CAPITOLO 2
ANALISI DI STABILITA' E IMPLEMENTAZIONE

2.1 Il modello

Nell'analisi effettuata nel documento [Katabi03] è stato utilizzato un modello fluido del traffico per analizzare la stabilità di XCP. L'analisi consiste di un singolo link con capacità ***c*** attraversato da **N** flussi XCP; per semplicità è stato assunto che tutti i flussi abbiano lo stesso round trip delay **d** e un rate di invio pari a $r_i(t)$ dove ***i*** è l'***i***-esimo utente al tempo ***t***. Il rate complessivo del traffico è pari a:

$$y(t)=\sum r_i(t)\ ,$$

mentre il rate del traffico mescolato ***h(t)***(shuffled traffic) è pari a:

$$h(t)=0.1\cdot y(t).$$

Il router invia lo stesso feedback totale in ogni intervallo di controllo **d**. Il feedback raggiunge le sorgenti dopo un round trip time modificando la somma delle loro cwnd ($\sum w(t)$). Quindi il feedback cumulativo inviato per ogni unità di tempo è la somma della derivata della congestion window come segue:

$$\sum \frac{dw}{dt}=\frac{1}{d}(-\alpha\cdot d\cdot(y(t-d)-c)-\beta\cdot q(t-d))\ .\ [1]$$

Dato che il rate del traffico di imput è $y(t)=\sum \frac{w_i(t)}{d}$, la derivata del rate del traffico $\dot{y}(t)$ è :

$$\dot{y}(t)=\frac{1}{d^2}(-\alpha\cdot d\cdot(y(t-d)-c)-\beta\cdot q(t-d))\ .\qquad [2]$$

Trascurando la condizione in cui la coda sia limitata, complessivamente il sistema può essere rappresentato usando le seguenti equazioni differenziali:

$$\dot{q}(t)=y(t)-c\Rightarrow \text{ risolvendo l'equazione 2 } \Rightarrow$$

$$\Rightarrow \dot{y}(t)=-\frac{\alpha}{d}((y(t-d)-c)-\frac{\beta}{d^2}q(t-d))\Rightarrow \qquad [3]$$

$$\Rightarrow \dot{r}(t)=\frac{1}{N}([\dot{y}(t-d)]^+ + h(t-d))-\frac{r_i(t-d)}{y(t-d)}([-\dot{y}(t-d)]^+ + h(t-d)) \quad [4]$$

La notazione $[\dot{y}(t-d)]^+$ è equivalente a $\max(0,\dot{y}(t-d))$. L'equazione 4 esprime la politica AIMD adottata dal controllore d'imparzialità, cioè viene allocato lo stesso

feedback positivo per tutti i flussi, mentre il feedback negativo allocato è proporzionale al throughput corrente di ogni flusso.

Possiamo chiamare **x(t)** la differenza tra input rate e capacità del link (cioè, $x(t)=y(t)-c$). Quindi il sistema lineare seguente:

$$\dot{q}(t)=x(t)$$
$$\dot{x}(t)=-K_1\cdot x(t-d)-K_2\cdot q(t-d)\ ,$$

è stabile per ogni ritardo **d** > 0 se:

$$K_1=\frac{\alpha}{d} \quad \text{e} \quad K_2=\frac{\beta}{d^2}\ ,$$

dove **α** e **β** sono le costanti che rispettivamente soddisfano le seguenti relazioni:

$$0<\alpha<\frac{\pi}{4\sqrt{2}}$$
$$\beta=\alpha^2\sqrt{2}$$

Vediamo come si ricavano le due costanti su scritte. Il sistema può essere espresso usando un feedback ritardato come in figura seguente:

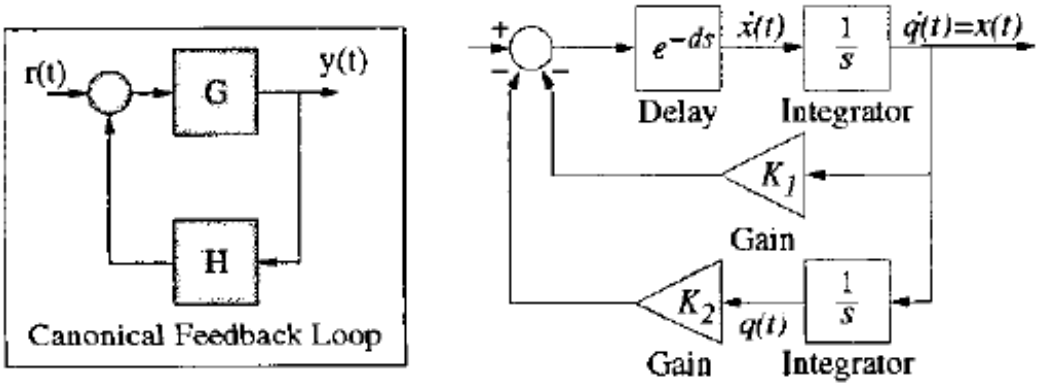

Fig. 2.1: schema a blocchi del sistema retroazionato

Il sistema rappresentato in fig. 2.1, ha la seguente funzione di trasferimento ad anello aperto con i rispettivi diagrammi di Bode e Nyquist:

$$G(s)=\frac{K_1\cdot s+K_2}{s^2}e^{-ds}\ ,$$

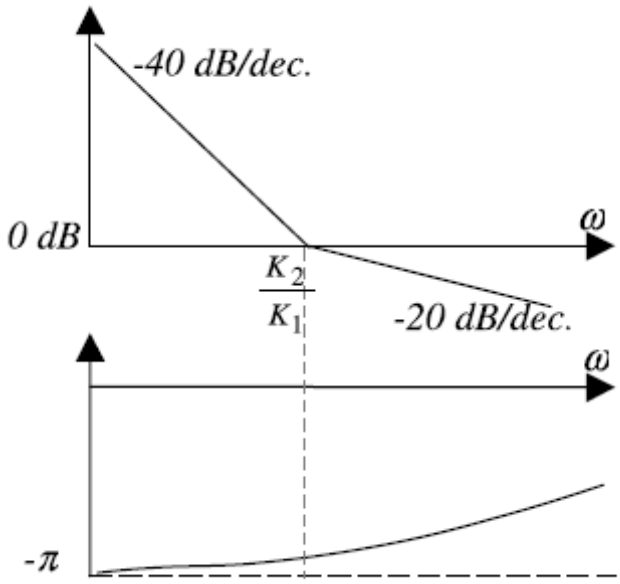

Fig. 2.2: diagramma di Bode della funzione ad anello aperto

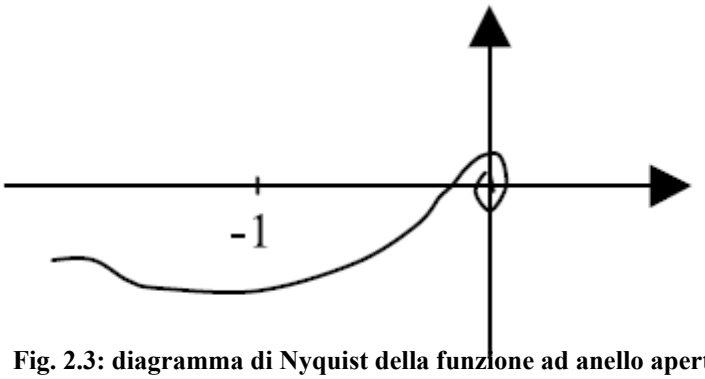

Fig. 2.3: diagramma di Nyquist della funzione ad anello aperto

Per piccoli valori del ritardo **d**, il sistema ad anello chiuso è stabile. Si può notare che il diagramma di Nyquist (fig. 2.3) non circonda -1. Inoltre, dal diagramma di Bode (fig. 2.2) si nota che il margine di fase rimane positivo indipendentemente dal ritardo. L'ampiezza e la fase della funzione di trasferimento ad anello aperto sono rispettivamente:

$$|G| = \frac{\sqrt{K_1{}^2 \cdot \omega^2 + K_2{}^2}}{\omega^2},$$

$$\angle G = -\pi + arctg\,\frac{\omega K_1}{K_2} - \omega \cdot d \ .$$

1

La frequenza di taglio si presenta su $\omega_t = \frac{K_2}{Kalignl\ ¿\ ¿_¿}$ ¿.

