Università di Genova
Facoltà di Ingegneria

Telematica 3
5. TCP/IP - UDP/TCP

Prof. Raffaele Bolla

Recupero di errore

- Alcuni protocolli di trasporto (TCP) applicano tecniche di recupero dell'errore con ritrasmissione, che in questo caso operano *end-to-end*.
- Queste tecniche sono applicate principalmente a livello di linea, fra due nodi adiacenti (scelta efficace se le linee sono soggette a tassi di errori significativi).
- Vediamo alcune delle tecniche più diffuse, considerando per il momento il contesto punto-punto del livello di linea.

5.2

Protocolli di trasporto

- Si consideri per prima una rete affidabile, ossia che consegni i pacchetti in ordine e senza errori (per esempio grazie ad un livello di linea affidabile che realizzi il recupero dell'errore).
- I compiti che devono essere svolti dal livello di trasporto in questo caso sono:
  - Indirizzamento
  - Multiplexing
  - Controllo di flusso
  - Apertura e chiusura delle connessioni

5.3

Protocolli di Trasporto
Indirizzamento

- Per individuare l'entità di applicazione a cui inviare l'informazione, l'entità di trasporto (ET) di sorgente ha bisogno:
  - Identificatore dell'utente (porta/SAP)
  - Identificatore entità di trasporto (di destinazione)
  - Indirizzo dell'host (rete)
- In genere l'utente è individuato dalla coppia (porta, host) di cui la porta viene inserita nell'intestazione dell'ET, mentre l'indirizzo di host viene passato al livello di rete.

5.4

Protocolli di Trasporto
Multiplexing

- Il livello di trasporto può operare un multiplexing diretto o inverso nei confronti del livello di rete:
  - Diretto: inviando più flussi contemporaneamente su un unico servizio del livello di rete (per diminuire l'*overhead*)
  - Inverso: suddividendo più flussi su servizi del livello di rete (per migliorare le prestazioni).

5.5

Protocolli di Trasporto
Controllo di flusso

- Controllo di flusso significa regolare l'immissione di dati nella rete da parte della sorgente.
- La ragione per la quale può essere necessaria questa azione è (in questo caso):
  - L'utente dell'ET ricevente non è in grado di accettare il flusso di dati attuale
  - L'entità stessa di trasporto non è in grado di accettare il flusso (per es. buffer esauriti)
- La situazione è diversa da quella presente a livello di linea perché il ritardo di andata e ritorno (*Round Trip Delay*) è
  - Molto più lungo
  - Potenzialmente molto variabile

5.6

Protocolli di Trasporto

Controllo di flusso

- Quattro possibili strategie:
  - Non fare nulla (scartare i pacchetti)
  - Rifiutare ulteriori pacchetti dal livello di rete
  - Usare un protocollo a *sliding window* (finestra scivolante) fissa
  - Usare un meccanismo a credito
- Il secondo caso significa demandare il controllo di flusso al livello di rete o comunque ai livelli inferiori

5.7

Protocolli di Trasporto

Controllo di flusso - *Sliding Window*

- Il controllo di flusso *sliding window* è in sostanza basato sul mancato invio delle conferme.
- Il trasmettitore non conferma l'ultimo (o gli ultimi) pacchetto arrivato che gli satura il buffer
- Siccome la rete è affidabile, il trasmettitore può interpretare il mancato invio di una conferma come indicazione di eccessivo invio e quindi (sapendo che i pacchetti devono essere arrivati) non ritrasmetterli fino a che non riceve le conferme mancanti.
- Non funziona correttamente in caso di rete non affidabile (non è in grado di distinguere fra perdite reali e indicazioni di rallentare).

5.8

Protocolli di Trasporto

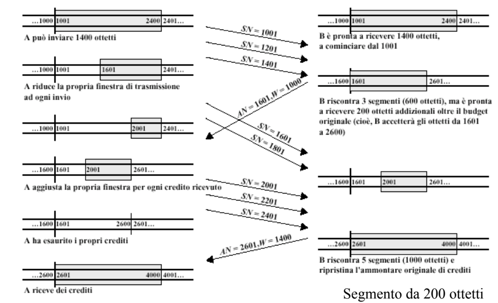
Controllo di flusso - Meccanismi a credito

- Separano la conferma dalla dimensione della finestra.
- Quando il trasporto invia dati inserisce un SN (*Sequence Number*) che indica il numero del primo ottetto inviato
- Ogni entità ricevente fornisce una conferma con le seguenti informazioni
  - AN (*Ack Number*) che indica che tutti gli ottetti con SN = AN-1 sono arrivati correttamente
  - W che indica che è possibile trasmettere ancora W ottetti, quelli da AN a AN+W-1.

