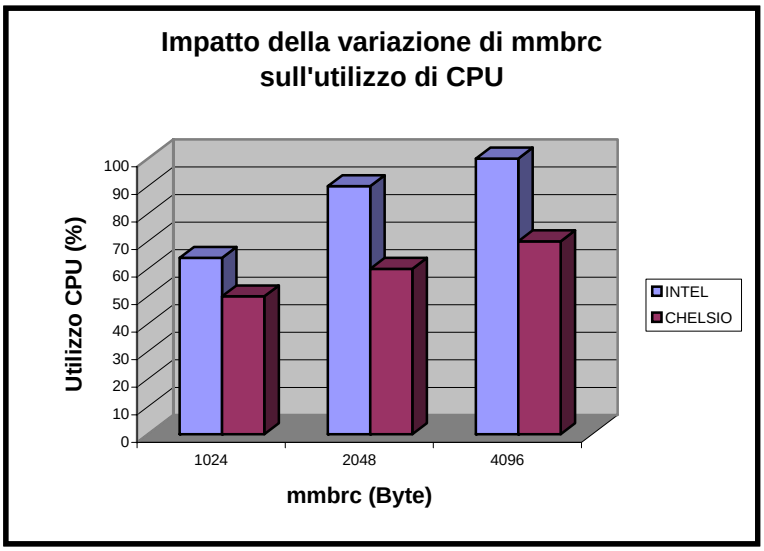

Fig. 6: Comparazione dell'utilizzo di CPU tra schede Intel e Chelsio.

Appare evidente come nel caso delle schede Intel il collo di bottiglia sia rappresentato dalla CPU che già a 6,2 Gb/s con singolo flusso è sfruttata al 100% della sua potenza di calcolo. Al contrario utilizzando le schede Chelsio la totale processazione del traffico TCP-IP viene fatta a livello di scheda liberando, la CPU da tale lavoro questa, a parità di throughput, viene sfruttata al 70% lasciando ancora libero un margine per aumentare le performance. Inoltre le schede Chelsio, con l'utilizzo di MTU pari a 1500 B, permettono di raggiungere alte prestazioni in qualsiasi topologia di rete. Lo svantaggio maggiore delle schedio Chelsio è rappresentato dal fatto che la dimensione della window deve essere configurata sulla scheda e quindi è costante e non è soggetta al meccanismo di autotuning del kernel[10]. Al contrario nelle schede Intel permettono di configurare la dimensione dei receive e transmit socket buffer nel kernel e quindi è possibile usufruire del meccanismo di autotuning con conseguente risparmio di risorse utilizzate.

5 CONFRONTO TRA PROTOCOLLI DI TRASPORTO: TCP-RENO VS TCP-BIC

Le reti a larga banda con grandi ritardi sono il tipico scenario in cui il protocollo di trasporto TCP potrebbe incontrare problemi ad utilizzare completamente la banda disponibile.

Il protocollo TCP-Reno aumenta la propria congestion window di 1 MSS ad ogni RTT e la dimezza ogni volta che una perdita si manifesta. In questa maniera, se consideriamo una connessione a 10 Gb/s con pacchetti di dimensione pari a 1500 B, serviranno 83.333 intervalli RTT affinché la window sia cresciuta a sufficienza per sfruttare a pieno la banda. Con un RTT di 100 ms, tale tempo aumenta a 1,5 ore. Inoltre, perché ciò avvenga in questi tempi, la probabilità di perdita (Pe) deve essere 1 pacchetto ogni 5 miliardi trasmessi (Pe = 0,2*10-9), un dato che è molto minore del limite teorico del bit error rate della rete. Quindi uno dei problemi del TCP è il lungo tempo di convergenza nelle reti ad alta velocità; un preciso e ben fatto tuning dei parametri TCP, ad esempio send/receive socket buffer e della dimensione delle code di trasmissione e backlog per ogni interfaccia, possono eliminare questo problema. Una semplice soluzione è aumentare la dimensione dei pacchetti usando i jumbo frame e usare più flussi in parallelo.

