Alma Mater Studiorum - Università di Bologna
FACOLTÀ DI SCIENZE MATEMATICHE FISICHE E NATURALI
Sede di Cesena

Corso di Laurea in Scienze dell'Informazione

PROTOCOLLI DI TRASPORTO IN AMBIENTE WIRELESS

Tesi di laurea in

Sistemi per l'Elaborazione dell'Informazione I

Relatore: Presentata da:

Prof. *Marco Roccetti* *Claudio Enrico Palazzi*

Sessione I

Anno accademico 2001 – 2002

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

SOMMARIO

1

INTRODUZIONE

Negli ultimi anni, la tecnologia di Internet è emersa come forza portante degli sviluppi nell'area delle reti di telecomunicazione. Il volume complessivo delle trasmissioni di dati è notevolmente cresciuto rispetto al livello di pochi anni fa. Risulta evidente un futuro prossimo in cui Internet avrà un ruolo centrale nella vita di tutti i giorni, ancor più di quanto già oggi avviene. Allo stesso tempo, reti mobili hanno incontrato un'analoga crescita esponenziale del traffico gestito e dell'importanza attribuitagli dagli utenti. Basti pensare al fatto che diversi paesi europei, tra cui anche l'Italia, hanno oggi un quantitativo superiore di numeri telefonici cellulari rispetto a quelli fissi [KMM00]. Più in generale, occorre riconoscere che negli anni recenti è stato possibile osservare una vera e propria esplosione della diffusione di nuove tecnologie di trasmissione wireless [GLMS02].

Al momento della sua creazione e durante lo sviluppo dei suoi attuali protocolli, Internet non è stata pensata per un suo utilizzo in uno scenario wireless; conseguentemente, si trova ora ad operare con algoritmi e funzionalità che vanno bene per una rete fissa ma che entrano in crisi quando parte del collegamento è mobile. Uno degli argomenti di studio più stimolanti in questo contesto è senz'altro costituito dal TCP [BKVP97, BPKS97, BSAK95, CI95, GMPV00, SNVS99, TB00]. Il protocollo di trasporto, infatti, con le sue numerose

funzionalità, è quello che più si trova coinvolto nel calo di prestazioni complessive derivanti dall'interazione con un ambiente wireless.

Questa tesi si propone di ordinare in maniera sistematica le proposte più rilevanti tra i nuovi protocolli di trasporto nati appositamente per realizzare connessioni tra dispositivi mobili e Internet in modo da evitare i pesanti pedaggi prestazionali che condizionano il TCP attuale. Il nostro obiettivo è quello di esaminare a fondo i problemi derivanti da questo contesto e al tempo stesso fornire una chiave di lettura delle diverse soluzioni già proposte. Oltre ad illustrare i meccanismi implementati dai nuovi protocolli di trasporto infatti, cercheremo di analizzare criticamente ciascun nuovo schema ponendo in evidenza vantaggi e controindicazioni di ognuno di essi. In second'ordine, reputiamo che i risultati discussi in questa tesi possano costituire un utile manuale introduttivo per i ricercatori che volessero interessarsi all'argomento delle connessioni TCP in ambiente wireless.

La tesi è strutturata su cinque capitoli, il primo dei quali è rappresentato dalla presente introduzione al problema studiato. Nel secondo capitolo, diviso in cinque paragrafi, introdurremo i TCP tradizionali. In particolare nel par. 2.1 illustreremo le architetture di riferimento per Internet, mentre nel paragrafo 2.2 descriveremo l'implementazione del TCP attraverso la sua versione più diffusa: la Reno. Nei tre paragrafi successivi passeremo in rassegna altri tre protocolli di trasporto cosiddetti "tradizionali": rispettivamente TCP New Reno, TCP Vegas e TCP Sack. Il terzo capitolo ci servirà per comprendere meglio le motivazioni di tanto interesse attorno all'argomento di questa tesi. Descriveremo infatti le problematiche che scaturiscono dall'utilizzo del protocollo TCP in un contesto wireless. Nel par. 3.1 introdurremo l'ambiente wireless descrivendone le peculiarità che rendono necessario un trattamento diverso rispetto alle reti fisse. Nel par. 3.2 procederemo

ad analizzare come queste caratteristiche siano causa di errati comportamenti da parte dei TCP tradizionali, con conseguenti cali delle prestazioni. Il quarto capitolo costituisce il cuore di questa tesi e si divide in dieci paragrafi. Il primo di questi introduce le tipologie di approccio possibili alle problematiche presentate nel par. 3.2, creando tre gruppi principali in cui far ricadere le varie soluzioni. I restanti nove paragrafi illustrano ciascuno un nuovo protocollo di trasporto creato per operare in ambiente wireless. All'interno di ogni paragrafo vengono esposti gli schemi implementati riportando anche alcuni dati sperimentali riferiti a simulazioni a cui i ricercatori li hanno sottoposti. Sono inoltre posti in evidenza sia i punti di forza e sia le eventuali lacune o controindicazioni relative all'utilizzo di questi nuovi protocolli. Nel quinto e ultimo capitolo procederemo all'ordinamento di tutti i ragionamenti esposti nei capitoli precedenti, particolarmente quelli del quarto. Inseriremo infatti pregi e difetti di ciascun protocollo esaminato in una griglia comune per consentirci di effettuare un confronto su basi omogenee tra le varie soluzioni proposte.

2

UN PROTOCOLLO DI TRASPORTO: TCP

L'architettura di protocolli della suite IP, da decenni, permette la comunicazione tra computer di qualsiasi dimensione, marca o sistema operativo. Esamineremo in questo capitolo le funzionalità e le principali implementazioni di una parte fondamentale di questa architettura di successo: il protocollo di trasporto TCP.

2.1 Architetture OSI e Internet

In questa tesi vengono discussi vari protocolli di trasporto e le loro caratteristiche in rapporto all'ambiente wireless. Per capire le motivazioni che hanno portato a focalizzare l'attenzione su questo argomento dobbiamo prima comprendere in che contesto si collocano i protocolli di trasporto e di quali funzionalità sono dotati. E' quindi necessario illustrare, innanzitutto, l'architettura di rete *Open System Interconnection* (OSI) definita dall'International Standards Organisation (ISO). Non si tratta di una architettura realmente utilizzata, ma è un utile strumento per comprendere e descrivere le architetture reali, come quella di Internet che illustreremo più avanti [Ste94]. Come si evince dalla Fig. 2.1, l'architettura si basa su sette strati software. La scelta di una struttura su più livelli rende la progettazione più semplice e facilita la realizzazione di eventuali

modifiche: fattori importanti in un sistema complesso come quello delle reti. Ogni strato, infatti, si occupa di una parte specifica del problema complessivo e qualora si decida di operare delle modifiche ad una delle funzioni implementate nella rete, sarà sufficiente intervenire unicamente sullo strato in cui risiedono tali funzioni. Ciascuno strato dell'architettura fornisce dei servizi al livello superiore, sfruttando quelli che gli vengono offerti dal livello sottostante.

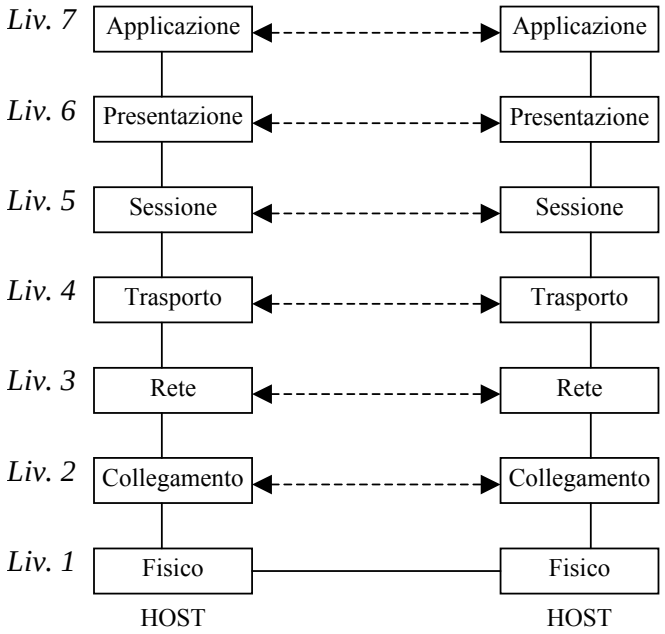

Fig. 2.1 Architettura di rete OSI.

Le comunicazioni tra due host, avvengono realmente solo nel livello più basso. I livelli superiori realizzano una comunicazione di tipo concettuale con il loro corrispondente, devono infatti utilizzare il livello sottostante, il quale a sua volta utilizza quello sotto di lui, fino ad arrivare al livello più basso. Ogni strato implementa uno o più protocolli, ognuno dei quali definisce due interfacce: interfaccia di servizio e interfaccia peer-to-peer. La prima serve a definire le operazioni che lo strato fornisce a quello superiore, la seconda stabilisce

le regole ed il formato per lo scambio di messaggi con il medesimo livello presente nell'host con cui si sta effettuando una comunicazione.

L'utilizzo dei servizi forniti dal livello inferiore, avviene tramite una tecnica detta *incapsulamento* [Ste94]. L'invio di un messaggio da un host ad un altro passa attraverso i vari strati dell'architettura del mittente, ognuno dei quali ne elabora i dati, aggiunge in testa al messaggio un'intestazione (*header*), ovvero incapsula il messaggio, e lo trasmette al livello sottostante fino a raggiungere quello fisico. Il messaggio così costruito (Fig. 2.2) viaggia in rete fino a raggiungere il destinatario, qui ognuno degli strati rimuove l'intestazione corrispondente che verrà utilizzata per le funzionalità di quello strato, elabora il messaggio e passa poi il messaggio risultante allo strato superiore fino a raggiungere quello più alto.

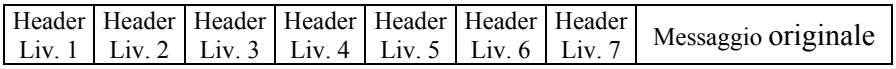

Fig. 2.2 Forma del messaggio che attraversa la rete

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

Livello fisico:

Gestisce la trasmissione di flussi di bit sul canale fisico.

Livello di collegamento:

Gestisce la comunicazione affidabile tra coppie di nodi direttamente connesse, trasmettendo e ricevendo unità di dati dette *frame*. Nel caso di canale ad accesso multiplo (broadcast), si occupa anche di controllare gli accesi per permetterne la condivisione.

Livello di rete:

Assegna gli indirizzi ed effettua l'instradamento dei dati attraverso vari nodi della rete per far giungere i pacchetti all'host destinatario. Permette dunque la comunicazione tra due nodi qualunque arbitrariamente connessi. Consente inoltre comunicazioni multicast[DEE89].

Livello di trasporto:

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

Livello di sessione:

Integra i diversi flussi di trasporto di una medesima applicazione. Gestisce le sessioni (ad esempio, login su di un computer remoto).

Livello di presentazione:

Gestisce le differenze di rappresentazione dei dati fra i due host (little endian o big endian, dimensioni dello stesso tipo di dati a 16 o 32 o 64 bit) [Ste94].

Livello di applicazione:

Implementa protocolli applicativi specifici: *File Transfer Protocol* (FTP) [PR85], *Telnet* [PR83], *Simple Mail*

Transfer Protocol (SMTP) [Pos82], *Simple Network Management Protocol* (SNMP) [CFSD90] ecc.

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

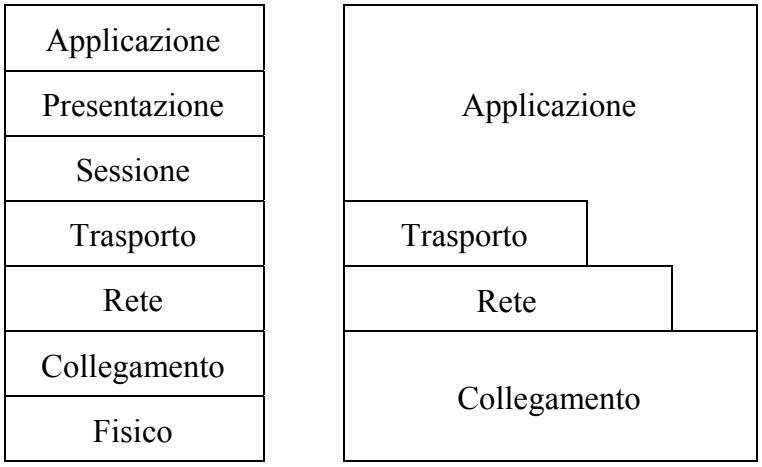

Fig. 2.3 Confronto tra architettura OSI ed architettura Internet.

Come già detto all'inizio del paragrafo, l'architettura OSI non è realmente impiegata in un contesto pratico, ma è utile per capire quelle che sono le reali architetture utilizzate. L'architettura di Internet, ad esempio, è divisa in quattro strati soltanto, le cui corrispondenze con quelli dell'architettura OSI sono evidenziate in Fig. 2.3. Notiamo che i primi due livelli dell'architettura OSI sono riuniti in un unico livello nell'architettura di Internet che è il livello di collegamento. Inoltre i livelli OSI di rete e di trasporto corrispondono rispettivamente ai livelli rete e di trasporto di Internet. Infine, il livello applicazione di Internet riunisce i rimanenti livelli OSI e può utilizzare le funzionalità, oltre che del livello di trasporto, anche dei livelli più bassi.

Cerchiamo di comprendere più approfonditamente le funzionalità dei quattro strati dell'architettura di Internet:

Livello di collegamento:

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

Livello di rete:

Effettua l'instradamento dei pacchetti attraverso la rete per far giungere i dati al destinatario. Nella suite di protocolli IP, in questo strato sono presenti *Internet Protocol* (IP) [Pos81a], *Internet Control Message Protocol* (ICMP) [Pos81b], *Internet Group Management Protocol* (IGMP) [Dee89, Fen97]. Il servizio di consegna dei dati offerto dallo strato di rete a quello superiore non è affidabile in quanto possono giungere dati non ordinati, o in più copie, o anche non arrivare affatto perché scartati da router intermedi. I blocchi di dati inviati e ricevuti da questo livello sono detti *datagram*.

Livello di trasporto:

Gestisce la consegna dei dati tra due host per conto dello strato di applicazione. Nella suite di protocolli IP, in questo strato sono presenti il *Transmission Control Protocol* (TCP) [Pos81c] ed il *User Datagram Protocol* (UDP) [Pos80]. Il servizio di consegna dei dati offerto dal TCP al livello superiore è affidabile. Lo strato di applicazione può quindi ignorare dettagli riguardanti questo aspetto. UDP invece fornisce un servizio più semplice che si limita ad inviare i pacchetti tra due host,

senza però offrire alcun tipo di garanzia sull'effettiva ricezione dei dati a destinazione. In questo caso, eventuali criteri di affidabilità richiesti dovranno essere implementati nello strato di applicazione. Una ulteriore differenza tra i due protocolli è che, diversamente dall'UDP, TCP consente al ricevente di regolare il flusso dei dati. La maggior parte delle applicazioni utilizza TCP (UDP è impiegato ad es. da SNMP) [Ste94].

Livello di applicazione:

Gestisce i vari dettagli riguardanti le specifiche applicazioni.

Analizzandolo in maniera più approfondita notiamo che TCP è un protocollo orientato alla connessione che effettua la consegna affidabile di flussi di byte ordinati, da processo a processo in maniera full-duplex. TCP assume di ottenere dal livello inferiore un servizio potenzialmente inaffidabile di consegna di datagram, i quali possono giungere disordinati, duplicati, o anche venire persi perché scartati da router intermedi, e si preoccupa di consegnare i pacchetti in ordine di trasmissione ed in un'unica copia [Pos81c]. Oltre a gestire il flusso dei dati in maniera affidabile, TCP si occupa anche di instaurare la connessione, ovvero di realizzare uno scambio di messaggi tra i due host che vogliono iniziare una comunicazione, di regolare il flusso di dati fra questi due host e di monitorare il livello di congestione della rete intervenendo, quando reputato necessario, per ridurne il carico.

Come già riportato, il TCP è il protocollo di trasporto maggiormente utilizzato nelle reti di computer. I suoi obiettivi lo rendono particolarmente interessante e ancor di più lo sarà quando lo

analizzeremo attraverso un'ottica wireless. Infatti, proprio le funzionalità del TCP, create per operare in un ambiente wired, sono quelle che risentono maggiormente dell'immersione in un contesto wireless, causando un significativo degradamento delle prestazioni. Allo scopo di ottenere buone prestazioni complessive occorre quindi intervenire proprio su tali funzionalità, correggendone le disfunzioni causate dall'operare in un ambiente che non è quello per cui sono state progettate.

2.2 Implementazione base del TCP: TCP Reno

Cerchiamo ora di comprendere più approfonditamente le varie funzioni del TCP illustrando il funzionamento della sua versione maggiormente diffusa: TCP Reno. Nell'effettuare questa analisi occorre tenere conto che è possibile individuare due funzioni principali in TCP: il controllo del flusso ed il controllo della congestione.

Il controllo del flusso permette al TCP mittente di inviare verso il ricevente un quantitativo adeguato di dati, allo scopo di evitare di sprecare risorse di bandwidth ma anche di non eccedere le capacità di ricezione del mittente. L'elemento principale di questa funzionalità è il meccanismo di *sliding window* [PD00].

Per impostare una connessione, TCP ricorre al meccanismo denominato three-way handshake. Un processo invia un segmento SYN contenente il numero di sequenza iniziale ad un altro processo che si trova in stato di attesa passiva, questi risponde con un segmento di conferma denominato *ack*, il quale contiene il suo numero di sequenza; infine il

primo processo risponde con un altro segmento di ack [Hus00a]. Dopo questa procedura, la comunicazione è stabilita e si può cominciare a trasferire dati. Tramite il primo segmento SYN viene anche negoziata la dimensione massima di un segmento inviabile sulla connessione, detta *Maximum Segment Size* (MSS), ed eventuali opzioni TCP.

Il quantitativo iniziale di dati inviati viene chiamato *initial window* (IW) ed è, escluse le situazioni sperimentali [AFP98, JBB92, PN98], della dimensione di uno o due segmenti [APS99]. Il valore iniziale della ssthresh è invece di 65535 byte.

Per fornire al livello superiore tutti i segmenti corretti, non duplicati e in ordine di trasmissione, TCP assegna un numero di sequenza ad ogni segmento inviato e ne richiede un riscontro positivo da parte del TCP ricevente, tramite pacchetti ack entro un intervallo di tempo definito. I numeri di sequenza vengono utilizzati dal TCP ricevente per ordinare correttamente i segmenti giunti e per individuare quelli duplicati. Per riconoscere e gestire gli errori eventualmente contenuti nei segmenti, viene aggiunto ad ognuno di essi un valore di checksum.

L'invio dei pacchetti ack da parte del ricevente avviene seguendo il meccanismo cumulativo. Questi pacchetti contengono al loro interno un numero di sequenza che permette di individuare quali sono i dati ricevuti correttamente e che riempiono esattamente la sequenza fino a quel momento, indica quindi anche il numero di sequenza del prossimo segmento atteso. Il TCP che riceve correttamente un segmento non invia l'ack al mittente se tutti i segmenti che precedono quello appena arrivato non sono giunti anch'essi. Se invece si rileva una mancanza nel flusso dei dati, il ricevente invia al mittente un ack duplicato (detto *dupack*) in cui si conferma ancora una volta l'arrivo dell'ultimo segmento giunto correttamente ed in ordine. Quando il pacchetto

mancante raggiungerà la destinazione, il TCP ricevente invierà al mittente un nuovo ack. L'arrivo di ogni nuovo ack provoca la trasmissione di nuovi dati da parte del mittente secondo criteri che illustreremo parlando del controllo della congestione.

L'host che riceve i dati potrebbe non essere in grado di processarli con la stessa velocità con cui il mittente li invia in rete, provocando così un accumulo di segmenti all'interno dei buffer a destinazione. Per prevenire l'esaurimento del suo spazio di memoria, il TCP ricevente invia al mittente, con ogni ack, una "finestra" indicante il numero massimo di segmenti che il mittente può inviare prima di ricevere ulteriori autorizzazioni. Tale finestra prende il nome di *receiver advertised window* (rwnd).

Nonostante la rwnd, i segmenti inviati potrebbero divenire troppi per essere gestiti dalla rete. Nulla vieta a diversi processi contemporaneamente attivi di immettere una quantità di dati superiore allo spazio disponibile nei buffer dei router intermedi lungo il percorso verso la destinazione. L'IP può pertanto perdere dei pacchetti per cause riconducibili a congestione della rete. In mancanza di reazioni adeguate, questo comportamento porterebbe ad un intasamento ulteriore della rete, con conseguente peggioramento della situazione. Proprio allo scopo di gestire questa problematica, TCP implementa, come annunciato all'inizio di questo paragrafo, dei meccanismi per il controllo della congestione.

La perdita di dati lungo il percorso non permette al ricevente di inviare il pacchetto di ack relativo. Se l'ack di un segmento non arriva entro l'intervallo di tempo stabilito, avviene un Retransmission Timeout (RTO) che provoca la ritrasmissione del segmento in questione. Il TCP

mittente effettua un costante monitoraggio dei RTT dei segmenti e calcola il RTO con le formule di Jacobson [JK88]:

```
Diff = RTT_misurato – Stima_corr_RTT

Stima_corr_RTT = Stima_corr_RTT + g*Diff

Dev = Dev + h*(|Err| - Dev)

RTO = Stima_corr_RTT + 4*Dev
```

In queste formule, `Diff` è la differenza tra il valore di RTT appena misurato, grazie all'arrivo di un ack, e la stima correntemente utilizzata del RTT. La costante `g` rappresenta il contributo apportato alla media da `Diff` ed è impostata a `1/8`. `Dev` corrisponde alla deviazione media che fornisce una buona stima di quella standard con un calcolo più semplice. La costante `h` rappresenta il contributo apportato da ogni nuovo calcolo ed è impostata a `1/4`.

Per evitare fenomeni di collassamento di Internet, sono stati introdotti due meccanismi denominati *slow start* e *congestion avoidance* che partecipano alla determinazione del numero di segmenti da inviare. Il loro compito è quello di testare continuamente la capacità della rete incrementando gradualmente la frequenza di invio dei segmenti. Gli algoritmi di slow start e congestion avoidance necessitano di due variabili di stato: la *congestion window* (cwnd) e la *slow start threshold* (ssthresh) [Ste97]. La prima di queste indica l'ammontare massimo di dati che il mittente può inviare in rete prima di ricevere un ack; il minimo tra questo valore e quello contenuto nella variabile rwnd determina il flusso trasmissivo. La seconda indica invece quando utilizzare slow start e quando congestion avoidance. Slow start si utilizza all'inizio della connessione e, a meno che si rilevi una situazione di rete congestionata, fino a che cwnd rimane inferiore a

ssthresh, dopodiché si utilizza congestion avoidance. La modalità di congestion avoidance continua invece fino a che non scade un RTO, lasciando quindi nuovamente il posto a slow start, oppure fino alla fine della connessione. Per determinare quando si è di fronte ad una congestione il mittente utilizza due tipi di segnale: la scadenza di un RTO e l'arrivo consecutivo di tre dupack. I dupack vengono inviati dal TCP ricevente per segnalare l'arrivo di segmenti fuori ordine e all'arrivo del terzo dupack, il TCP mittente entra in modalità *fast retransmit/fast recovery*. Le caratteristiche di questa modalità verranno illustrate più avanti.

Durante la fase di slow start, il mittente aumenta il valore della cwnd ogni volta che riceve un ack di conferma dell'arrivo di nuovi dati. L'aumento è pari alla dimensione massima che un segmento può avere per non essere ulteriormente frammentato dai nodi intermedi lungo il suo percorso verso la destinazione. Si tratta dunque di un aumento moltiplicativo: ogni volta che il destinatario riceve una intera finestra di dati corretti, la cwnd viene raddoppiata.

Nel caso in cui slow start termini perché la cwnd ha raggiunto il valore di ssthresh, si entra in modalità congestion avoidance nella quale la variabile cwnd viene aumentata solo di un segmento per ogni intera finestra di dati che è giunta a destinazione. Si tratta dunque di un incremento additivo e poiché il valore della cwnd viene mantenuto in byte, l'aumento per ogni ack ricevuto risulta pari a:

```
MSS*MSS/cwnd
```

TCP utilizza l'arrivo degli ack per interpretare le condizioni della rete. In particolare, qualora l'ack di un segmento inviato ritardi oltre il tempo di RTO, si assume che vi sia stata una perdita e che ciò sia riconducibile

ad una situazione di congestione. Pertanto, TCP provvede alla ritrasmissione del segmento e a ritornare in modalità slow start con un valore di cwnd pari ad un segmento. Quando il mittente riceverà un ack non sarà in grado di sapere se si tratta della conferma di avvenuta ricezione per il primo segmento inviato o della sua ritrasmissione. Si crea dunque un'ambiguità che non consente di utilizzare le stime degli RTT per calcolare il RTO [KP87]. Il valore di RTO viene allora ricalcolato attraverso il meccanismo di *exponential backoff*, moltiplicando il valore precedente per una costante che deve essere uguale o superiore a 2 [PA00]. Il valore di ssthresh sarà invece impostato come:

```
max (flightsize/2, 2*MSS)
```

dove `flightsize` è la dimensione dei segmenti inviati, e di cui il mittente non ha ancora ricevuto conferma del loro arrivo a destinazione, al momento della scadenza del timeout. Come conseguenza, la fase di slow start successiva terminerà prima per lasciare posto a quella di congestion avoidance, questo comportamento è volto ad evitare di eccedere la capacità della rete.

Allo scopo di recuperare rapidamente dalla perdita di un pacchetto segnalata dall'arrivo di tre dupack senza però che si debba attendere il timeout, viene utilizzato, come già annunciato, l'algoritmo di fast retransmit/fast recovery. Grazie ad esso si riesce ad evitare l'invocazione da parte del TCP mittente del meccanismo di slow start, il quale ridurrebbe la finestra di invio ad un singolo segmento in una situazione in cui i tre dupack ricevuti testimoniano comunque la presenza di bandwidth [APS99]. Per essere messo in pratica, il TCP ricevente deve inviare immediatamente un dupack al mittente per ogni pacchetto fuori ordine ricevuto. La ricezione di tre dupack consecutivi

viene interpretato dal mittente come la perdita del pacchetto in sequenza atteso. L'algoritmo, al ricevere di tre dupack, imposta il valore di ssthresh come metà dei segmenti in viaggio in quell'istante e ritrasmette il pacchetto di cui era attesa la conferma. Inoltre, in considerazione dei tre pacchetti fuori sequenza giunti al mittente causando l'invio dei tre dupack, "gonfia" la cwnd ponendola uguale a:

```
ssthresh + 3*MSS
```

Per ogni dupack ulteriore ricevuto (indicante un segmento che ha lasciato la rete e che si trova nel buffer del ricevente), il valore di cwnd viene aumentato di un altro MSS. A questo punto, se cwnd e rwnd lo permettono, vengono inviati altri segmenti. All'arrivo di un nuovo ack, la variabile cwnd viene "sgonfiata" ritornando al valore contenuto in ssthresh.

2.3 La versione TCP New Reno

E' risaputo che i meccanismi di fast retransmit e di fast recovery funzionano bene in caso di un'unica perdita di segmento all'interno di una singola finestra, mentre non riescono a fornire delle buone prestazioni nel caso di perdite multiple all'interno di una singola finestra [MMFR96]. Questo avviene perché l'algoritmo, al ricevere di tre dupack consecutivi, ritrasmette solo il primo pacchetto e la prima informazione utile al prosieguo arriva solo al ricevimento dell'ack per il pacchetto ritrasmesso. Gli altri segmenti persi dovranno attendere lo scadere del timeout. Per risolvere questo problema, la versione TCP denominata New Reno introduce la possibilità di inviare, oltre agli ack cumulativi tradizionali, anche degli ack parziali [FH99]. Si tratta di

conferme, inviate dal ricevente al mittente, per l'arrivo corretto di alcuni, ma non tutti, i segmenti che risultavano in viaggio prima che venisse invocato il meccanismo di fast retransmit. Il meccanismo degli ack parziali richiede che venga memorizzata dal mittente un'ulteriore variabile di stato denominata *recover*.

L'algoritmo di fast retransmit/fast recovery implementato dallo schema New Reno ricalca in parte quello della versione Reno. Infatti, prevede di ritrasmettere il segmento perso, quando vengano ricevuti tre dupack e la variabile ssthresh viene impostata come:

```
max (flightsize/2, 2*MSS)     (1)
```

dove `flightsize` corrisponde all'ammontare dei dati inviati ma di cui ancora non è stato ricevuto alcun ack [APS99]. La variabile cwnd viene incrementata artificialmente per tenere conto dei tre segmenti giunti a destinazione fuori ordine e viene quindi posta uguale a:

```
ssthresh + 3*MSS
```

La versione New Reno dell'algoritmo di fast retransmit/fast recovery prevede però che nella variabile recover venga memorizzato il numero di sequenza più elevato tra quelli dei segmenti trasmessi. Per ogni ulteriore dupack ricevuto oltre al terzo, la cwnd viene incrementata di un ulteriore MSS. Se le condizioni espresse dal nuovo valore della cwnd e di quello della rwnd lo permettono, viene inviato un nuovo segmento. Le differenze rispetto all'algoritmo tradizionale di fast retransmit/fast recovery risiedono quindi nella possibilità di utilizzare ack parziali e di tenere aggiornata la variabile recover.

Fast recovery termina quando arriva un ack completo, quindi con la conferma di un numero di sequenza non inferiore a quello contenuto nella variabile recover, oppure alla scadenza del timeout. Se il mittente riceve un ack che dichiara l'avvenuta ricezione di tutti i pacchetti, incluso quello il cui numero di sequenza è memorizzato nella variabile recover, allora si esce dallo schema di fast recovery e si provvede ad eliminare dalla cwnd l'incremento artificiale precedentemente operato. Il valore dei ssthresh viene calcolato come in (1), mentre quello della variabile cwnd viene impostata secondo la formula:

```
min (ssthresh, flightsize + MSS)
```

Qualora l'ack ricevuto sia un ack parziale, ovvero un ack che non fa riferimento a tutti i segmenti compresi fino al numero di sequenza memorizzato in recover, TCP New Reno ritrasmette il primo segmento non ancora confermato. La ritrasmissione di un unico pacchetto dopo un ack parziale rispecchia un atteggiamento conservativo; se si volesse essere più aggressivi si potrebbe effettuare la ritrasmissione di più pacchetti. Oltre alla ritrasmissione viene operata una riduzione della variabile cwnd corrispondente al numero di segmenti giunti a destinazione (indicati dall'arrivo dell'ack parziale) ed un aumento di un MSS per tenere conto dell'arrivo a destinazione dei dati che hanno generato l'ack parziale. Inoltre, ma solo per il primo ack parziale giunto al mittente, viene anche azzerato il timer di ritrasmissione.

Perdite multiple all'interno di una singola finestra di invio potrebbero inutilmente far invocare più volte la procedura di fast retransmit [Flo94]. Per risolvere questo problema sono state proposte alcune modifiche al TCP New Reno originale [FH99]. In particolare, si utilizza una nuova variabile, denominata *send_high*, il cui valore iniziale corrisponde al primo numero di sequenza inviato e nella quale, dopo

ogni timeout, viene memorizzato il numero di sequenza più elevato tra quelli trasmessi. Inoltre, se il mittente riceve tre dupack ed ancora non si trova in modalità fast recovery, si verificano i numeri di sequenza riconosciuti dai dupack. Qualora tali numeri di sequenza risultano inferiori a quello memorizzato nella variabile send_high, il mittente non compie alcuna azione. Viceversa, la variabile ssthresh viene impostata secondo l'equazione (1), il numero di sequenza più elevato tra quelli dei dati trasmessi viene memorizzato in recover e si continua poi con la procedura precedentemente illustrata.

L'algoritmo proposto da TCP New Reno potrebbe comunque provocare l'invocazione dell'algoritmo di slow start, ad esempio nel caso in cui siano stati persi molti pacchetti all'interno della stessa finestra di invio oppure se il RTO è impostato di poco superiore al RTT [FF96].

2.4 La versione TCP Vegas

TCP Vegas parte da un approccio diverso rispetto a TCP Reno e New Reno. Questo protocollo di trasporto cerca infatti di stimare il livello di congestione prima che accada, e quindi di evitarla. TCP Vegas mantiene le regole di equità nei confronti degli altri utilizzatori della rete, correggendo in maniera moltiplicativa la frequenza di invio quando si rileva una congestione: applica infatti slow start quando si rileva un timeout. Inoltre, è in grado di operare con qualsiasi altra valida implementazione del TCP e le modifiche richieste si limitano al mittente [BOP94].