5.9

Protocolli di Trasporto

Controllo di flusso - Meccanismi a credito



Segmento da 200 ottetti

SN = Serial Number; AN = Acknowledge Number; W = dimensione della finestra

5.10

Protocolli di Trasporto

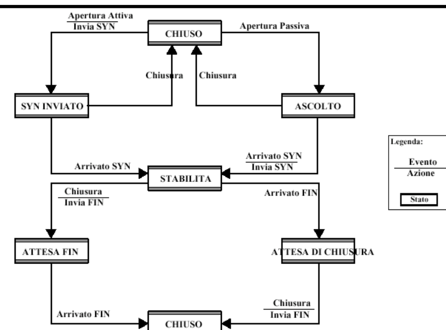
Apertura e chiusura della connessione

- L'apertura della connessione serve a tre scopi:
  - Assicurarsi che la destinazione esiste ed è consenziente alla comunicazione;
  - Permettere la negoziazione di parametri;
  - Attivare l'allocazione di risorse (spazio nei buffer);

5.11

Protocolli di Trasporto

Apertura e chiusura della connessione



5.12

Protocolli di Trasporto
Rete non affidabile

- Nel caso la rete non sia affidabile, ossia possano avvenire perdite/errori ed i pacchetti possano arrivare fuori sequenza, allora bisogna affrontare o riaffrontare i seguenti aspetti:
  - Consegna ordinata (numeri di sequenza)
  - Strategia di ritrasmissione
  - Identificazione dei duplicati
  - Controllo di flusso
  - Apertura della connessione
  - Chiusura della connessione

5.13

Protocolli di Trasporto
Strategia di ritrasmissione

- Si possono utilizzare i sistemi ARQ.
- Il problema principale e stimare la durata del *timeout*, ossia la durata del ***Round Trip Time*** (RTT, tempo di andata e ritorno).
- In genere si tenta di usare una stima adattativa di tale parametro, per esempio facendo la media dei ritardi con cui giungono le conferme; questo modo di procedere può non funzionare perché:
  - Il ricevitore potrebbe non inviare l'Ack immediatamente
  - Se il segmento è stato ritrasmesso, il ricevitore non può sapere se l'Ack è riferito alla prima o alla seconda trasmissione
  - Le condizioni delle rete possono cambiare molto velocemente

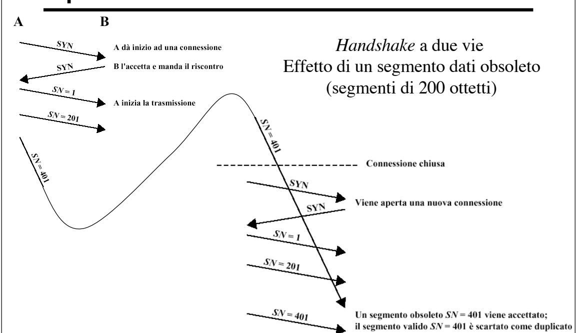
5.14

Protocolli di Trasporto
Controllo di flusso

- Il controllo di flusso a credito funziona bene anche nelle reti non affidabili.
- Anche se va persa una conferma che stabilisce un nuovo valore di finestra più piccolo, il risultato sarà che il trasmettitore ri-invierà dei pacchetti e le nuove conferme ripristineranno la situazione giusta.
- L'unica situazione di *deadlock* può avvenire se fra due stazioni A e B, B invia (AckNumber =AN = i, Window = W = 0) e poco dopo (AN = i, W = j) ma quest'ultimo pacchetto va perso. In questo caso A non invia più nulla perché ha la finestra chiusa e B neanche perché attende nuovi pacchetti. Soluzione: Timer legato alla finestra.

5.15

Protocolli di Trasporto
Apertura della connessione



5.16

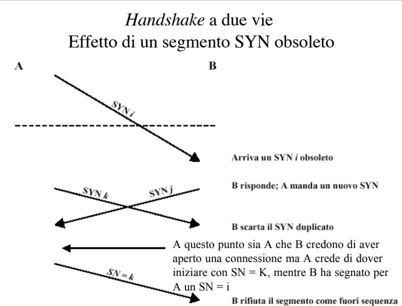
Protocolli di Trasporto
Apertura della connessione

- Per evitare che pacchetti di connessioni già chiuse confondano quella attiva si può, ad esempio, usare un numero iniziale di sequenza diverso per connessione (supponendo che lo spazio dei numeri di sequenza sia grande)
- Quindi, in pacchetto di apertura SYN trasporterà anche il numero di sequenza (SYN i).

5.17

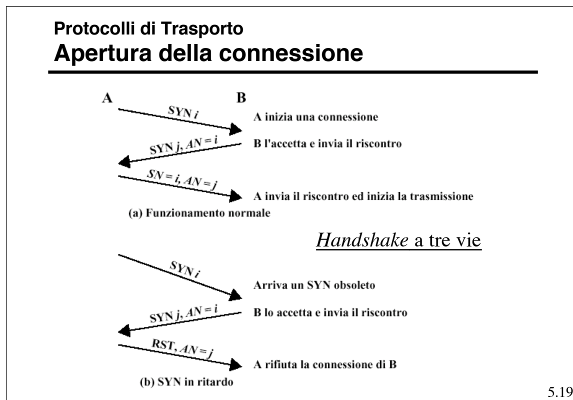
Protocolli di Trasporto
Apertura della connessione