Per semplificare il sistema, si scelgono **α** e **β** tali che la frequenza di taglio ω_t sia uguale alla frequenza di crossover ω_c (frequenza con cui si ottiene ampiezza unitaria, $|G(\omega_c)|=1$).

Sostituendo $\omega_c=\omega_t=\frac{K_2}{K_1}$ in $|G(\omega_c)|=1$ si ottiene $\beta=\alpha^2\sqrt{2}$. Per mantenere la stabilità per alcuni ritardi, bisogna rendere il margine di fase sempre positivo ed indipendente dal ritardo **d**. Questo significa :

$$\angle G=-\pi+\frac{\pi}{4}-\frac{\beta}{\alpha}>-\pi\Rightarrow\frac{\beta}{\alpha}<\frac{\pi}{4}\ .$$

Sostituendo β con la relazione vista prima, si ottiene $\alpha<\frac{\pi}{4\sqrt{2}}$, in cui il margine di guadagno è unitario e il margine di fase sarà sempre positivo. Questa analisi è vera solo per alcuni ritardi **d**, capacità e numero di sorgenti.

2.2 Implementazione del router XCP

I calcoli del router sono suddivisi in tre parti: 1) fase in cui arrivano i pacchetti (on packet arrival); 2) fase in cui vengono inviati pacchetti (on packet departure); 3) fase in cui viene valutata la coda persistente (persistent queue) che utilizza un timer separato. I calcoli sono presentati qui di seguito [draft – falk – spec – 01].

2.2.1 Fase di arrivo dei pacchetti (on packet arrival)

Quando un pacchetto arriva al router, diversi parametri utilizzati da XCP necessitano di essere aggiornati. I passaggi sono descritti nel seguente codice:

on packet arrival do:

1. *input_traffic += Pkt_size*
2. *sum_x += X*
3. *if(Rtt < MAX_INTERVAL) then*
4. *sum_xrtt += X*Rtt*
5. *else*
6. *sum_xrtt += X*MAX_INTERVAL*

Riga 1): la variabile *input_traffic* accumula la quantità totale di informazione che arriva al router durante un intervallo di controllo. Quando arriva un pacchetto, la grandezza del pacchetto è presa dall'IP header ed è aggiunta al conto in corso.

Riga 2): la variabile *sum_x* è impiegata nel calcolo dell'intervallo di controllo e della capacità allocata. Per ogni pacchetto viene sommato il valore di X utilizzando

l'informazione della *congestion header*. È preferibile che *sum_x* sia memorizzato in una variabile intera a 64 bit.

Righe 3) e 5): si esegue un test per verificare se il round trip time del flusso supera il massimo valore permesso dell'intervallo di controllo. Se è così, il massimo intervallo di controllo (*MAX_INTERVAL*) è utilizzato nei calcoli successivi. Un intervallo di controllo troppo elevato causa errori consistenti per quanto riguarda l'allocazione di banda dei flussi.

Righe 4) e 6): come la riga 2), la variabile *sum_xrtt* è usata nel calcolo dell'intervallo di controllo. E' rappresentata da una variabile senza segno di 96 bit.

2.2.2 Calcolo dell'intervallo di controllo

Quando l'intervallo di controllo scade, devono essere aggiornate diverse variabili come mostrato di seguito. Noteremo che diversi calcoli presentano delle divisioni. Queste dovrebbero essere realizzate utilizzando calcoli in virgola mobile o interi con appropriate scale di rappresentazione per evitare sovra/sotto - allocazione di banda.

estimation – control timeout do:

7. *avg_rtt = sum_xrtt / sumx*
8. *input_bw = input_traffic / ctl_interval*
9. *F = α * (capacity – input_bw) – β * queue / avg_rtt*
10. *shuffled_traffic = shuffle_function(...)*
11. *residue_pos_fbk = shuffled_traffic + max(F,0)*
12. *residue_neg_fbk = shuffled_traffic + max(-F,0)*
13. *Cp = residue_pos_fbk / sum_x*
14. *Cn = residue_neg_fbk / input_traffic*
15. *input_traffic = 0*
16. *sum_x = 0*
17. *sum_xrtt = 0*
18. *ctl_interval = max(avg_rtt, MIN_INTERVAL)*
19. *timer.rescheduler(ctl_interval)*

Riga 7): aggiornamento dell'RTT medio *avg_rtt* dato dal rapporto tra le due sommatorie calcolate nella sezione precedente. Questo valore è utilizzato per determinare l'intervallo di controllo.

Riga 8): la banda media del traffico in arrivo si ricava dal rapporto tra i bytes ricevuti e l'intervallo di controllo precedente.

Riga 9): viene calcolato l'ammontare del feedback. La variabile *capacity* è la capacità di trasmissione dei pacchetti IP del link, misurata in bytes/secondi. la variabile *avg_rtt* (RTT medio) è stata calcolata nella riga 7). La variabile *queue* è la coda persistente che sarà definita successivamente. I valori **α** e **β** sono le costanti definite nei paragrafi precedenti. Come visto prima, la costante **α** deve essere un numero positivo minore di $\frac{\pi}{4\sqrt{2}}$; è consigliato utilizzare **0.4** come valore nominale. La costante **β,** invece, è pari a $\beta = \alpha^2\sqrt{2}$ (se si utilizza 0.4 come valore nominale di **α**, **β= 0.226)**. Notiamo che *F* può essere sia positivo che negativo.

Riga 10): in questa riga è stabilito l'ammontare della capacità che sarà mischiata (shuffled) nel successivo intervallo di controllo utilizzando la *shuffle_function.* Il mescolamento (shuffling) prende una piccola quantità di capacità accessibile e la ridistribuisce aggiungendola sia al feedback positivo che a quello negativo. Questo permette a nuovi flussi di acquisire capacità in un sistema a pieno carico. La funzione *shuffle_function* consigliata nel documento[draft – falk – spec – 01] è la seguente:

$$shuffled_{traffic} = \max(0{,}0.1 \cdot input_{bw} - |F|)$$

dove la variabile *input_bw* è stata definita nella riga 8). Si possono scegliere altre funzioni; l'importante è considerare che all'aumentare del *shuffled_traffic* diminuisce il tempo del nuovo flusso di acquisire capacità e convergere alla stabilità. Tuttavia, un elevato *shuffled_traffic* può impedire ai flussi di acquisire la stessa quantità di banda disponibile. La funzione *shuffled_traffic* è sempre positiva.

Riga 11): la variabile *residue_pos_fbk* tiene conto della quantità di capacità positiva che il router ha allocato. Esso è inizializzato con il valore del feedback totale positivo.

Riga 12): la variabile *residue_neg_fbk* tiene conto della quantità di capacità negativa che il router ha allocato. Esso è inizializzato con il valore del feedback totale negativo. Questa variabile è sempre positiva.

Riga 13): in questa riga si calcola il fattore di scala del feedback positivo, *Cp*. Le variabili *residue_pos_fbk* e *sum_x* sono definite prima.

Riga 14): in questa riga, invece, si calcola il fattore di scala del feedback negativo, *Cn*. Questo è sempre un valore positivo e le variabili *residue_neg_fbk* e *input_traffic* sono definite prima.

Righe 15) – 17): reset delle variabili per l'intervallo di controllo successivo.

Riga 18): impostazione del successivo intervallo di controllo. L'uso di *MIN_INTERVAL* è importante per stabilire un intervallo di controllo esatto quando il router è inattivo.

Riga 19): set del timer.