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

TCP Vegas estende il meccanismo di ritrasmissione della versione Reno tenendo traccia dei tempi di arrivo degli ack e stimando in questo modo i RTT. All'arrivo di un dupack, se la differenza tra il tempo corrente ed il timestamp del pacchetto risulta maggiore del tempo di timeout, allora il segmento mancante viene immediatamente ritrasmesso senza dover attendere altri dupack [BOP94]. Il tempo che intercorre tra l'invio di un pacchetto e la ricezione dell'ack viene verificato anche per l'arrivo dei primi due nuovi ack successivi all'invio dei dati nel caso che si tratti di una ritrasmissione. Se questo intervallo di tempo è più ampio del valore di timeout, allora TCP Vegas ritrasmette un altro segmento. Con questo nuovo meccanismo si riesce a ridurre il tempo di reazione del protocollo di trasporto alle perdite di segmenti. Inoltre, se si verifica una perdita multipla e con l'invocazione più di una volta di fast retransmit, la cwnd viene ridotta solo per la prima di queste.

Il secondo meccanismo innovativo implementato da TCP Vegas è quello che gli consente di prevenire le congestioni senza dover aspettare di rilevare delle perdite nei segmenti inviati. L'idea di base è quella di avere la bandwidth pienamente utilizzata e allo stesso tempo tenere sotto controllo la quantità di occupazione dei buffer nei router lungo il percorso dal mittente al ricevente. Infatti buffer troppo pieni provocherebbero perdite di segmenti ma, allo stesso tempo, buffer troppo vuoti non permetterebbero tempestivi adattamenti della

frequenza trasmissiva in situazioni di aumenti transienti della bandwidth disponibile. Per realizzare il controllo della congestione, TCP Vegas utilizza alcune formule e parametri. Innanzitutto, se chiamiamo `RTT_Base` il minimo RTT misurato per i segmenti inviati e `Dim_Finestra` la dimensione della finestra di invio corrente, il throughput atteso corrisponde alla seguente formula:

```
Throughput_Atteso = Dim_Finestra / RTT_Base
```

Solitamente il segmento con il RTT minimo corrisponde al primo inviato, poiché i router non devono ancora sopportare il peso della nuova connessione [BOP94].

Una ulteriore operazione effettuata da TCP Vegas è quella di calcolare il throughput reale una volta ogni RTT, rilevando il tempo trascorso tra l'invio di un segmento contenente un certo ammontare di dati ed il ricevimento del corrispondente ack, e dividendo poi il tempo impiegato per i byte trasmessi. Si calcola quindi la differenza tra il throughput atteso e quello reale:

```
Diff = Throughput_Atteso - Throughput_Reale
```

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

$$\alpha < \texttt{Diff} < \beta$$

Gli estremi α e β sono due costanti che devono rappresentare rispettivamente la situazione di avere troppi oppure troppo pochi pacchetti nei buffer dei router lungo il percorso verso il ricevente.

L'ultimo meccanismo introdotto da TCP Vegas riguarda alcune modifiche da apportare alla procedura di slow start per renderla in grado di riconoscere una congestione incipiente e di evitarla. La crescita della finestra di invio, in maniera esponenziale, viene effettuata solo una volta per ogni RTT, nel restante periodo rimane fissa. In questo modo il protocollo di trasporto riesce a stimare la frequenza di invio effettiva ed applicare le formule per stimare quanti pacchetti inviare sul canale. Si cerca così di evitare perdite nella fase iniziale di slow start.

TCP Vegas, grazie alle sue capacità predittive offre uno schema per evitare il verificarsi di perdite dovute a congestione. Le versioni, Reno e New Reno del TCP inviano pacchetti sulla connessione con una frequenza sempre più elevata, attendendo di rilevare una perdita per comprendere quando la rete è divenuta congestionata e ridurre così la frequenza di invio. Si può quindi dire che questi protocolli provochino loro stessi delle perdite. TCP Vegas invece utilizza un approccio proattivo che simulazioni hanno dimostrato produrre un throughput anche del 71% migliore rispetto alla versione Reno [BP95, ADLY95].

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

da quelle dovute a congestione, situazioni nelle quali viene ugualmente invocata la procedura di slow start.

2.5 La versione TCP Sack

E' risaputo che TCP, soprattutto nella versione Reno, ma anche in quella New Reno, subisce un calo delle prestazioni in condizioni di perdite multiple in una singola finestra [FF95]. Ciò è dovuto al meccanismo di notifica cumulativa dei pacchetti giunti a destinazione operato dagli ack. Quando un segmento fuori sequenza giunge al ricevente, questi non può inviare il relativo ack. Trattandosi di ack cumulativi infatti, ciò equivarrebbe a dichiarare di aver ricevuto anche tutti i segmenti precedenti. TCP reagisce invece inviando un dupack dell'ultimo segmento ricevuto in sequenza, ma questo non fornisce alcuna informazione al mittente su quali pacchetti siano effettivamente andati persi. Il risultato finale è che verranno invocati meccanismi di controllo della congestione anche se non fosse stato necessario. L'utilizzo di pacchetti *selective acknowledgment* (sack) costituisce una possibile risposta a questo problema in quanto consente al ricevente di informare il mittente su quali siano esattamente i segmenti giunti a destinazione e quali siano quelli mancanti [MMFR96, FMMP00].

L'impiego dei pacchetti sack prevede l'utilizzo della normale struttura dei pacchetti TCP [Pos81c], sfruttando il campo opzioni nell'header. Vi sono due tipi di impostazione delle opzioni. Il primo serve ad instaurare una comunicazione di tipo sack; il suo invio avviene tramite un segmento SYN. Questo tipo di opzione è costituita da due byte, e i suoi campi sono mostrati in Fig. 2.4.

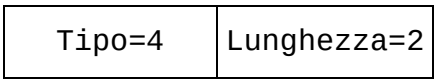

Fig. 2.4 Opzione per l'abilitazione dei sack.

Il secondo tipo di opzione è costituito dalla notifica selettiva dei pacchetti correttamente ricevuti. Il suo invio è possibile solo su di una connessione che abbia ricevuto il permesso. La sua lunghezza di tale opzione è variabile ed il contenuto è illustrato in Fig. 2.5.

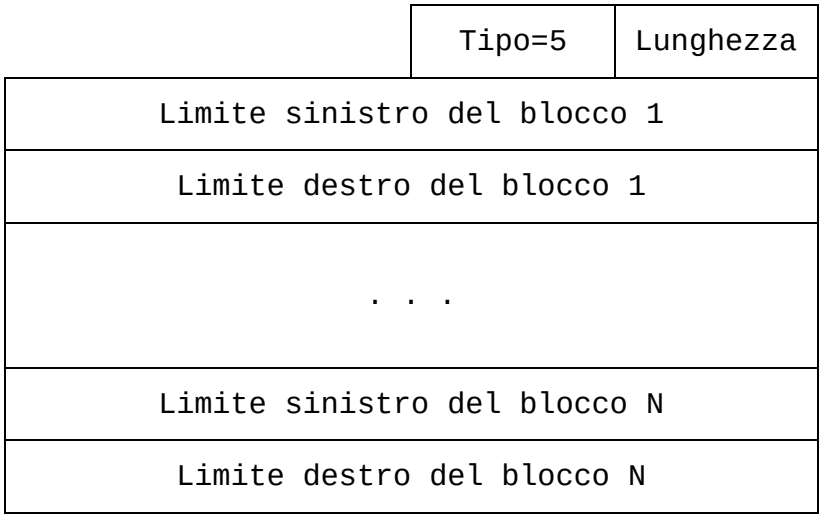

Fig. 2.5 Opzione per l'utilizzo dei sack.

L'opzione sack serve ad informare il mittente del ricevimento di blocchi non contigui di dati, viceversa si utilizzano normali ack. I blocchi non contigui sono individuati attraverso l'indicazione dei numeri di sequenza che li delimitano a destra e a sinistra. Quando alcuni pacchetti mancanti giungono al ricevente, questi invia un altro sack per informare il mittente della nuova situazione.

Un opzione sack, che specifica n blocchi, ha una lunghezza di 8*n+2 byte. Considerando i 40 byte disponibili nelle opzioni TCP, si riescono

a specificare al massimo 4 blocchi. Qualora si utilizzino altre opzioni [JBB92], i blocchi riportabili dall'opzione sack possono essere in numero inferiore. In ogni caso non è detto che sia possibile evidenziare tutti i blocchi di dati presenti nel ricevente.

Dopo aver ricevuto l'opzione per l'abilitazione dei sack, e solo in questo caso, il ricevente può decidere se generare pacchetti sack. Qualora decida di farlo, dovrà generare pacchetti di tipo sack tutte le volte che le circostanze lo consentono. Le regole da applicare per l'invio dei sack sono le seguenti:

- Il primo blocco nelle opzioni sack deve sempre specificare l'insieme contiguo di dati contenenti il segmento che ha provocato l'invio del sack stesso. In questo modo vengono poste in evidenza i cambiamenti più recenti nell'insieme dei dati giunti al ricevente. I blocchi successivi possono essere ordinati liberamente.

- E' preferibile che il ricevente includa nelle opzioni, il maggior numero possibile di blocchi sack distinti. Le ripetizioni di blocchi già precedentemente inviati che non siano sottoinsiemi di nuovi blocchi, consentono di recuperare da possibili perdite di pacchetti sack. Ciò porta a ripetere la notifica di ricevimento di un dato, tante volte quanti sono i blocchi distinti riportabili nelle opzioni del nostro protocollo, rendendo lo schema più robusto.

Il mittente utilizza i segmenti spediti ed i pacchetti sack ricevuti per gestire una coda di trasmissione. In questa coda, ogni segmento spedito

è caratterizzato da un flag chiamato *sacked*, che specifica se ne è giunta la conferma di ricezione. Ad ogni ricevimento di sack, i flag della coda di trasmissione vengono aggiornati. Quando il mittente ritrasmette uno dei segmenti con il flag non marcato, può sceglierne uno qualsiasi. Unica condizione è che il numero di sequenza del segmento che si sceglie di ritrasmettere deve essere inferiore a quello più alto tra i blocchi di cui è giunta notifica di ricezione.

Qualora si verifichi un timeout, il mittente procede a ritrasmettere tutti i segmenti nella coda di trasmissione, a prescindere dall'impostazione del flag sacked. Il ricevente potrebbe infatti aver rinnegato tutti i dati fuori sequenza ricevuti. Inoltre si attenderà un normale ack cumulativo prima di togliere i segmenti dalla coda.

L'opzione sack non cambia il comportamento del protocollo di trasporto in caso di scadenza del timeout per un pacchetto: verranno invocati i normali meccanismi di controllo della congestione.

TCP Sack offre il grande vantaggio rispetto ai protocolli di trasporto tradizionali di poter indicare in modo preciso e non cumulativo i segmenti che hanno raggiunto correttamente il destinatario. Questo consente un utilizzo delle ritrasmissioni più efficace, con guadagni prestazionali evidenti soprattutto in situazioni di perdite multiple all'interno di una singola finestra di dati. Le sue buone prestazioni, hanno permesso a TCP Sack di diffondersi in Internet, sostituendo gradualmente il TCP Reno e di essere implementato su Windows98, Windows2000 e su alcune versioni Linux.

La capacità di evitare inutili ridondanze o attese di timeout, rende TCP Sack particolarmente adatto su connessioni ad alto BER[1] come ad esempio quelle wireless. Allo stesso tempo però, occorre notare che i 40 byte di opzioni nell'header aggiungono un overhead che può risultare pesante in situazioni di bandwidth limitata. Il pieno sfruttamento dello spazio per le opzioni nell'header limita inoltre la possibilità di implementare altri meccanismi per il miglioramento delle performance [JB88]. Ulteriori svantaggi sono costituiti dal fatto che TCP Sack non permette di adottare tecniche di compressione degli header [Jac90], né di distinguere tra i vari tipi di perdita (errore, congestione, handoff, fading).

[1] Bit Error Rate, ovvero la frequenza di BIT errati sul totale di quelli trasmessi.

3

TCP IN AMBIENTE WIRELESS

I nuovi scenari wireless pongono problemi inaspettati al tempo in cui Internet fu concepita e i suoi protocolli realizzati. Esamineremo in questo capitolo le problematiche che scaturiscono al livello di trasporto dall'interazione di dispositivi mobili con la rete fissa, in ordine sia del controllo del flusso, sia del controllo della congestione e sia delle problematiche di consumo delle risorse computative ed energetiche.

3.1 Caratteristiche dell'ambiente analizzato.

Le reti wireless stanno diventando sempre più importanti nelle comunicazioni odierne, grazie alla loro capacità di soddisfare la richiesta di informazioni da parte degli utenti, in ogni momento e in ogni luogo. Appare quindi particolarmente interessante l'interazione tra dispositivi mobili e Internet. Un numero crescente di utenti usufruisce della tecnologia mobile per ottenere informazioni da Internet o per utilizzare la posta elettronica. La nostra trattazione dell'ambiente wireless si limiterà proprio agli aspetti inerenti il caso di collegamento tra dispositivi mobili e Internet. In particolare analizzeremo le problematiche scaturite dall'utilizzo del TCP per questo tipo di collegamenti. Abbiamo visto nel capitolo precedente l'importanza delle varie funzionalità delegate al protocollo di trasporto, occorre sottolineare che queste stesse funzionalità risentono vistosamente

dell'interazione di Internet con dispositivi mobili, producendo risultati deludenti. Sono quindi, queste stesse funzionalità, quelle su cui maggiormente occorre agire se si vogliono ottenere adeguati risultati prestazionali.

La situazione da noi studiata riguarda collegamenti in cui la parte wireless è limitata all'ultimo tratto. La separazione tra la parte wired e quella wireless della connessione, viene realizzata tramite un server proxy denominato *base station* che si occupa di mantenere la connessione tra l'host fisso e quello mobile. Lo schema illustrato in Fig. 3.1 è tipico di situazioni in cui un utente si colleghi ad Internet utilizzando un dispositivo mobile quale un telefonino od un computer portatile.

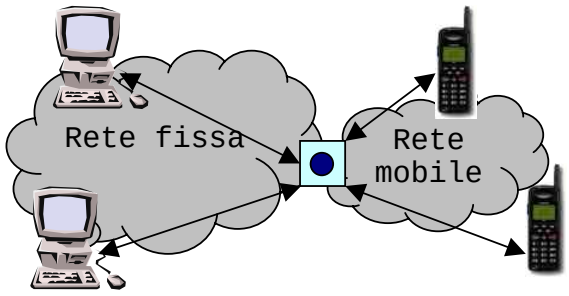

Fig. 3.1 Collegamento wireless

L'area geografica da coprire con il segnale trasmissivo viene divisa in *celle* ognuna delle quali è controllata da una base station. Quando un dispositivo mobile attraversa la zona di confine tra due celle si verifica un *handoff*, ovvero il trasferimento di tutti i dati utili alla gestione della connessione dalla vecchia base station a quella nuova [CI94]. Tale procedimento può risultare pesante perché comporta lunghi tempi di trasferimento dei dati, nonché l'esistenza di buffer sufficientemente capienti da permettere la presenza simultanea di più utenti nella stessa cella. Occorre inoltre ricordare che alcuni segmenti in viaggio durante il

passaggio dalla base station precedente a quella successiva potrebbero andare persi [SB98].

Un aspetto importante è costituito dal dimensionamento delle celle. Celle più piccole hanno un numero minore di utenti al loro interno e possono quindi garantire loro bandwidth maggiori. Inoltre, si ottiene una distanza mediamente più breve tra dispositivo mobile e base station; una latenza wireless minore permette di realizzare reazioni tempestive alle condizioni variabili del collegamento, ottenendo così prestazioni migliori. D'altra parte, celle più ampie comportano un numero minore di handoff, in quanto gli utenti mobili tendono a trascorrere più tempo all'interno di ogni singola cella. Realizzando celle più grandi, inoltre, si rende necessario un numero complessivamente inferiore di base station e quindi si ha un costo realizzativo minore.

Nel tentativo di evitare handoff dispendiosi che farebbero calare sensibilmente le prestazioni complessive, sono state proposte diverse soluzioni. Alcune prevedono l'esistenza di celle parzialmente sovrapposte, in modo che, quando un utente entra in una di queste zone di sovrapposizione, i pacchetti destinati al ricevente transitino sia sulla vecchia, sia sulla nuova base station, creando le condizioni per il nuovo collegamento ancor prima di essere costretti ad abbandonare il vecchio [CI95]. Altre soluzioni propongono strutture gerarchiche in cui esistono più strati di indirizzamento [BS97]. In questo caso i dati necessari al mantenimento della connessione risiedono nel livello più alto, su di un host supervisore che controlla diverse zone nel livello sottostante. Così facendo si riesce a gestire un'area più ampia limitando gli handoff solo al caso in cui il dispositivo mobile stia transitando verso una nuova area controllata da un nuovo supervisore. Quando invece la mobilità avviene tra due aree gestite dallo stesso supervisore, non c'è bisogno di operare un handoff. Un ulteriore tentativo di minimizzare l'influenza degli

handoff sulle prestazioni del collegamento prevede l'utilizzo di tecniche multicast per mantenere aggiornate tutte le base station vicine al dispositivo mobile [MB97, TPL99]. Con questo meccanismo, l'utente continua a ricevere dati senza soluzione di continuità. Allo stesso tempo però, l'invio multicast di dati a più base station comporta un utilizzo dispendioso della bandwidth.

L'accesso wireless a Internet differisce dal caso wired sotto diversi aspetti. Per facilitare la comprensione di queste differenze elenchiamo le peculiarità più significative degli ambienti wireless nella colonna di sinistra della Tab. 3.1, nella colonna di destra elenchiamo invece le problematiche che scaturiscono dall'utilizzo del TCP in tale ambiente.

Notiamo innanzitutto che i dispositivi wireless sono costruiti per essere facilmente trasportabili dall'utente, hanno dunque dimensioni ridotte e sono alimentati tramite batterie. Ne consegue che l'energia è una risorsa limitata di cui occorre evitare sprechi e che analogamente anche la quantità di memoria interna è generalmente poco elevata. I dispositivi mobili vengono solitamente impiegati per una particolare tipologia di collegamenti: quelli di breve durata. Ciò è valido soprattutto per quel che concerne i telefoni cellulari.

Proprio perché mobili, i dispositivi che utilizzano reti wireless possono rilevare, in un breve lasso di tempo, situazioni ambientali molto diverse tra loro: a seconda della zona in cui si trova l'host mobile, si possono incontrare differenti intensità del segnale trasmissivo. La presenza di ostacoli fisici lungo il percorso e l'esistenza di "zone d'ombra" rendono ancor più variabile la qualità della connessione. La mobilità può quindi

provocare disconnessioni temporanee, handoff e fading[2], con il rischio che alcuni pacchetti vadano persi.

Caratteristiche degli ambienti Wireless	Problematiche che scaturiscono dall'utilizzo di TCP in ambiente Wireless
▪ Scarsità di memoria ▪ Dispositivi a batteria: scarsità di energia ▪ Sessioni di breve durata ▪ Disconnessioni, handoff e fading ▪ Latenza elevata e variabile ▪ Bandwidth scarsa e variabile ▪ BER elevato ▪ Errori in burst ▪ Mobilità della connessione	1. Incapacità di distinzione fra disconnessione, handoff, congestione, rumore 2. Finestra di invio ridotta ed incapace di crescere in modo adeguato (Spreco di Bandwidth) 3. Difficoltà nel calcolare il RTO 4. Ritrasmissioni ridondanti (Spreco di energia) 5. Lentezza nel recupero da disconnessioni (Spreco di Tempo) 6. Header voluminosi in rapporto alle dimensioni contenute dei pacchetti

Tab. 3.1 Caratteristiche dell'ambiente wireless e interazione con il TCP.

Nei collegamenti mobili la distanza fisica percorsa dai pacchetti in viaggio tra mittente e ricevente varia a seconda della posizione geografica del dispositivo. La latenza del collegamento pertanto, oltre

[2] Attenuazione del segnale. La causa può essere l'ingresso dell'utente in una zona in cui ostacoli fisici, oppure la distanza dalla sorgente trasmissiva, rendono difficoltose la comunicazione.

ad essere generalmente elevata, può anche variare fortemente in un breve lasso di tempo. L'eventuale presenza di meccanismi di correzione degli errori o di ritrasmissione a livello di collegamento, quali *Forward Error Correction* (FEC) e *Automatic Repeat Request* (ARQ) [CZ99], possono provocare ulteriore variabilità nella latenza misurata sulla connessione. Non ci addentriamo nei dettagli di questi meccanismi in quanto, se implementati, fanno parte dello strato di collegamento. Osserviamo comunque che FEC e ARQ permettono di recuperare da perdite dovute ad errori, al prezzo però di una parte della bandwidth e di una latenza ancor più variabile [Hus01].

La bandwidth disponibile nei collegamenti wireless risulta variabile e poco elevata; ciò si unisce ad una situazione generalmente poco affidabile delle connessioni mobili, dove si rilevano BER molto più elevati rispetto a quelle fisse. Le perdite nei collegamenti con dispositivi mobili sono spesso dovute a fattori diversi dalla congestione, quali ad esempio fading, handoff o rumore, e tendono a verificarsi raggruppati (burst) [BB95, BSAK95, Hus00b].

3.2 Interazione tra TCP e ambiente wireless

La caratteristiche di un ambiente wireless sopra esposte combinate con il comportamento del TCP provocano effetti non desiderati. Grazie alla Fig. 3.3 riusciamo a comprendere meglio le varie correlazioni tra le proprietà di un ambiente wireless e le problematiche che ne scaturiscono dal punto di vista del protocollo di trasporto. I riquadri bianchi riportano alcune delle caratteristiche di un ambiente wireless indicate precedentemente dalla Tab. 3.1, tra questi ce n'è uno con i bordi tratteggiati; la distinzione serve ad indicare che si tratta in realtà

di una caratteristica del TCP. I riquadri grigi evidenziano invece le reazioni anomale che scaturiscono dall'immersione del protocollo di trasporto in un ambiente wireless. Le frecce permettono di determinare l'origine dei problemi, ovvero partono da una causa e indicano una conseguenza.

Analizziamo la Fig. 3.3 partendo dal problema più evidente, ovvero la perdita di numerosi pacchetti. Per spiegare questo fenomeno sarebbe sufficiente ricordare che un ambiente wireless è condizionato da numerosi fattori, che vanno dagli aspetti climatici fino alla presenza o meno di ostacoli fisici lungo il percorso del segnale. Pertanto un utente mobile vedrà una connessione caratterizzata da un BER elevato [Hus00b]. Inoltre situazioni di disconnessione temporanea, handoff e fading possono produrre ulteriori perdite di pacchetti. Occorre anche considerare che la bandwidth disponibile per un collegamento wireless è in genere variabile[3] e dipende dal numero di utenti presenti in un dato istante in una cella, oltre che dalle condizioni del canale [BS97].

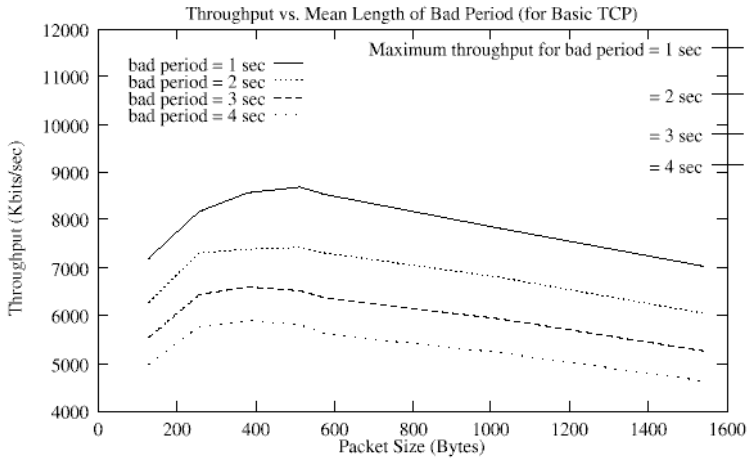

Fig. 3.2 Throughput ottenuti al variare delle dimensioni dei pacchetti [BKVP97].

[3] La variabilità aumenta in presenza di meccanismi di adattamento segnale-rumore [Hus01].

Come dimostrato dalla Fig. 3.2, una scelta ottimale nella dimensione dei pacchetti inviati può far ottenere alla connessione prestazioni migliori. Sull'asse delle ordinate si notano i diversi throughput realizzati dal TCP variando la dimensione, riportata sulle ascisse, dei pacchetti inviati. L'esperimento, presentato in [BKVP97], è stato condotto simulando un collegamento tra un host mobile ed uno fisso, separati dalla base station. Sul tratto wireless la bandwidth è stata impostata a 10 Mbps mentre su quello wireless la bandwidth era di 2 Mbps. Il canale alternava 10 secondi con un BER di 10^{-6} con un periodo di tempo in cui il BER saliva a 10^{-2}. L'esperimento è stato ripetuto utilizzando diversi intervalli di tempo di canale condizionato dagli errori: in particolare con 1, 2, 3 e 4 secondi. Dalle curve raffigurate, risulta evidente il guadagno prestazionale ottenuto utilizzando la dimensione più adeguata per i pacchetti inviati; ovviamente tale dimensione varia secondo le differenti condizioni del canale.

Le reti wireless sono tipicamente caratterizzate da pacchetti di dimensione più piccola rispetto a quelli che circolano nelle reti wired. La dimensione ridotta dei pacchetti sul collegamento wireless riduce la probabilità dei singoli pacchetti di giungere errati alla destinazione. I pacchetti in viaggio potrebbero dunque venire ulteriormente frammentati quando transitano sulla base station dirigendosi verso il collegamento wireless. La frammentazione dei pacchetti andrebbe però evitata ogni volta che è possibile [KM87]. Infatti, la perdita di uno di questi frammenti causa la perdita dell'intero pacchetto spedito dalla sorgente e quindi la sua ritrasmissione. Inoltre, ciascun nuovo pacchetto necessita di un header completo ed aumenta dunque il totale dei byte in transito. Complessivamente, oltre ad aumentare l'overhead totale, si verifica anche un aumento sensibile del BER rilevato.

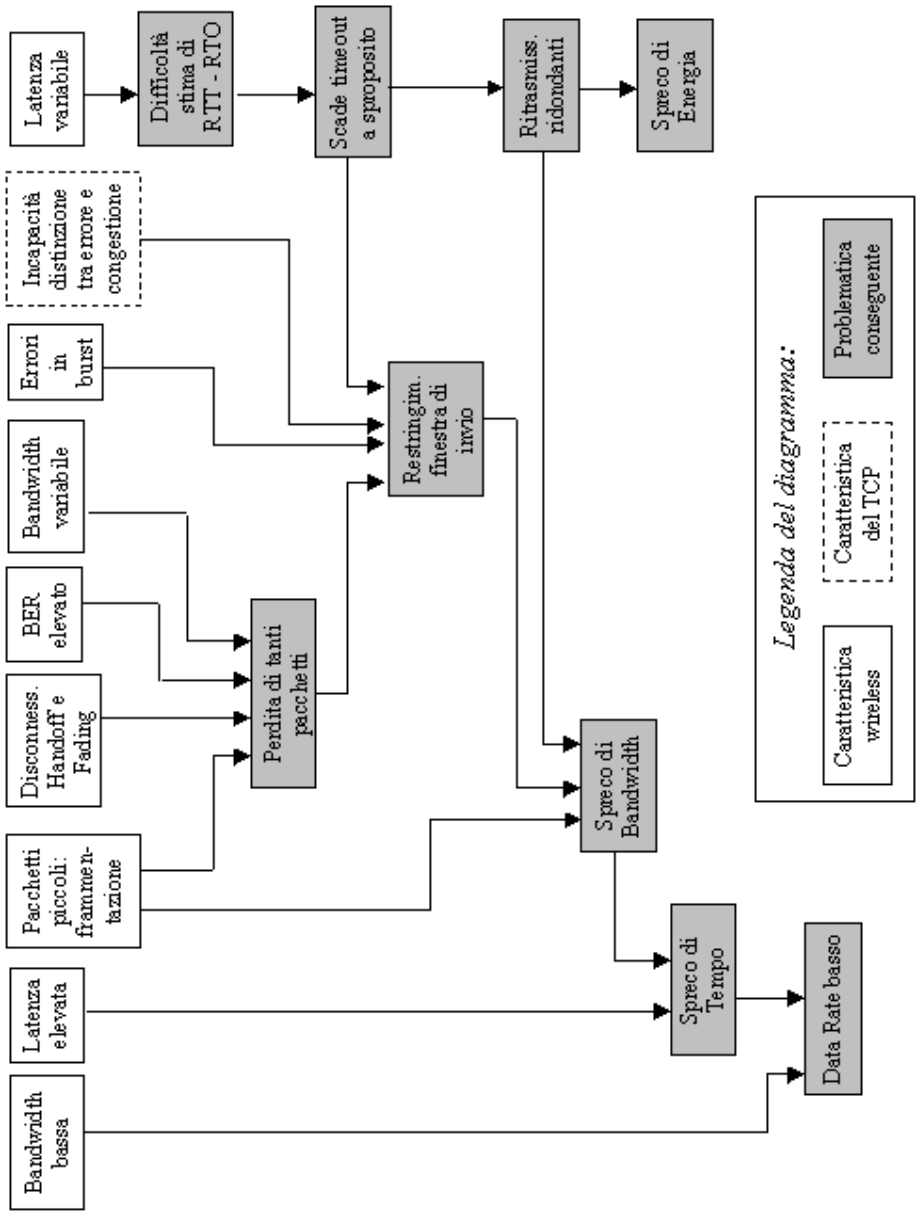

Fig. 3.2 Tipiche problematiche relative all'utilizzo di TCP in ambiente wireless: diagramma di causalità.

La perdita di tanti pacchetti è fra le cause che provocano un eccessivo restringimento della finestra di invio del mittente. Questa situazione viene esasperata dall'incapacità del TCP di distinguere tra le varie cause che portano alla perdita di un pacchetto. Ogni volta che lo scadere di un timeout o la ricezione di tre dupack indica che potrebbe essersi verificata la perdita di un pacchetto, TCP assume che la causa sia un livello di congestione della rete troppo elevato. Invece potrebbero essersi verificati errori transienti nonostante ci si trovi in presenza di bandwidth disponibile; in questi casi occorre ritrasmettere il pacchetto perso, ma non è necessario ed anzi risulta deleterio invocare meccanismi per il controllo della congestione. La costante interpretazione come congestione dei motivi che hanno portato ad una perdita causa quindi un errato atteggiamento del TCP e risulta in un utilizzo non efficiente delle risorse. Questa univocità interpretativa sarebbe giustificabile se ci trovassimo in un ambiente wired, non è più corrispondente alla realtà se parte del collegamento è di tipo wireless. In ambiente wireless le perdite causate da problemi diversi dalla congestione (errori, fading, handoff) sono tutt'altro che rari o irrilevanti. In questa tipologia di errori, l'assunto sbagliato che si tratti di congestione porta il TCP a restringere inutilmente la finestra di invio. La caratteristica degli errori di presentarsi in burst fa sì che questo errore interpretativo si ripeta più volte di seguito e che la finestra di invio venga ridotta diverse volte. Il ripetersi di questi errori impedisce che tale finestra possa crescere adeguatamente prima che si verifichi una nuova perdita. La conseguenza è evidente: la finestra di invio rimane ridotta rispetto all'effettiva disponibilità di bandwidth per tutto il collegamento.

Un'ulteriore causa che porta ad eccessivi restringimenti della finestra di invio trae origine dalla caratteristica degli ambienti wireless di avere una latenza fortemente variabile. TCP incontra quindi notevoli difficoltà ad effettuare una stima adeguata per il RTT e di conseguenza

ad impostare i timeout. Una stima ridotta del RTT rispetto alle condizioni correnti causa, infatti, lo scadere precoce dei timeout, prima che i dati in viaggio abbiano il tempo di raggiungere il destinatario. Non si tratta di vere perdite ma il TCP si comporta come se lo fossero, invocando il meccanismo di slow start e restringendo la finestra di invio. Se viceversa il RTT viene sovrastimato, TCP non può reagire tempestivamente ad eventuali perdite dovute a congestione, attendendo inutilmente che scada un timeout impostato troppo ampio prima di poter effettuare la ritrasmissione dei pacchetti persi.