Handshake a due vie
Effetto di un segmento SYN obsoleto



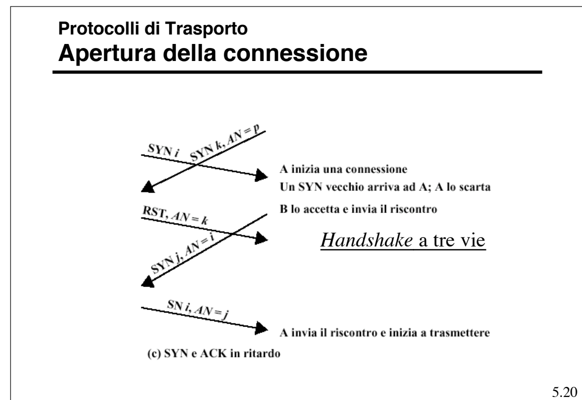
5.18

Protocolli di Trasporto
Apertura della connessione



5.19

Protocolli di Trasporto
Apertura della connessione



5.20

Protocolli di trasporto nel TCP/IP

- La suite TCP/IP propone due diversi protocolli di trasporto, che hanno delle caratteristiche opposte:
  - ***User Datagram Protocol*** (**UDP**) semplicissimo, che non fornisce quasi nessuna funzionalità (è circa come usare l'IP direttamente)
  - ***Transmission Control Protocol*** (**TCP**) che invece fornisce tutte le possibili funzionalità previste a livello di trasporto

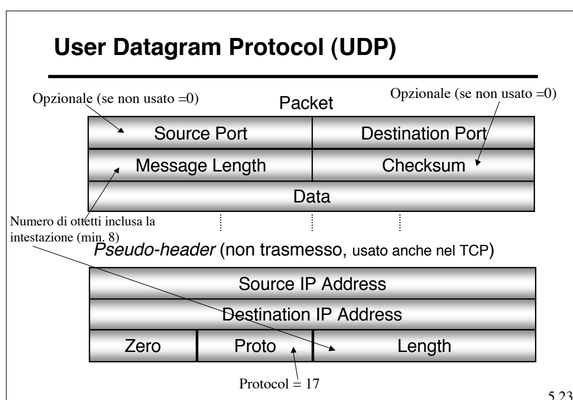
5.21

User Datagram Protocol (UDP)

- UDP realizza un protocollo di trasporto non orientato alla connessione e non affidabile (senza recupero d'errore).
- Aggiunge alle funzionalità di IP la possibilità di distinguere fra diverse destinazioni all'interno della stessa macchina, ma non garantisce l'integrità dei dati e consegna ordinata dei dati.
- E' identificato dal *Protocol Number* 17 nell'*header* IP.
- In sostanza permette solo di identificare le applicazioni sorgenti e destinazioni tramite un ***port number***.

5.22

User Datagram Protocol (UDP)



5.23

User Datagram Protocol (UDP)

- I numeri di porta sono di due tipi:
  - Preassegnati (*well-known*) in modo univoco a certi servizi, e sono i numeri più bassi;
  - Assegnati dinamicamente, tutti gli altri.
- Alcuni dei preassegnati sono:
  - 0 ECHO *echo*
  - 11 USERS Utenti attivi
  - 37 TIME Data e ora
  - 42 NAMESERVER *Domain name server*
  - 69 TFTP *Trivial File Transport*
  - 520 RIP *Routing Internet Prot.*

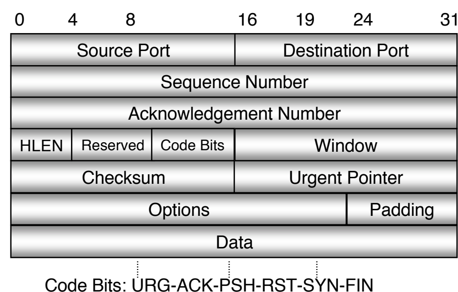
5.24

Transmission Control Protocol (TCP)

- Il TCP è definito nel RFC 793, ed ha le seguenti caratteristiche principali:
  - È orientato alla connessione.
  - Full-duplex.
  - Unicast.
  - Trasporta flussi di dati.
  - Realizza un trasmissione affidabile (recupero d'errore e consegna ordinata) su un servizio di rete datagram non affidabile (IP).
  - Applica un controllo di flusso/congestione adattativo.

5.25

Transmission Control Protocol (TCP)



5.26

Transmission Control Protocol (TCP)

- TCP usa soltanto un singolo tipo di unità dati di protocollo, chiamato segmento TCP. L'intestazione è piuttosto lunga (minima 20 ottetti), visto che è unica per tutti i meccanismi del protocollo.
  - *Source Port* (porta di sorgente) **(16 bit):** Utente TCP sorgente
  - *Destination Port* (porta di destinazione) **(16 bit):** Utente TCP destinazione
  - *Sequence Number* (numero di sequenza) **(32 bit):** Numero di sequenza del primo ottetto dati in questo segmento, tranne quando il flag SYN è attivo. Se il flag SYN è attivo, contiene L'*Initial Sequence Number* (ISN, numero di sequenza iniziale) ed il primo ottetto di dati è ISN +1.

5.27

Transmission Control Protocol (TCP)

- *Acknowledgement Number* (numero di riscontro) **(32 bit):** Un riscontro *piggybacked* (inserito in un invio di dati in senso opposto). Contiene il numero di sequenza dell'ottetto dati successivo che l'entità TCP si aspetta di ricevere.
- *Data Offset* (offset dei dati) **(4 bit):** Numero di parole di 32 bit nell'intestazione.
- **Flag (6 bit):**
  - » URG: Campo *Urgent Pointer* significativo.
  - » ACK: Campo Acknowledgement Number significativo.
  - » PSH: Funzione push.
  - » RST: Re-inizializza la connessione.
  - » SYN: Sincronizza i numeri di sequenza.
  - » FIN: Nessun dato in più dal trasmettitore.