2.2.3 Fase di partenza dei pacchetti (on packet departure)

Un router XCP elabora ogni pacchetto utilizzando i parametri del feedback calcolati precedentemente. Come specificato prima, ogni pacchetto indica l'intervallo corrente che intercorre tra un pacchetto e l'altro (***X***) e la variazione di throughput desiderata *Delta_Throughput*. Il router calcola un cambiamento di capacità per pacchetto che sarà confrontato con il valore *Delta_Throughput* dell'header del pacchetto. Utilizzando la politica AIMD, il feedback positivo è applicato in modo imparziale per flusso, mentre il feedback negativo è realizzato proporzionalmente alla capacità di ogni flusso.

Per adattare routers ad alta velocità, XCP utilizza una rappresentazione numerica in virgola – fissa per i valori della *congestion header*. Questo significa che i calcoli per pacchetto definiti di seguito presentano un errore residuo che è più basso di 1Bps. Questi errori accumulati attraverso tutti i pacchetti che intercorrono in un intervallo di controllo, portano a delle imprecisioni per quanto riguarda l'allocazione di banda disponibile dei flussi. Saranno necessari ulteriori studi per capire se queste imprecisioni portano a problemi significativii.

L'elaborazione dei dati viene effettuata secondo il seguente codice:

on packet departure do:

*20. pos_fbk = Cp * X*

*21. neg_fbk = Cn * Pkt_size*

```
22. feedback =  pos_fbk – neg_fbk
23. if ( Delta_Throughput > feedback) then
24.       Delta_Throughput = feedback
25. else
26.      neg_fbk = min(residue_neg_fbk, neg_fbk +
                          (feedback  - Delta_Throughput))
27.      pos_fbk = Delta_Throughput + neg_fbk
28. residue_pos_fbk = max(0, residue_pos_fbk – pos_fbk)
29. residue_neg_fbk = max(0, residue_neg_fbk – neg_fbk)
30. if (residue_pos_fbk <= 0) then Cp = 0
31. if (residue_neg_fbk <= 0) then Cn = 0
```

Riga 20): contributo del feedback positivo per pacchetto corrente calcolato utilizzando *Cp*, definito in riga *13*, e X (tempo che intercorre tra i pacchetti) preso dalla *congestion header*. Notiamo che se *Cp* (e *Cn* in riga *21*) è rappresentato in virgola – mobile, questo calcolo sarà effettuato moltiplicando la mantissa di *Cp* con il valore di X, e successivamente si scalerà il risultato a seconda del valore dell'esponente di *Cp*.

Riga 21): contributo del feedback negativo calcolato utilizzando *Cn*, definito in riga *14*, e *Pkt_size* (grandezza di un pacchetto) definito nell' header IP. Il valore di *neg_fbk* è sempre positivo.

Riga 22): ammontare del feedback, ricavato dal router attraverso la differenza tra il feedback positivo e quello negativo per ogni pacchetto. Questo valore può essere sia positivo che negativo.

Righe 23) e 24): nella riga *23* si verifica se il pacchetto richiede un incremento di capacità (attraverso il valore *Delta_Thoughput)* superiore a quello allocato dal router. Se questo è vero, il throughput desiderato dal sender dovrà essere ridotto riportandolo al valore del feedback allocato dal router. Questo viene effettuato in riga *24*.

Riga 25): questa sezione viene eseguita quando il pacchetto riporta un incremento del throughput più piccolo di quello allocato dal router.

Riga 26): attraverso questa istruzione, il contributo del feedback negativo per pacchetto, *neg_fbk*, è settato i maniera tale da essere il più piccolo dei due termini in parentesi. Il primo termine, *residue_neg_fbk*, è l'ammontare del feedback negatvo. Il secondo termine incrementa il feedback negativo nominale del valore ottenuto tramite la differenza tra il throughput desiderato (*Delta_Throughput*) e la ripartizione del router (*feedback)*. Questo permette al router di salvare il feedback che è stato allocato a monte del bottleneck

Riga 27): l'allocazione positiva, *pos_fbk,* è aggiornata attraverso la somma tra *Delta_Throughput* e *neg_fbk*. In questo modo la somma aritmetica tra *pos_fbk* e *neg_fbk* eguaglia il throughput desiderato.