La scadenza a sproposito dei timeout, provoca un ulteriore problema che è quello delle ritrasmissioni inutili e ridondanti. Inviare ulteriori copie dello stesso segmento quando la copia (o le copie) precedente è ancora in viaggio è, oltre che superfluo, anche deleterio per le prestazioni. Infatti, si occupa inutilmente della bandwidth che potrebbe essere impiegata più efficientemente inviando nuovi dati. Come conseguenza si ottiene uno spreco di risorse: di bandwidth ma anche di energia. Il fattore energetico non deve essere sottovalutato: i dispositivi mobili hanno batterie che consentono loro un periodo di utilizzo effettivo molto limitato, evitare sprechi significa quindi fornire un servizio migliore all'utente. Ulteriore ridondanza nell'invio di dati può essere causato dalla presenza di meccanismi di ritrasmissione nello strato di collegamento. La base station potrebbe infatti implementare dei meccanismi autonomi di ritrasmissione sul canale wireless. Con questa funzionalità aggiuntiva si cerca di ridurre le ritrasmissioni totali da parte del mittente: diverse perdite potrebbero infatti essere recuperate "in loco". Ciò non impedisce però che possa scadere nel frattempo anche il timeout del mittente, soprattutto in situazioni di ritrasmissioni multiple nello strato di collegamento oppure se il RTT sul percorso wireless è piuttosto elevato. Il dispositivo mobile potrebbe quindi ricevere entrambe le ritrasmissioni, alla prima della quale risponde con un ack diretto verso il mittente. Il mittente non è però in

grado di determinare se l'ack ricevuto fa riferimento al primo pacchetto inviato, o alla sua ritrasmissione in seguito al timeout, oppure ancora se sia dovuto a ritrasmissioni locali dello strato di collegamento. Abbiamo quindi trovato un ulteriore elemento che ostacola la stima corretta del RTT.

Lo spreco di bandwidth è uno dei punti centrali nella comprensione delle cause che portano a performance non positive quando si utilizza il TCP in un ambiente wireless. La bandwidth utilizzabile dai dispositivi mobili è infatti una risorsa generalmente limitata, va dunque gestita con attenzione se non si vuole che una frequenza di trasmissione troppo bassa scoraggi possibili utenti dall'utilizzare questa tecnologia. Proprio analizzando l'efficienza nell'utilizzo della bandwidth, notiamo che TCP imposta la finestra iniziale di invio in maniera prudente, ponendola uguale ad 1 MSS. In questo modo potrebbero passare diversi RTT prima che si cominci ad utilizzare in maniera adeguata la bandwidth disponibile [PN98]. Le connessioni wireless sono generalmente di breve durata e dunque il periodo iniziale con finestre di invio molto piccole diviene percentualmente consistente rispetto al tempo totale del collegamento. Le ridotte dimensioni della finestra iniziale di invio, provocano quindi uno spreco significativo di bandwidth e quindi un aumento della durata della connessione.

Abbiamo già ampiamente illustrato come i dispositivi mobili potrebbero trovarsi in una zona in cui il segnale trasmissivo risulta attenuato o del tutto assente. Ciò può causare una disconnessione e la perdita di diversi pacchetti. Ma non solo, al ristabilirsi della connessione, TCP mostra ancora una volta le sue lacune nell'interazione con dispositivi mobili. Infatti, anziché riprendere immediatamente ad inviare dati, TCP deve attendere che scada un RTO. Quando ciò avviene, TCP entra in modalità slow start e ricomincia ad

inviare dati. Allo stesso tempo però, interpretando l'evento come il segnale di una congestione della rete, TCP riporta la finestra di invio alla dimensione di un singolo segmento. E' evidente che se al momento in cui la connessione riprende, la rete non si trova affatto congestionata, la finestra di invio ridotta comporta un spreco delle risorse disponibili.

La combinazione di tutti i fattori esposti rende quindi le versioni tradizionali di TCP inadeguate ad affrontare un ambiente wireless. L'appetibilità di ogni nuova tecnologia dipende fortemente dalla qualità del servizio che essa può offrire. Le potenzialità dei dispositivi mobili di soddisfare le necessità di informazione da parte degli utenti in ogni luogo ed in ogni momento, necessitano quindi di essere supportate da un servizio efficiente. Per realizzare questo obiettivo è indispensabile disporre di un protocollo di trasporto in grado di gestire le nuove caratteristiche delle connessioni wireless. Analizzeremo quindi nel capitolo successivo le più rilevanti di soluzioni proposte, evidenziandone pregi e lacune.

4

ANALISI CRITICA DELLE SOLUZIONI PROPOSTE

In questo capitolo presentiamo un'analisi critica dei principali schemi proposti per implementare il protocollo di trasporto TCP in ambiente wireless. In particolare vengono esaminati i nuovi meccanismi alla luce dei benefici prestazionali apportati ma anche delle implicazioni negative che il loro utilizzo comporta.

4.1 Analisi delle diverse tipologie di approccio

Dopo aver visto quali sono i problemi generati dall'utilizzo del TCP in un ambiente wireless, vediamo ora quali sono i diversi tipi di approccio ideati per affrontarli. In particolare si possono identificare tre gruppi principali di soluzioni [Ewe01]. Un primo schema è quello di cercare di ridurre l'incidenza degli errori sul wireless utilizzando soluzioni che richiedono l'intervento dello strato di collegamento. Altre soluzioni partono dal presupposto che il TCP è nato quando ancora non si prospettava l'odierna diffusione di dispositivi mobili per il collegamento ad Internet; occorre pertanto creare un nuovo protocollo per il livello di trasporto al fine di ottenere prestazioni significativamente migliori. Un terzo tipo di approccio è invece volto a mantenere la struttura attuale di Internet, modificandola il meno possibile ed in misura circoscritta alla parte wireless. Per realizzare

questo schema, il collegamento tra l'host fisso ed il dispositivo mobile viene diviso in due parti dalla base station.

Le soluzioni basate su di uno strato di collegamento affidabile esulano dall'argomento di interesse di questa tesi. Ci limitiamo ad alcune osservazioni relative al rapporto tra questo tipo di soluzioni e lo strato di trasporto. Rileviamo innanzitutto che gli schemi basati su ritrasmissioni dello strato di collegamento sono utilizzabili anche in presenza di altre soluzioni che operano nello strato di trasporto. Segnaliamo inoltre che, grazie alla loro capacità di recupero locale dei pacchetti persi, questi schemi forniscono al protocollo dello strato di trasporto un servizio più affidabile. Il protocollo di trasporto viene così gravato di una percentuale inferiore di errori, ottenendo prestazioni migliori. D'altra parte, le ritrasmissioni nello strato di collegamento potrebbero interferire con quelle dello strato di trasporto, causando ritrasmissioni ridondanti ed errori nella stima del RTT[4].

Gli schemi che affrontano i problemi legati al collegamento wireless tramite la realizzazione di un protocollo specifico, cercano di rendere il protocollo di trasporto cosciente della presenza di una componente mobile. Questo tipo di soluzione però potrebbe richiedere modifiche, anche ampie, al mittente, al ricevente e ai nodi intermedi. A seconda di dove sono richiesti gli interventi per l'implementazione del nuovo protocollo, cambiano le probabilità per una sua applicazione reale. Il caso più semplice, e quindi preferibile, è quello in cui le modifiche siano necessarie soltanto nel dispositivo mobile. In questo caso è sufficiente implementare il nuovo protocollo di trasporto sui nuovi

[4] Si pensi al caso in cui il protocollo di trasporto effettui una ritrasmissione in seguito a timeout o a dupack. Se però nel frattempo, lo strato di collegamento opera anch'esso una ritrasmissione, la quale ha buon esito, il ricevente informerà il mittente del successo della trasmissione tramite ack. Al mittente però non è dato di sapere se tale ack fa riferimento al primo segmento inviato o alla sua ritrasmissione, con conseguenti possibili errori nel calcolo del RTT.

dispositivi messi in vendita. Le aziende venditrici potrebbero anche fornire ai propri clienti un servizio gratuito di aggiornamento del software, operazione già praticata dalle principali aziende produttrici di telefoni cellulari. Anche il caso in cui le modifiche richieste per installare il nuovo protocollo di trasporto siano limitate all'host fisso è accettabile. Le aziende fornitrici di servizi via Internet troveranno infatti conveniente aggiornare il proprio software pur di offrire alla propria clientela un prodotto migliore. Il caso in cui le modifiche siano richieste contemporaneamente al mittente e al ricevente, invece, crea qualche problema. Se infatti lo schema innovativo è implementato solo in uno dei due host, si perdono i vantaggi derivanti dal suo utilizzo; risulta anzi fondamentale che l'host dotato del nuovo protocollo di trasporto possa interagire anche quello tradizionale, viceversa la comunicazione non sarà possibile. L'implementazione di protocolli di trasporto che richiedano modifiche al codice presente nei router della rete è altamente improbabile. Data la loro quantità e in considerazione del numero e della variegata tipologia di proprietari, risulta praticamente impossibile modificare tutti i nodi intermedi di Internet.

Modifiche ampie del TCP attuale potrebbero produrre incrementi prestazionali più elevati, occorre però tenere conto delle considerazioni esposte sopra a proposito di dove si concentrino tali modifiche. Inoltre, è necessario evitare che l'ansia di produrre risultati positivi non porti a realizzare nuovi protocolli eccessivamente aggressivi, con cali conseguenti nelle prestazioni complessive di Internet [Hus00b]. Ricordiamo infine che le soluzioni che prevedono la modifica del TCP solitamente mantengono il paradigma end-to-end e vengono anche chiamate soluzioni *All-IP*. La conservazione del paradigma end-to-end è importante perché garantisce la consegna dei dati su qualsiasi tipo di rete eterogenea [Cla88]. La rete deve essere vista come qualcosa di indefinito, che non offre informazioni esplicite sul livello di congestione.

Le soluzioni che prevedono la divisione della connessione in due parti nascono dall'idea di risolvere localmente i problemi relativi al collegamento wireless. Per questi schemi viene spesso usato l'appellativo di *walled garden*, proprio per indicare la loro caratteristica di nascondere all'host fisso le problematiche della connessione mobile [Hus01]. L'host fisso non è infatti cosciente della presenza di un tratto wireless nel collegamento. In questa tipologia di soluzioni, sul collegamento tra la sorgente fissa e la base station viene utilizzato il TCP tradizionale, mentre sul collegamento tra la base station ed il dispositivo mobile è auspicabile l'utilizzo di un protocollo di trasporto appositamente adattato. Al confine tra le due parti viene posto un server proxy che permette all'applicazione di interagire con il dispositivo wireless. Questo tipo di approccio permette di isolare i problemi legati alla presenza di una componente wireless nel collegamento.

La divisione in due della connessione apporta dei benefici anche nel caso in cui il protocollo di trasporto utilizzato nel tratto wireless rimanga quello tradizionale. Infatti, la connessione tra base station e dispositivo mobile ha solitamente un RTT minore, anche notevolmente, rispetto a quello tra host fisso ed host mobile. Di conseguenza, le problematiche inerenti il tratto wireless possono quindi essere affrontate più tempestivamente. Tuttavia, incrementi prestazionali più evidenti si ottengono nel caso in cui il collegamento tra base station e dispositivo mobile sia dotato di un protocollo di trasporto appositamente ottimizzato per operare in un ambiente wireless.

L'utilizzo di soluzioni che affrontano localmente i problemi dell'ambiente wireless permette di ottenere incrementi prestazionali rispetto ai protocolli di trasporto tradizionali. Tuttavia, questa tipologia di soluzioni può richiedere il monitoraggio da parte della base station

dei pacchetti in transito, rendendo impossibile l'invio di dati criptati. L'esigenza di trasmissioni sicure si sta invece diffondendo sempre più fra tutte le tipologie di utenti. Un ulteriore svantaggio dei protocolli di trasporto che prevedono la divisione della connessione è che spesso non mantengono intatto il paradigma end-to-end di Internet. Oltre ai già citati problemi di interoperabilità tra reti di diverso tipo, l'intervento attivo della base station su dati e ack può far perdere le garanzie di sicurezza e di affidabilità nelle trasmissioni (si veda ad esempio il par. 4.2). Infine, la gestione di due connessioni richiede alla base station un aumento del carico computazionale e delle risorse di memoria disponibili.

Soluzioni con divisione della connessione	**Soluzioni senza divisione della connessione.**	**Soluzioni nello strato di collegamento.**
4.1 I-TCP 4.2 M-TCP 4.3 Snoop Protocol	4.4 Delayed Dupacks 4.5 TCP-Aware 4.6 Freeze-TCP 4.7 TCP Probing 4.8 WTCP 4.9 TCP Westwood	ECN ELN EBSN FEQ ARQ

Tab. 4.1 Categoria dei diversi approcci presentati.

Prima di analizzarle nel dettaglio, riportiamo in Tab. 4.1 i nomi dei protocolli di trasporto scientificamente più rilevanti tra quelli appositamente realizzati per l'ambiente wireless. Per facilitare una immediata classificazione, abbiamo diviso l'elenco dei nuovi protocolli a seconda della tipologia di soluzione attuata. Alla sinistra di ciascun

nome è riportato il numero di paragrafo corrispondente al protocollo. Nella colonna di destra elenchiamo, per completezza, alcuni schemi implementabili nello strato di collegamento per incrementare le prestazioni delle connessioni con dispositivi wireless. Oltre ai meccanismi di FEQ e ARQ già nominati nel par. 3.1, ricordiamo gli schemi di *Explicit Congestion Notification* (ECN) [RF99], *Explicit Loss Notification* (ELN) [BPSK97] e *Explicit Bad State Notification* (EBSN) [BKVP97]. Si tratta di meccanismi per la notifica esplicita delle cause delle perdite, ovvero rispettivamente congestione, errore e cattivo stato del canale. Su questi meccanismi non ci soffermeremo oltre in quanto argomento che esula dall'oggetto trattato in questa tesi.

4.2 I-TCP (Bakre e Badrinath, 1995)

Il protocollo I-TCP fu ideato nel 1995 da Ajay Bakre e B. R. Badrinath della Rutger University, in risposta alla necessità di affrontare i problemi causati dalla mobilità e dalla inaffidabilità dei collegamenti wireless. Questo protocollo di trasporto indiretto utilizza le risorse di router, denominati *Mobility Support Router* (MSR), posti tra la parte fissa e quella mobile della rete, per gestire le problematiche wireless completamente all'interno della parte wireless del collegamento. In questa maniera si riescono ad incrementare le prestazioni di un dispositivo mobile senza aver modificato il codice TCP/IP nell'host fisso. L'utilizzo di un protocollo completamente nuovo creerebbe, infatti, problemi di interoperabilità con il resto della rete.

Soluzioni precedenti, come i protocolli Thinwire [FDC84] o la compressione degli header [Jac90], avevano già cercato di alleviare

alcuni problemi nei collegamenti lenti ma non avevano trattato adeguatamente il problema della mobilità di uno dei due host. Anche le ritrasmissioni effettuate nello strato di collegamento, sebbene permettano di portare la frequenza degli errori di una connessione wireless ad un livello vicino a quello di una wired, non sono esenti da problemi poiché possono interferire con le ritrasmissioni end-to-end proprie del TCP, ottenendo quindi prestazioni non all'altezza delle aspettative. La mobilità di uno dei due host può essere all'origine di cali consistenti nelle performance del collegamento. Diversi segmenti TCP potrebbero andare persi durante l'attraversamento del dispositivo mobile da una cella ad un'altra, specialmente nel caso in cui le due celle non siano sovrapposte, facendo così erroneamente attivare i meccanismi di controllo della congestione.

L'approccio di I-TCP è quello di spezzare la connessione TCP in due parti, una tra l'host fisso e il MSR e l'altra tra il MSR e l'host mobile, in modo da nascondere i problemi della parte wireless a quella wired. L'uso della indirezione permette infatti di separare il controllo del flusso e della congestione nella parte wireless da quelli della parte wired, consentendo così di trattare efficacemente le diverse caratteristiche. Un protocollo di trasporto separato per il mezzo wireless potrebbe anche supportare notifiche esplicite di eventi quali disconnessioni, movimenti, informazioni in genere (bandwidth disponibile ad esempio). L'indirezione permette inoltre al MSR di gestire vari servizi della rete fissa per conto dell'host mobile, il quale può utilizzare un protocollo semplificato rispetto al TCP/IP completo.

I-TCP è un protocollo di trasporto pienamente compatibile con TCP/IP e si fonda su pochi principi base. Innanzitutto, la connessione tra l'host fisso e quello mobile è in realtà formata da due connessioni distinte, separate da una base station (MSR). La gestione della connessione

corrente viene trasferita dal vecchio al nuovo MSR qualora il dispositivo mobile cambi cella. Occorre inoltre ricordare che l'host fisso non è consapevole della indirezione, né interessato dal cambiamento di cella da parte dell'host mobile.

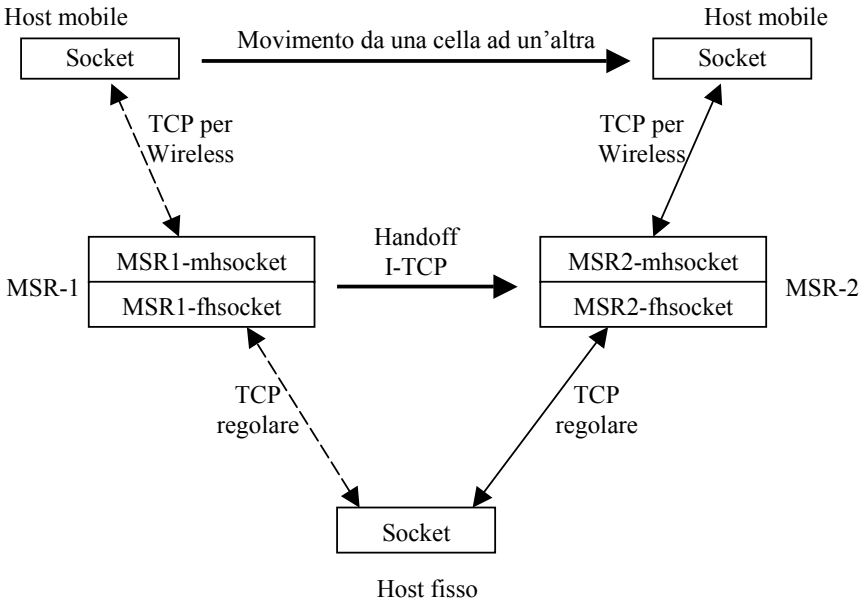

Fig. 4.1 Handoff di una connessione I-TCP [BB95].

L'host mobile che voglia comunicare con un host fisso tramite I-TCP deve fare richiesta, tramite speciali socket del protocollo I-TCP, al MSR corrente perché apra una connessione con l'host fisso per suo conto. Una volta stabilita la connessione, si potranno utilizzare le socket normali per inviare o ricevere dati. L'host mobile comunica con la base station utilizzando un TCP modificato per offrire prestazioni migliori in ambiente wireless. L'host fisso vede solo una immagine dell'host mobile, tale immagine risiede sul MSR corrente e viene trasferita al nuovo MSR nel caso in cui il dispositivo mobile cambi cella. In caso di handoff, il nuovo MSR crea le socket corrispondenti alle due connessioni, quella wired e quella wireless, come illustrato in Fig. 4.1.

Siccome le parti finali delle due connessioni non cambiano (cambia solo la base station), non c'è bisogno di ristabilire la connessione.

Il contributo principale apportato dal I-TCP risiede nella suddivisione della connessione in due parti autonome, una per la parte wired e una per quella wireless del collegamento. Sulla seconda, possono essere utilizzati protocolli specificamente adattati per l'ambiente wireless e le sue problematiche, in modo da migliorare le prestazioni globali. Il recupero dalle perdite dovute alle condizioni della parte wireless vengono recuperate tempestivamente con ritrasmissioni da parte del MSR, anziché far intervenire l'host fisso. I-TCP utilizza ack distinti per la parte wireless del collegamento e per quella wired. Il MSR invia immediatamente l'ack al mittente quando riceve dei dati, dopodiché cerca di inviare tali dati verso l'host mobile. In questo modo, l'host fisso ignora completamente gli eventuali problemi legati alla componente wireless del collegamento.

Per dimostrare le migliori prestazioni di I-TCP rispetto al TCP tradizionale nella gestione degli handoff, Bakre e Badrinath hanno realizzato una serie di esperimenti in cui si confrontavano i due protocolli sotto varie condizioni. In particolare sono stati registrati i throughput relativi all'invio di alcuni dati su brevi distanze e su lunghe distanze. Ciascuna di queste due tipologie di trasmissione è stata ulteriormente suddivisa in quattro casi: nessun movimento del dispositivo mobile, movimento del dispositivo in un contesto avente celle sovrapposte, movimento in un contesto senza celle sovrapposte ma anche senza perdita di tempo nel riprendere le trasmissioni nella nuova cella, e movimento tra celle non sovrapposte e con un ritardo di un secondo prima di riprendere le trasmissioni nella nuova cella. Nel caso di trasmissioni su brevi distanze sono stati inviati 4 MB, mentre su lunghe distanze sono stati trasmessi 2 MB. Nei casi sperimentali in cui

siano previsti degli spostamenti del dispositivo tra due celle, questi avvengono ogni 8 secondi. I risultati pubblicati in [BB95], sono da noi riportati nella Tab. 4.2 per il caso su brevi distanze e nella Tab. 4.3 per il caso su lunghe distanze.

Tipo di connessione	Nessun movimento	Celle sovrapposte	Celle non sovrapposte con 0 sec tra le celle	Celle non sovrapposte con 1 sec tra le celle
TCP	65,49 KB/s	62,59 KB/s	38,66 KB/s	23,73 KB/s
I-TCP	70,06 KB/s	65,37 KB/s	44,83 KB/s	36,31 KB/s

Tab. 4.2 Throughput su brevi distanze [BB95].

Tipo di connessione	Nessun movimento	Celle sovrapposte	Celle non sovrapposte con 0 sec tra le celle	Celle non sovrapposte con 1 sec tra le celle
TCP	13,35 KB/s	13,26 KB/s	8,89 KB/s	5,19 KB/s
I-TCP	26,78 KB/s	27,97 KB/s	19,12 KB/s	16,01 KB/s

Tab. 4.3 Throughput su lunghe distanze [BB95].

In tutti i casi le prestazioni misurate del I-TCP risultano migliori rispetto a quelle del TCP tradizionale. Questo è dovuto al fatto che il protocollo I-TCP riesce a proteggere il mittente dalle perdite e dunque ad evitare di invocare meccanismi quali slow start e fast retransmit. Le differenze appaiono ancor più evidenti nei throughput relativi a trasmissioni su lunghe distanze, dove i lunghi RTT amplificano la degradazione delle prestazioni del TCP tradizionale dovuta all'errata invocazione dei meccanismi di controllo della congestione. Con I-TCP

invece le perdite relative al collegamento wireless vengono recuperate localmente e, essendo la connessione tra host mobile ed MSR molto più breve rispetto a quella tra host mobile ed host fisso, il throughput misurato per I-TCP risulta, nel caso peggiore, il triplo di quello del TCP tradizionale.

Le buone prestazioni di I-TCP si accompagnano però ad alcuni difetti. Proprio la modalità con cui il protocollo I-TCP realizza la separazione della connessione fa perdere la semantica end-to-end che è propria del TCP tradizionale. Infatti la presenza di ack distinti per la parte wireless del collegamento e per la parte wired causa inaffidabilità del sistema. Può accadere che un pacchetto raggiunga il MSR e che questa invii quindi un ack verso il mittente per confermare l'avvenuta ricezione corretta dei dati. La semplice ricezione dei dati da parte del MSR non garantisce però che tali dati raggiungano poi anche il destinatario finale. Se immediatamente dopo l'invio dell'ack dal MSR all'host fisso, vi fosse un malfunzionamento del primo, si verificherebbe una perdita irrecuperabile dei dati. Occorre inoltre rilevare che le procedure di handoff, soprattutto se frequenti ed in presenza di più connessioni, potrebbero risultare dispendiose nel trasferimento di tutti i pacchetti in memoria da una base station all'altra [BS97]. Un ulteriore degradamento delle prestazioni potrebbe essere dovuto a perdite che si verificano qualora vi sia una differenza elevata tra la bandwidth disponibile sulla parte wired del collegamento e quella disponibile nel tratto wireless. In questo caso, infatti, i segmenti inviati dal mittente tendono ad accumularsi nei buffer della base station fino ad essere scartati una volta che questi buffer divengono pieni. Infine, dobbiamo rilevare che la gestione di due connessioni distinte porta ad un aumento della latenza totale end-to-end.

4.3 M-TCP (Brown e Singh, 1997)

Il protocollo M-TCP fu sviluppato nel 1997 da Kevin Brown e Suresh Singh della University of South Carolina, Columbia. Analogamente al I-TCP, propone di dividere la connessione in due parti, la prima sulla parte wired del collegamento e la seconda sulla parte wireless. L'obiettivo era quello di creare un protocollo di trasporto che potesse risultare efficace su collegamenti caratterizzati da bandwidth bassa e variabile, anche in presenza di frequenti disconnessioni. Diversamente da I-TCP, il nuovo schema doveva inoltre mantenere la semantica end-to-end.

Per capire le motivazioni che hanno portato a scegliere la struttura dell'architettura, dobbiamo innanzitutto ricordare le implicazioni nella scelta delle dimensioni di una cella. Come già illustrato nel paragrafo 3.1, reti picocellulari offrono generalmente bandwidth più elevate ma comportano anche un numero maggiore di handoff. Inoltre, occorrono investimenti consistenti per realizzare tante celle, ognuna delle quali deve essere controllata da una base station.

L'architettura proposta, illustrata in Fig. 4.2, si basa su una gerarchia a tre livelli [BS97]: *Supervisor Host* (SH), *Mobile Support Station* (MSS) e *Mobile Host* (MH). Al livello più basso troviamo i dispositivi mobili (MH) i quali comunicano, all'interno di ciascuna cella, con il relativo MSS. Al livello più alto troviamo i SH, ciascuno dei quali controlla più MSS. Il SH è connesso con la rete fissa e funge da gateway provvedendo all'instradamento dei segmenti. Deve inoltre gestire la maggior parte dei dettagli necessari alla gestione degli utenti mobili e mantenere le condizioni di QoS negoziate. Lo svantaggio della complessità dei SH viene ripagato dalla semplicità dei MSS che,

costando poco, permettono il loro acquisto in numero elevato e dunque la creazione di celle di piccole dimensioni. Inoltre non vi è alcun trasferimento di informazioni nello spostamento del dispositivo mobile tra due MSS appartenenti allo stesso SH. Ciò risulta in $O(\sqrt{n})$ disconnessioni per unità di tempo, nel caso in cui vi siano regioni di n celle governate da un SH, contro $O(n)$ disconnessioni nel caso in cui siano gli MSS da soli a gestire ciascuna regione.

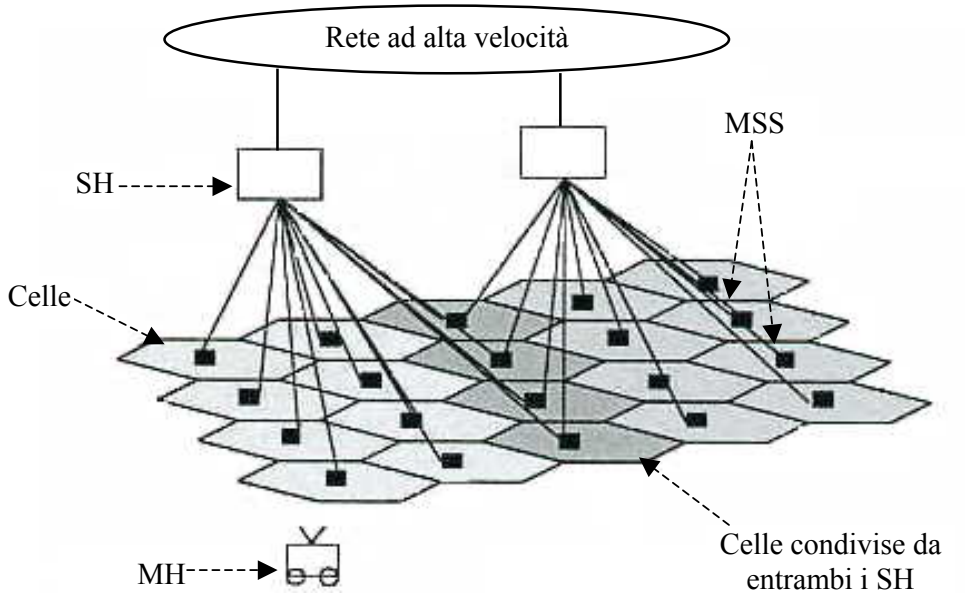

Fig. 4.2 Architettura del M-TCP [BS97].

Per gestire i cambiamenti dinamici nella bandwidth disponibile all'interno di una cella, dovuti all'ingresso o all'uscita di utenti, il protocollo M-TCP prevede di realizzare un apposito modulo di gestione della bandwidth nel SH. Questo modulo assegna una quantità fissata di bandwidth a ciascuna connessione e si assicura che i dati vengano trasmessi con la quantità assegnata di banda, permettendo alcune forme limitate di QoS. La Bandwidth assegnata viene periodicamente ricalcolata in base alle modificate esigenze dei vari utenti.

Le frequenti disconnessioni e l'alimentazione a batteria, tipiche dei dispositivi mobili, impongono ulteriori attenzioni da parte del protocollo di trasporto. Nello schema proposto dal M-TCP, i SH forniscono un recupero "locale" contro le perdite tipiche degli ambienti wireless e tengono traccia del movimento degli utenti, evitando che il numero di pacchetti duplicati trasmessi divenga elevato e sprechi inutilmente energia.

Analizzando più in dettaglio il meccanismo del M-TCP, notiamo che il client TCP nel SH (che chiameremo SH-TCP) riceve i segmenti trasmessi dal mittente e li passa al client M-TCP perché li propaghi verso il dispositivo mobile. Analogamente, gli ack ricevuti dal M-TCP del SH sono passati al SH-TCP per essere inoltrati al TCP del mittente TCP. Come illustrato in Fig. 4.3, nel SH risiedono sia SH-TCP, sia M-TCP. Quando il SH riceve un pacchetto dal mittente, non risponde immediatamente inviando un ack, come invece accadeva per i protocolli I-TCP, bensì attende di ricevere l'ack del pacchetto dall'host mobile, in modo da essere sicuro che il pacchetto abbia effettivamente raggiunto la destinazione. Questo comportamento garantisce il mantenimento della semantica end-to-end.

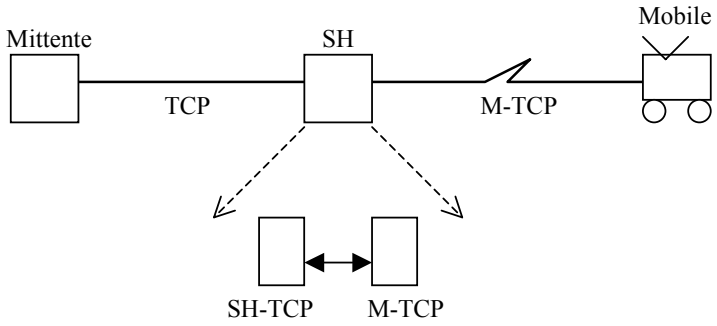

Fig. 4.3 Impostazione di una connessione [BS97].

Quando viene rilevata una disconnessione, M-TCP "soffoca" il TCP del mittente, così da non invocare erroneamente procedure per il controllo della congestione. Per realizzare questo meccanismo, vengono inviati gli ack di notifica per tutti i byte correttamente ricevuti tranne che per l'ultimo. Qualora si verifichi una disconnessione, l'ack trattenuto viene inviato verso il mittente con all'interno la direttiva di aggiornare la dimensione della finestra di invio, ponendola uguale a zero. Quando l'host fisso riceve questo ack, entra in *persist mode* e in questa modalità non subisce timeout, né aggiustamenti del RTT e nemmeno restringimenti della finestra di congestione. L'ack trattenuto viene inviato, con indicazione di porre la dimensione della finestra di invio uguale a zero, anche nel caso in cui non ci si trovi di fronte ad una disconnessione, bensì ad una situazione in cui la bandwidth disponibile sia molto scarsa. In questo modo SH-TCP può stimare il RTT e l'intervallo di RTO, ed usare queste informazioni per ridurre la finestra di invio prima che il mittente sia costretto ad invocare meccanismi di exponential backoff. A questo scopo, viene mantenuto nel SH un timer inizializzato col valore stimato del RTO.

Per comprendere quando è avvenuta una disconnessione, M-TCP monitora il flusso di ack: se non ne riceve per un determinato periodo di tempo, allora ipotizza che il dispositivo mobile si sia disconnesso. Ne informa quindi il SH-TCP che provvede a congelare il mittente come già spiegato. Il SH riconosce il ripristino di una connessione perché il suo M-TCP riceve di nuovo l'ultimo ack, contrassegnato come ack di riconnessione, da parte dell'host mobile. La trasmissione dei dati dovrebbe ripartire a piena velocità non appena l'host mobile riacquisisce la connessione. A questo scopo, il dispositivo mobile invia immediatamente un messaggio di connessione ristabilita al SH. Qui M-TCP lo raccoglie e passa l'informazione al SH-TCP, il quale invia un ack al mittente e fa riaprire la sua rwnd. Il mittente può quindi uscire dal persist mode e ricominciare a spedire dati con la stessa velocità di

quando aveva smesso poiché, nel frattempo, non è stato invocato alcun meccanismo di controllo della congestione.

Quando SH-TCP ritiene che non ci saranno più nuovi ack da parte del dispositivo mobile, ovvero quando sta per scadere il timeout del mittente e l'host mobile si trova in una cella con una buona bandwidth, provvede all'invio dell'ultimo ack.