5.28

Transmission Control Protocol (TCP)

- *Window* (finestra) **(16 bit):** Allocazione di crediti del controllo di flusso, in ottetti. Contiene il numero di ottetti dati, a partire da quello indicato nel campo di Acknowledgement, che il trasmettitore è disposto ad accettare.
- **Checksum (16 bit):** Complemento ad uno della somma modulo 2^{16} – 1 di tutte le parole a 16 bit nel segmento più una pseudo-intestazione
- *Urgent Pointer* (puntatore d'urgenza) **(16 bit):** Indica l'ultimo ottetto nella sequenza di dati di tipo *urgent*.
- *Option* (opzioni) **(variabile):** Ad esempio:
  - » Dim max del segmento (MSS, Maximum Segment Size)
  - » **Window- scale**
  - » **SACK (*Selective* Ack).**

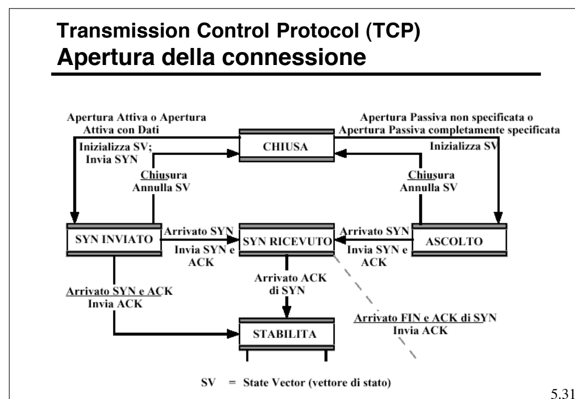
5.29

Transmission Control Protocol (TCP)

- Apertura/chiusura della connessione
  - Utilizza un meccanismo di *handshake* a tre vie, la connessione è individuata dalla coppia porta di destinazione di sorgente. Quindi può esserci una solo connessione aperta per coppia. La chiusura può anche essere "brutale" tramite la primitiva ABORT.
- Trasferimento dei dati
  - Avviene numerando gli ottetti inviati, sono previste due modalità aggiuntive
    - » **Flusso dati *push*:** TCP decide autonomamente quando sono stati accumulati un numero sufficiente di dati per formare un segmento da trasmettere. L'utente può richiedere che il TCP trasmetta tutti i dati in sospeso fino a, o inclusi quelli, etichettati con un *push flag*. Dal lato del ricevitore, il TCP inoltra questi dati all'utente in modo analogo.
    - » **Flusso dati *urgent*:** Fornisce uno strumento per informare l'utente TCP di destinazione che dei dati significativi o "urgenti" si trovano nel flusso dati in arrivo. Dipende dall'utente di destinazione determinare come comportarsi in presenza di questo tipo di dati.

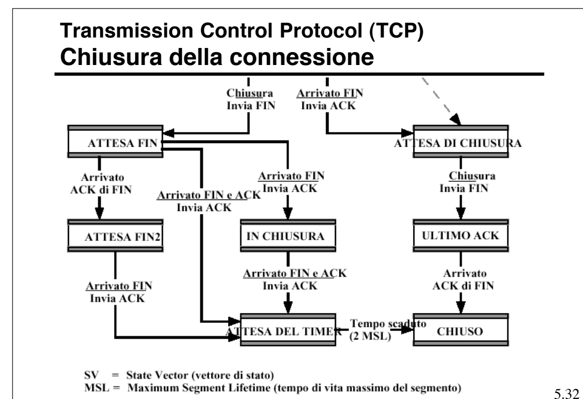
5.30

Transmission Control Protocol (TCP)
Apertura della connessione



5.31

Transmission Control Protocol (TCP)
Chiusura della connessione



5.32

Transmission Control Protocol (TCP)
Realizzazioni

- Il protocollo lascia alcune scelte sostanzialmente libere, ossia permette diverse realizzazioni tra loro comunque interoperabili. Questo in particolare per quanto concerne
  - Strategia di trasmissione (immediato o ritardato)
  - Strategia di inoltro
  - Strategia di accettazione (in ordine, o nella finestra (invia Ack solo per dati in ordine))
  - Strategia di ritrasmissione
  - Strategia di riscontro (singolo o cumulativo) per i bit arrivati nell'ordine corretto

5.33

Transmission Control Protocol (TCP)
Strategia di ritrasmissioni

- Il TCP mantiene una coda di segmenti già trasmessi ma non riscontrati. La specifica TCP stabilisce che si ritrasmetta un segmento se non si riceve un riscontro entro un certo lasso di tempo. Una realizzazione di TCP può impiegare una di tre diverse strategie di ritrasmissione.
  - **Solo-il-primo**
  - **A lotto**
  - **Individuale**

5.34

Transmission Control Protocol (TCP)
Strategia di ritrasmissioni

- **Solo-il-primo:**
  - » Mantiene un solo timer di ritrasmissione per l'intera coda.
  - » Se riceve un riscontro, rimuove il relativo segmento o segmenti dalla coda ed ri-inizializza il timer.
  - » Se il tempo si esaurisce, ritrasmette il segmento in cima alla coda e ri-inizializza il timer.
  - E' efficiente in termini di traffico generato, in quanto vengono ritrasmessi solo i segmenti persi (o i segmenti il cui ACK è stato perso).
  - Tuttavia, poiché il timer nel secondo segmento della coda non viene attivato fino a quando non arriva il riscontro del primo segmento, si possono verificare ritardi considerevoli.