Righe 28) e 29): in queste due righe, i valori *residue_pos_fbk* e *residue_neg_fbk* sono ridotti rispettivamente dei valori *pos_fbk* e *neg_fbk*, evitando di ottenere risultati negativi.

Righe 30) e 31): quando l'ammontare del feedback diventa zero, vengono settati a zero i fattori di scala cioè viene stoppato il feedback.

2.2.4 L'intervallo di controllo

L'algoritmo di allocazione di capacità del router XCP, aggiorna diversi parametri ad ogni intervallo di controllo. Quest'ultimo è generalmente definito come l'RTT medio dei flussi che attraversano il router, visto in riga *7* (*avg_rtt*). Sono ancora in fase di studio altre possibili scelte dell'intervallo di controllo. Notiamo che:

- il valore di *avg_rtt* visto in precedenza, si riferisce all'ultimo calcolo effettuato. In altre parole, l'*avg_rtt* è calcolato basandosi sui pacchetti arrivati nell'intervallo di controllo precedente.
- L'*avg_rtt* ignora i pacchetti che riportano un RTT pari a zero.
- L'*avg_rtt* deve avere un valore minimo. Questo permette ai flussi di acquisire banda già da subito. Il valore minimo di default, *MIN_INTERVAL*, è massimo 5-10ms .

- L'*avg_rtt* deve avere un valore massimo. Il valore massimo di default, *MAX_INTERVAL*, è massimo 0.5-1sec.

2.2.5 Coda Persistente

Nel calcolo dell'intervallo di controllo si è utilizzata la variabile *queue* che rappresenta la coda persistente; questa rappresenta il minimo livello di coda stimato su un intervallo di tempo. Le procedure sono le seguenti:

on packet departure do:

32. *min_queue = min(min_queue, inst_queue)*

when the queue-computation timer expires do:

33. *queue = min_queue*
34. *min_queue = inst_queue*
35. *Tq = max(ALLOWED_QUEUE, (avg_rtt – inst_queue/capacity)/2)*
36. *queue_timer.reschedule(Tq)*

Riga 32): la lunghezza della coda istantanea corrente è esaminata ogni volta che un pacchetto in volo calcola la grandezza minima della coda.

Righe 33) e 34): subito dopo lo scadere del tempo di stima della coda, *Tq,* la variabile *queue* è pari al minimo livello di coda dell'ultimo intervallo *Tq*, mentre la variabile *min_queue* è pari al valore della coda corrente.

Riga 35): il primo termine della funzione *max*, *ALLOWED_QUEUE*, è il tempo di smaltimento della coda tollerabile (in genere si raccomanda un valore nominale pari a 2ms). Il secondo termine è una stima del ritardo di propagazione. In altre parole la coda persistente è una coda che non si esaurisce in un ritardo di propagazione. La divisione per 2 si effettua per evitare sovrastima del ritardo di propagazione.

Riga 36): si setta il tempo di calcolo della coda.

2.3 Problemi irrisolti

XCP è un protocollo tuttora in fase di studio per cui ci sono ancora dei problemi che non sono stati risolti.

2.3.1 XCP con routers non XCP

Ovviamente oggi in rete ci sono routers non XCP e rimarranno tali anche se XCP dovesse diventare un protocollo attivo e funzionante; esso quindi dovrà saper interagire anche con sistemi di controllo diversi. Gli elementi non XCP presenti in rete includono alcune specie di "link-level switches" con accodamento, cioè switches ATM e sofisticati switches Ethernet. Persino semplici multiplexers sono code non XCP con bassissimi livelli di buffering.

Le sorgenti e la rete si preoccupano della presenza di questi elementi non XCP poiché ognuno di questi può essere luogo di congestione, e se una stazione XCP è bloccata da uno degli elementi non XCP, il feedback del router non informerà alla stazione di rallentare il rate del traffico.

Sebbene il comportamento delle sorgenti XCP, in condizioni appena descritte, sia un problema importante da risolvere, una soluzione promettente potrebbe essere quella di azionare un algoritmo di rilevamento di congestione end – to – end in parallelo con l'algoritmo XCP e scambiare l'uso di questi algoritmi quando si rileva congestione non controllata da XCP. Per esempio, se una sorgente XCP riceve 3 ACK duplicati cambierà il suo comportamento e passerà all'algoritmo TCP Reno.