Gli sviluppatori del M-TCP hanno deciso di non implementare il meccanismo *sack* nel loro protocollo, giustificando questa scelta con l'affermazione che l'utilizzo dei pacchetti di tipo sack risulta di utilità limitata in presenza di disconnessioni [BS97].

Un'altra caratteristica da segnalare è costituita dal fatto che il M-TCP presente nel SH non restringe la sua cwnd allo scadere del timeout. La motivazione di questo comportamento è che questi timeout sono dovuti per lo più a disconnessioni e non a congestioni. Lo slow start viene quindi invocato solo all'inizio di una connessione. Gli ideatori del protocollo M-TCP arrivano addirittura ad ipotizzare l'eliminazione di meccanismi di controllo della congestione, e quindi dello slow start, poiché il SH gestisce già l'allocazione della bandwidth per ogni dispositivo mobile. Appare quindi non necessario eseguire la crescita progressiva del throughput per saggiare la disponibilità di bandwidth, come invece tipicamente avviene nei tradizionali protocolli di trasporto. La rimozione dei meccanismi per il controllo della congestione potrebbero avere effetti positivi soprattutto nel caso di piccoli trasferimenti di dati, quando il tempo perso per saggiare la disponibilità di banda risulta in percentuale consistente, rispetto ad utilizzare da subito la bandwidth assegnata dal SH.

Con collegamenti wireless lenti potrebbe risultare conveniente l'utilizzo della compressione degli header dei pacchetti [Jac90]. D'altra parte però, questo metodo si basa sulle differenze incrementali e obbliga a ricevere i pacchetti consecutivamente: ciò costituisce un problema in presenza di BER elevati. In ogni caso, la compressione degli header porta ad un guadagno percentuale dal 2 al 7% sulla dimensione totale di ogni pacchetto; Brown e Singh suggeriscono quindi di comprimere i dati stessi con il codice gzip. La compressione dei dati potrebbe ridurre il tempo di trasmissione, ma comporta anche un incremento significativo dei tempi di calcolo effettuati dal SH; il suo utilizzo risulta quindi conveniente solo in casi di bandwidth disponibile estremamente limitata [BS97].

Un pregio importante del M-TCP è quello di aver ideato un'architettura che permette una gestione delle celle efficace. Con la gerarchia a tre livelli proposta si riesce ad ottenere celle di piccola dimensione, e quindi bandwidth maggiori, e allo stesso tempo handoff meno pesanti.

Analizzando le qualità del protocollo M-TCP, dobbiamo riconoscere che la capacità di congelare le trasmissioni permette un buon comportamento in presenza di disconnessioni; si evitano, infatti, i timeout in serie tipici dei TCP tradizionali in contesti analoghi e i dati vengono inviati sempre alla massima velocità disponibile. Contrariamente al I-TCP, il timeout del mittente si basa sul RTT calcolato nel percorso fino al dispositivo mobile e la frequenza trasmissiva dell'host fisso è adattata alla bandwidth wireless. Inoltre gli ack dei pacchetti ricevuti dalla base station non vengono inviati da questa verso l'host fisso immediatamente, bensì solo dopo aver ricevuto la conferma dal dispositivo mobile che quei pacchetti sono effettivamente giunti a destinazione: rimane quindi soddisfatta la semantica end-to-end della connessione. Analogamente al I-TCP, la

divisione della connessione in due parti permette il recupero locale degli errori, con una operazione più veloce rispetto al caso in cui dovesse essere l'host fisso ad occuparsene.

Le migliori prestazioni di M-TCP rispetto a TCP sono evidenti in Fig. 4.4 che riporta alcuni dati sperimentali presentati in [BS97]. In tale grafico si notano i diversi tempi di trasmissione necessari ai due protocolli per inviare un file di 1MB su di una connessione in cui il mittente dista cinque passi dal ricevente al variare della lunghezza dei periodi di disconnessione. Per realizzare l'esperimento è stato supposto che la permanenza media del dispositivo mobile all'interno di una cella sia di 5 secondi e che la velocità di scaricamento del file sia di 32Kbps.

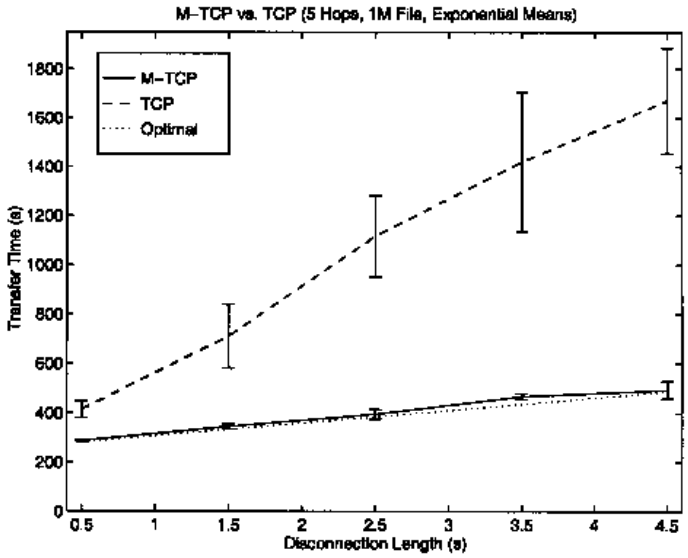

Fig. 4.4 Prestazioni del M-TCP e del TCP a confronto [BS97].

Una delle critiche più ovvie che potrebbe essere rivolta ai protocolli che applicano la tecnica della connessione divisa è quella di causare un aumento delle elaborazioni necessarie, e quindi dei ritardi. Brown e Singh hanno quindi realizzato una serie di esperimenti per tentare di

dimostrare che il tempo di elaborazione nel gateway non crea "colli di bottiglia". I calcoli extra per ogni finestra richiesti dal M-TCP lo portano a realizzare prestazioni inferiori rispetto al TCP tradizionale, in condizioni di piccole finestre. Si tratta di un effetto cumulativo che può però essere ridotto aumentando le dimensioni dei buffer(vedi Fig. 4.5); pertanto, Brown e Singh hanno impostato i buffer a 16KB. D'altro canto, più aumentano le dimensioni dei buffer alla base station necessarie per ogni connessione e più risulta dispendiosa l'occupazione delle risorse. Ovviamente, l'elaborazione extra fa perdere un tempo percentualmente inferiore nelle connessioni su percorsi lunghi rispetto a quello su una connessione di lunghezza breve. M-TCP ottiene quindi maggiori guadagni prestazionali rispetto a TCP se applicato su reti ad ampia scala.

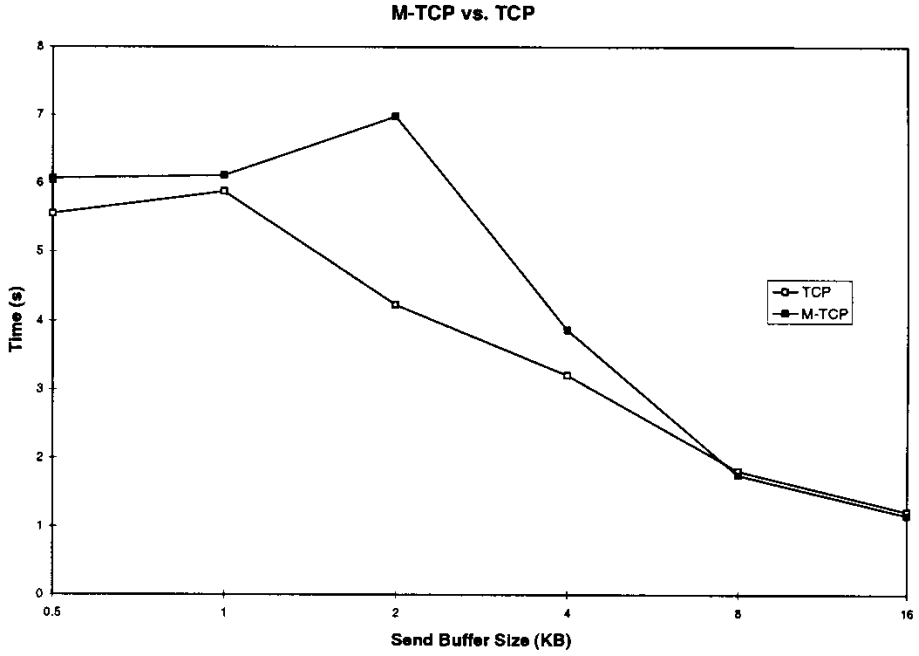

Fig. 4.5 Tempi di trasmissione al variare della dimensione dei buffer [BS97].

Rimane da chiedersi cosa accadrebbe qualora il BER risultasse particolarmente alto, provocando così molte perdite dovute ad errori

piuttosto che a disconnessioni. In questo influisce la scelta di non inserire il meccanismo sack nel protocollo M-TCP, nonostante la sua utilità riconosciuta in situazioni di BER elevato. Brown e Singh sostengono che, grazie all'utilizzo di FEC e di schemi di ritrasmissione appropriati nello strato di collegamento dell'architettura, il numero di errori può essere ridotto significativamente. Occorre però ricordare che quando non necessario, il FEC causa uno spreco di bandwidth, risorsa limitata in ambienti wireless.

4.4 Snoop Protocol (Balakrishnan et al., 1995)

Lo Snoop Protocol nasce nel 1995 con lo scopo di migliorare le prestazioni di collegamenti aventi un tratto wireless preservando però la semantica end-to-end del TCP. L'idea principale consiste nel memorizzare nella base station tutti i pacchetti in transito, così da consentire delle ritrasmissioni locali sul tratto wireless. I problemi causati da un collegamento in parte fisso ed in parte mobile, sono riconducibili a quest'ultimo tratto e di conseguenza, una soluzione che agisca direttamente sul tratto wireless permette risultati più efficienti. L'utilizzo dei dupack per riconoscere perdite ed effettuare ritrasmissioni locali, consente di proteggere il mittente dalle tipiche situazioni transienti di collegamento debole o inesistente, che periodicamente colpiscono il tratto wireless. Lo Snoop Protocol modifica il software di rete della base station e del dispositivo mobile e, allo stesso tempo, non costituisce un elemento essenziale per la connessione end-to-end: la mancanza o l'interruzione del funzionamento del protocollo non pregiudica la connessione. Le modifiche che sono richieste alla base station servono a gestire la memorizzazione dei pacchetti transitati dalla base station al dispositivo

mobile e per i quali la base station non ha ancora ricevuto il relativo ack. E' importante notare che il controllo di ogni pacchetto che transita nella base station, in entrambe le direzioni, pone lo Snoop Protocol ad un livello intermedio tra il livello di rete e quello di trasporto.

Cercando di comprendere più approfonditamente il meccanismo dello Snoop Protocol, partiamo dal caso in cui dei dati vengano inviati da un host fisso ad uno mobile. La base station, attraverso cui passeranno i dati, necessita di alcune modifiche del codice; è infatti necessario aggiungere un modulo chiamato proprio *snoop*. Questo modulo si occupa di mantenere traccia di ogni pacchetto in transito in modo da individuare e recuperare attraverso la ritrasmissione, quelli persi. Per realizzare questo obiettivo, viene mantenuta alla base station una cache di pacchetti inviati dall'host fisso dei quali però non si è ancora ricevuto riscontro di ricezione dall'host mobile. Il modulo snoop effettua inoltre un monitoraggio di tutti gli ack provenienti dal dispositivo mobile e, quando l'arrivo di un dupack o lo scadere di un timeout locale indica che un pacchetto presente nella cache è stato perso, provvede alla sua ritrasmissione senza inoltrare eventuali dupack alla sorgente fissa. In questo modo si riesce a nascondere all'host fisso molte perdite legate alla parte wireless del collegamento, prevenendo quindi l'invocazione di meccanismi inutili e deleteri di controllo della congestione. Nel caso in cui il pacchetto da ritrasmettere non sia presente nella cache della base station, allora si può desumere che la perdita sia dovuta a congestione sulla rete wired. In questo caso i dupack non vengono soppressi, bensì inoltrati verso la sorgente fissa, in modo che possano essere attivati i meccanismi di fast retransmit o, nel caso di scadenza del timeout, di slow start.

Il modulo snoop utilizza due procedure collegate denominate *snoop_data( )* e *snoop_ack( )*. La prima gestisce e memorizza i

pacchetti indirizzati al host mobile, la seconda gestisce invece gli ack provenienti dal dispositivo mobile e si occupa di ritrasmettere localmente i pacchetti persi sul tratto wireless del collegamento. Riportiamo in Fig. 4.6 e in Fig. 4.7 i flow chart riassuntivi del funzionamento delle due procedure.

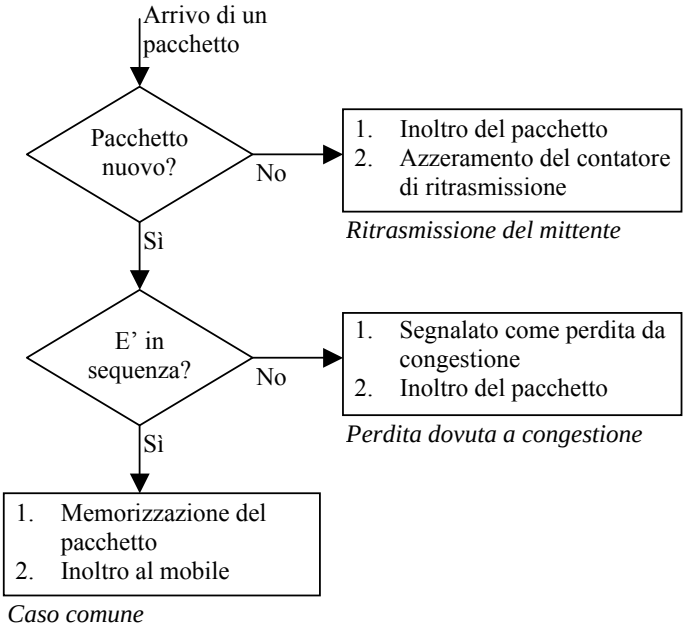

Fig. 4.6 Flow chart della procedura *snoop_data( )* [BSAK95].

Il modulo snoop tiene traccia dell'ultimo numero di sequenza della connessione transitato: sappiamo infatti che il TCP identifica univocamente ogni suo pacchetto tramite il numero di sequenza del suo primo byte di dati e la sua dimensione. Alla base station potrebbero arrivare diverse tipologie di pacchetti e per ognuna di queste vi è un modalità di gestione differente. Il caso comune è quello in cui arrivi un nuovo pacchetto, con il numero di sequenza atteso. In questo caso il pacchetto viene memorizzato nella cache della base station ed inoltrato verso il dispositivo mobile. Inoltre viene scritto un timestamp in un pacchetto per ogni finestra trasmessa, in modo da poter stimare il RTT

sulla parte wireless del collegamento. La cache è implementata come un buffer circolare di pacchetti e la ritrasmissione di un pacchetto presente nella cache avviene utilizzando dei timeout. In particolare si usano due tipi di contatori: *round-trip timer* e *persist timer*. Il primo si basa sulla stima dello *smoothed round-trip time* (srtt) la cui formula è data da:

$$srtt = (1-\alpha)*srtt\_vecchio + \alpha*srtt\_corrente$$

con $\alpha = 0,25$. La ritrasmissione avviene quando la base station non riceve l'ack atteso in un periodo di tempo doppio rispetto al srtt. Un limite al numero degli interrupt dovuti allo scadere del tempo è fornito da una soglia minima per i timer di 40ms. Questa impostazione dei timer viene realizzata solo dopo la prima ritrasmissione di un pacchetto dalla cache, alcune inutili ritrasmissioni di pacchetti già arrivati al dispositivo mobile possono quindi avvenire. Il persist timer provoca le ritrasmissioni di pacchetti presenti nella cache, di cui ancora non è stato ricevuto alcun ack, nel caso in cui non vi siano state attività da parte del mittente o del ricevente per 200ms. In questo caso il numero di dupack previsti viene riportato a zero ed il prossimo ack atteso viene impostato con un valore maggiore di uno rispetto all'ultimo ack transitato fino a quel momento. Questi timer e le conseguenti ritrasmissioni assumono un ruolo critico all'interno del meccanismo dello Snoop Protocol poiché incrementano il numero di tentativi di trasmissione e di conseguenza la possibilità che si tratti di ritrasmissioni ridondanti; allo stesso tempo però aumentano anche le possibilità che i dati raggiungano il dispositivo mobile prima possibile.

Un diverso tipo di pacchetti che può raggiungere la base station è rappresentato dai pacchetti fuori sequenza che sono stati precedentemente memorizzati nella cache gestita dallo Snoop Protocol;

ciò avviene quando la perdita di pacchetti causa lo scadere del timeout o l'invocazione del meccanismo di fast retransmit nel mittente. Le azioni seguenti dipendono dal numero di sequenza di tale pacchetto: se è maggiore di quello a cui fa riferimento l'ultimo ack ricevuto, allora viene inoltrato verso il dispositivo mobile poiché si ipotizza che tale pacchetto non avesse in precedenza raggiunto la destinazione. Se viceversa il numero di sequenza del pacchetto in questione risulta inferiore a quello riportato dall'ultimo ack ricevuto dalla base station, si può ritenere che l'ack precedente sia andato perso. Non è dunque necessario inoltrare il pacchetto verso il dispositivo mobile, bensì occorre ritrasmettere verso il host fisso l'ultimo ack transitato.

Un terzo tipo di pacchetti che possono essere inviati dal host fisso alla base station è costituito dai pacchetti fuori sequenza che non sono però stati precedentemente memorizzati nella cache. Questa situazione può essere dovuta a due fattori: la perdita da parte della rete wired di un pacchetto oppure il rilascio non in sequenza del pacchetto. La prima causa risulta essere più probabile nel caso in cui il numero di sequenza del pacchetto arrivato fuori ordine è maggiore rispetto all'ultimo di più di uno o due pacchetti. In questo caso, il pacchetto viene marcato come ritrasmesso dal mittente e propagato verso il dispositivo mobile. La procedura *snoop_ack( )* utilizzerà poi questa informazione per gestire gli ack di questo pacchetto da parte del host mobile.

La procedura *snoop_ack( )* controlla e gestisce gli ack di ritorno dal dispositivo mobile agendo in conseguenza del tipo e del numero di ack che riceve. Tali ack possono distinguersi in tre diverse tipologie. Il caso più comune è quello che indica il ricevimento corretto di un nuovo pacchetto da parte del host mobile. Quando ciò avviene, il pacchetto in questione viene cancellato dalla memoria della base station e si aggiorna il valore del RTT stimato relativo alla parte wireless del

collegamento. L'aggiornamento del RTT stimato avviene solo per un pacchetto in ogni finestra di invio e nel caso in cui non vi sia stata ritrasmissione del pacchetto; viceversa non si potrebbe sapere se l'ack ricevuto si riferisce al primo invio o alla sua ritrasmissione, e quindi la stima del RTT potrebbe risultare completamente errata. Un ulteriore operazione, peraltro ovvia, consiste nel propagare l'ack ricevuto verso il mittente fisso.

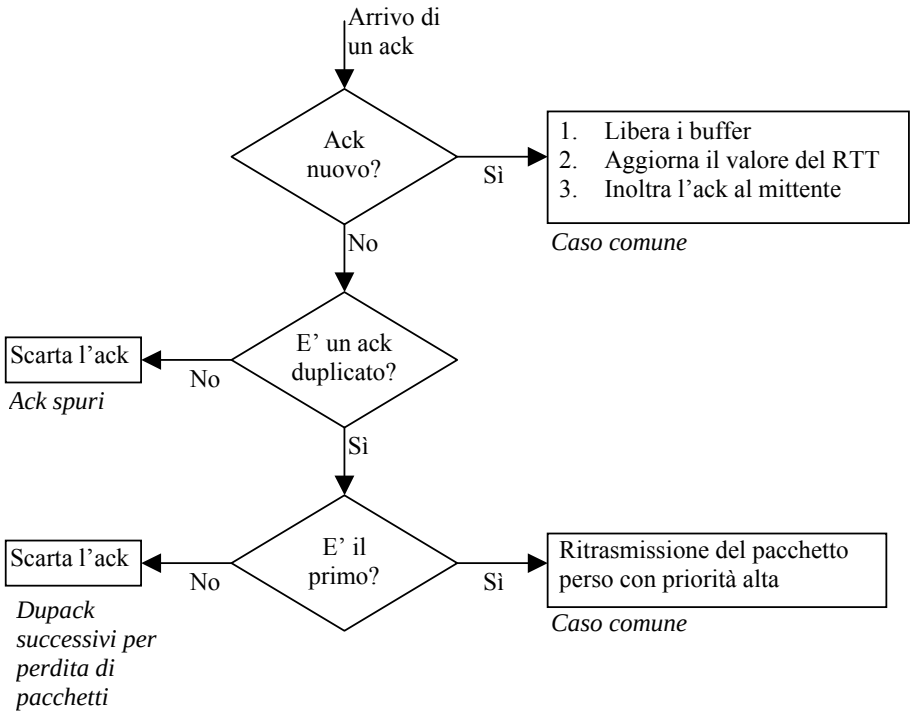

Fig. 4.7 Flow chart della procedura *snoop_ack()* [BSAK95].

Un altra tipologia di ack è rappresentata da quelli che comunicano l'avvenuto ricevimento di un pacchetto avente un numero di sequenza inferiore rispetto a quanto già riconosciuto. Ovvero un ack che conferma l'avvenuta ricezione corretta di una quantità di dati inferiore a quanto già riportato da altri ack. Si tratta di una situazione piuttosto rara e *snoop_ack( )* semplicemente li scarta.

Un terzo tipo di ack ricevibili dalla base station è costituito dai dupack, ovvero duplicati identici ad ack già ricevuti. La ricezione di questa tipologia di ack indica che il pacchetto successivo a quello indicato non è correttamente giunto al dispositivo mobile, ma sono comunque pervenuti alcuni pacchetti successivi. Nel caso in cui i dupack riguardino pacchetti non più presenti in memoria, oppure pacchetti che sono stati marcati come ritrasmessi dal mittente, è necessario inoltrare il dupack verso il host fisso. Il mittente infatti dovrà provvedere a rispedire il pacchetto mancante invocando i suoi meccanismi di controllo della congestione e aggiornando il suo stato tenendo conto del numero dei dupack per il pacchetto ritrasmesso. Se i dupack riguardano pacchetti presenti nella cache, occorre distinguere tra due situazioni. Qualora il dupack non fosse atteso, si provvede immediatamente a recuperare la perdita trasmettendo nuovamente il pacchetto con una priorità elevata; si conta inoltre il numero di pacchetti inviati tra la perdita del pacchetto e la sua ritrasmissione, in modo da stimare il numero massimo di dupack che potrebbero arrivare. I dupack successivi non saranno quindi una sorpresa e costituiscono il secondo tipo di situazione. Avendo già ritrasmesso il pacchetto perso, il dupack viene semplicemente scartato. Il pacchetto ritrasmesso raggiungerà il dispositivo mobile prima che la maggior parte dei pacchetti successivi producendo un ack con un numero di sequenza superiore a quelli ricevuti fino a quel momento.

Per creare delle priorità diverse, vengono mantenute due code nello strato di collegamento che consentono di gestire separatamente i pacchetti a priorità normale da quelli a priorità elevata. Questa tecnica permette di migliorare le prestazioni soprattutto con una frequenza di errori media o bassa, poiché i pacchetti ritrasmessi raggiungono la destinazione prima. Non vi sono invece grandi vantaggi a mantenere

due code nel caso in cui la frequenza di errore nei pacchetti inviati sia alta, poiché un'alta percentuale dei pacchetti dovrà essere ritrasmessa.

Passiamo ora ad analizzare come avviene il trasferimento di dati in uscita dal dispositivo mobile verso la rete fissa, attraverso una base station. Difficilmente si possono ottenere buoni risultati con la sola modifica della base station per memorizzare i pacchetti ed effettuare le ritrasmissioni necessarie. Questo perché molte delle perdite potrebbero verificarsi lungo il tragitto dal dispositivo mobile alla base station e non c'è modo per il mittente di sapere se la perdita di un pacchetto è avvenuta sulla parte wireless del collegamento, oppure sulla rete fissa in seguito a congestione. Inoltre, poiché la ritrasmissione di un pacchetto da parte del mittente avverrebbe dopo la scadenza di un timeout basato sul calcolo del RTT totale sul percorso tra i due host, il caso in cui un pacchetto sia perso nel breve tragitto wireless non verrebbe trattato tempestivamente. Pertanto, la soluzione proposta dallo Snoop Protocol prevede anche alcune modifiche al dispositivo mobile allo scopo di renderlo capace di tenere traccia dei pacchetti persi e di generare dei *negative acknowledgment* (nack). Questo tipo di pacchetti di conferma serve ad informare con precisione il mittente su quali pacchetti non sono correttamente giunti a destinazione; la loro implementazione realizzata si basa sull'utilizzo di sack. [JB88]. Questa soluzione è particolarmente utile in situazioni tipiche di collegamenti wireless con segnale debole, nelle quali sono frequenti perdite multiple all'interno di una singola finestra di invio. La trasmissione di questi nack avviene in seguito allo scadere di un certo intervallo di tempo, oppure quando un certo numero di pacchetti, di una singola finestra di invio, raggiunga la base station.

Il trasferimento dei dati relativi ad una connessione, da una base station ad un'altra, detto anche handoff, costituisce un momento molto

importante in collegamenti con parti wireless. Quando il dispositivo mobile si sposta da un'area coperta dal segnale di una base station, ad una adiacente, controllata da un'altra base station, la connessione deve essere mantenuta attiva reindirizzando il traffico dei dati verso la nuova stazione. Questa operazione va effettuata cercando di ridurre al minimo i tempi di mancanza di segnale, in modo da perdere minor quantità di dati possibile. Gli handoff vengono solitamente iniziati dal dispositivo mobile quando identifica una base station con un segnale più forte di quella utilizzata al momento, ma possono anche essere anticipati dalle base station. Quando il processo ha inizio, la nuova base station inizia a ricevere anch'essa i pacchetti destinati al dispositivo mobile, allo scopo di crearsi la memoria cache prevista dal meccanismo dello Snoop Protocol per gestire la connessione. Durante l'handoff, le due stazioni coinvolte non possono però monitorare gli ack provenienti dal host mobile. Lo schema proposto, realizzando una creazione preventiva della cache sulla nuova base station, permette una durata di handoff abbastanza breve, misurata dagli ideatori dello Snoop Protocol in un intervallo compreso tra i 10 e i 25ms [BSAK95].

Procedendo ora ad un'analisi critica dello Snoop Protocol, appare doveroso sottolineare come le sue buone prestazioni di throughput, anche in ambienti con BER elevati e soprattutto nella versione implementata con l'integrazione dei pacchetti di tipo nack, lo abbiamo reso un punto di riferimento comparativo per le successive proposte di modifica o creazione di un protocollo di trasporto che potesse operare efficientemente su collegamenti wireless [BS97, GMPG00, VMPM99]. I risultati delle sperimentazioni effettuate dai ricercatori di Berkeley che hanno ideato lo Snoop Protocol sono pubblicati in [BSAK95] e in [BPSK97]. I problemi relativi al tratto wireless del collegamento vengono affrontati localmente dallo snoop agent posto nella base station, nascondendo così gran parte delle carenze tipiche dei collegamenti wireless alla rete fissa. La Fig. 4.8, riportata da

[BSAK95], mostra il buon comportamento dello Snoop Protocol rispetto a TCP in presenza di errori. Il grafico è costruito con un'ascisse in scala $\log_2$ con un BER di 1 errore ogni 64kbit all'estremo di sinistra e nessun errore all'estremo di destra. Le barre verticali rappresentano la deviazione standard del throughput del receiver. Si può notare come, in presenza di BER elevati, Snoop Protocol offra prestazioni anche di venti volte superiori a quelle di TCP.

Il mantenimento della semantica end-to-end e la caratteristica di "soft state", per cui un blocco del funzionamento delle soluzioni implementate non interrompe la connessione, permettono allo Snoop Protocol di essere utilizzabile in quasi ogni contesto. Tra i suoi pregi è importante ricordare anche il mantenimento della semantica end-to-end in quanto ciò che avviene è un semplice inoltro degli ack generati dal dispositivo mobile.

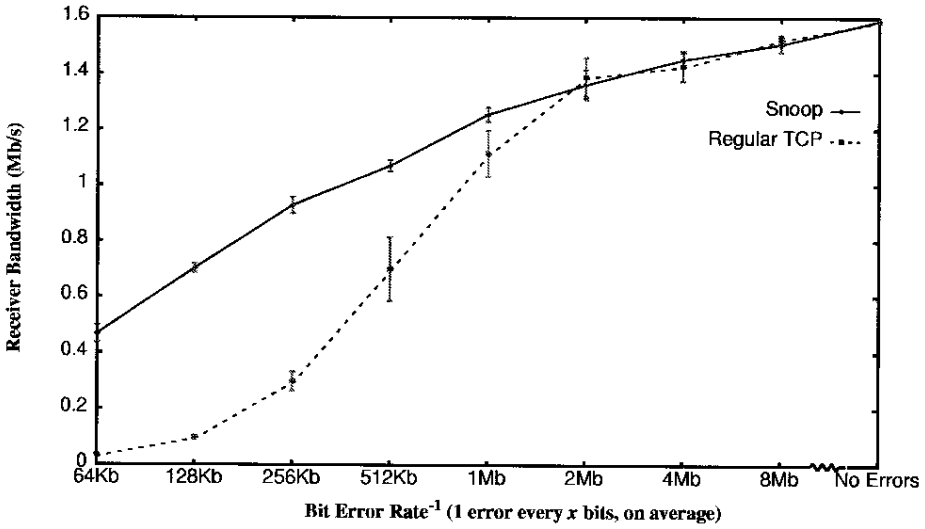

Fig. 4.8 Throughput ricevuto dall'host mobile al variare del BER [BSAK95].

La pecca principale dello Snoop Protocol risiede proprio nel fatto che necessita di monitorare tutti i pacchetti in transito per la base station. Non è quindi possibile il suo utilizzo con trasmissioni sicure che prevedano la cifratura del header TCP. Snoop Protocol non gode inoltre di buone prestazioni in presenza di lunghe disconnessioni: di fronte a periodi di disconnessione superiori al timeout, il mittente invocherà comunque il meccanismo di slow start. Anche handoff frequenti causano problemi, infatti quando un dispositivo mobile cambia cella, la nuova base station deve copiare la cache dalla precedente. Nel frattempo, l'host mobile godrà di un throughput scarso poiché non potranno ancora essere utilizzate ritrasmissioni o funzioni di filtraggio da parte del modulo di snoop. Qualora le celle siano di piccole dimensioni, il problema assume connotazioni significative causando un serio degradamento delle prestazioni. Una possibile soluzione potrebbe essere quella di adottare un approccio "soft state" e far iniziare la preparazione della cache nella nuova base station prima che la vecchia smetta di ricevere pacchetti dal mittente. Occorrerebbe però poter sapere sempre in quale cella si sta trasferendo il dispositivo mobile, mentre sta per lasciare la vecchia. Ciò non è sempre possibile, potrebbero ad esempio esservi più celle adiacenti e in questo caso si dovrebbe iniziare ad impostare in anticipo le nuove cache sulle base station di tutte le celle che costituiscono la possibile destinazione dell'host mobile; chiaramente l'eccessivo dispendio di risorse computative e di memoria rendono questa possibilità non praticabile.

4.5 Delayed Dupacks (Vaidya et al., 1999)

Risalente al 1999 ed ispirato allo Snoop Protocol, il protocollo Delayed Duplicate Acknowledgement, detto anche Delayed Dupacks, presenta

un approccio in cui il protocollo di trasporto sia di tipo *TCP-unaware*, ovvero non consapevole della presenza di una parte wireless nel collegamento e che quindi non richieda azioni specifiche da parte dei nodi intermedi. In questa maniera, la base station non ha bisogno di controllare gli header dei pacchetti TCP (come ad es. Snoop Protocol) e può essere utilizzato anche in presenza di traffico criptato.

Lo schema del Delayed Duplicate Acknowledgments propone innanzitutto di realizzare delle ritrasmissioni nello strato di collegamento al fine di recuperare i pacchetti persi sulla parte wireless dovuti ad errori trasmissivi. Inoltre, mentre lo schema dello Snoop Protocol prevede di utilizzare gli ack duplicati del TCP per impostare le ritrasmissioni dello strato di collegamento, il Delayed Duplicate Acknowledgments utilizza gli ack dello strato di collegamento. Altra peculiarità dello schema proposto è quella di ridurre le interferenze tra le ritrasmissioni a livello TCP e quelle operate dal livello di collegamento. I pacchetti duplicati dal terzo in poi, vengono ritardati per un intervallo di tempo stabilito. In particolare, all'arrivo di pacchetti fuori ordine, il TCP ricevente risponde immediatamente solo per i primi due, inviando dei dupack al mittente. L'invio di ulteriori dupack viene ritardato per un tempo *d* allo scopo di permettere allo strato di collegamento di effettuare le sue ritrasmissioni e recuperare così dalle perdite senza eseguire fast retransmit. In questa maniera si riducono le ritrasmissioni ridondanti ed i dimezzamenti inutili della finestra di invio. Se allo scadere dell'intervallo di tempo prestabilito non si è riusciti ancora a recuperare i pacchetti mancanti, i dupack ritardati vengono inviati tutti insieme.