Sebbene questi approcci migliorino le prestazioni, non assicurano in caso di protocolli TCP estesi, la TCP friendliness (un flusso è normalmente considerato TCP-friendly se si comporta, in caso di congestione, allo stesso modo di un flusso TCP standard) quando si è in situazione di probabilità di perdita tra 10^{-2} e 10^{-4} (definito come il range di "buon - comportamento" del protocollo TCP standard), perché la velocità con cui la window cresce è tipicamente superiore al TCP. Garantire contemporaneamente friendliness e bandwidth scalability (l'abilità nel raggiungere un certo valore di traffico in presenza di una ragionevole probabilità di perdita con una velocità di crescita della window fissa) è una sfida.

Sono stati suggeriti diversi algoritmi di controllo di congestione per risolvere questo problema[11]: High Speed TCP (HSTCP), Scalable TCP (STCP), FAST, ecc. Questi protocolli aggiustano la rate di crescita della finestra basandosi sulla sua dimensione corrente. In questo modo più è grande la congestion window, più velocemente questa cresce. Questi protocolli sono TCP friendly in quei scenari in cui la rate di perdita è molto elevata e sono scalabili in quelli in cui le perdite sono basse, ma non soddisfano la caratteristica di RTT fairness.

È davvero impegnativo costruire un protocollo che soddisfi tutte queste caratteristiche: RTT fairness, TCP friendliness e scalabilità. Inanzitutto è bene specificare che nessun protocollo potrà garantire queste tre caratteristiche contemporaneamente per qualsiasi frequenza di perdita di pacchetti.

In particolare prendiamo in esame il protocollo chiamato Binary Increase TCP (BI-

TCP)[12)], costituito da due algoritmi:

- crescita “binary search”
- crescita “additiva”

che nei prossimi due capitoli descriveremo in dettaglio.

5.1 Crescita “binary search”

La minima window corrente può essere stimata come la dimensione della window alla quale non si verifica nessuna perdita di pacchetti. Se la dimensione massima della window è conosciuta, allora possiamo applicare la tecnica chiamata binary search per fissare la dimensione della “target window”, che in BIC viene posta a metà tra il valore massimo e quello minimo. Se si verificano perdite di pacchetti, la dimensione corrente della window diventa il nuovo massimo e la window ridotta dopo la perdita sarà il nuovo minimo. La metà tra questi nuovi valori diventerà il nuovo target.

La giustificazione di questo approccio è che la rete tipicamente è soggetta a perdite nell’intorno del nuovo massimo ma non nell’intorno del nuovo minimo, ne consegue che il nuovo target può essere stimato a metà tra i due estremi. Se non si verificano perdite, la dimensione corrente della window diventa il nuovo minimo e un nuovo target viene calcolato. Questo processo è ripetuto finché la differenza tra il massimo e il minimo è al di sotto di un valore di soglia Smin.

Questa tecnica è chiamata crescita “binary search” e permette una ricerca della massima larghezza di banda in modo aggressivo inizialmente (cioè quando la differenza tra la corrente dimensione della window e la dimensione del target window è grande) e in modo meno aggressivo mano a mano che ci si avvicina al valore di target. Ne consegue che la funzione di crescita della dimensione della window è di tipo logaritmico, ovvero la pendenza della curva (**Fig. 7**) gradatamente decresce quando si è vicini al valore di target. Gli altri protocolli tendono ad aumentare la propria rate in modo tale che al punto di saturazione l’incremento è al massimo. Tipicamente il numero di pacchetti perduti è proporzionale alla dimensione dell’ultimo incremento della window prima della perdita. Ne consegue che questa tecnica riduce il numero di pacchetti perduti. Per esempio, assumendo una congestion window di 128 pacchetti, in caso di perdita, i parametri vengono così ricalcolati:

Min_Window=128/2= 64
Target = (Max_Window + Min_Window)/2.

In altre parole la congestion window evolve in questo modo:

64 → 96 → 112 → 120 → 124 → 126 → 127

La fig. 7 evidenzia la differenza tra l’incremento lineare e quello di tipo binario e si intuisce subito come quest’ultimo sia più aggressivo nella fase iniziale.