Una terminale che è controllata dall'algoritmo di congestione end – to – end informerà i routers XCP settando un bit specifico in testa al pacchetto. Un router può trattare questi pacchetti differentemente rispetto ai pacchetti controllati da XCP. Per esempio, il router potrebbe distribuire meno feedback ai pacchetti controllati end – to – end.

Nonostante si utilizzi l'algoritmo di controllo end – to – end del rate di trasmissione, un terminale intercetterà anche il feedback XCP e, se la sorgente scopre che esso sarà più piccolo rispetto a quello dato dal controllo end – to – end in un RTT, il terminale restituirà il feedback XCP seguente. Il feedback XCP che è più piccolo in un RTT, indica che l'intasamento del terminale è ancora una volta su un router XCP e il terminale utilizzerebbe l'informazione XCP poichè risulta più precisa.

Questi algoritmi sono ancora in fase di studio.

2.3.2 Links a rate variabile

Come discusso in [Zhang05], XCP può funzionare in maniera non ottimale su links condivisi. Quando un link è condiviso, come l'algoritmo CSMA delle reti ethernet e wireless, il singolo rate di scarico della coda è spesso funzione del carico mediamente condiviso. Così, utilizzando un valore costante per la variabile 'capacità' nell'algoritmo di

controllo di istradamento, non si ottiene una buona reazione. Per ottenere operazioni corrette, la capacità del router XCP dovrebbe considerare anche la maniera con la quale è condivisa la capacità del link.

2.3.3 XCP come TCP PEP (performance – enhancing proxy)

Oggi in internet, a volte, sono utilizzate (in alcune reti particolari) tecniche di miglioramento per il protocollo TCP (TCP PEPs – TCP performance – enhancing proxies [RFC3135]). Un meccanismo ordinario usato in TCP PEPs è dividere una connessione TCP in tre parti dove, la prima e l'ultima utilizzano TCP, mentre la parte centrale utilizza un protocollo di trasporto più aggressivo. Questo migliora le prestazioni sulla parte di percorso "problematica" e non richiede cambi di livelli di protocolli sul sistema finale. Per esempio, se un link satellitare ad alta velocità è usato per connettere una rete LAN ad Internet, un TCP PEP può essere implementato su entrambi i lati del link satellitare.
Oggi non è insolito trovare diversi TCP PEPs che, per offrire alta velocità di trasmissione, non utilizzano affatto un controllo di congestione. Di conseguenza, questo limita l'ambiente in cui possono essere applicati. Tuttavia, XCP può essere utilizzato tra due TCP PEPs sia per garantire una velocità di trasferimento elevata, sia per ottenere risposta di congestione corretta. Gli obiettivi di tale meccanismo saranno:

- conservare la semantica end – to – end TCP;
- consentire alcune forme di rilevamento su un PEP per determinarne la presenza sul percorso;
- consentire aggregati di flussi multipli tra due PEPs.

Un sistema che affronta gli obiettivi su scritti può anche essere usato per migliorare le caratteristiche di XCP. Un operatore di rete può utilizzare routers XCP e utilizzare PEPs sulla periferia della rete per passare dal tradizionale controllo di congestione TCP a quello XCP. In questa maniera si possono ottenere piccole code e miglioramenti riguardo l'utilizzo dei link con la rete XCP (soprattutto per le reti wireless).

2.3.4 Condivisione di risorse tra XCP e TCP

Katabi descrive un sistema per la condivisione di banda tra flussi XCP e TCP che si basa su throughputs medi delle sorgenti XCP e TCP utilizzando il router. Sono presenti due code sul router servite con differenti pesi. L'algoritmo permette il mantenimento di pacchetti, cioè, quando la coda è piena, cerca di evitare perdite.