Consideriamo due esempi di perdita di pacchetto: perdita dovuta ad errori di trasmissione sulla parte wireless e perdita dovuta a congestione sulla parte wired. Nel primo caso, qualora l'intervallo di tempo *d* sul

dispositivo mobile venga impostato grande abbastanza da consentire la ritrasmissione dei pacchetti perduti nello strato di collegamento, da parte della base station, e gli ack di questi siano ricevuti prima dello scadere di *d*, i pacchetti duplicati non sono inoltrati al mittente. Il mittente infatti, non ricevendo più di due pacchetti duplicati, non inizia la procedura di fast retransmit. Se però l'intervallo *d* viene impostato di dimensioni troppo grandi, potrebbe rimanere tempo sufficiente da far scadere il timeout nel TCP del mittente, causando così l'invocazione del meccanismo slow start ed abbattendo le prestazioni di throughput. Appare evidente l'importanza di un opportuno dimensionamento del ritardo di spedizione dei dupack successivi al secondo. Nel secondo esempio invece, la perdita avviene nella parte wired del collegamento ed è dovuta a congestione del traffico in rete, pertanto il meccanismo di ritrasmissione sulla parte wireless non è utile. Il dispositivo mobile anzi, ritardando l'invio del terzo e dei successivi dupack, ritarda la notifica di congestione della rete fissa e conseguentemente delle misure atte a limitarla. Il meccanismo di Fast retransmit non viene invocato immediatamente con il risultato che le prestazioni rispetto al TCP standard calano. Allo scadere dell'intervallo *d*, il TCP ricevente invierà simultaneamente tutti i dupack trattenuti.

In Fig. 4.9 riportiamo un illustrazione del Delayed Duplicate Acknowledgements. Come già detto, questo schema utilizza gli ack dello strato di collegamento per impostare le ritrasmissioni ma in figura sono riportati solo i pacchetti dello strato di trasporto. Dall'esempio notiamo che il pacchetto *2000:2999* viene perso e che lo strato di collegamento ne effettua la ritrasmissione. Nel frattempo però, giungono al dispositivo mobile i pacchetti successivi, che risultano pertanto essere fuori ordine. Il dispositivo mobile effettua immediatamente l'invio di dupack solo per i primi due, mentre l'invio del terzo dupack e dei successivi viene ritardato dando il tempo allo strato di collegamento di effettuare la ritrasmissione del pacchetto

mancante ed evitando l'invocazione del meccanismo di fast retransmit. In figura si ipotizza il caso in cui il pacchetto ritrasmesso riesca a raggiungere correttamente il ricevente entro l'intervallo di tempo *d*. Al suo arrivo, quindi, il dispositivo mobile scarterà tutti i dupack ritardati ed invierà l'ack dell'ultimo pacchetto in sequenza giunto.

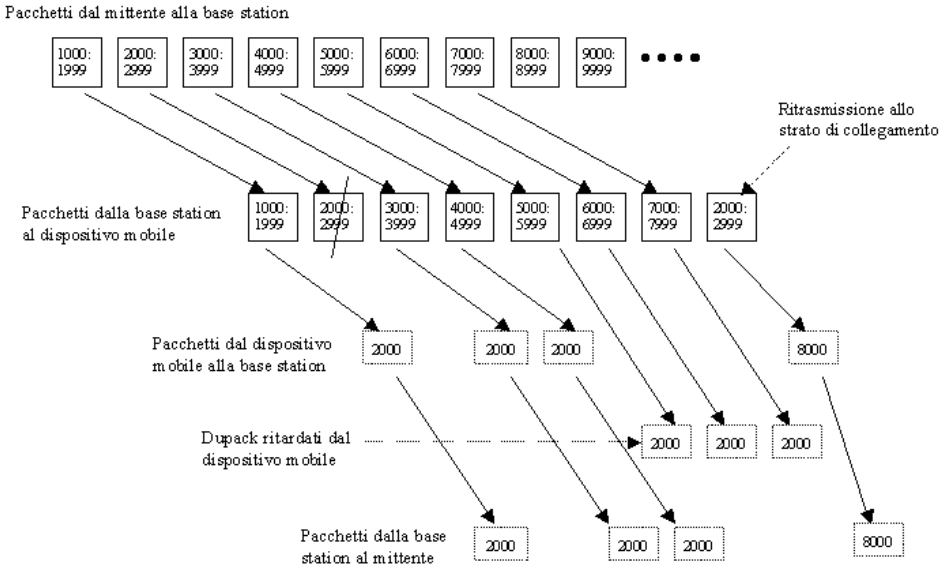

Fig. 4.9 Esempio dello schema Delayed Duplicate Acknowledgments [VMPM99].

L'analisi dello schema Delayed Dupacks ci suggerisce che i benefici maggiori si ottengono nel caso in cui le perdite siano dovute ad errori trasmissivi sulla parte wireless, mentre nel caso in cui le perdite di pacchetti siano dovute a congestione sulla rete fissa, si riscontra un calo delle prestazioni. Il risultato finale dipende dunque dalle occorrenze dei due casi ed è migliore se il RTT nella parte wireless rimane relativamente basso rispetto a quello totale tra il mittente ed il ricevente. La Fig. 4.10 ci mostra il throughput relativo ad una connessione con diverse percentuali di perdite dovute a congestione nel caso in cui la latenza sul collegamento wireless sia di appena 1ms e non vi siano errori trasmissivi. Il confronto avviene tra il TCP base (senza

meccanismi di recupero al livello di collegamento), lo Snoop Protocol e quattro protocolli Delayed Dupacks con differenti ritardi. I ritardi utilizzati per il confronto variano da 0 a 0,2 secondi. La mancanza di errori trasmissivi rende inutile il meccanismo di invio ritardato dei dupack, il quale si limita a ritardare il recupero delle perdite da congestione.

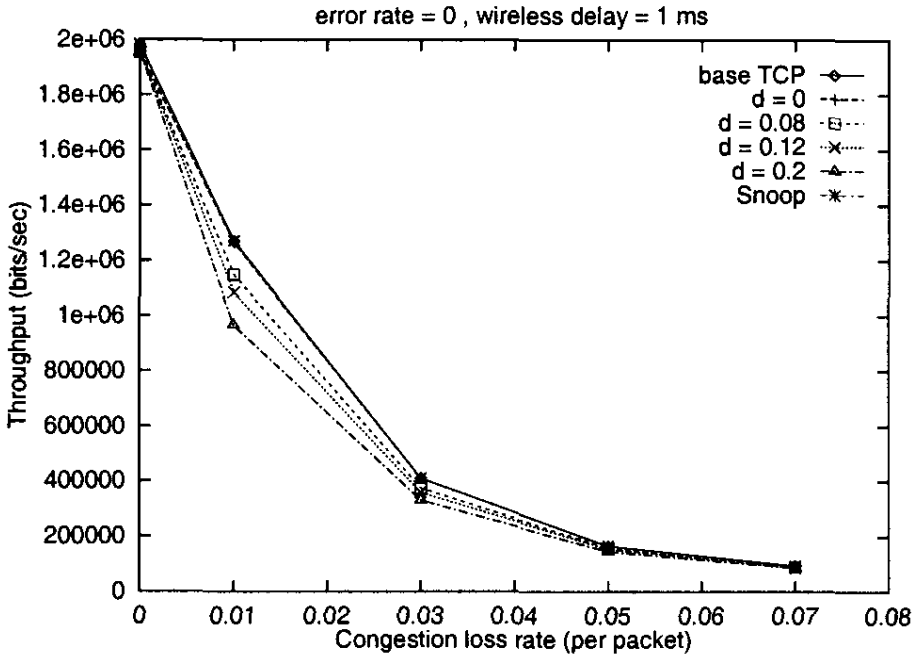

Fig. 4.10 Throughput con latenza wireless di 1ms e BER=0 [VMPM99].

Come era ovvio aspettarsi, all'aumentare delle perdite da congestione diminuisce il throughput. Inoltre, in assenza di errori trasmissivi, TCP e Snoop Protocol hanno delle prestazioni simili mentre il protocollo Delayed Dupacks ottiene risultati inferiori e che peggiorano all'aumentare del ritardo *d*.

La Fig. 4.11 ci mostra il throughput ottenuto dai protocolli confrontati, al variare della percentuale di errori trasmissivi e tenendo costante una

frequenza di 0,01 perdite per pacchetto dovute a congestione. In questo caso il protocollo che sembra comportarsi meglio è il Delayed Dupacks. Analizzando i valori espressi, notiamo però che il risultato migliore è quello prodotto con *d=0*, ovvero utilizzando i meccanismi di ritrasmissione al livello di collegamento, ma senza introdurre alcun ritardo nell'invio di dupack. In situazioni di latenza wireless ridotta, Delayed Dupack non sembra quindi in grado ottenere prestazioni superiori a quelle di un TCP tradizionale con l'aggiunta di meccanismi di ritrasmissione al livello di collegamento.

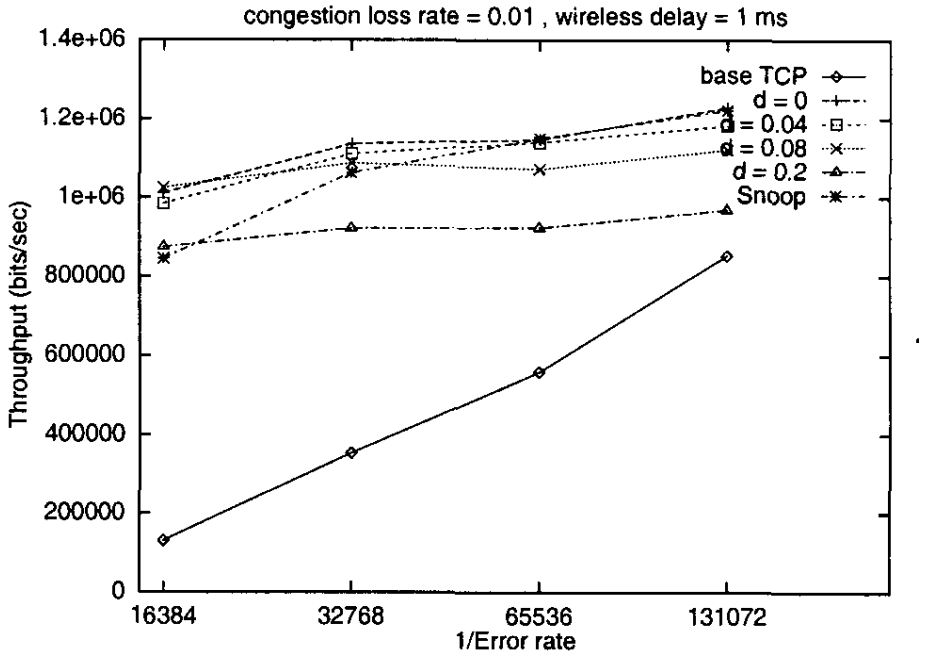

Fig. 4.11 Throughput con 1ms di latenza wireless e diversi BER [VMPM99].

All'aumentare dei tempi trasmissivi sulla parte wireless del collegamento, si riescono ad ottenere prestazioni migliori da parte dello schema di Delayed Dupacks. La Fig. 4.12 illustra il throughput ottenuto dai protocolli confrontati al variare del BER e in un contesto con latenza sul tratto wireless di 20ms.

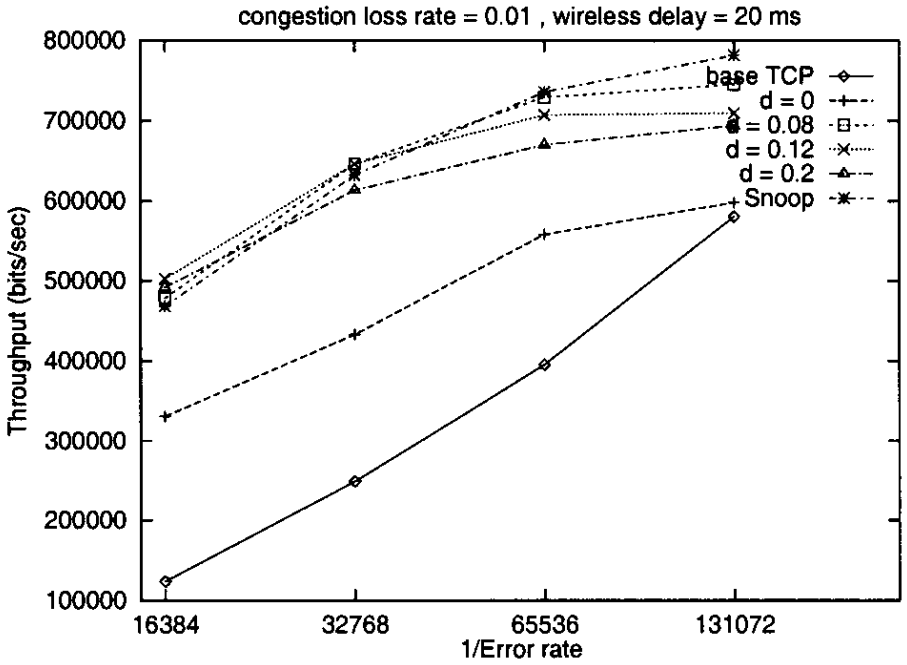

Fig. 4.12 Throughput con 20ms di latenza wireless e diversi BER [VMPM99].

In questo caso Delayed Dupacks ottiene delle buone prestazioni, migliori di quelle di TCP con meccanismi di ritrasmissione al livello di collegamento (caso *d=0*). Utilizzando il ritardo più appropriato, i risultati sono confrontabili con quelli dello Snoop Protocol, con la differenza che il primo richiede modifiche solo nell'host ricevente. La chiave per comprendere questi risultati risiede nel fatto che con tempi trasmissivi sul tratto wireless più elevati, si riesce ad ottenere su tale tratto un prodotto latenza-bandwidth maggiore. Infatti, condizione essenziale perché lo schema proposto funzioni è che il prodotto latenza-bandwidth sia maggiore di 4 pacchetti, viceversa non sarebbe possibile far scattare il meccanismo di fast retransmit su cui Delayed Dupacks interviene. Inoltre una latenza di 20ms corrisponde ad un RTT di circa 40ms. Conseguentemente se il ritardo *d* è inferiore a questo valore, il dupack ritardato verrà rilasciato prima che meccanismi di ritrasmissione dello strato di collegamento possano recuperare il pacchetto, invocando quindi meccanismi per il controllo della congestione.

Concludendo lo schema Delayed Dupacks produce dei benefici solo se il prodotto latenza-bandwidth è maggiore di quattro, viceversa risulta più efficace limitarsi ad aggiungere al TCP dei meccanismi di ritrasmissione al livello di collegamento. Nei casi in cui Delayed Dupacks è utile, le sue prestazioni sono comparabili con quelle dello Snoop Protocol, dal quale si differenzia però per non richiedere alcuna conoscenza del contenuto dell'header al livello di trasporto da parte della base station. Grazie a questo approccio, lo schema del Delayed Duplicate Acknowledgements risulta applicabile anche in presenza di trasmissioni criptate a livello TCP. Allo stesso tempo però, la scelta dell'intervallo di ritardo *d* assume un'importanza cruciale per valutare l'efficienza di questo schema.

4.6 TCP-Aware (Biaz e Vaidya, 1999)

Una delle principali cause delle cattive prestazioni dei TCP tradizionali in un ambiente wireless risiede, come già illustrato, nell'incapacità di questi protocolli di distinguere le perdite di segmenti dovute a congestione da quelle dovute ad errori di altro tipo. Se il protocollo di trasporto potesse riconoscere i motivi che hanno portato alla perdita di un pacchetto, potrebbe anche rispondervi nella maniera più appropriata. Per cercare di dotare TCP Reno di un meccanismo che gli permetta di discriminare le tipologie di perdita, Saad Biaz e Nitin H. Vaidya della Texas A&M University hanno ideato nel 1999 uno schema che tiene conto delle differenze tra i tempi di arrivo dei vari pacchetti.

L'esperienza precedente nel cercare di adattare alcuni noti meccanismi orientati ad evitare congestioni aveva portato ad un risultato negativo. Le funzioni predittive utilizzate in alcune situazioni avevano rivelato un comportamento non migliore di quello ottenibile un predittore casuale [BV98]. Una delle cause che avevano portato a questo parziale fallimento era la necessità, impossibile da soddisfare, di intervalli adeguatamente lunghi di creazione delle code nei router, in modo che le funzioni predittive potessero aver tempo di riconoscere una situazione di congestione prima che si verificasse. Un'altra lacuna risiedeva nel punto di osservazione scelto per le funzioni predittive: il mittente. L'utilizzo degli ack cumulativi da parte del TCP, non permette al mittente di sapere esattamente quale pacchetto è andato perso, mentre il ricevente ha una visione più precisa della situazione.

Lo schema successivamente sviluppato da Biaz e Vaidya, che qui riportiamo, utilizza una tecnica end-to-end di rilevamento delle cause di perdita dei segmenti TCP che risiede nel ricevente. Le assunzioni di partenza, necessarie per un buon funzionamento di tale schema sono:

- Solo l'ultimo passo del collegamento è di tipo wireless.

- Il tratto wireless sia il "collo di bottiglia" dell'intero collegamento, ovvero la bandwidth disponibile nella parte wireless sia molto inferiore rispetto a quella disponibile sulla parte wired.

- Il mittente effettua il trasferimento di dati in blocco.

- Si possono ignorare i tempi di elaborazione alla base station ed al ricevente.

- La frequenza complessiva delle perdite di pacchetti deve essere bassa.

Attraverso la descrizione dei tre esempi di Fig. 4.13, comprendiamo in dettaglio le osservazioni che hanno portato all'ideazione di questo schema.

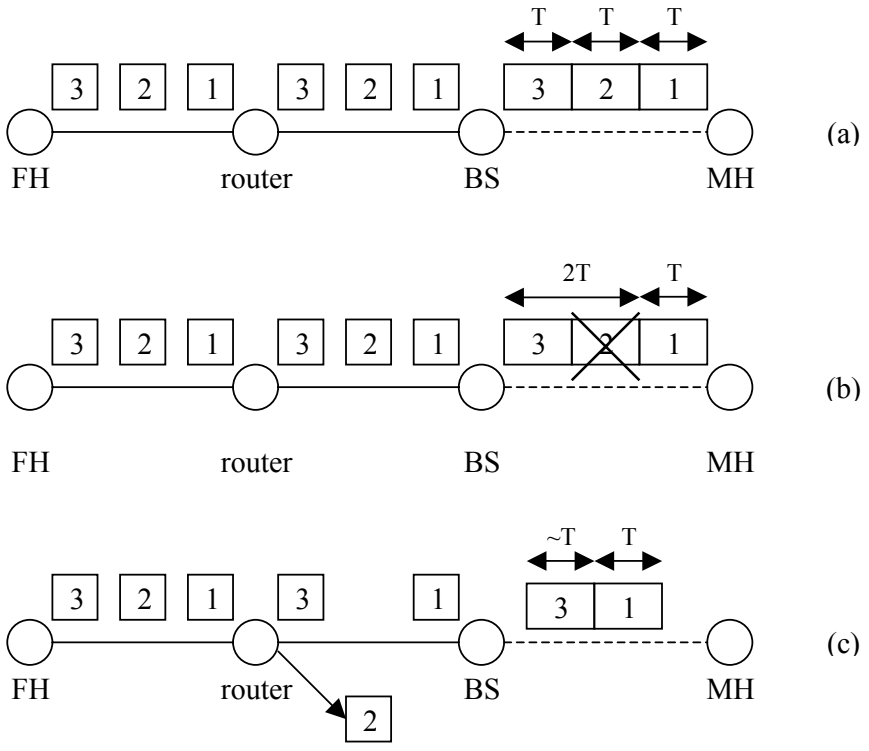

Fig. 4.13 Esempi di tempi di interarrivo [BV99].

Nell'esempio (a) è rappresentata la situazione in cui non si verifica alcuna perdita; la differenza di tempo tra gli arrivi di pacchetti

consecutivi rimane invariata e corrisponde al tempo *T* necessario per trasmettere un pacchetto sulla parte wireless del collegamento. In (b) viene invece descritto il caso in cui venga perso uno pacchetto sulla parte wireless. Il tempo di interarrivo tra i due pacchetti ricevuti correttamente è pari a circa *2T*; infatti anche se il pacchetto 2 è andato perso durante la trasmissione, ha comunque utilizzato il collegamento per un tempo *T*. Nella situazione rappresentata da (c), la perdita del pacchetto 2 si è verificata lungo la rete fissa per problemi di congestione. In questo caso, assumendo che il pacchetto 3 arrivi alla base station prima o subito dopo che questa abbia trasmesso il pacchetto 1 verso il destinatario, il tempo di interarrivo tra i due pacchetti sarà dell'ordine di *T*.

L'algoritmo che segue dalle osservazioni sopra esposte è il seguente (si assume che tutti i pacchetti abbiano la stessa dimensione):

- Indichiamo con T_{min} il tempo di interarrivo minimo osservato dal ricevente durante la connessione.

- Indichiamo con P_o un pacchetto ricevuto fuori ordine di sequenza dal ricevitore e con P_i l'ultimo pacchetto ricevuto in sequenza prima di P_o. Indichiamo inoltre con T_g l'intervallo di tempo tra l'arrivo dei pacchetti P_i e P_o e con n il numero di pacchetti mancanti tra i numeri di sequenza di P_i e di P_o.

- Se $(n+1)T_{min} \leq T_g \leq (n+2)T_{min}$, allora si assume che gli n pacchetti mancanti siano dovuti ad errori trasmissivi in ambiente wireless; viceversa si assume che la perdita dei pacchetti sia dovuta a congestione.

L'algoritmo proposto pone delle condizioni piuttosto restrittive per identificare una perdita dovuta all'ambiente wireless, questo perché è preferibile interpretare erroneamente una perdita sul tratto wireless come dovuta a congestione, piuttosto che incorrere nello sbaglio opposto.

Indichiamo con A_w l'accuratezza nel riconoscere le perdite come dovute ad errori wireless, con b_{w1} e b_{w2} le bandwidth rispettivamente sul canale wired e su quello wireless e con r_c e r_w la frequenza delle perdite dovute rispettivamente a congestione e ad errore wireless. Come mostrato in Fig. 4.14, riportata da [BV99], il valore di A_w dipende dal rapporto b_{w1}/b_{w2}. In tale esperimento sono state implementate quattro connessioni che condividevano lo stesso canale ed è stato possibile distinguere tre casi:

$b_{w1}/b_{w2} > 1$:

L'accuratezza A_w migliora in maniera direttamente proporzionale all'aumentare del rapporto b_{w1}/b_{w2}. In questa situazione infatti, aumentano le probabilità che una perdita sia dovuta alla componente wireless del collegamento.

$b_{w1}/b_{w2} \approx 1$:

In questa situazione i pacchetti in viaggio hanno uguali probabilità di trovarsi accodati in un router qualsiasi o nella base station. Pertanto è la probabilità che la perdita sia dovuta al tratto wireless cala e di conseguenza anche il valore di A_w.

$b_{w1}/b_{w2} < 1$:

In questo caso il bottleneck è rappresentato dalla parte wired del collegamento. Vi saranno pertanto più perdite dovute a congestione sul lato wired che non perdite imputabili al tratto wireless del collegamento e, di conseguenza, l'accuratezza di A_w ne risente.

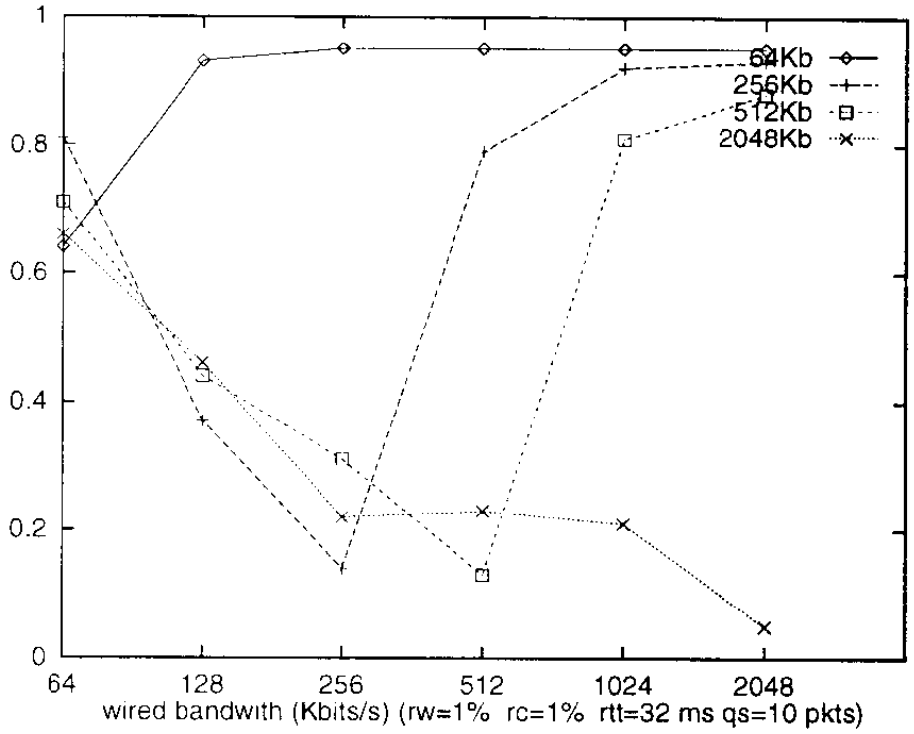

Fig. 4.14 Accuratezza A_w nel riconoscere le perdite wireless [BV99].

Nelle Figg. 4.15 e 4.16 si nota l'incremento prestazionale in termini di magnitudo ottenuto dal TCP Aware in rapporto al TCP Reno. *RA* indica il rapporto tra le prestazioni del TCP Aware e quello Reno, mentre *RI* indica il rapporto tra un TCP ideale e TCP Reno. Per TCP ideale si intende un TCP che sappia sempre ed esattamente le cause che hanno portato alla perdita di un pacchetto, invocando quindi i

meccanismi per il controllo della congestione solo nei casi in cui si sia appunto verificata una perdita dovuta a congestione.

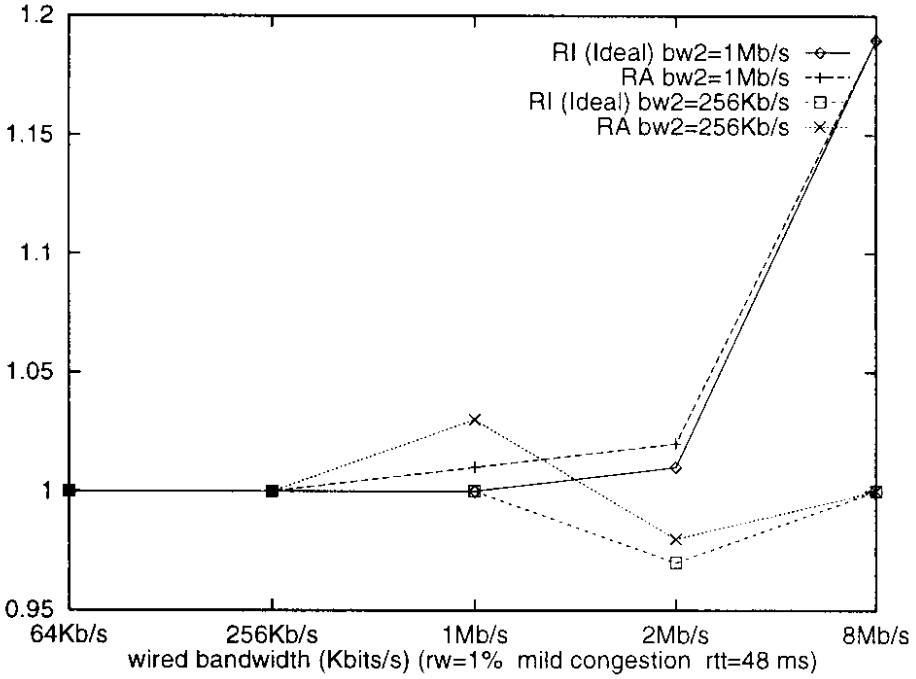

Fig. 4.15 Miglioramento prestazionali con r_w=1% per congestione media [BV99].

Per illustrare i grafici ottenuti, Biaz e Vaidya hanno utilizzato la formula approssimata presente in [MSMO97] per calcolare il throughput:

$$T = \frac{MSS}{RTT}\frac{C}{\sqrt{p}}$$

In tale formula *p* rappresenta la probabilità che si verifichino delle perdite casuali mentre *C* è una costante. L'approssimazione si basa sul fatto che la cwnd viene dimezzata ad ogni perdita. Nel nostro caso le

perdite sono dovute sia a congestione e sia a errori wireless e dunque $p = r_c + r_w$. Conseguentemente il throughput per il TCP normale può essere approssimato come:

$$T_{TCP} = \frac{MSS}{RTT} \frac{C}{\sqrt{r_c + r_w}}$$

Il TCP ideale dimezza la sua cwnd solo quando la perdita è dovuta effettivamente a congestione e dunque il suo throughput approssimato risulta:

$$T_{Ideale} = \frac{MSS}{RTT} \frac{C}{\sqrt{r_c}}$$

Il rapporto tra i due throughput stimati ci permette di comprendere l'importanza del rapporto r_w / r_c nel miglioramento ottenibile da un TCP che sia in grado di discriminare le perdite da congestione da quelle derivanti da problematiche wireless.

$$\frac{T_{Ideale}}{T_{TCP}} = \sqrt{1 + \frac{r_w}{r_c}}$$

Notiamo infatti che il miglioramento ottenuto dal TCP ideale rispetto quello tradizionale aumenta in maniera direttamente proporzionale all'aumentare di r_w rispetto ad r_c.

Le Figg. 4.15 e 4.16 evidenziano inoltre un miglior comportamento di TCP Aware perfino rispetto al TCP ideale. Ciò è probabilmente dovuto al fatto che TCP Aware aumenta, anche se erroneamente, il numero di

occasioni in cui non fa intervenire meccanismi di controllo della congestione, operando così un numero inferiore di restringimenti della finestra di invio. Questo comportamento però equivale ad assumere un atteggiamento aggressivo nei confronti della bandwidth disponibile, il risultato su ampia scala potrebbe essere un peggioramento del livello di congestione con conseguente calo delle prestazioni complessive.

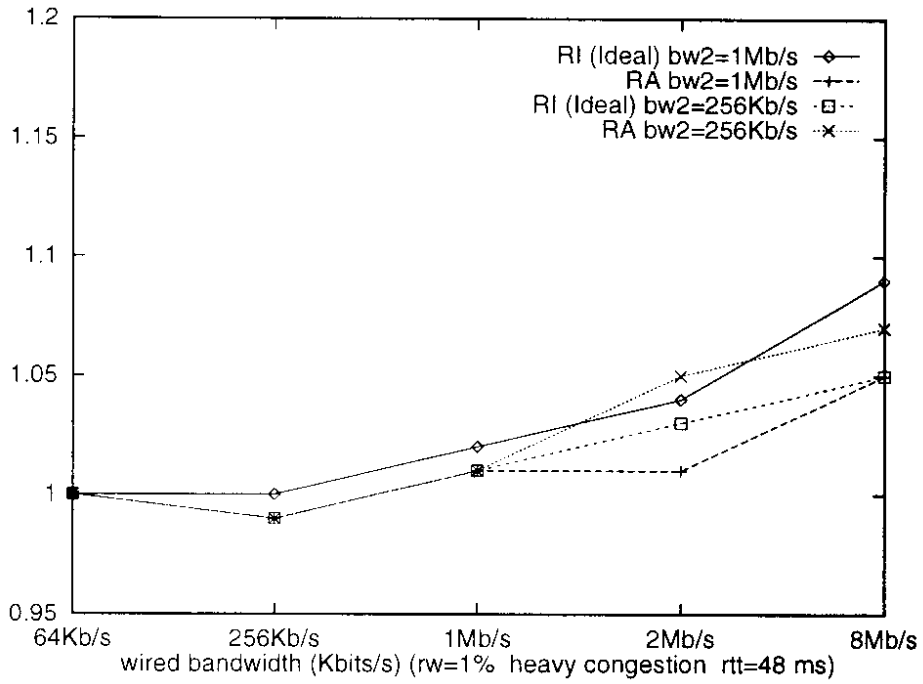

Fig. 4.16 Miglioramento prestazionali con r_w=1% per congestione elevata [BV99].