5.35

Transmission Control Protocol (TCP)
Strategia di ritrasmissioni

- **A lotto:**
  - » Mantiene un solo timer di ritrasmissione per l'intera coda.
  - » Se riceve un riscontro, rimuove il relativo segmento o segmenti dalla coda e ri-inizializza il timer.
  - » Se il tempo si esaurisce, ritrasmette tutti i segmenti nella coda e ri-inizializza il timer.
  - Riduce i ritardi, ma può dare origine a ritrasmissioni inutili (va bene per implementare un GO-BACK-N)
- **Individuale:**
  - » Mantiene un timer differente per ogni segmento nella coda
  - » Se riceve un riscontro, rimuove il relativo segmento o segmenti dalla coda ed elimina il timer o i timer corrispondenti.
  - » Se il tempo di un qualunque timer si esaurisce, ritrasmette il segmento corrispondente e rinizializza quel timer.
  - Ottimale, ma più complesso. Adatto, oltre che per una strategia GO-BACK-N, anche per il *Selective Repeat* (Sack).

5.36

Transmission Control Protocol (TCP)

Controllo di congestione

- Il meccanismo TCP di controllo di flusso basato sui crediti è stato esteso per realizzare anche un controllo di congestione sorgente -destinazione.
- La congestione ha due effetti principali:
  - come inizia la congestione, il tempo di transito lungo la rete aumenta.
  - come la congestione diventa pesante, vengono persi pacchetti o segmenti
- Il meccanismo di controllo di flusso del TCP può servire per riconoscere l'inizio della congestione (osservando l'incremento dei ritardi e dei segmenti persi) e per reagire riducendo il flusso di dati.

5.37

Transmission Control Protocol (TCP)

Controllo di congestione

- La prima cosa da fare è stimare il RRT (*round-trip-time*)
  - Media semplice

Segmento i-esimo

$$\mathrm{ARTT}(K+1) = \frac{1}{K+1}\sum_{i=1}^{K+1}\mathrm{RTT}(i)$$

RRT medio (*Average*)

$$\mathrm{ARTT}(K+1) = \frac{K}{K+1}\mathrm{ARTT}(K) + \frac{1}{K+1}\mathrm{RTT}(K+1)$$

Si noti che ad ogni termine nella sommatoria viene dato lo stesso peso; ovvero, ogni termine è moltiplicato per la stessa costante $1/(K+1)$.

5.38

Transmission Control Protocol (TCP)

Controllo di congestione

- Media esponenziale: sarebbe meglio dar maggior peso ai valori più recenti, in quanto rappresentano con più probabilità il comportamento futuro. Una tecnica comune di predizione, basata su un certo numero di valori passati, è la media esponenziale:

$$\mathrm{SRTT}(K+1) = \alpha \times \mathrm{SRTT}(K) + (1-\alpha) \times \mathrm{RTT}(K+1)$$

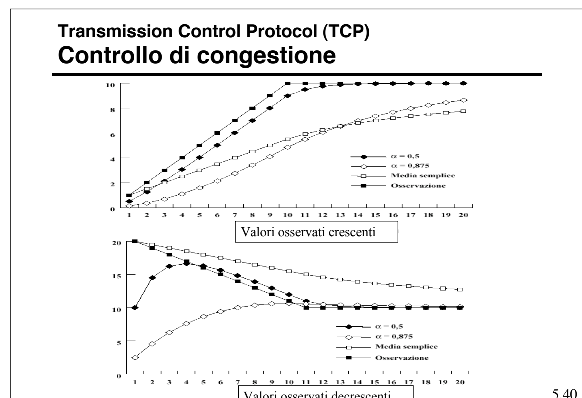
oppure

$$\mathrm{SRTT}(K+1) = (1-\alpha) \times \mathrm{RTT}(K+1) + \alpha(1-\alpha) \times \mathrm{RTT}(K) + \alpha^2(1-\alpha) \times \mathrm{RTT}(K-1) + \ldots + \alpha^K(1-\alpha) \times \mathrm{RTT}(1)$$

5.39

Transmission Control Protocol (TCP)

Controllo di congestione



5.40

Transmission Control Protocol (TCP)

Controllo di congestione

- Il timeout (***Retrasmission TimeOut***, RTO) può essere calcolato come:

Costante additiva

$$\mathrm{RTO}(K+1) = \mathrm{SRTT}(K+1) + \Delta$$

- In questo caso però Δ non è legato a SRTT, quindi una formulazione più opportuna (usata nella versione originale del TCP) potrebbe essere

$$\mathrm{RTO}(K+1) = \mathrm{MIN}(\mathrm{UBOUND}, \mathrm{MAX}(\mathrm{LBOUND}, \beta \times \mathrm{SRTT}(K+1)))$$

>1

5.41

Transmission Control Protocol (TCP)

Controllo di congestione

- La varianza nella rete può essere elevata, ma specialmente può variare nel tempo.
- Quando la varianza è bassa, un valore alto di β sovrastima RTO e quindi in caso di pacchetto perso si attende più tempo del dovuto
- Ma quando la varianza è alta il valore β = 2 può non essere comunque sufficiente ad evitare ritrasmissioni inutili.
- Per cui è stato proposta una stima della varianza dell'RTT (algoritmo di Jacobson)