Un algoritmo simile al TFRC stima la congestion window media delle sorgenti TCP e l'algoritmo XCP stima il throughput medio delle sorgenti XCP. Queste medie sono utilizzate per pesare dinamicamente il tempo speso dagli elaboratori per smaltire ogni coda. Primi esperimenti indicano che questo sistema può fornire fairness fra due flussi di classi diverse (XCP/TCP).
Rimangono comunque vari problemi da risolvere; per esempio rimangono dubbi su quale algoritmo simile al TFRC potrebbe funzionare meglio per la stima del throughput TCP considerando soltanto il rate di perdita, vedi(par5.4).

2.3.5 Operazione di Host back – to – back

Gli hosts XCP potrebbero essere in grado di operare in maniera back – to – back, cioè senza l'uso di router sul percorso. Teoricamente questo non dovrebbe essere un problema. Un sender inizializza il *Delta_Throughput* del valore desiderato, senza che il router apporti delle modifiche, in maniera tale che sia automaticamente accettato. Non si è ancora deciso se il receiver XCP sarà in grado di modificare il *Delta_Throughput* per richiedere un controllo di flusso da receiver a sender. A questo punto, XCP non offre alcun meccanismo di controllo di flusso. Il problema si potrebbe risolvere applicando XCP sulla coda di uscita di un host. Chiaramente, in questo modo, si complicherebbe ancora più il sistema.

2.4 Considerazioni di sicurezza

La presenza di un header che può essere letta e scritta da entità non partecipanti alla comunicazione end – to – end, porta alla nascita di alcune potenziali vulnerabilità di sicurezza. Vediamo i rischi a cui gli utenti possono essere esposti:

- *Attacchi di utenti intermedi (Man – in – the – Middle Attacks):* l'attacco di un utente "maligno" avviene forzando un sender a non inviare dati, inserendo un feedback negativo nel flusso. L'attacco può avvenire anche eliminando pacchetti lungo un flusso, utilizzando il controllo di congestione di Jacobson, ovvero modificando i bits ECN (Explicit Congestion Notification).
- *Convertire i dati dei canali:* IPsec [RFC2401] necessita di essere modificato, per consentire ai router di leggere l'intera *congestion header* e scrivere il campo *Delta_feedback*. Questa può diventare una conversione di dati del canale, cioè in maniera tale che un terminale può produrre dati osservabili in rete compromettendo i terminali.
- *Sorgenti Dannose:* l'algoritmo XCP si basa sull'informazione dei senders riguardo il round trip time (RTT) e l'intervallo intercorrente tra i pacchetti (X) e, di

conseguenza, reagisce correttamente al feedback datogli dalla rete. Naturalmente c'è la possibilità che il sender non fornisca queste informazioni in maniera corretta. Una sorgente che falsa i valori di X e di conseguenza del throughput, non può influire sull'utilizzo del link ma, nel caso peggiore, può acquisire capacità imparziale. Questo equivale anche per un sender aperto a più flussi TCP. Comunque, poiché X è dichiarato esplicitamente in ogni *congestion header*, è più facile da sorvegliare all'estremità della rete.

Se un sender falsa il proprio RTT può inficiare l'algoritmo di controllo del router, soprattutto quando ci sono più sender che falsano il loro RTT e l'intervallo di controllo del router si basa sull'RTT medio. Tuttavia, c'è una piccola nota positiva e cioè che un RTT falsato non influenzerebbe l'allocazione equa di banda.

Un flusso può anche ignorare i feedback negativi ricevuti dal router. In questo modo un flusso può ottenere un throughput ingiusto in un router congestionato. Comunque, la natura esplicita di XCP permette di verificare se i flussi reagiscono correttamente al feedback. Per esempio, una funzione di controllo, lungo il percorso, può ispezionare i *congestion header* di un flusso in entrambe le direzioni. Se si notifica un comportamento della sorgente non coerente con i feedback ricevuti (per esempio diminuire il throughput alla ricezione di un feedback negativo), allora si punisce il flusso interessato attraverso una drastica riduzione del suo throughput. Notiamo che questo può essere applicato in maniera probabilistica, verificando i flussi occasionalmente.