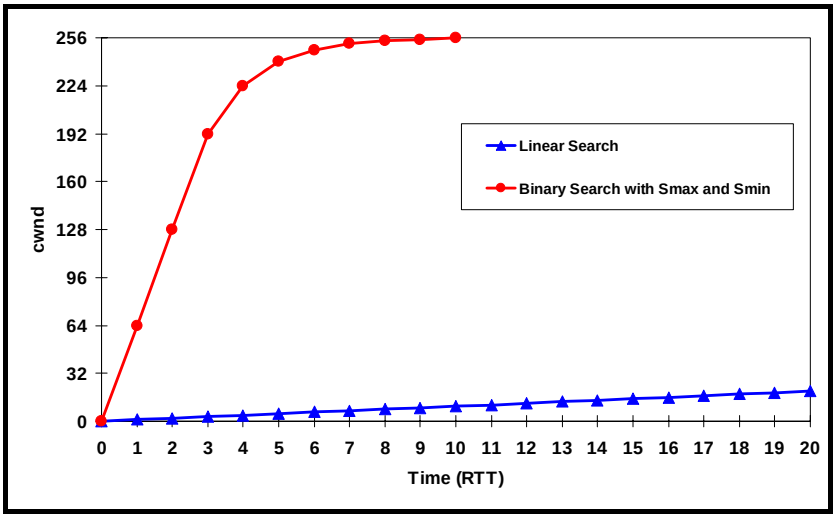

Fig. 7: Confronto tra crescita lineare e crescita "binary search" della finestra di congestione.

5.2 Crescita "additiva"

Per assicurare una rapida convergenza e RTT-fairness, si aggiunge alla binary search increase un'altra strategia di incremento della window denominata additive increase.

Quando tra la dimensione corrente della window minima e il punto intermedio (tra il suo valore massimo e quello minimo) c'è una distanza elevata, aumentare la dimensione della window direttamente al punto intermedio potrebbe comportare una crescita troppo aggressiva. Si fissa quindi un valore Smax che limite superiormente l'incremento della dimensione della window.

Invece che aumentare la window direttamente al punto intermedio al prossimo RTT, la si aumenta di un valore pari a Smax finché la distanza tra window minima e punto intermedio, non è inferiore di Smax. Dopo di questo, l'incremento torna ad essere al punto intermedio.

La conseguenza dell'aggiunta di questa tecnica è che dopo una forte riduzione della window, la funzione di crescita è inizialmente di tipo lineare per poi diventare logaritmica.

La combinazione di queste due tecniche viene detta binary increase ed è riassumibile dal punto di vista del codice in questo modo:

In caso di perdita:

max' = cwnd
min' = cwnd / 2
ta rget = (max' + min') / 2

In caso di crescita (while max – min < S_min):

max' = max
min' = cwnd
arget = (max' + min') / 2
f target – cwnd > S_max
Cwnd' = cwnd + S_max (additive increase)

Dal punto di vista sperimentale abbiamo testato la differenza di rapidità di convergenza tra i due protocolli in esame. Disponendo al CNAF di una macchina con kernel 2.4 e TCP Reno e di un'altra che dispone di kernel 2.6 e TCP BIC, il confronto è stato effettuato come mostrato nella **Fig. 8** in cui un solo flusso alla volta è stato attivato.

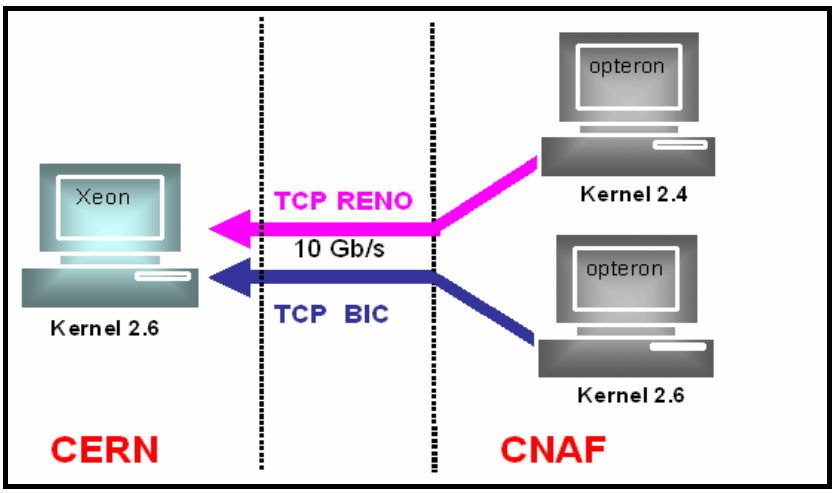

Fig. 8: Topologia di rete per i test di rapidità di convergenza di TCP BIC.