5.42

Transmission Control Protocol (TCP)

Controllo di congestione

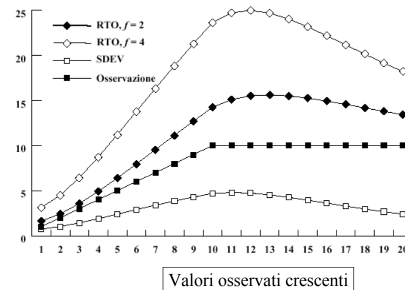
- La stima della deviazione standard viene fatta in modo analogo a quello della media:

$$SRTT(K+1) = (1-g) \times SRTT(K) + g \times RTT(K+1) \qquad g = 1/8$$
$$SERR(K+1) = RTT(K+1) - SRTT(K)$$
$$SDEV(K+1) = (1-h) \times SDEV(K) + h \times |SERR(K+1)| \qquad h = 1/4$$
$$RTO(K+1) = SRTT(K+1) + f \times SDEV(K+1) \qquad f = 4$$

5.43

Transmission Control Protocol (TCP)

Controllo di congestione

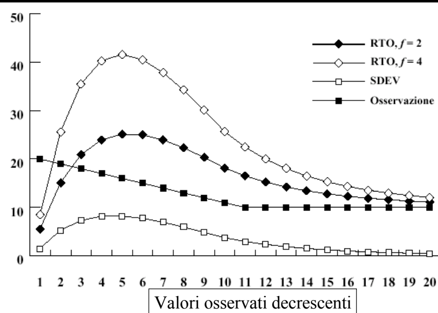


Valori osservati crescenti

5.44

Transmission Control Protocol (TCP)

Controllo di congestione



Valori osservati decrescenti

5.45

Transmission Control Protocol (TCP)

Controllo di congestione

- Bisogna considerare altri due fattori:
  - Quale valore RTO si deve usare per un segmento ritrasmesso?
    - » A questo scopo viene usato un algoritmo di *backoff* esponenziale dell'RTO.
  - Quali campioni di tempo di andata e ritorno si dovrebbe usare come ingresso all'algoritmo di Jacobson (in relazione a segm. ritrasmessi) ?
    - » L'algoritmo di Karn determina quali campioni usare.

5.46

Transmission Control Protocol (TCP)

Controllo di congestione

- Una prima ipotesi è usare il RTO precedente.
- Una politica più ragionevole suggerisce che una sorgente TCP aumenti il proprio RTO ogni volta che un segmento viene ritrasmesso (perché ipotizza congestione); questo meccanismo viene chiamato **algoritmo di *backoff***. In particolare, ad ogni ritrasmissione viene posto

$$RTO = q \times RTO$$

- Il valore più frequentemente usato per q è 2, da cui il nome associato a questa tecnica di *backoff* esponenziale binario (come nel CSMA/CD)

5.47

Transmission Control Protocol (TCP)

Controllo di congestione

- Quando arriva un ACK di un segmento ritrasmesso, il trasmettitore può interpretarlo in due modi:
  - Si tratta dell'ACK relativo alla prima trasmissione del segmento. In questo caso, l'RTT è semplicemente più lungo del previsto ma riflette l'effettive condizioni di rete.
  - Si tratta l'ACK relativo alla seconda ritrasmissione.
- Non potendo distiguere le due situazioni si è scelto di ignorare l'informazione.

5.48

Transmission Control Protocol (TCP)

Controllo di congestione

- Quindi **l'algoritmo di Karn** aggiunge le seguenti regole:
  - Non usare l'RTT, misurato su di un segmento ritrasmesso, per aggiornare SRTT e SDEV.
  - Quando ha luogo una ritrasmissione, calcolare l'RTO usando la procedura di *backoff*.
  - Continuare ad usare la procedura di *backoff* per il calcolo dell'RTO nei successivi segmenti, fino a che non arriva un riscontro di un segmento che non sia stato ritrasmesso.
  - Quando viene ricevuto un riscontro di un segmento non ritrasmesso, l'algoritmo di Jacobson è di nuovo attivato per il calcolo dei futuri valori di RTO.

5.49

Transmission Control Protocol (TCP)

Controllo di congestione

- Gestione della finestra
  - Il TCP usa un meccanismo a credito per gestire la finestra di trasmissione
  - Le modalità di crescita e decrescita della finestra vengono però controllate opportunamente in relazione alla congestione
  - In particolare si ha che
    awnd = MIN[*credito*, *cwnd*]
  - Dove
    - » ***awnd*** = finestra concessa, in segmenti.
    - » ***credito*** = la quantità di credito accordato nel riscontro più recente, in segmenti
    - » ***cwnd*** = finestra di congestione, in segmenti

5.50

Controllo di congestione (TCP)

Tahoe

- In fase di apertura della connessione, la scelta della finestra e critica, perché non si hanno informazioni sullo stato della rete.
- Allora si attua lo ***slow start*** (partenza lenta)
  - inizializza *cwnd* = 1 segmento
  - Per ogni riscontro ricevuto si aumenta *cwnd* di uno
- Se tutti i segmenti vengono riscontrati la finestra raddoppia ogni RTT circa e quindi n realtà la crescita è esponenziale

5.51

Controllo di congestione (TCP)