Lo schema proposto da Biaz e Vaidya permette di operare una distinzione tra i diversi tipi di perdita grazie all'utilizzo dei tempi di interarrivo al ricevente, mantenendo la semantica end-to-end della connessione. Allo stesso tempo però, le assunzioni poste in partenza e le condizioni necessarie a far sì che il meccanismo funzioni, ne limitano il campo di applicazione negandone l'utilizzo in contesti generali di comunicazione tra dispositivi di varia natura.

4.7 Freeze-TCP (Goff et al., 2000)

Abbiamo visto nei capitoli iniziali come i protocolli di trasporto che richiedono ai nodi intermedi di monitorare o addirittura di intervenire direttamente nella gestione del traffico TCP, non sono utilizzabili in presenza di traffico IP criptato. Allo scopo di realizzare un protocollo di trasporto che alleviasse le problematiche dei collegamenti con tratti wireless senza richiedere l'intervento attivo di alcun nodo intermedio, nel 2000 è stato ideato Freeze-TCP. Nato dalla collaborazione di Tom Goff, James Moronski e D. S. Phatak della State University of New York, con Vipul Gupta della Sun Microsystems Inc., questo protocollo di trasporto, oltre a non richiedere modifiche ai nodi intermedi, si distingue anche per non modificare nemmeno il codice del mittente, bensì unicamente quello del ricevente. In questo modo è possibile implementare questo protocollo sui nuovi dispositivi mobili, senza richiedere alcun intervento sulla rete esistente.

Freeze-TCP utilizza attivamente una funzione che è già presente nei TCP tradizionali: la possibilità di "congelare" le trasmissioni tramite la notifica di una rwnd di dimensione zero [Ste94]. Abbiamo già incontrato questa funzione discutendo del protocollo M-TCP (vedi par. 4.3), nel quale però veniva invocata da parte della base station. Tramite questo schema, il mittente entra in uno stato detto *persist mode*, che gli consente di bloccare i contatori dei timeout; inizia inoltre ad inviare dei pacchetti di prova detti *Zero Window Probe* (ZWP), i quali verificano il collegamento fino a che il ricevente non riapre la sua finestra di ricezione. L'intervallo di invio dei ZWP cresce esponenzialmente fino a che raggiunge il tempo limite di 1 minuto, dopodiché rimane costante.

L'invio della notifica di rwnd di dimensione zero, tramite un pacchetto che chiameremo *Zero Window Advertisment* (ZWA), avviene quando il dispositivo mobile si accorge che la forza del segnale trasmissivo si sta indebolendo e che quindi è probabile una imminente disconnessione. Freeze-TCP si basa sull'assunto che l'host mobile riesca ad accorgersi in tempo dell'imminente perdita del segnale in modo da inoltrare almeno un ZWA verso l'host fisso. Allo stesso tempo però, è bene evitare di far entrare il mittente in persist mode troppo presto, altrimenti si aggiungeranno inutili periodi di inattività. L'anticipo ideale con cui far partire il ZWA è uguale a quanto occorrerebbe per inviare esattamente un ZWA dal ricevente al mittente, prima che si verifichi la disconnessione. Un valore accettabile nella scelta dell'anticipo è la lunghezza di un RTT, che corrisponde al tempo che impiega un pacchetto ad arrivare a destinazione ed il corrispondente ack a tornare indietro. Gli autori di Freeze-TCP sostengono che dati sperimentali dimostrano che utilizzare periodi più lunghi o più brevi di un RTT porta ad un peggioramento delle prestazioni medie [GMPG00].

Qualora il periodo di disconnessione sia prolungato, non è detto che la ripresa delle trasmissioni sia immediata con il ristabilirsi della connessione. Poiché l'invio dei ZWP viene effettuato ad intervalli di tempo che aumentano esponenzialmente, se il ripristino della connessione avviene subito dopo la perdita dell'ultimo ZWP inviato dal mittente, allora il dispositivo mobile attende inattivo fino al successivo ZWP. Per ovviare a questo problema, è stato implementato anche lo schema suggerito in [CI95] e che gli autori di Freeze-TCP hanno denominato *Triplicate Reconnection ACKs* (TR-ACKs). Questo schema consiste nell'invio, non appena la connessione viene ristabilita di 3 dupack riferiti all'ultimo segmento di dati ricevuto correttamente prima della disconnessione.

Freeze-TCP si presenta come un protocollo di trasporto end-to-end con l'indubbio vantaggio di non richiedere alcuna modifica alla rete esistente, ma soltanto la sua implementazione sui dispositivi mobili, garantendo così completa interoperabilità; questo gli permette inoltre di essere utilizzabile anche con traffico criptato. Per il suo funzionamento, questo protocollo utilizza meccanismi già presenti nei TCP tradizionali (rwnd, sack...) ampliandone le funzionalità per migliorare le prestazioni in ambiente wireless con frequenti disconnessioni.

Durata disconnes-sione	Tempo medio di trasferimento (in secondi) per 10 tentativi con 10 disconnessioni ciascuno				Guadagno Freeze-TCP +TR-ACKs sul TCP
	Con TR-ACKs		Senza TR-ACKs		
	TCP	Freeze-TCP	TCP	Freeze-TCP	
2,6 ms	18,7	13,0 (+30,4%)	18,6	17,1 (+8,1%)	[+30,1%]
30 ms	17,9	13,2 (+26,2%)	17,8	16,9 (+5,4%)	[+25,8%]
0,1 s	18,4	13,8 (+25,2%)	18,7	17,5 (+6,3%)	[+26,2%]
0,5 s	19,4	17,3 (+10,9%)	21,2	21,8 (-3,2%)	[+18,4%]
1 s	25,7	22,4 (+12,7%)	28,2	28,6 (-1,5%)	[+20,6%]
2 s	40,0	32,5 (+16,6%)	66,1	66,6 (-0,8%)	[+50,8%]
5 s	71,0	63,5 (+10,4%)	116,3	95,0 (+18,4%)	[+45,3%]
10 s	143,8	116,6 (+18,9%)	190,6	184,3 (+3,3%)	[+38,8%]

Tab. 4.4 Prestazioni del Freeze-TCP con un collegamento locale a 10 Mbps [GMPG00].

Per dimostrare la qualità del protocollo Freeze-TCP in un ambiente con frequenti disconnessioni, i suoi autori hanno condotto vari esperimenti. Il primo riguarda un collegamento a breve distanza con un'elevata bandwidth disponibile. In particolare sono state effettuate dieci trasmissioni di un flusso di dati della dimensione di 10 MB su di un collegamento Ethernet a 10 Mbps con 10 disconnessioni per ogni trasmissione ed un intervallo di un secondo tra gli eventi. La distanza tra mittente e ricevente era di soli 3 collegamenti e l'intervallo di allarme è stato impostato uguale a 2,6 ms, valore corrispondente al RTT di un pacchetto di 1000 byte. La Tab. 4.4 riporta i dati sperimentali

presenti in [GMPG00], relativi a questo esperimento. Risultati analogamente buoni per il caso di un collegamento remoto possono essere reperiti sempre in [GMPG00]. Si tratta comunque ancora di collegamenti caratterizzati da bandwidth elevata.

Durata disconnes-sione	Tempo medio di trasferimento (in secondi) per 10 tentativi con 10 disconnessioni ciascuno				Guadagno Freeze-TCP +TR-ACKs sul TCP
	Con TR-ACKs		Senza TR-ACKs		
	TCP	Freeze-TCP	TCP	Freeze-TCP	
2,6 ms	185,7	178,8 (+3,7%)	183,1	179,7 (+1,9%)	[+2,35%]
30 ms	185,2	181,5 (+2,0%)	181,8	177,4 (+2,4%)	[+0,17%]
0,1 s	190,2	176,7 (+7,1%)	178,2	183,0 (-2,6%)	[+0,84%]
0,5 s	178,3	179,3 (-0,6%)	188,4	189,2 (-0,4%)	[+4,83%]
1 s	196,0	193,7 (+1,1%)	198,2	193,7 (+2,2%)	[+2,27%]
2 s	206,9	208,3 (-0,7%)	234,4	217,5 (+7,2%)	[+11,1%]
5 s	243,5	240,1 (+1,4%)			

Tab. 4.5 Prestazioni del Freeze-TCP con un collegamento locale a 38,4 kbps [GMPG00].

Più interessante dal punto di vista del collegamento wireless ad Internet da parte di dispositivi mobili, è invece il caso in cui la bandwidth disponibile sia limitata. In questo caso Freeze-TCP continua a comportarsi meglio del TCP tradizionale, ma in misura ridotta rispetto alla situazione precedente. Ciò risulta evidente dai dati inseriti in Tab. 4.5 che fanno riferimento ad un esperimento riportato in [GMPG00]. In tale esperimento dieci connessioni con distanze tra mittente e ricevente di tre collegamenti condividono una bandwidth di 38,4 kbps nel tentativo di trasmettere ciascuno un flusso di dati della dimensione di 500 kbyte. Il numero di disconnessioni è di 12 ciascuno con un intervallo di 10 secondi tra due eventi e l'intervallo di allarme è impostato uguale a 650ms che corrisponde all'incirca al RTT per un pacchetto di 1000 byte. Con una bandwidth limitata, il collegamento si satura molto prima rispetto ad una connessione con una bandwidth elevata. Conseguentemente, prevenire la congestione evitando di

invocare meccanismi per il controllo della stessa, porta a guadagni prestazionali minori.

L'aspetto critico di Freeze-TCP risiede nell'impostazione dell'intervallo di allarme. Prevenire il momento della disconnessione permette di ottenere risultati migliori rispetto ad una reazione dopo che questa si è già verificata. D'altra parte però, i pacchetti ZWA devono partire al momento giusto, in modo da non sprecare bandwidth diventando inattivi troppo presto. Allo stesso tempo, a tali pacchetti deve anche essere lasciato il tempo necessario per arrivare al mittente. L'errore nell'impostazione dell'istante di partenza dei pacchetti ZWA è direttamente proporzionale al degradamento delle prestazioni generali.

Occorre inoltre rilevare che Freeze-TCP riprende le trasmissioni dei dati dopo una disconnessione con la stessa velocità precedente. Questo comportamento permette di evitare sprechi di bandwidth in situazioni in cui le condizioni del canale prima e dopo la disconnessione rimangono sostanzialmente invariate. Gli autori stessi di Freeze-TCP ammettono che tale comportamento potrebbe invece risultare errato nel caso in cui sia stato effettuato un cambiamento di cella [GMPG00]. Le condizioni trasmissive della nuova cella potrebbero rivelarsi differenti, anche profondamente, da quelle della vecchia; l'utilizzo di tecniche aggressive durante la ripresa delle trasmissioni potrebbe portare ad un degradamento delle prestazioni complessive. In ragione di ciò potrebbe risultare consigliabile un atteggiamento più prudente dopo un handoff, riprendendo le trasmissioni utilizzando slow start o comunque una frequenza trasmissiva ridotta rispetto ai parametri precedenti. Un caso analogo, in cui Freeze-TCP assume un comportamento troppo aggressivo, è quello in cui oltre alla disconnessione si verifica anche una situazione di rete congestionata. Anche in questa situazione sarebbe preferibile riprendere le trasmissioni con una frequenza di invio dei dati

ridotta. In caso contrario, il livello di congestione verrà ulteriormente incrementato, con conseguente aumento delle perdite e calo delle prestazioni complessive.

La capacità di operare in situazioni con frequenti o lunghe disconnessioni, risulta però di limitata utilità in situazioni di BER elevato. Il verificarsi di perdite comporta comunque l'invocazione di meccanismi per il controllo della congestione, quali slow start o fast retransmit, rendendo inutili i tentativi di prevenire la congestione proprio per evitare perdite.

4.8 TCP-Probing (Tsaoussidis e Badr, 2000)

Abbiamo già ampiamente illustrato come gli errori causati dalla presenza di un tratto wireless del collegamento, vengano interpretati dai TCP tradizionali unicamente come congestione del canale, causando quindi un restringimento della finestra di invio. Il meccanismo di riduzione moltiplicativa ed incremento additivo della cwnd porta a sprecare la bandwidth disponibile. Ciò si verifica anche nel caso di errori casuali e sporadici, quando invece sarebbe opportuno un comportamento più aggressivo del protocollo di trasporto. Una conseguenza di questo comportamento è anche lo spreco di energia, che per i dispositivi mobili rappresenta una risorsa limitata e quindi preziosa. Infatti, non solo tempi di connessione più lunghi comportano maggior dispendio energetico, ma occorre anche aggiungere l'incapacità dei TCP tradizionali di monitorare la rete senza incorrere in perdite di segmenti. Infine, la ritrasmissione dei pacchetti persi durante un periodo di segnale fortemente attenuato, porterebbe solo ad altre

perdite e quindi ad ulteriori ritrasmissioni; anche in questo caso si verificherebbe un inutile dispendio di energia. Proprio per contrastare queste problematiche, Vassilios Tsaoussidis della Northeastern University e Ussein Badr della SUNY Stony Brook, hanno sviluppato nel 2000 un protocollo di trasporto alternativo che presta particolare attenzione alla conservazione energetica. Il loro protocollo, chiamato *TCP-Probing*, utilizza uno schema detto *Probe Cycle* che prevede lo scambio di pacchetti di prova tra mittente e ricevente aventi lo scopo di sondare la rete per poter adottare poi il comportamento più opportuno.

Nei TCP tradizionali, quando un segmento di dati è in ritardo, probabilmente perché è andato perso, il mittente effettua la ritrasmissione e restringe la cwnd e la ssthresh. Viceversa, con TCP-Probing, il mittente sospende l'invio di dati ed inizia un Probe Cycle. In pratica, vengono inviati verso il mittente soltanto dei particolari pacchetti di prova allo scopo di monitorare le condizioni del canale. Vi sono quattro tipi diversi di pacchetti di prova, due viaggiano dal mittente al ricevente (`probe1` e `probe2`) e altri due costituiscono le relative conferme di avvenuta ricezione (`pr1_ack` e `pr2_ack`). Tali pacchetti sono privi di payload ed utilizzano estensioni dei campi opzione degli header TCP. Infatti all'interno di ciascun pacchetto di prova vi è un campo `type` per distinguere tra i quattro tipi di pacchetto appena enunciati, un campo `lenght` per indicare la sua lunghezza complessiva ed un campo `id_number` per distinguere un gruppo da un altro.

Il mittente inizia un probe cycle inviando il segmento `probe1`, a cui il ricevente risponde con il segmento `pr1_ack`. Non appena `pr1_ack` raggiunge il mittente, questi invia anche il segmento `probe2`, al quale il ricevente risponderà con `pr2_ack`. Con questo scambio di pacchetti, il mittente può effettuare delle misure del RTT che utilizza poi per

comprendere le condizioni del canale trasmissivo ed adottare quindi le misure più appropriate. L'algoritmo discriminante proposto dagli ideatori, del TCP-Probing consiste in:

- Se i due RTT misurati sono compresi nell'intervallo [RTT migliore, RTT stimato al momento in cui inizia il Probe Cycle], si ipotizza che le condizioni del canale siano buone e che la causa che ha fatto scattare il probe cycle sia un errore transiente.

- Viceversa, si ipotizza che le condizioni del canale siano tali da suggerire un approccio conservativo.

Durante il Probe Cycle, il mittente utilizza due timer. Il primo, che chiameremo *probe timer*, serve a determinare se i pacchetti di prova sono andati persi. Il secondo, *measurement timer*, viene invece utilizzato per misurare gli RTT nei due scambi dei pacchetti di prova. L'insieme dei 4 pacchetti di un Probe Cycle viene identificato, come già spiegato, dal campo `id_number`; se uno dei pacchetti di prova viene perso, si riparte da capo con un altro Probe Cycle, con nuovi pacchetti di prova e timer reinizializzati. Se quindi persistono le cattive condizioni del canale trasmissivo, la durata del Probe Cycle viene automaticamente estesa fino a che le condizioni non ritornano sufficienti per effettuare due successive misure di RTT. Viceversa, con errori transienti, il Probe Cycle termina velocemente, in proporzione alla densità degli errori.

Qualora il mittente riceva un ack durante il Probe Cycle, provvede ad aggiornare la sua finestra di invio come farebbero i TCP tradizionali; allo stesso tempo però, non invia nuovi dati verso il ricevente finché il

Probe Cycle non è stato completato. Il mittente entra in fase di Probe Cycle alla scadenza di un timeout oppure al ricevimento di tre dupack. In entrambi i casi, se le condizioni del canale verificate dai pacchetti di prova risultano accettabili, TCP-Probing non entra in modalità slow start, bensì in modalità *Immediate Recovery*. In questa modalità, il protocollo di trasporto riprende dal punto in cui si trovava al ricevimento del timeout lasciando invariati i valori di cwnd e di ssthresh. Viceversa si invoca il meccanismo di Slow Start.

Durante il Probe Cycle viene comunque tenuto conto di tutti i dupack che arrivano, in questo modo si può procedere a "gonfiare" la finestra di congestione in maniera analoga a quanto previsto dallo schema di Fast Retransmit presente sui TCP tradizionali. Se le condizioni del canale rilevate dai pacchetti di prova non risultano soddisfacenti, contrariamente a quanto previsto dai TCP tradizionali, il mittente entra in modalità slow start. Le condizioni della connessione richiedono infatti un atteggiamento più prudente allo scopo di evitare ulteriori perdite.

TCP-Probing risulta più efficace delle versioni Tahoe[5], Reno e New Reno del TCP in situazioni con canale debole, come dimostrato dalla Fig. 4.17. Per realizzare il grafico è stato effettuato un esperimento in cui l'utente passava da uno stato con condizioni ottimali del canale ad un altro con trasmissioni condizionate da errori casuali con una frequenza stabilita. L'intervallo trascorso in ciascuno dei due stati prima di transitare nell'altro corrisponde a 10 secondi. L'esperimento è stato ripetuto con diversi BER per misurarne il goodput medio, ovvero il throughput effettivo misurato al ricevente.

[5] La versione TCP Tahoe precede quella Reno (illustrata nel par. 2.2) e si differenzia da essa per avere un approccio più conservativo all'arrivo di tre dupack: non utilizza, infatti, il meccanismo di fast recovery, bensì invoca slow start.

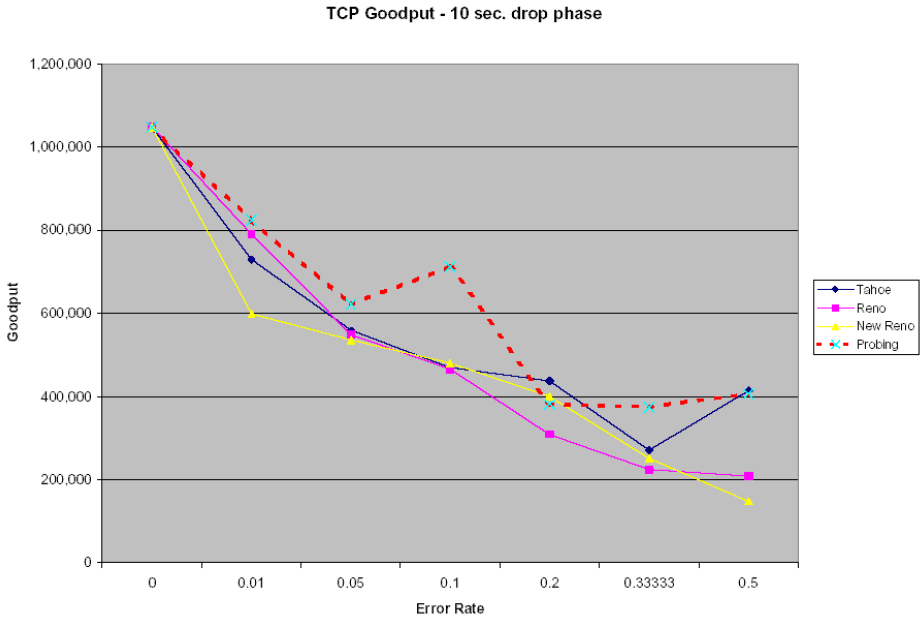

Fig. 4.17 Goodput con permanenza di 10 secondi in ciascuno stato [TB00].

Tranne il caso in cui le perdite nello stato di canale condizionato dagli errori costituivano il 20% del totale, TCP-Probing ottiene prestazioni superiori alle versioni Tahoe, Reno, New Reno. L'utilizzo dei pacchetti di prova permette di reagire con maggior consapevolezza, e quindi più efficacemente, alle diverse situazioni di congestione, errori transienti o errori persistenti.

Questi risultati positivi sono però anche dovuti ad una variabilità del canale piuttosto limitata. Se infatti restringiamo il periodo di permanenza del dispositivo mobile all'interno di ciascuno stato, TCP-Probing ottiene prestazioni meno positive. La Fig. 4.18 mostra il caso in gli stati del canale cambino ogni secondo. In questo caso, TCP-Probing ottiene prestazioni migliori degli altri protocolli confrontati soltanto nel caso in cui vi sia un BER superiore al 33% nel periodo di canale condizionato dagli errori.

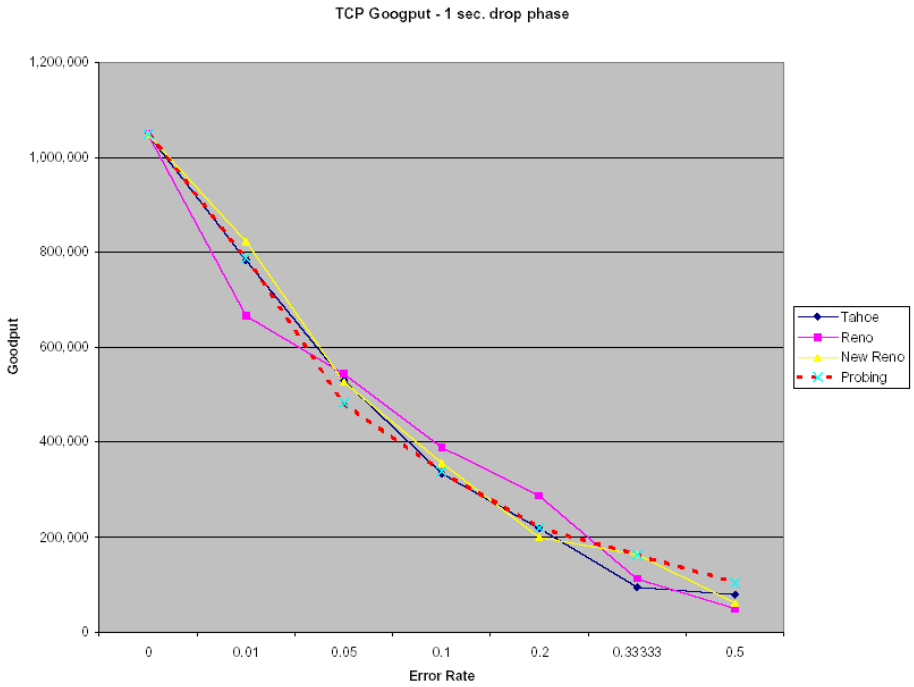

Fig. 4.18 Goodput con permanenza di 1 secondo in ciascuno stato [TB00].

TCP-Probing fornisce un modo per realizzare un utilizzo più efficiente delle risorse energetiche dei dispositivi mobili e della bandwidth disponibile su collegamenti wireless. L'invio dei pacchetti di prova durante il Probe Cycle richiede, data l'assenza di payload, minori risorse rispetto ai pacchetti normali. Ciò permette di monitorare lo stato della rete in maniera più efficiente, anche dal punto di vista del consumo energetico.

Il meccanismo dei pacchetti di prova utilizzato per saggiare le condizioni del canale prima di riprendere la trasmissione dei dati, conferisce al TCP-Probing buone capacità di conservazione dell'energia. Il suo utilizzo delle limitate risorse energetiche dei dispositivi mobili risulta più efficiente di quello dei protocolli di trasporto tradizionali. Durante il Probe Cycle non vengono infatti inutilmente spediti pacchetti completi di dati sul canale se le condizioni

non lo consentono, si attende invece che i periodici invii dei pacchetti di prova, privi di payload, confermino il ripristino delle condizioni di invio. La capacità di risparmio energetico di TCP-Probing rispetto ad altri protocolli è dimostrata dalle Figg. 4.19 e 4.20.

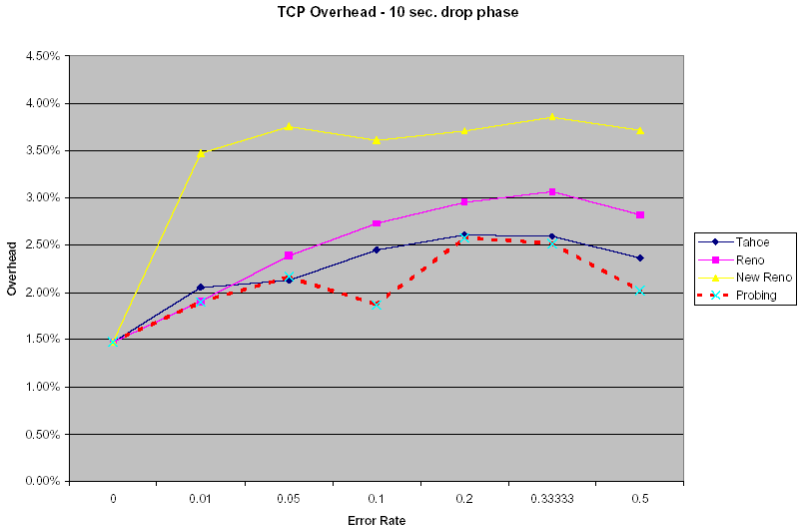

Fig. 4.19 Overhead con permanenza di 10 secondi in ciascuno stato[TB00].

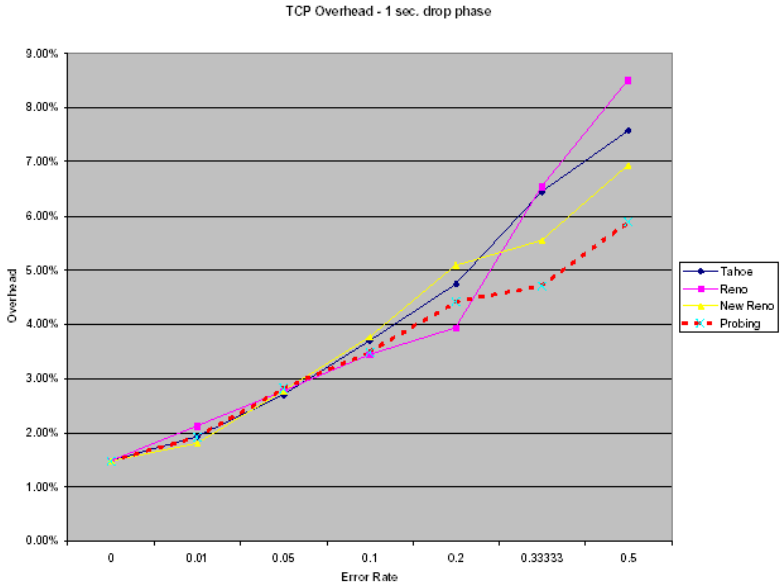

Fig. 4.19 Overhead con permanenza di 1 secondo in ciascuno stato [TB00].

4.9 WTCP (Sinha et al., 1999)

Nel 1999 Prasun Sinha, Narayanan Venkitaraman, Raghupathy Sivakumar e Vaduvur Bharghavan della University of Illinois, idearono il protocollo di trasporto WTCP per risolvere le problematiche di TCP su collegamenti wireless in un ambiente WWAN[6]. WTCP si presenta come uno schema composto di diverse soluzioni ai principali problemi legati ai collegamenti wireless. I principi chiave su cui si basa sono:

- Utilizzo di meccanismi end-to-end.

- Controllo della trasmissione basato sulla frequenza di arrivo al ricevente dei pacchetti, piuttosto che sul tradizionale sistema a "finestre".

- Utilizzo delle differenze tra i tempi di arrivo dei pacchetti al ricevente e di quelle al mittente, come metrica principale per stabilire la frequenza trasmissiva, anziché utilizzare le perdite dei pacchetti ed i relativi timeout.

- Utilizzo di pacchetti sack per il recupero efficiente delle perdite.

I principali problemi wireless che gli autori ritengono debbano essere trattati al fine di ottenere i migliori risultati sono: le perdite non dovute

[6] Wireless Wide-Area Network

a congestione, la latenza ampia e variabile e la possibile asimmetria tra il canale dei dati e quello degli ack [BPK97].

Analizzando il controllo della trasmissione, notiamo che, contrariamente ai protocolli di trasporto tradizionali, WTCP non utilizza gli ack come clock implicito, bensì fa affidamento su di un vero e proprio clock e adatta il traffico inviato in rete con la frequenza trasmissiva misurata per la connessione. Vengono monitorati i tempi di arrivo medi dei pacchetti al ricevente e, grazie alle differenze tra i ritardi ed il confronto con gli intervalli di invio al mittente, si ottiene una misura della bandwidth disponibile. Il mittente infatti include in ogni pacchetto inviato anche la separazione temporale dal precedente. In questo modo si può prevenire la congestione e ridurre la frequenza di invio dei pacchetti in modo da evitare perdite e non far scadere timeout. La frequenza di invio dei pacchetti viene quindi decisa dal ricevente e notificata al mittente includendola negli ack. Tra l'altro, il fatto che sia il receiver a calcolare la frequenza di invio grazie ai pacchetti ricevuti dal mittente, mette al riparo da eventuali ritardi nel percorso degli ack, qualora questo sia più lento (collegamento asimmetrico [BPK97]).

Nel caso in cui si verifichino delle perdite, WTCP è in grado di distinguere se si tratta di congestione o di errori analizzando i tempi di interarrivo; il mittente sarà così informato di quale frequenza trasmissiva adottare. Per poter discriminare le perdite dovute ad improvvise congestioni, WTCP mantiene traccia delle perdite di pacchetti in situazioni non congestionate e calcola la media e la deviazione nel numero di perdite di pacchetti non imputabili a congestione. Grazie al meccanismo di anticipo delle congestioni, le perdite imputabili a questa causa risultano scarse. Se si verificano periodi di disconnessione, questo schema ipotizza di trovarsi in una situazione di rete congestionata e riduce la frequenza di invio dei

pacchetti. Quando il ricevente determina che si tratta di disconnessione e non di congestione (capiremo più avanti che il riconoscimento avviene grazie all'invio di una coppia di pacchetti speciali), attende che venga ripristinata la connessione e poi reimposta la frequenza di invio sul valore precedente alla sua riduzione.

WTCP mantiene anche una "storia" dei recenti aumenti o diminuzioni della frequenza di invio [SNVS99]; questo allo scopo di realizzare la reazione più corretta possibile alla situazione della rete. Uno degli obiettivi del protocollo di trasporto è infatti quello di mantenere l'equità nel garantire a tutti gli utenti la possibilità di accedere alle risorse di bandwidth disponibile; la diminuzione delle frequenze di invio deve quindi essere di tipo moltiplicativo. Allo stesso tempo però, occorre utilizzare pesi e misure adeguate ai differenti casi. Nel caso in cui si stia reagendo ad una reale congestione, occorre agire con una riduzione consistente della frequenza di invio dei pacchetti, dimezzandone il valore. Viceversa, quando la congestione non è ancora avvenuta ma è comunque prevista, appare maggiormente appropriata una diminuzione più leggera.

WTCP è stato pensato appositamente per le WWAN e, considerando l'alto RTT di questo ambiente e la generale brevità delle trasmissioni, cerca di partire da subito con la frequenza trasmissiva più appropriata anziché utilizzare il meccanismo di slow start. Lo schema implementato da WTCP è quello *packet-pair*, ovvero mentre si stabilisce la connessione si inviano due segmenti di dimensione massima (MSS) e si calcola il loro ritardo interpacchetto allo scopo di ottenere una stima della frequenza di invio accettabile. Questo meccanismo può essere usato anche per recuperare da una disconnessione. Un altro metodo proposto per effettuare il recupero da disconnessione è quello di inviare ad intervalli regolari un pacchetto di prova (detto *probe packet*) dal

mittente al ricevente, dopo un periodo di tempo stabilito in cui non si sia ricevuto alcun ack. La differenza tra i due schemi è che nel primo, al ristabilirsi della connessione, viene ricalcolata la frequenza di invio dei segmenti in base alle nuove condizioni del canale, mentre nel secondo si riparte con i vecchi valori.

Per garantire l'affidabilità delle trasmissioni WTCP utilizza, come già annunciato, pacchetti di tipo sack. Tali pacchetti vengono inviati verso il mittente con frequenza stabilita dal mittente stesso e gli permettono di capire precisamente quali dei dati inviati sono andati persi. Confrontando il contenuto dei sack ricevuti con l'ultima ritrasmissione effettuata (mantenuta in memoria), il mittente riesce a comprendere se la ritrasmissione è andata persa o se potrebbe ancora essere in viaggio, e decidere quindi se sia il caso o meno di effettuare un invio ulteriore degli stessi dati.