I risultati ottenuti, riportati nella **Fig. 9**, si riferiscono ad un test in cui si è utilizzato iperf come generatore di traffico, un solo flusso, una dimensione di socket buffer pari a 20 MB una dimensione sufficiente per massimizzare le prestazioni), una MTU pari a 9000 B e txqueuelen e coda backlog di lunghezza pari a 10000 pacchetti.

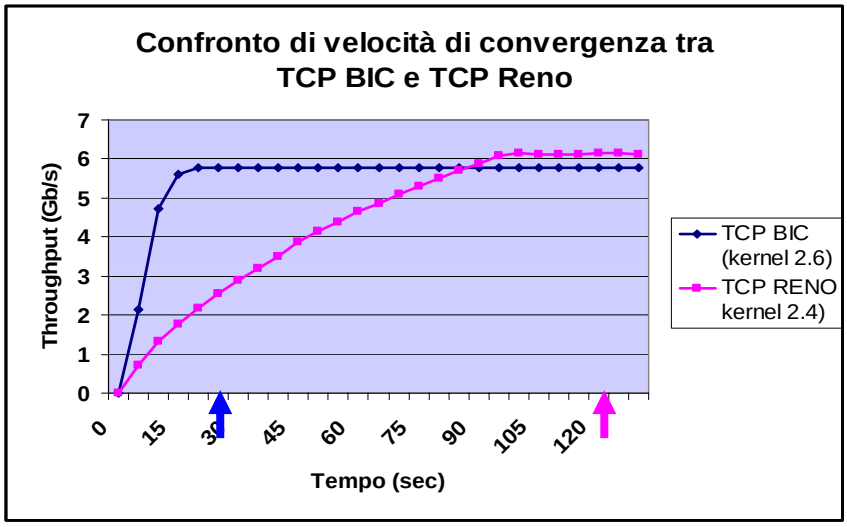

Fig. 9: Confronto di velocità tra TCP BIC e TCP Reno.

Come sostenuto dalla trattazione teorica, i risultati dei test confermano che i meccanismi implementati nel BIC (incremento binary search e incremento additivo) garantiscono una convergenza verso il valore massimo di throughput più di cinque volte più rapida rispetto alla versione Reno: in **Fig. 9** è mostrato come TCP BIC converga circa dopo 20 secondi (freccia blu) contro i 110 secondi (freccia rosa). Questa proprietà è estremamente importante, in quanto garantisce una più elevata capacità di recupero in caso di perdite sporadiche, e quindi un maggior efficiente utilizzo delle risorse offerte dalla connessione.

6 CONCLUSIONI

L'obiettivo del lavoro è stato quello di verificare le prestazioni e le configurazioni di vari elementi di un sistema di comunicazione a 10 Gb/s: server, interfacce di rete per nodi terminali, linee locali e geografiche sperimentali; al fine di individuare una configurazione ottimale che permetta l'ottimizzazione delle prestazioni di un'applicazione basata sul protocollo di trasmissione TCP.

Sono stati quindi individuati una serie di parametri chiave sia di tipo hardware che software, la cui configurazione risulta particolarmente rilevante.

A livello del bus interno PCI-X di un server è stata dimostrata l'importanza di un parametro, il maximum memory byte count, che indica il massimo numero di byte che possono essere trasferiti su un bus PCI-X dalla memoria della scheda di rete per ogni sequenza. Abbiamo verificato che un valore alto di questo parametro permette di migliorare le prestazioni perché rende il rapporto dati-overhead più vantaggioso è perché in questo modo si riesce ad utilizzare a pieno la capacità di calcolo della CPU.

Sono inoltre stati comparati in termini di prestazione e di utilizzo della CPU, due diversi kernel Linux: le versioni 2.6 e 2.4. Sono stati individuati diversi parametri del kernel estremamente significativi al fine di massimizzare l'efficacia del processo trasmissivo a 10 Gb/s, in particolare: la dimensione delle code txqueuelen e backlog, la dimensione dei receive e transmit socket buffer.

La txqueuelen e backlog sono rispettivamente la coda di trasmissione e ricezione che interfacciano la memoria RAM della macchina alla memoria RAM della scheda di rete (NIC). Una corretta configurazione delle di queste code è essenziale per evitare fenomeni di perdita di pacchetti all'interno dei server di trasmissione e ricezione che causano un abbassamento delle performance.