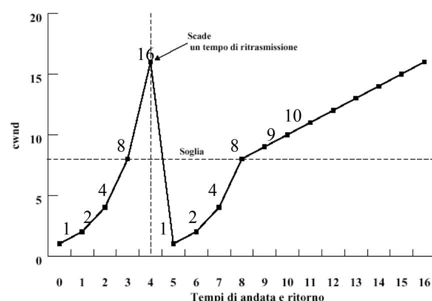
Tahoe

- Come reagire in caso di congestione (identificata dallo scadere di un timer)?
- Siccome uscire della condizioni di congestione è difficile, quando si esaurisce il tempo di ritrasmissione di un segmento, si attua la seguente strategia (**TCP Tahoe**):
  - Si inizializza una soglia a partenza lenta pari a metà della finestra di congestione corrente; ovvero, si fissa *ssthresh* = *cwnd*/2.
  - Si pone *cwnd* = 1 e si procede con la tecnica a partenza lenta fino a che *cwnd* = *ssthresh* (*cwnd* è incrementato di 1 per ogni ACK ricevuto).
  - Per *cwnd* ≥ *ssthresh*, si aumenta *cwnd* di uno per ogni tempo di andata e ritorno.

5.52

Controllo di congestione (TCP)

Tahoe



5.53

Controllo di congestione (TCP)

Tahoe

- La versione più recente di Tahoe (quella originale viene oggi chiamata "old Tahoe") realizza un meccanismo detto di ***Fast Retrasmit***:
  - In corrispondenza dell'arrivo di tre ACK che confermano sempre lo stesso numero di sequenza, il trasmettitore si comporta come fosse scaduto un *timeout*.
  - Infatti, se un segmento viene perduto ma non i successivi, ogni segmento arrivato fuori sequenza comporta la generazione di un ACK con la conferma degli ultimi dati in sequenza ricevuti corretti; siccome in virtù del meccanismo a finestra i segmenti sono spesso inviati uno di seguito all'altro in gruppi, se quello perso non è l'ultimo, è facile che gli ACK arrivino prima dello scadere del timeout.

5.54

Controllo di congestione (TCP)
Reno

- Alcune realizzazioni di TCP usano la strategia ***fast retrasmit*** in modo differente:
  - In corrispondenza dell'arrivo di tre ACK che confermano sempre lo stesso numero di sequenza, il trasmettitore ritrasmette **il solo** segmento successivo a quello confermato senza attendere la scadenza del *timeout*.
  - L'idea di base è supporre che una perdita identificata da tre ACK consecutivi sia essenzialmente una perdita estemporanea (isolata) e che quindi non indichi necessariamente la presenza di una vera e propria situazione di congestione.

5.55

Controllo di congestione (TCP)
Reno

- Insieme al *Fast-retrasmit*, si può attuare un meccanismo modificato per l'aggiornamento della finestra detto ***fast-recovery***:
  - Nel caso in cui scada il *timeout*, il comportamento è quello del TCP Taohe, la finestra riparte da 1.
  - Nel caso in cui arrivino tre ACK con lo stesso numero di sequenza, la fase di *slow-start* non viene attivata immediatamente, ma viene attivato un complesso meccanismo, illustrato successivamente, a supporto del recupero del "solo" segmento perso e per una gestione della finestra adatta ad una condizione di moderata congestione.
- La realizzazione di TCP che applica il *fast-retransmit* e il *fast-recovery* si chiama "**Reno**".

5.56

Controllo di congestione (TCP)
Reno

- Alla ricezione del terzo ACK duplicato, l'agente TCP Reno di inoltro entra nella fase di *Fast-retransmit/Fast-recovery*, eseguendo le seguenti operazioni:
  1. Salva nella variabile ***recover*** il numero di sequenza più alto tra quelli dei segmenti trasmessi, ed imposta:

     ***ssthresh* = max (*FlightSize*/2, 2)**

     FlightSize è il numero dei segmenti trasmessi e non ancora riscontrati nell'istante attuale.
  2. Ritrasmette, quindi, il segmento perso ed imposta:

     ***cwnd* = *ssthresh* + 3**

     la finestra di trasmissione viene artificialmente aumentata di tre unità per tenere conto dei due segmenti già ricevuti e che hanno generato gli ACK duplicati più il segmento perso.

5.57

Controllo di congestione (TCP)
Reno

3. Da qui in avanti, per ogni ACK duplicato ricevuto (per via dei segmenti già trasmessi prima della ricezione dei tre ACK), incrementa di una unità la propria *cwnd*.
4. La trasmissione è regolata in modo usuale: trasmetto nuovi segmenti se *flightsize* < MIN {credito, *cwnd*}.
5. Se arriva l'ACK che riscontra tutti i dati fino al *sequence number* contenuto in *recover*, allora si imposta:

   cwnd =*ssthresh* = min (*ssthresh* , *FlightSize* + 1)

   Oppure: *cwnd* = *ssthresh* = max (*FlightSize*/2, 2)

   » N.B.: Il *FlightSize* si intende aggiornato all'ultimo passo del meccanismo di *Fast Recovery*.

   Il meccanismo di *Fast Recovery* viene così terminato.

– Nel caso in cui siano avvenute più perdite nel periodo antecedente all'attivazione del *fast retrasmitt/recover*, quindi con numero di sequenza minore della variabile *recover*, necessariamente scade un *timeout* e si attiva la fase di *slowstart*.