Il meccanismo per l'invio dei pacchetti sack è ulteriormente modificato dal fatto che sono stati esclusi i timeout. La grande variabilità della latenza in ambiente wireless provoca, come abbiamo visto nei capitoli iniziali, difficoltà nel calcolare in maniera attendibile gli RTT, causando cali vistosi nelle prestazioni del TCP. Il protocollo WTCP supera questo problema eliminando i timeout e delegando al mittente il compito di capire, grazie ai sack ricevuti dal ricevente, quando è necessario effettuare una ritrasmissione. Il mittente riceve questi pacchetti di conferma periodicamente, secondo una frequenza che egli stesso stabilisce e di cui il ricevente viene informato attraverso i pacchetti di dati. Per calcolare la frequenza con cui i sack devono essere inviati si tiene conto delle perdite dei sack osservate dal mittente, della natura del canale trasmissivo (half duplex o full duplex), della media e della deviazione standard negli intertempi nella ricezione dei sack. Se il

mittente non riceve alcun sack per un determinato periodo di tempo[7], si entra in *blackout mode* ipotizzando una disconnessione. In questo stato il mittente invia dei pacchetti di prova, con frequenza periodica e con numero di sequenza incrementale, lungo la connessione attendendo un ack dal ricevente che lo faccia uscire dallo stato di blackout.

L'utilizzo di tecniche differenti, combinate in maniera coordinata per affrontare efficacemente un largo spettro di problematiche tipiche degli ambienti wireless, permette al WTCP di ottenere buone prestazioni in fase simulativa. Riportiamo in Figg. 4.21 e 4.22 due grafici presentati in [SNVS99] che illustrano chiaramente la maggiore velocità trasmissiva del protocollo WTCP rispetto alle versioni Vegas e New Reno del TCP, in particolare in condizioni di BER elevati. In Fig. 4.21 vengono infatti mostrati i risultati migliori di WTCP ottenuti in una simulazione che prevedeva un singolo flusso trasmissivo con 50kbps di bandwidth ed una frequenza di errore casualmente distribuita del 4%. Sotto queste condizioni, WTCP riesce ad inviare un quantitativo di dati superiore, circa il doppio, rispetto a quello inviato nello stesso tempo da TCP New Reno o da TCP Vegas.

Anche in presenza di un canale trasmissivo ottimale, ovvero senza perdite, WTCP non offre prestazioni inferiori a quelle del TCP Vegas e mantiene comunque un vantaggio su TCP New Reno, come illustrato dalla Fig. 4.22. La linea rossa corrispondente al protocollo WTCP è infatti sovrapposta a quella verde del TCP Vegas. Leggermente più in basso, risultando quindi un po' più lento, si trova invece la versione New Reno del TCP.

[7] Gli autori suggeriscono di usare un periodo soglia di 5 secondi [SNVS99].

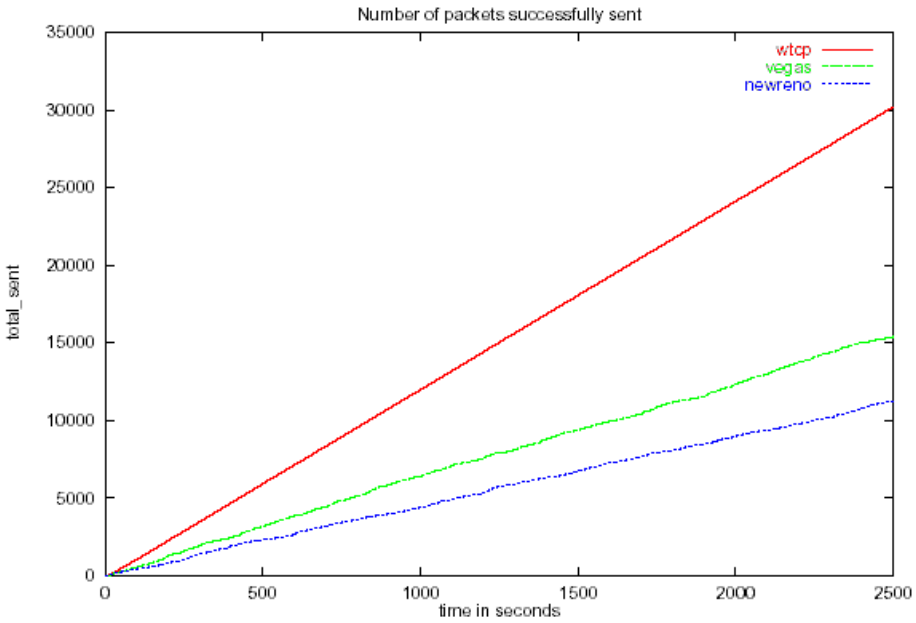

Fig. 4.21 Pacchetti inviati con successo (frequenza di errore del canale: 4%) [SNVS99].

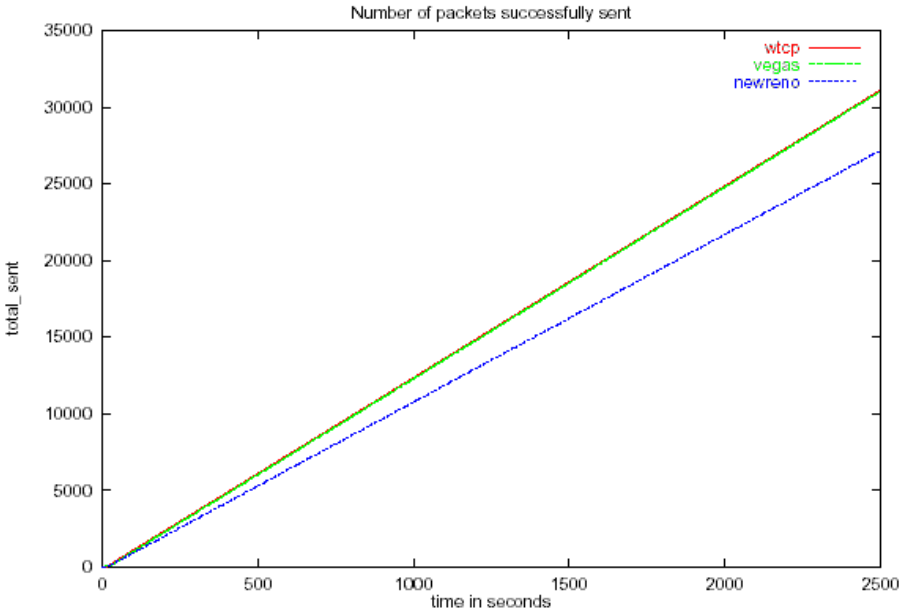

Fig. 4.22 Pacchetti inviati con successo (frequenza di errore del canale: 0%) [SNVS99].

Grazie all'aggiustamento dinamico della frequenza trasmissiva ed all'eliminazione dei timeout, WTCP permette l'invio dei dati in maniera fluida superando i problemi legati alla difficoltà di calcolare i RTT in ambiente wireless. L'impiego di sack consente un recupero delle perdite più efficiente. I pacchetti probe possono garantire un veloce recupero da disconnessioni ed un inizio di trasmissione a pieno regime; il secondo aspetto in particolare migliora notevolmente le prestazioni delle connessioni di breve durata.

Tra i possibili lati negativi di questo protocollo vi è una eccessiva complessità computazionale delegata al ricevente, che di solito è il dispositivo mobile. Sappiamo infatti che questi dispositivi sono caratterizzati da risorse limitate mentre con questo protocollo si trovano gravati da pesanti consumi nell'utilizzo del processore e quindi anche energetici. Rimane inoltre da verificare che WTCP non abbia un atteggiamento troppo aggressivo nell'accaparrarsi le risorse disponibili.

4.10 TCP Westwood (Mascolo et al., 2001)

Nel 2001 Saverio Mascolo del Politecnico di Bari, Claudio Casetti del Politecnico di Torino, insieme a Mario Gerla, M. Y. Sanadidi e Ren Wang della UCLA, hanno proposto un nuovo protocollo di trasporto chiamato TCP Westwood. Grazie a modifiche del sistema di gestione delle finestre di invio del mittente, questo protocollo consente di ottenere prestazioni migliori rispetto ai TCP tradizionali su rete mobile [MGPGC02].

Le principali meccanismi innovativi introdotti da TCP Westwood sono riassumibili in tre punti principali:

- Stima end-to-end della bandwidth disponibile allo scopo di discriminare le cause delle perdite di pacchetti.

- Funzionamento senza bisogno di ispezioni o intercettazioni di pacchetti ai nodi intermedi.

- Monitoraggio continuo della connessione da parte del mittente grazie agli ack ricevuti ed utilizzo, al verificarsi di ack duplicati o di timeout, della bandwidth stimata per impostare i valori di ssthresh e di cwnd.

Già altre proposte di nuovi protocolli di trasporto presentavano schemi per la stima della bandwidth [GLMW99, GWL99, GWL00]. Questi schemi tuttavia richiedevano interventi da parte dello strato di rete, come ad esempio la misurazione da parte dei router intermedi della bandwidth disponibile, ovvero di operazioni che richiedono l'utilizzo di nuove funzioni dello strato di trasporto. TCP Westwood invece fa affidamento solo su informazioni già disponibili negli attuali header TCP e non richiede nessun tipo di supporto da parte dello strato di trasporto, mantenendo così il principio della separazione e modularità dei diversi strati dell'architettura. Allo stesso tempo TCP Westwood mantiene la semantica end-to-end del collegamento.

Vediamo in dettaglio come la frequenza media di arrivo degli ack viene utilizzata dal mittente per stimare la bandwidth. Dopo l'arrivo di tre dupack o lo scadere di un timeout, la sorgente ipotizza che si possa essere di fronte ad una congestione della rete. Il canale trasmissivo si trova quindi ad un livello saturo e la frequenza di invio equivale all'incirca alla miglior quantità di banda concedibile agli utenti. Ogni volta che un ack raggiunge il mittente, porta con sé informazioni sul quantitativo di dati che ha raggiunto il ricevente e quindi sulla bandwidth disponibile. TCP Westwood utilizza la stima della bandwidth per impostare in maniera appropriata i valori di cwnd e di ssthresh, secondo uno schema chiamato *Faster Recovery* [CGLMS00]. Impostando il valore di ssthresh vicino alla capacità della rete, infatti, si potrebbero ottenere prestazioni migliori.

Il meccanismo di Faster Recovery si basa sull'ipotesi di stimare la bandwidth disponibile grazie alla frequenza di ricevimento degli ack. Per far questo vengono presi in considerazione anche i dupack, pure questi infatti sono testimonianza dell'avvenuta ricezione di dati. Indichiamo con `BWE` la bandwidth stimata, con `dim_pacch` la dimensione dei pacchetti, con `t_corr` il tempo corrente e con `t_ult_ack` il tempo di ricevimento dell'ultimo ack che ha raggiunto il mittente. L'algoritmo utilizzato, ad ogni ricevimento di ack, risulta quindi:

```
BWE_sempl = dim_pacch*8/(t_corr-t_ult_ack)

BWE = BWE*α + BWE_sempl*(1 - α)
```

La costante α rappresenta un coefficiente incrementale e gli autori suggeriscono di utilizzare 0,8 come suo valore. Qualora la dimensione dei pacchetti rimanga costante, al valore `dim_pacch` occorre sostituire la media delle dimensioni degli ultimi *n* pacchetti. Analogo problema si

presenta per i dupack poiché non trasportano informazioni inerenti la quantità di dati ricevuta. In questo caso gli autori propongono di utilizzare la media precedente al ricevimento dei dupack e di aggiornare tale media solo dopo che siano stati ricevuti degli ack riferita a nuovi dati.

Lo schema Faster Recovery utilizza la Bandwidth così stimata (BWE) per impostare i valori di cwnd e di ssthresh secondo gli algoritmi:

- Caso ricevimento di *n* (solitamente tre) dupack:

```
ssthresh = (BWE * RTTmin)/dim_segmento;
if (cwnd > ssthresh) /*congest. avoid.*/
 cwnd = ssthresh;
endif
```

- Caso scadenza di un timeout:

```
ssthresh = (BWE * RTTmin)/dim_segmento;
if (ssthresh < 2)
 ssthresh = 2;
endif;
cwnd = 1;
```

Il valore di RTT_{min} rappresenta il RTT più piccolo rilevato durante tutta la connessione. Il tentativo degli autori è quello di far convergere lo schema verso la situazione di utilizzo della bandwidth realmente disponibile. Analizzando il caso in cui venga ricevuto un certo numero di dupack, notiamo che il valore di ssthresh viene impostato uguale alla capacità della connessione in quel momento. Infatti il ricevimento di dupack è indice che si è raggiunto il massimo trasmissibile correntemente dal canale. La variabile cwnd viene posta uguale a ssthresh solo nel caso che sia maggiore di quest'ultima. Se invece ci si trova di fronte alla scadenza di un timeout, il valore di cwnd viene impostato uguale a 1 mentre ssthresh prende il valore corrispondente

alla bandwidth stimata al momento dello scadere del timeout, in modo da garantire un recupero veloce.

Faster Recovery offre un meccanismo per decrementare la finestra di invio in maniera consapevole della bandwidth disponibile. Allo stesso modo, sarebbe utile uno schema che riconosca quando la frequenza di immissione dei pacchetti in rete può essere incrementata con tranquillità. Proprio per questo, gli autori hanno completato Faster Recovery con il meccanismo *Gradual Faster Recovery*, rappresentato dal seguente algoritmo:

```
if (cwnd > ssthresh) AND (cwnd < BWE*RTT_min)
then
     ssthresh += (BWE*RTT_min – ssthresh)/2;
```

Questo algoritmo viene invocato ogni 500ms per incrementare il valore di ssthresh qualora le condizioni del canale lo consentano.

Considerando t_k il momento temporale in cui si è ricevuto un certo ack che notifica l'avvenuto ricevimento di d_k dati, una semplice stima della bandwidth b_k potrebbe essere:

$$b_k = \frac{d_k}{t_k - t_{k-1}} \qquad (1)$$

Allo scopo di non creare congestione dovuta ad una cattiva stima è opportuno applicare un filtro a questa formula, in modo da eliminare le componenti estreme. Gli autori propongono la (2) come filtro discreto da utilizzare.

$$\hat{b}_k = \frac{\frac{2\tau}{t_k - t_{k-1}} - 1}{\frac{2\tau}{t_k - t_{k-1}} + 1} \hat{b}_{k-1} + \frac{b_k + b_{k-1}}{\frac{2\tau}{t_k - t_{k-1}} + 1} \qquad (2)$$

In questa formula $\hat{b}_k$ rappresenta la stima filtrata della bandwidth disponibile al tempo t_k. Il valore $1/\tau$ è la frequenza di taglio del filtro, ovvero tutte le componenti al di sopra di questo valore vengono escluse. Per comprendere meglio il comportamento della (2), consideriamo la semplificazione che si ottiene supponendo un tempo di interarrivo costante tra i pacchetti, ad esempio:

$$t_k - t_{k-1} = \Delta_k = \tau / 10$$

Il filtro (2) diventa allora il seguente filtro con coefficienti costanti in cui la costante *a* corrisponde a *0,9*:

$$\hat{b}_k = a\hat{b}_{k-1} + \frac{(1-a)}{2}[b_k + b_{k-1}] \qquad (3)$$

Ovviamente la (3) non può essere utilizzata in un contesto reale, in quanto l'ipotesi di intervalli di arrivo uguali tra i pacchetti è troppo lontana dalla realtà. Tuttavia si può notare come la formula porti ad avere un nuovo valore $\hat{b}_k$ che proviene al 90% dal precedente valore $\hat{b}_{k-1}$ ed il restante 10% dalla media aritmetica delle stime semplici di tipo (1), b_k e b_{k-1}. Nella formula reale occorre tenere conto del tempo effettivamente trascorso tra l'arrivo dei pacchetti, ovvero delle differenze t_k - t_{k-1}. Il valore rappresentato da *a* risulta inversamente

proporzionale a quello dei tempi di interarrivo, ciò corrisponde al fatto che quanto più $\hat{b}_{k-1}$ risulta vecchio, tanto meno occorrerà tenerne conto per calcolare la nuova stima.

La stima della bandwidth deve tenere conto, come già spiegato, anche dei dupack e potrebbe ugualmente essere stravolta da una combinazione di ack ritardati e cumulativi. Pertanto occorre che il mittente sia in grado di:

- tenere traccia del numero di dupack ricevuti prima che nuovi dati siano stati ricevuti;

- rilevare gli ack ritardati e comportarsi di conseguenza.

Per realizzare questo obiettivo gli autori hanno realizzato una procedura chiamata *AckedCount*, la quale si occupa di realizzare un conteggio corretto del numero di pacchetti da considerare per realizzare una buona stima della bandwidth. Questa procedura verifica per ogni ack la quantità di nuovi dati di cui si sta dichiarando l'avvenuta ricezione. Se i nuovi dati ricevuti dal mittente sono uguali a zero, allora si tratta sicuramente di un dupack e nel calcolo per stimare la bandwidth viene conteggiato un singolo pacchetto in più (anche se fuori ordine). Qualora il numero di dati per cui si è ricevuto l'ack corrispondono a più di un segmento, la procedura considera il ricevimento di un ack ritardato o cumulativo e durante la stima della bandwidth considera tutti i nuovi segmenti giunti a destinazione; per far questo tiene anche conto di quanti segmenti erano già stati precedentemente conteggiati in seguito al ricevimento di dupack.

Gli autori hanno studiato il comportamento di TCP Westwood in presenza di più connessioni, chiedendosi in particolare se fosse mantenuta l'equità nel concedere a tutti le stesse possibilità di accesso al canale (fairness). Il risultato della loro sperimentazione è stato che TCP Westwood si dimostra più equo rispetto a TCP Reno in situazioni di presenza simultanea di connessioni aventi differenti periodi di propagazione. Questo perché, diversamente da TCP Reno, TCP Westwood tiene conto del RTT nella riduzione del valore di cwnd.

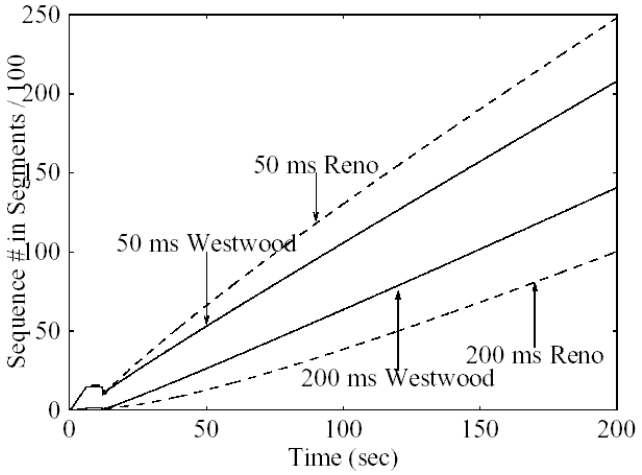

Fig. 4.23 Segmenti trasferiti con diversi RTT [MCGSW01].

La Fig. 4.23 illustra il caso in cui due connessioni aventi RTT diversi condividano lo stesso canale. L'esperimento è stato compiuto utilizzando prima con due connessioni utilizzanti TCP Reno, con RTT uguali rispettivamente a 50 ms e 200 ms, ed è stato successivamente ripetuto con due analoghe connessioni che impiegavano però TCP Westwood. Dalla figura si comprende che entrambi i protocolli di trasporto offrono migliori prestazioni con RTT bassi. TCP Westwood ha però un divario inferiore tra le due connessioni, con una distribuzione più equa della bandwidth complessivamente disponibile.

Un protocollo di trasporto che preservi uguali possibilità di trasmissione per tutte le connessioni deve anche evitare di togliere bandwidth a connessioni utilizzanti protocolli di trasporto diversi. Da questo punto di vista TCP Westwood sembra non garantire in pieno un atteggiamento amichevole nei confronti degli altri protocolli, come dimostrato da un esperimento condotto dagli ideatori di TCP Westwood e riportato in [MCGSW01]. In tale esperimento sono state misurate le prestazioni ottenute da 20 connessioni su di un canale avente una bandwidth di 2 Mbps. Il risultato ottenuto da 20 flussi di dati utilizzanti TCP Westwood è stato molto simile a quello ottenuto dal TCP Reno: 0,0994 Mbps di throughput medio per il primo caso e 0,0992 Mbps per il secondo. Si è poi provato ad utilizzare contemporaneamente 10 connessioni utilizzanti TCP Westwood ed altre 10 utilizzanti TCP Reno. Per le prime il throughput medio misurato è stato di 0,1078, con un guadagno superiore all'8%, mentre per le seconde è stato di 0,0913 con una perdita dell'8%: in totale si è quindi registrata una perdita di equità di circa il 16%.

Tra i vantaggi di TCP Westwood dobbiamo rilevare la sua capacità, grazie al meccanismo di Faster Recovery ed alla stima della bandwidth, di evitare inutili restringimenti della finestra di invio in situazioni di errori casuali e sporadici. I vantaggi prestazionali ottenuti rispetto a TCP Reno e TCP Sack sono evidenziate in Fig. 4.24, riportata da [MCGSW01]. In tale figura viene illustrato il throughput ottenuto al variare della percentuale di perdite di pacchetti media. I dati sono stati ottenuti supponendo di avere un collegamento tra un host fisso ed uno mobile separato da una base station. Il tratto tra host fisso e base station ha una latenza di 45 ms ed una bandwidth di 10 Mb, mentre quello tra base station e host mobile ha una latenza di 0,01 ed una bandwidth di 2 Mb.

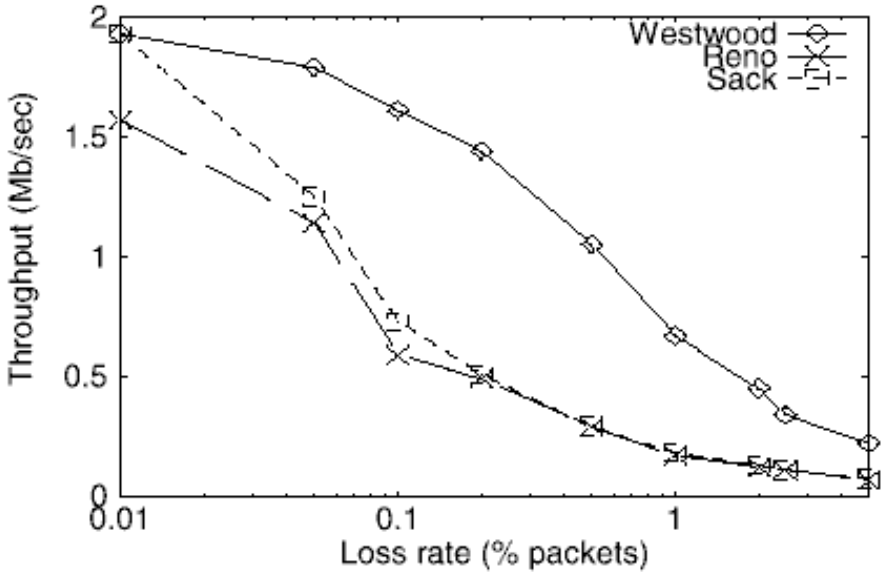

Fig. 4.24 Throughput ottenuto con diverse percentuali di perdite [MCGSW01].

Quando invece la frequenza degli errori diviene elevata, TCP Westwood potrebbe risultare preferibile l'utilizzo di schemi che prevedono il recupero al livello di collegamento, come ad esempio il protocollo Snoop. Un recupero end-to-end infatti parte svantaggiato rispetto ad un recupero locale, allo stesso tempo però TCP Westwood non richiede modifiche nei nodi intermedi della rete e offre quindi maggiore scalabilità.

TCP Westwood si dimostra efficace anche in condizioni di segnale debole o di brevi disconnessioni. Tali prestazioni sono dimostrate attraverso i risultati di un esperimento svolto dagli autori di TCP Westwood e presentato in [MCGSW01]. In particolare si è ipotizzato un modello a due stati del collegamento: nel primo stato la percentuale di pacchetti persi è dello 0,001%, mentre nel secondo corrisponde ad un valore variabile tra 0 e 30% per il caso di segnale debole , oppure al 100% per il caso di disconnessione. In Fig. 4.24 riportiamo i risultati di throughput ottenuti da TCP Westwood, TCP Reno e TCP Sack,

considerando 8 secondi di durata media dello stato di segnale forte e 4 secondi per quello debole. Risultano evidenti i benefici apportati dal meccanismo di stima della bandwidth contro i tradizionali schemi di controllo della congestione. Sull'asse delle ordinate sono presentati i valori di throughput ottenuti, mentre su quello delle ascisse vi è la percentuale di perdite utilizzata per il periodo di segnale debole.

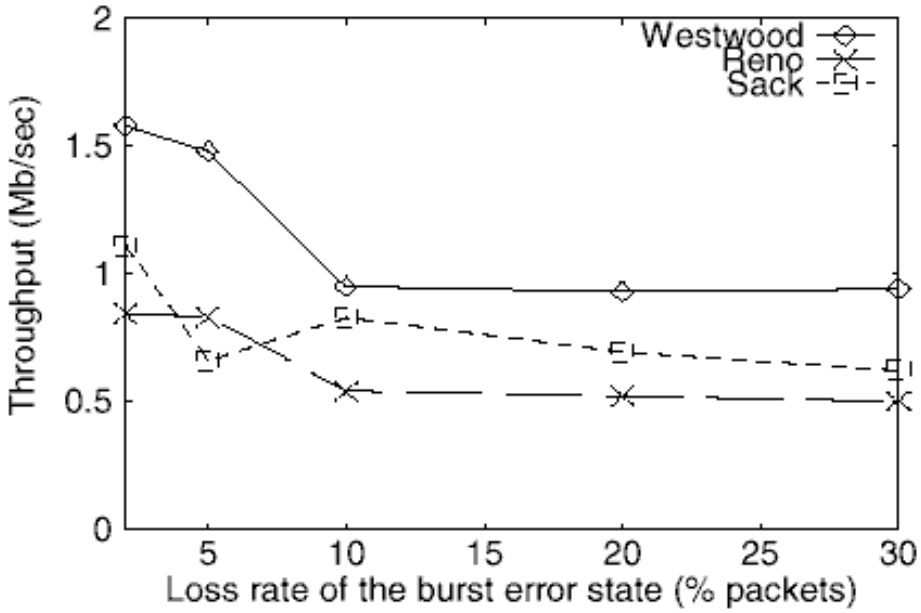

Fig. 4.24 Throughput con canale debole [MCGSW01].

In Fig. 4.25 riportiamo invece i risultati relativi all'esperimento condotto per misurare le prestazioni di TCP Westwood nei confronti di TCP Reno e TCP Sack, in presenza di periodi di disconnessione. In questo caso la durata media del periodo di buona condizione del canale è di 4 secondi. Per i periodi di disconnessione sono stati simulati vari intervalli di tempo, compresi tra 0 e 0,5 secondi. Anche in questo contesto TCP Westwood ottiene prestazioni superiori a quelle dei due protocolli tradizionali. Sull'asse delle ordinate vengono presentati i valori di throughput, mentre su quello delle ascisse si trova la durata media in secondi del periodo di disconnessione simulato.

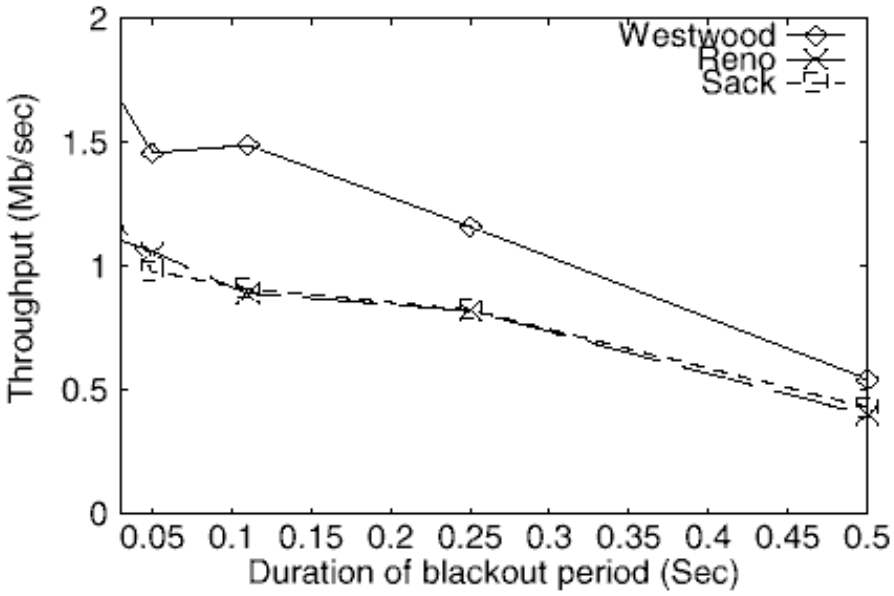

Fig. 4.25 Throughput in presenza di disconnessioni [MCGSW01].

La stima della bandwidth effettuata al mittente tiene conto sia del percorso dei dati verso il ricevente e sia di quello degli ack verso il mittente. Pertanto, TCP Westwood potrebbe non garantire adeguate prestazioni con un canale asimmetrico, soprattutto in situazioni di BER elevato. La stima della bandwidth risentirebbe infatti delle difficoltà degli ack di raggiungere il mittente. Un ulteriore calo delle prestazioni si ha quando il tempo di propagazione tra mittente e ricevente è troppo basso o troppo elevato, come mostrato in Fig. 4.26 riportata da [MCGSW01]. Nel primo caso non sono necessarie finestre di invio ampie e dunque i vantaggi di stimare la bandwidth disponibile per garantire maggiori frequenze di invio sono meno evidenti, mentre nel secondo le informazioni relative alle condizioni del canale giungono troppo tardi al mittente per essere ancora valide. La figura presenta sull'asse delle ordinate i valori di throughput misurato al variare dei ritardi di propagazione; questi ultimi sono indicati sull'asse delle ascisse.

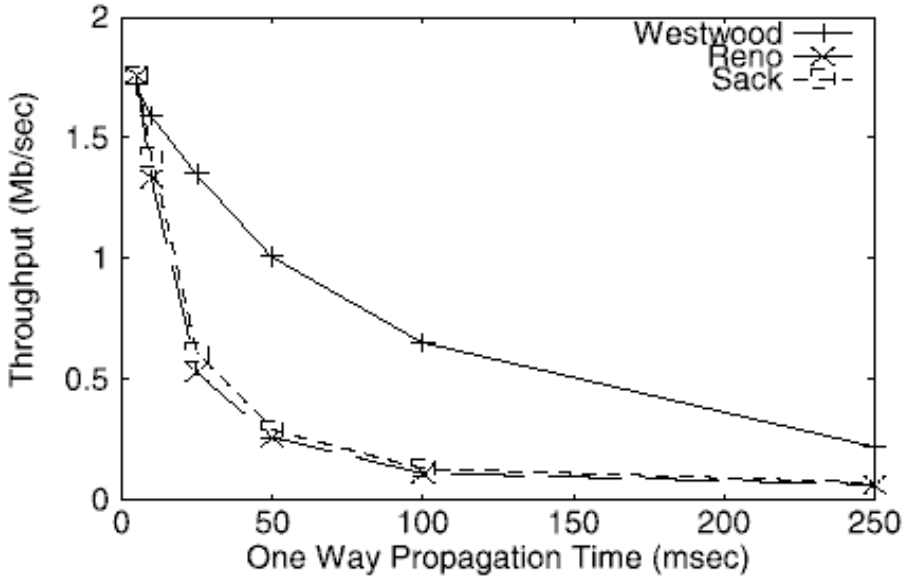

Fig. 4.26 Throughput ottenuti su diversi ritardi di propagazione [MCGSW01].

5

CONSIDERAZIONI FINALI

I protocolli di trasporto finora esposti presentano vantaggi e lacune. In questo capitolo riassumeremo in maniera coesa ed ordinata le peculiarità di ogni proposta nel tentativo di far emergere dal confronto le soluzioni migliori.

Nel secondo capitolo di questa tesi abbiamo presentato i TCP tradizionali, ovvero quelli più frequentemente utilizzati nelle attuali comunicazioni via Internet. Nessuno di essi appare però adatto ad operare in un ambiente wireless. Nel terzo capitolo abbiamo infatti evidenziato le problematiche che scaturiscono dal loro utilizzo da parte di dispositivi mobili che si connettono ad Internet. La necessità di realizzare un protocollo di trasporto capace di superare queste problematiche ha portato i ricercatori a sviluppare diversi TCP alternativi. Nel capitolo quarto abbiamo quindi riportato, descrivendole ed analizzandole, le proposte più significative nel panorama dei protocolli di trasporto ideati appositamente per collegamenti wireless.

Cerchiamo ora di riordinare le idee esposte per ciascuno dei nuovi protocolli presentati allo scopo di individuare, se esiste, una soluzione che possa risolvere i problemi prestazionali del TCP in un contesto wireless.