Con il termine receive e transmit socket buffer si intende la quantità di memoria che la CPU riserva, ad ogni singola connessione, all'interno della memoria RAM del server. Un corretto dimensionamento di questo parametro è fondamentale sia per poter fare sì che la capacità del link venga totalmente sfruttata, sia per avere un corretto bilanciamento tra prestazioni raggiunte e quantità di risorse di sistema utilizzate.

Inoltre, sono state confrontate le prestazioni di due diverse implementazioni del protocollo TCP: TCP Reno e TCP BIC. TCP BIC è una versione ottimizzata del protocollo, esplicitamente introdotta nei kernel Linux 2.6 per migliorare le prestazioni di un flusso TCP in caso di scenari basati su cammini di rate a larghissima banda e altro ritardo trasmissivo. Particolare attenzione è stata posta alle caratteristiche di convergenza.

Abbiamo sperimentato che il protocollo TCP BIC risulta avere una rapidità di convergenza circa sei volte migliore rispetto al TCP Reno. Questa proprietà è estremamente importante, in quanto garantisce una più elevata capacità di recupero in caso di perdite sporadiche, e quindi un maggior efficiente utilizzo delle risorse offerte dalla connessione.

Anche alcune caratteristiche del Kernel 2.6 sono risultate importanti nel tentativo di raggiungere le massime prestazioni. È stato infatti verificato che la versione 2.6 del kernel ha un utilizzo della CPU di circa il 15% superiore rispetto alla versione 2.4. Ne deriva che l'attivazione di un unico flusso in un kernel 2.6 porta la CPU alla saturazione ad un valore di throughput più basso rispetto allo stesso flusso attivato nel kernel 2.4. Ciò non avviene però quando i flussi attivati sono molteplici. Generalmente, l'attivazione di flussi multipli su un dato nodo trasmissivo o in ricezione, comporta un maggiore utilizzo di risorse di calcolo e una leggera perdita di efficienza di throughput complessiva, a causa della necessità di effettuare un elevato numero di "context switch" a livello di scheduler, tra i vari processi attivi. Il fatto che il kernel 2.6 permetta di raggiungere throughput aggregati superiori nel caso di molteplici

flussi, può essere attribuito alla maggiore efficienza degli algoritmi di scheduling[13] implementati nella versione 2.6.

7 BIBLIOGRAFIA

(1) GARR (La Rete dell'Università e della Ricerca Scientifica Italiana)
http://www.garr.it

(2) GEANT2 (La Rete dell'Educazione e della Ricerca Scientifica Europea)
http://www.geant2.net

(3) PCI-X Addendum to the PCI Local Bus Specification

(4) Iperf Version 2.0.2
http://dast.nlanr.net/Projects/Iperf/

(5) TCP: Transport Control Protocol – Tiziana Ferrari
www.cnaf.infn.it/~ferrari/seminari/fe-sys-**di**st/**tcp**-fe.ppt

(6) TCP Tuning Guide
http://dsd.lbl.gov/TCP-tuning/

(7) Technical Report DataTAG-2004-1 FP5/IST DataTAG Project:
A Map of the Networking Code in Linux Kernel 2.4.20 - *M. Rio et al.*
http://datatag.web.cern.ch/datatag/papers/tr-datatag-2004-1.doc

(8) INTEL PRO10Gbe – SR
http://www.intel.com/cd/network/connectivity/emea/ita/169513.htm

(9) CHELSIO T210 – SR T110 – SR
http://www.chelsio.com/products/T210.php

(10) TCP auto-tuning zoo
http://www.csm.ornl.gov/~dunigan/net100/auto.html

(11) Evaluation of High-Speed TCP Proposals Michele Weigle
www.hamilton.ie/net/eval/results_HI2005.pdf

(12) Binary Increase Congestion Control (BIC) for Fast Long-Distance Networks
Lisong Xu, Khaled Harfoush, and Injong Rhee
www.csc.ncsu.edu/faculty/rhee/export/bitcp.pdf

(13) LINUX KERNEL 2.6 - Le nuove feature - Gian Paolo Ghilardi
http://carapax.crema.unimi.it/eventi/ld2003/pdf/ghilardi.pdf

(14) RFC 1157 - A Simple Network Management Protocol (SNMP)
http://www.ietf.org/rfc/rfc1157.txt