La Tab. 5.1 è una matrice a doppia entrata che riassume secondo una metrica comune i pregi e le lacune dei vari protocolli di trasporto. In cima alle colonne elenchiamo i nomi dei protocolli e all'inizio di ogni riga riportiamo un problema che occorre affrontare per ottenere un TCP in grado di operare in ambiente wireless. La tabella è divisa orizzontalmente in due parti per distinguere le problematiche che causano cali prestazionali da quelle di altro tipo. Per completezza riportiamo nelle prime quattro colonne anche i TCP tradizionali; in questa maniera risulterà anche più semplice effettuare eventuali confronti con essi. Le problematiche riportate all'invio di ogni riga sono state verbalmente formulate in modo che, se nella casella corrispondente ad un protocollo vi è il simbolo √, allora quel protocollo affronta in maniera positiva quella problematica. Ad ogni √ corrisponde dunque una caratteristica positiva mentre ad ogni casella vuota corrisponde una lacuna od un atteggiamento negativo.

Analizziamo ora le varie proposte partendo da alcune considerazioni riguardo al primo protocollo di trasporto incontrato. TCP Reno è il protocollo di trasporto più diffuso in internet ed è quindi il riferimento principale quando si vogliano individuare delle lacune nel sistema attuale oppure misurare le prestazioni ottenute da nuovi protocolli. Per questo motivo appare ovvio che tutte le questioni nella prima parte della tabella siano soddisfatte, le modifiche eventuali richieste dagli altri schemi proposti sarebbero da attuare proprio sulla versione Reno. Allo stesso modo è ovvio che tutte le caselle corrispondenti a problematiche prestazionali rimangano vuote, poiché tali problematiche relative all'ambiente wireless sono scaturite proprio nell'osservazione del TCP Reno.

La versione New Reno introduce gli ack parziali nei meccanismi del TCP Reno e richiede quindi delle modifiche al codice sia del mittente e sia del ricevente. Questo meccanismo gli consente di recuperare anche in situazioni di perdite multiple all'interno di una singola finestra di invio. Il recupero avviene per un singolo pacchetto ogni RTT ma consente comunque una limitata riduzione del numero dei timeout e quindi del numero totale di pacchetti trasmessi per inviare una certa quantità di dati.

TCP Vegas parte da un approccio differente rispetto alle versioni Reno e New Reno, cercando di rilevare una congestione incipiente prima che si realizzi. Grazie ad un meccanismo di stima della bandwidth disponibile tramite i tempi di arrivo degli ack, è possibile ridurre l'invio dei dati per evitare di perdere pacchetti e di doverli poi ritrasmettere.

Il protocollo di trasporto fra quelli tradizionali che meglio si adatta a situazioni di BER elevato è senz'altro il TCP Sack. Utilizzando i pacchetti di conferma di tipo sack, questo protocollo riesce infatti a recuperare anche da perdite multiple all'interno di una singola finestra di invio, limitando così il numero di timeout e di conseguenza i restringimenti della finestra di invio. La precisa consapevolezza di quali dati sono stati persi permette, inoltre, di limitare le ritrasmissioni ridondanti. Ricordiamo però che l'implementazione del meccanismo sack occupa tutto lo spazio delle opzioni nell'header TCP, rendendo impossibile l'eventuale utilizzo di altre estensioni [JB88]. Inoltre, non essendo stato pensato appositamente per operare in ambiente wireless, TCP Sack non affronta altri problemi quali, ad esempio, le disconnessioni o gli handoff. Le modifiche richieste per implementare il TCP Sack si concentrano su ricevente e mittente: il primo deve poter costruire i pacchetti di tipo sack ed il secondo deve essere in grado di interpretarli.

Il protocollo I-TCP realizza una netta divisione del collegamento. Vengono infatti realizzate due distinte connessioni, la prima tra l'host fisso e la base station e la seconda tra la base station e l'host mobile, utilizzando ack separati. Se da una parte questo permette di nascondere le problematiche wireless all'host fisso e di effettuare in loco tempestive ritrasmissioni dei pacchetti persi, dall'altra non rispetta la semantica end-to-end che è una delle basi su cui si fonda Internet. Abbiamo visto nel par.4.2 come un guasto della base station può portare alla perdita irrimediabile di alcuni dati. Inoltre, il controllo e l'eventuale ritrasmissione dei pacchetti in transito richiesto alla base station comporta l'impossibilità per questo schema di gestire trasmissioni criptate. Le modifiche necessarie per utilizzare I-TCP riguardano principalmente la base station. Come abbiamo già spiegato nel par. 4.1, risultano preferibili i protocolli che richiedono modifiche solo al ricevente o al mittente; nel caso in cui le modifiche siano richieste ad entrambi gli host, occorre assicurarsi che possano utilizzare anche un TCP tradizionale. Invece, i protocolli che necessitano di modifiche ai nodi intermedi, siano anche limitate alla base station come in questo caso, incontrano difficoltà considerevoli ad essere implementati su larga scala.

Un altro protocollo che effettua una divisione della connessione in due parti è il M-TCP, il quale però rispetta il paradigma end-to-end: gli ack inoltrati verso il mittente sono gli stessi spediti dal ricevente. Quando si accorge che sta per avvenire una disconnessione, M-TCP "soffoca" il mittente facendogli sospendere le trasmissioni ma anche i timeout. Inoltre, mentre TCP Reno deve attendere lo scadere di un timeout per riprendere le trasmissioni al ristabilirsi di una connessione, M-TCP consente la ripresa immediata a piena velocità grazie all'invio di un ack trattenuto dal ricevente. In questo modo si evitano inutili perdite di tempo. Tramite i meccanismi di ritrasmissione dalla base station al

dispositivo mobile, M-TCP riesce a recuperare dalle perdite dovute ad errori incorsi sul tratto wireless senza gravarne l'host fisso. D'altra parte però, tali ritrasmissioni concorrono a rendere variabile il RTT misurato e non è detto che nel frattempo non scada anche un timeout al mittente provocando la simultanea ritrasmissione del pacchetto anche da parte di quest'ultimo. Rimane inoltre il dubbio che il mancato utilizzo dei pacchetti sack limiti in parte i vantaggi ottenibili. Nella Tab.5.1 abbiamo comunque segnalato il problema relativo a "Gestione BER elevato" come positivamente affrontato poiché la divisione della connessione permette di recuperare velocemente dagli errori, inoltre gli ideatori di M-TCP propongono di limitare il numero delle perdite grazie all'utilizzo di FEC. Analogamente al I-TCP, anche in questo caso il controllo e l'eventuale ritrasmissione dei pacchetti da parte della base station compromette la possibilità di gestire trasmissioni criptate. Le modifiche necessarie all'implementazione del M-TCP coinvolgono ricevente e nodi intermedi (le base station).

Lo Snoop Protocol garantisce delle buone prestazioni anche in presenza di BER elevati grazie al suo meccanismo di ritrasmissione dei pacchetti operato dalla base station. Eventuali perdite sul tratto wireless vengono quindi recuperate localmente e con maggiore tempestività rispetto a quanto potrebbe fare l'host fisso. Si riducono così il numero di timeout e gli inutili restringimenti della finestra di invio. D'altra parte, come per M-TCP, le ritrasmissioni locali potrebbero essere causa di ridondanze e di aumento della variabilità degli RTT misurati. Snoop Protocol non offre inoltre alcun meccanismo specifico per la gestione delle disconnessioni, con conseguente invocazione della procedura di slow start. Un ulteriore difetto dello Snoop Protocol risiede nell'impossibilità, dato il ruolo della base station nella gestione dei pacchetti in transito, di essere utilizzato con trasmissioni criptate. Notiamo infine che le modifiche richieste per implementare Snoop Protocol si concentrano nella base station.

Passando alle proposte di protocolli di trasporto che non dividono la connessione, troviamo innanzitutto lo schema Delayed Dupacks il quale offre un meccanismo end-to-end per limitare gli inutili restringimenti della finestra di invio. Condizione per il suo funzionamento è che lo strato di collegamento sia dotato di meccanismi di ritrasmissione dei pacchetti persi. Infatti, grazie al ritardato invio del terzo dupack, Delayed Dupacks cerca di lasciare il tempo a suddetti meccanismi di ritrasmissione per portare a termine la consegna dei dati, senza far intervenire subito l'algoritmo di fast retransmit/fast recovery. Questo contribuisce a limitare inutili e ridondanti ritrasmissioni di pacchetti da parte del mittente quando la perdita è riconducibile ad errore; se invece la perdita fosse dovuta a congestione il risultato sarebbe un dannoso ritardo nell'invocare meccanismi per il suo controllo. Delayed Dupacks è un meccanismo che gode dell'indubbio vantaggio di richiedere modifiche soltanto nel ricevente. D'altra parte però soffre di diverse lacune come quella di non avere meccanismi per la gestione delle disconnessioni o per distinguere le varie tipologie di perdite. Risulta inoltre inadeguato in situazioni di canale debole o comunque con BER molto elevato. Richiede inoltre molta attenzione nella determinazione del ritardo di invio più adeguato per il terzo dupack. Nei casi in cui è applicabile, Delayed Dupacks offre prestazioni compatibili con quelle dello Snoop Protocol dal quale si differenzia però per non richiedere modifiche alla base station, la quale inoltre non ha bisogno di controllare gli header TCP permettendo quindi anche l'utilizzo di trasmissioni criptate.

Il protocollo TCP Aware cerca di operare una distinzione tra le varie tipologie di perdite attraverso la misurazione dei tempi di interarrivo al mittente in modo da invocare i meccanismi per il controllo della congestione solo quando necessario. Tra i punti a favore di TCP Aware vanno ricordati la conservazione della semantica end-to-end e la

capacità di gestire anche traffico criptato. Gli aspetti negativi riguardano invece le situazioni di BER elevato o di variabilità accentuata degli RTT, nelle quali l'algoritmo discriminante delle perdite potrebbe risultare inefficace. Non vi sono inoltre meccanismi per la gestione delle disconnessioni che evitino l'invocazione di slow start. Le modifiche richieste per implementare TCP Aware riguardano unicamente il mittente e servono a consentirgli di operare scelte differenti a seconda del tipologia stimata di perdita.

Un ulteriore protocollo che preserva la semantica end-to-end della connessione e che consente la trasmissione criptata è il Freeze-TCP. Questo protocollo è in grado di interrompere le trasmissioni del mittente, compresi i timeout in modo da evitare inutili restringimenti nella finestra di invio, nel caso in cui si stia per verificare una disconnessione. Ciò avviene grazie alla possibilità, prevista anche dai TCP tradizionali, di inviare al mittente una advertised window uguale a zero. Le modifiche richieste da questo protocollo sono quindi da attuare unicamente nel ricevente. La chiave di questo meccanismo è costituito dalla scelta dell'anticipo con cui il ricevente invia il pacchetto per bloccare il mittente. Valori sbagliati per questo intervallo di tempo impedirebbero a tale pacchetto di raggiungere il mittente, oppure bloccherebbero le trasmissioni troppo presto. Freeze-TCP permette inoltre di riprendere le trasmissioni immediatamente dopo il ripristino della connessione tramite l'invio da parte del ricevente di tre dupack. La possibilità di sospendere l'invio di dati da parte del mittente in situazioni in cui il segnale trasmissivo sia assente, consente al protocollo Freeze-TCP di evitare inutili tentativi di trasmettere dati che continuerebbero ad andare persi. Si può quindi asserire che vi sia una limitazione al problema delle ritrasmissioni ridondanti. Una lacuna che sembra avere Freeze-TCP è quella di non gestire adeguatamente situazioni di BER elevati anche in considerazione del fatto che, seppure in grado di riconoscere una disconnessione, questo protocollo non

distingue le perdite dovute ad errore da quelle dovute a congestione. Pertanto il la segnalazione in Tab. 5.1 che Freeze-TCP effettua una "Distinzione tra congestione, rumore, handoff e disconnessione" è da intendersi come parziale.

Il TCP-Probing è uno tra i protocolli di trasporto che affrontano più problematiche legate all'interazione tra TCP ed ambiente wireless, come si vede dalla Tab. 5.1; si potrebbe pertanto pensare che sia uno dei migliori. Ciò risulta vero solo in parte in quanto le prestazioni complessive ottenute da questo protocollo al confronto con i TCP tradizionali non hanno rivelato miglioramenti significativi o generalizzati. Oltre a questo, TCP-Probing non sembra particolarmente adeguato in situazioni con BER elevati. Si tratta comunque di un protocollo di trasporto che affronta efficacemente il problema delle disconnessioni, soprattutto di quelle di lunghe durata. Ad ogni perdita di pacchetto infatti, TCP-Probing effettua una serie di Probe Cycle per saggiare le condizioni del canale fino a che le condizioni non consentono la ripresa delle trasmissioni. Questa operazione, utilizza dei pacchetti speciali privi di payload e consente di evitare di inviare dati su di un canale in assenza di segnale trasmissivo. In questo modo si risparmiano risorse energetiche e si evitano inutili ritrasmissioni che andrebbero ancora perse. All'uscita dal Probe Cycle, le trasmissioni riprendono a piena velocità senza sprecare tempo. Al termine di un periodo di disconnessione piuttosto ampio, i TCP tradizionali si troverebbero invece con la finestra di invio molto ridotta e gli RTO molto ampi. Il meccanismo del Probe Cycle consente, inoltre, di ripartire da una perdita con una stima del RTT più adeguata di quella che otterrebbero i TCP tradizionali in un ambiente wireless. Ulteriori punti a favore di TCP-Probing riguardano il mantenimento della semantica end-to-end e la capacità di gestione del traffico criptato. Le modifiche richieste dall'implementazione di TCP-Probing riguardano mittente e ricevente.

	TCP Reno	TCP New Reno	TCP Vegas	TCP Sack	I-TCP	M-TCP	Snoop Protocol	Delayed Dupacks	TCP Aware	Freeze TCP	TCP - Probing	WTCP	TCP Westwood
	Problematiche non prestazionali												
Codice ricevente inalterato	√		√		√		√		√				√
Codice mittente inalterato	√				√	√	√	√		√			
Codice nodi intermedi inalterato	√	√	√	√				√	√	√	√	√	√
Gestione del traffico criptato	√	√	√	√				√	√	√	√	√	√
Semantica E2E	√	√	√	√		√	√	√	√	√	√	√	√
	Problematiche prestazionali												
Gestione delle disconnessioni						√				√	√	√	
Limitazione dello spreco di tempo						√				√	√	√	
Gestione BER elevato				√	√	√	√					√	
Distinzione tra congestione, rumore, handoff e disconness.						√			√	√	√	√	√
Limitaz. problema finestra di invio ridotta e incapace di crescere				√	√	√	√	√	√	√	√	√	√
Limitaz. problema difficoltà calcolo RTT – RTO											√	√	
Limitaz. ritrasmiss. ridondanti				√				√		√	√	√	

Tab. 5.1 Risultati conseguiti dai vari protocolli di trasporto.

Un altro protocollo che colpisce per la quantità di problematiche affrontate (vedi Tab. 5.1) è il WTCP. Questo protocollo di trasporto ha l'indubbio pregio di integrare vari meccanismi per affrontare le diverse problematiche derivanti dall'ambiente wireless. Per affrontare efficacemente situazioni di BER, WTCP utilizza i pacchetti di tipo sack. Il controllo del flusso viene attuato utilizzando la misura dei tempi di interarrivo dei pacchetti al ricevente come indicatore della capacità del canale, il quale provvede poi ad informarne il mittente tramite i pacchetti sack. La stima della bandwidth disponibile effettuata da parte del ricevente mette al riparo da problematiche legate ad eventuali asimmetrie del canale trasmissivo [BPK97]. Altra peculiarità di WTCP è costituita dall'eliminazione dei timeout e dei conseguenti slow start tipici dei TCP tradizionali, delegando all'interpretazione dei vari sack ricevuti l'individuazione della perdita di un pacchetto. Viene quindi ridimensionata l'importanza di effettuare una stima corretta di RTT e RTO, come anche quella di distinguere tra le varie tipologie di perdita. Inoltre, l'eliminazione dei timeout e l'utilizzo dei pacchetti sack consentono di limitare il numero totale di pacchetti inviati per un determinato file. Qualora la ricezione di sack venga interrotta per un certo periodo di tempo, WTCP ipotizza sia avvenuta una disconnessione, sospende allora le trasmissioni di dati ed inizia ad inviare piccoli pacchetti di prova in attesa che riprenda il collegamento. Quando le condizioni lo consentono, la trasmissione viene fatta ripartire e la frequenza di invio viene rapidamente impostata su valori corrispondenti alle reali possibilità del canale grazie alla misurazione dei tempi di interarrivo dei pacchetti. Ulteriori pregi di WTCP risiedono nel mantenimento della semantica end-to-end e nell'essere utilizzabile anche con trasmissioni criptate. Per la sua implementazione WTCP richiede modifiche al mittente ed al ricevente. Rimangono comunque alcuni dubbi sulla capacità di sopportazione da parte del ricevente del carico di lavoro computazionale richiesto. Il ruolo di ricevente è infatti solitamente ricoperto dal dispositivo mobile che sappiamo essere generalmente dotato di risorse e capacità limitate.

TCP Westwood è il protocollo più recente tra quelli studiati ed il suo funzionamento si basa sulla stima della bandwidth effettuata dal mittente osservando la frequenza di arrivo degli ack. Si tratta dunque di un protocollo end-to-end la cui implementazione richiede modifiche unicamente al mittente. Diversamente da WTCP inoltre, lo sforzo computazionale per effettuare la stima della bandwidth disponibile è delegato al mittente, ovvero in genere all'host fisso. La bandwidth stimata viene utilizza per impostare cwnd e ssthresh dopo la perdita di un pacchetto. Ciò garantisce una certa funzionalità di distinzione fra i vari tipi di perdite e un limite all'eccessiva riduzione della finestra di invio. TCP Westwood non offre però meccanismi specifici per gestire disconnessioni o situazioni di BER particolarmente elevati. Inoltre, in contesti di canale asimmetrico la stima della bandwidth operata dal mittente grazie alla verifica degli RTT potrebbe risultare falsata. Resta infine da valutare con attenzione la capacità di TCP Westwood di garantire l'equa possibilità di utilizzo del canale a tutte le connessioni, a prescindere dal protocollo di trasporto utilizzato.

Per permetterci di dichiarare come vincitore uno dei protocolli presentati avremmo dovuto disporre di confronti prestazionali con caratteristiche di partenza comuni tra i vari contendenti. Ciò non è purtroppo attualmente possibile e anche se lo fosse scopriremmo probabilmente che talune proposte risultano eccellenti in certe situazioni ma perdono il confronto in altre. Abbiamo dunque proceduto ad analizzare i singoli protocolli e a riportare in maniera uniforme all'interno della Tab. 5.1 le caratteristiche che potevano rendere vantaggioso il loro utilizzo. Emerge da questo studio che i protocolli più interessanti sono quelli che affrontano le problematiche legate all'ambiente wireless sotto molteplici aspetti integrando più schemi. In particolare siamo rimasti colpiti dalla completezza di soluzioni proposta dal WTCP che è riuscito ad affrontare tutte le carenze dei TCP

tradizionali, ottenendo buoni risultati prestazionali e limitando gli aspetti negativi. Rimangono comunque da verificare per questo protocollo fattori quali il carico computazionale richiesto al dispositivo mobile ed il mantenimento dell'equità nel garantire a tutte le connessioni, a prescindere dal protocollo di trasporto utilizzato, le stesse possibilità di accedere al canale.

BIBLIOGRAFIA

[ADLY95] J. S. Ahn, P. Danzig, Z. Liu, L. Yan, "Evaluation of TCP Emulation and Experiment". *Proc. ACM SIGCOMM '95*, pp. 185 – 195, Cambridge, MA, USA, agosto 1995.

[APS99] M. Allman, V. Paxson, W. Stevens, "TCP Congestion Control", IETF RFC 2581, aprile 1999.

[AFP98] M. Allman, S. Floyd, C. Partridge, "Increasing TCP's initial window", IETF RFC 2414, settembre 1998.

[BB95] A. Bakre, B. R. Badrinath, "I-TCP: Indirect TCP for Mobile Hosts", *Proc. 15th IEEE International Conference on Distributed Computing Systems*, Vancouver, British Columbia, pp. 136 - 143, maggio 1995.

[BBJ92] D. Borman, R. Braden, V. Jacobson, "TCP Extensions for High Performance" IETF RFC 1323, maggio 1992

[BKVP97] B. S. Baksi, R. Krishna, N. H. Vaidya, and D. K. Pradhan, "Improving Performance of TCP over Wireless Networks", *Proc. 17th International*

Conference on Distributed Computing Systems, Berkeley, CA, USA, maggio 1997.

[BOP94] L. S. Bradmo, S. W. O'Malley, L. L. Peterson, "TCP Vegas: New Techniques for Congestion Detection and Avoidance". *Proc. ACM SIGCOMM '94*, Vancouver, Canada, pp. 24 – 35, ottobre 1994.

[BP95] L. S. Bradmo, L. L. Peterson, "TCP Vegas: End to End Congestion Avoidance on a Global Internet". *IEEE Journal on Selected Areas in Communications*, vol. 13, n. 8, pp. 1465 – 1480, ottobre 1995.

[BPK97] H. Balakrishnan, V. N. Padmanabhan, R. H. Katz, "The Effects of Asymmetry on TCP Performance", *Proc. ACM Mobicom '97*, settembre 1997.

[BPSK97] H. Balakrishnan, V. N. Padmanabhan, S. Sehan, and R. H. Katz, "A comparison of mechanism for improving TCP performance over wireless links", *IEEE/ACM Trans. Networking*, vol. 5, no. 6, pp. 756 - 769, dicembre 1997.

[BS97] K. Brown, S. Singh, "M-TCP: TCP for Mobile Cellular Networks", *ACM Computer Communication Review*, Vol. 27, n. 5, luglio 1997.

[BSAK95] H. Balakrishnan, S. Seshan, E. Amir, R. H. Katz, "Improving TCP/IP Performance over Wireless

Networks", *Proc. ACM MobiCom '95*, Berkeley, CA, USA, vol. 2, n. 11, pp. 2 - 11, novembre 1995.

[BV98] S. Biaz, N. H. Vaidya, "Distinguishing Congestion Losses from Wireless Transmission Losses: A Negative Result", *Proc. IEEE 7th International Conference on Computer Communications and Networks,* ottobre 1998.

[BV99] S. Biaz, N. H. Vaidya, "Discriminating Congestion Losses from Wireless Losses using Inter-Arrival Times at the Receiver", *Proc. IEEE Symposium ASSET '99*, Dallas, Texas, USA, marzo 1999.

[CGLMS00] C. Casetti, M. Gerla, S. S. Lee, S. Mascolo, M. Sanadidi, "TCP with Faster Recovery", MILCOM 2000, ottobre 2000.

[CI94] R. Càceres, L. Iftode, "The Effects of Mobility on Reliable Transport Protocols", *14th International Conference on Distributed Computing Systems*, pp. 12 - 20, giugno 1994.

[CI95] R. Càceres, L. Iftode, "Improving the Performance of Reliable Transport Protocols in Mobile Computing Environments", *IEEE Journal on Selected Areas in Communications*, vol. 13, n. 5, pp. 850 - 857, giugno 1995.

[Cla88] D. Clark, “The Design Philosophy of the DARPA Internet Protocols”, *ACM SIGCOMM ‘88*, agosto 1988.

[CFSD90] J. Case, M. Fedor, M. Schoffstall, J.Davin, “A Simple Network Management Protocol (SNMP)”, IETF RFC 1157, maggio 1990

[CZ99] A. Chockalingam, M. Zorzi, “Wireless TCP Performance with Link Layer FEC/ARQ”, *IEEE International Conference on Communications ‘99*, pp. 1212 – 1216, vol. 2, giugno 1999.

[Dee89] S. Deering, “Hot Extensions for IP Multicasting”, IETF RFC 1112, agosto 1989.

[Ewe01] A. Ewerlid, "Reliable communication over wireless links," *Proc. Nordic Radio Symposium*, Saltsjobaden, Svezia, aprile 2001.

[FDC84] S. Floyd, T. Henderson, “A Thinwire Protocol for connecting personal computers to the Internet”, IETF RFC 914, settembre 1984.

[FF95] K. Fall, S, Floyd, “Comparisons of Tahoe, Reno, and Sack TCP”, ftp://ftp.ee.lbl.gov/papers/sacks_v0pdf, ottobre 1994.

[FF96] K. Fall, S, Floyd, "Simulation-based Comparisons of Tahoe, Reno, and Sack TCP", *ACM Computer Communication Review*, luglio 1996

[FH99] S. Floyd, T. Henderson, "The NewReno Modification to TCP's Fast Recovery Algorithm", IETF RFC 2582, aprile 1999.

[Flo94] S. Floyd, "TCP and Successive Fast Retransmits" ftp://ftp.ee.lbl.gov/papers/fastretrans.ps, ottobre 1994.

[Fen97] W. Fenner, "Internet Group Management Protocol, Version 2", IETF RFC 2236, novembre 1997.

[FMMP00] S. Floyd, J. Mahdavi, M. Mathis, M. Podolsky, "An Extension to the Selective Acknowledgement (SACK) Option for TCP", IETF RFC 2883, luglio 2000.

[GLMS02] M. Guizani, W. W. Lu, P. Meche, M. Sawahashi, "Wideband Wireless Access Technologies to Broadband Internet", *IEEE Communications Magazine*, aprile 2002.

[GMPG00] T. Goff, J. Moronski, D. Phatak, V. Gupta, "Freeze-TCP: A true end-to-end Enhancement Mechanism for Mobile Environments", *IEEE INFOCOM 2000*, marzo 2000.

[Hus00a] G. Huston, “TCP Performance”, *The Internet Protocol Journal*, vol. 3, n. 2, giugno 2000.

[Hus00b] G. Huston, “The future for TCP”, *The Internet Protocol Journal*, vol. 3, n. 3, settembre 2000.

[Hus01] G. Huston, “TCP in a Wireless World”, *IEEE Internet Computing*, pp. 82 – 84, marzo – aprile 2001.

[Jac90] V. Jacobson, “Compressing TCP/IP Headers for Low-Speed Serial Links”, IETF RFC 1144, febbraio 1990.

[JB88] V. Jacobson, R. Braden, “TCP Extensions for Long-Delay Paths”, IETF RFC 1072, ottobre 1988.

[JBB92] V. Jacobson, R. Braden, D. Borman, “TCP Extensions for High Performance“, IETF RFC 1323, maggio 1992.

[JK88] V. Jacobson, M. J. Karrels, “Congestion Avoidance and Control”, *ACM SIGCOMM ’88*, novembre 1988.

[KB87] P. Karn, C. Partridge, “Improving Round-Trip Time Estimates in Reliable Transport Protocols”, *Computer Communication Review*, vol. 17, n. 5, pp. 3 – 7, agosto 1987.

[KM87] C. A. Kent, J. C. Mogul, “Fragmentation Considered Harmful”, *ACM Computer Communication Review*, vol. 17, n. 5, pp. 390 – 401, agosto 1987.

[KMM00] R. Kalden, I. Meirick, M. Meyer, “Wireless Internet Access Based on GPRS”, *IEEE Personal Communications*, pp. 8 – 18, aprile 2000.

[MB97] J. Mysore, V. Bharghavan, “A New Multicasting-based Architecture for Internet Host Mobility”, *ACM/IEEE MOBICOM ’97*, pp. 161 – 172, settembre 1997.

[MCGSW01] S. Mascolo, C. Casetti, M. Gerla, M. Y. Sanadidi, R. Wang, "TCP Westwood: Bandwidth Estimation for Enhanced Transport over Wireless Links", *MOBICOM 2001*, luglio 2001.

[MGPGC02] S. Mascolo, A. Grieco, G. Pau, M. Gerla, C. Casetti, “End-to-End Bandwidth Estimation in TCP to Improve Wireless Link Utilization, *European Wireless Conference*, Firenze, Italia, febbraio 2002.

[MMFR96] M. Mathis, J. Mahdavi, S. Floyd, A. Romanov, “TCP Selective Acknowledgment Options”, IETF RFC 2018, ottobre 1996.

[MSMO97] M. Mathis, J. Semke, J. Mahdavi, T. Ott, "The macroscopic behaviour of the TCP congestion

avoidance algorithm", *ACM Computer Communication Review*, vol. 27, n. 3, pp. 67 - 82, luglio 1997.

[PA00] V. Paxson, M. Allman, "Computing TCP's retransmission timer", IETF RFC 2988, novembre 2000.

[PD00] L. L. Peterson, B. Davie, *Computer Networks A System Approach*, second edition, Morgan Kaufmann Publishers, San Francisco, CA, USA, 2000.

[PN98] K. Poduri, K. Nichols, "Simulation studies of increased initial TCP window size" IETF RFC 2415, settembre 1998.

[Pos80] J. Postel, "User Datagram Protocol", IETF RFC 768, agosto 1980.

[Pos81a] J. Postel, "Internet Protocol", IETF RFC 791, settembre 1981.

[Pos81b] J. Postel, "Internet Control Message Protocol", IETF RFC 792, settembre 1981.

[Pos81c] J. Postel, "Transmission Control Protocol", IETF RFC 793, settembre 1981.

[Pos82] J. Postel, "Simple Mail Transfer Protocol", IETF RFC 721, agosto 1982.

[PR83] J. Postel, J. Reynolds, “Telnet Protocol Specification”, IETF RFC 854, maggio 1983.

[PR85] J. Postel, J. Reynolds, “File Transfer Protocol (FTP)”, IETF RFC 959, ottobre 1985.

[RF99] K. Ramakrishnan, S. Floyd, “A proposal to add Explicit Congestion Notification (ECN) to IP”, IETF RFC 2481, gennaio 1999.

[SB98] M. Stangel, V. Bharghavan, “Improving TCP Performance in Mobile Computing Environments”, *IEEE International Conference on Communications ’98*, giugno 1998.

[SNVS99] P. Sinha, N. Venkitaraman, R. Sivakumar, V. Bharghavan, "WTCP: A Reliable Transport Protocol for Wireless Wide-Area Networks", *ACM Mobicom '99*, agosto 1999.

[Ste94] W. R. Stevens, *TCP/IP Illustrated, Volume 1: The Protocols*, Addison Wesley, 1994.

[Ste97] W. Stevens, “TCP Slow Start, Congestion Avoidance, Fast Retransmit, and Fast Recovery Algorithms”, IETF RFC 2001, gennaio 1997.

[TPL99] C. Tan, S. Pink, K. Lye, “A Fast Handoff Scheme for Wireless Networks”, *Proc. 2th ACM International Workshop on Wireless Mobile Multimedia*, agosto 1999.

[TB00] V. Tsaoussidis, H. Badr, "TCP-Probing: Towards an Error Control Schema with Energy and Throughput Performance Gains", *The 8th IEEE Conference on Network Protocols*, novembre 2000.

[VMPM99] N. H. Vaidya, M. Mehta, C. Perkins, G. Montenegro, "Delayed Duplicate Acknowledgements: A TCP-Unaware Approach to Improve Performance of TCP over Wireless" Technical Report 99-003, Computer Science Department, Texas A&M University, febbraio 1999.

Luglio 2002

Questa tesi è la conclusione di un viaggio cominciato anni fa,
è il prodotto di una persona completamente diversa da quella che partì,
è il risultato delle vite di tante persone che si sono intrecciate alla sua.

A Ettore per l'iscrizione, a Fabione per la libertà mentale, a Tamara per l'affetto, a Fax per il basket, ad Andrea per la generosità, a Sonia per i gavettoni, al Moro per le costolette prima di un Consiglio, a Benellone per la coscienza, a Maria Grazia per i ricordi, ai bottegari e alle vismarose per l'amicizia, a Erika per il nastro sulle lettere, a Mascia per gli addominali, a Valentina per la pizza, a Jesù per la fantasia, a giorg per quanta ne sa!, a Papòscl per il pas de chat, a Vico perché non ha vissuto gratis, ad Antonio e a Jattone perché i terroni sono gente meravigliosa, a Carla per gli abbracci, ai Polci Bros per la LAN domestica, a Ghea perché farebbe lo stesso per me, a Silvia perché si prende cura dei gemellini da sola, a Simona perché è la mia pellegrina, a Ciccio, Luciano e Silvia perché mi sento un po' di famiglia anch'io, a Michela per il francese, a Cesare, Angelo, Boschi, Fullo, Bruce, Carruzzé, Sara, Micky, insomma alla S.P.R.I.Te. intera perché il volontariato, è una delle più belle invenzioni del mondo, a Stefano, Ale, John e Manuel per il coraggio, a Giò per le sfide, a Vic per la nobiltà d'espressione, ai docenti e al personale di questo corso di laurea per avermi sopportato, al mio relatore per i suggerimenti, a Romina, Luana e Diletta per l'ospitalità, a Vanessa per la luce del mattino, ai miei familiari per il sostegno, ai miei nipotini Sofia, Giacomo, Elena perché sono il futuro e a mio fratellino Matteo perché è il mio orgoglio più grande…

Grazie.

Questa tesi è la conclusione di un viaggio cominciato anni fa,
questa tesi è l'inizio di un viaggio che comincia ora…