5.58

Controllo di congestione (TCP)
Reno

- Comportamento del TCP Reno in presenza di una sola perdita:

5.59

Controllo di congestione (TCP)
Reno

- I meccanismi di *Fast-retransmit* e *Fast-recovery* del TCP Reno sono particolarmente inefficienti in presenza di perdite multiple di segmenti nella stessa finestra di trasmissione.

5.60

Controllo di congestione (TCP)
New Reno

- Nella versione Reno del TCP l'agente di trasmissione TCP ritrasmette un segmento dopo:
  - Un *timeout* (attivando la procedura di *Slow Start*),
  - Tre ACK duplicati (applicando il *Fast Recovery* e il *Fast Retransmit*).
- Un *timeout* può provocare la ritrasmissione di più segmenti (*congestion avoidance*), mentre la ricezione di tre ACK duplicati provoca l'applicazione del meccanismo di *Fast Retransmit*, il quale consente la ritrasmissione di solo un segmento.
- I problemi (eccessivo abbassamento del *throughput*) possono nascere quando viene perso più di un segmento nella stessa finestra di trasmissione e vengono invocati i meccanismi di *Fast Recovery* e *Fast Retransmit*.
- Sono state quindi proposte un serie di modifiche che cercano di risolvere questi problemi (più altri) che hanno dato luogo ad una ulteriore versione del TCP detta **New Reno.**

5.61

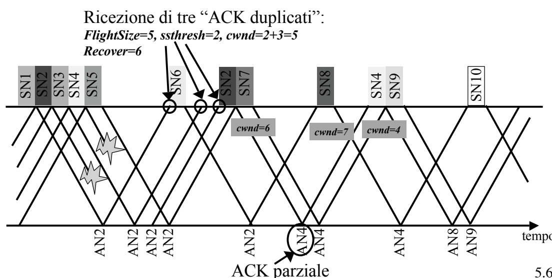
Controllo di congestione (TCP)
New Reno

- I meccanismi di *Fast-retransmit* e *Fast-recovery* nella versione New Reno compiono sostanzialmente le stesse operazioni precedentemente descritte, ma vengono modificati per il recupero di perdite multiple nella stessa finestra di trasmissione:
  - Un'ACK che riscontra nuovi dati ma non il numero di sequenza contenuto in *recover*, è un "ACK parziale".
  - Quando si riceve un ACK parziale
    - » si procede alla ritrasmissione del primo segmento non riscontrato
    - » si decrementa *cwnd* del numero dei segmenti appena riscontrati meno uno
    - » se possibile, si inoltrano nuovi segmenti.
  - L'agente TCP di inoltro continua a rimanere in *Fast Retransmit*.
- Il decremento "parziale" della finestra di congestione viene effettuato per garantire che, quando termina il meccanismo di *Fast Retransmit*, la nuova *cwnd* sia pari a circa *ssthresh* segmenti.

5.62

Controllo di congestione (TCP)
New Reno

- Comportamento con più perdite nella stessa finestra di trasmissione nel TCP New Reno:



5.63

Controllo di congestione (TCP)
New Reno

- In assenza dell'opzione *Sack*, un agente TCP di inoltro può interpretare la ricezione un ACK duplicato come l'effetto di:
  - Un segmento perso o ricevuto in ritardo dal TCP ricevente,
  - Un segmento ritrasmesso e che era già stato ricevuto dal TCP ricevente.
- Nel New Reno questa seconda condizione si può verificare solo se una fase di *Fast Retransmit* viene terminata a causa di un *timeout:*
  - In tale situazione, l'agente TCP di inoltro, passando alla fase di *Slow Start*, inizia a ritrasmettere tutti i segmenti a partire da quello che ha generato il *timeout*;
  - Se tra i segmenti ritrasmessi durante questa fase ve ne sono almeno tre consecutivi già ricevuti correttamente dall'agente di ricezione (durante il *Fast-recovery* precedente) verranno generati tre ACK duplicati.
  - L'agente TCP di inoltro ripasserà inutilmente in fase di *Fast-retransmit* cercando di recuperare un segmento erroneamente considerato perso.
- Quindi, nella situazione di cui sopra, si ha l'attivazione di più fasi di *Fast Retransmit/Recovery*, che producono eccessive riduzioni della finestra di trasmissione ed inutili ritrasmissioni.

5.64

Controllo di congestione (TCP)
New Reno

- Questo problema è stato risolto nel ***New Reno*** con l'introduzione del meccanismo di "*bug-fix*":
  - Viene introdotta nell'agente TCP di inoltro una variabile ***send_high*** in cui viene salvato il numero di sequenza più alto dei segmenti trasmessi prima dello scadere di un *timeout*.
  - Quando vengono ricevuti tre ACK duplicati consecutivi che:
    - » riscontrano un numero di sequenza più basso del valore in *send_high*, allora si presume che si tratti di ACK dovuti alla ritrasmissione non necessaria di alcuni segmenti, e quindi si prosegue la fase di *Slow Start*.

      (Nel caso che tali ACK duplicati fossero invece dovuti ad una nuova perdita, per il recupero del segmento perso si dovrà aspettare lo scadere del timeout ad esso associato)
    - » riscontrano un numero di sequenza più alto del valore in *send_high*, allora si tratta di ACK dovuti alla perdita di nuovi segmenti e quindi si attiva una fase di *Fast-recovery*.

5.65

Controllo di congestione (TCP)
New Reno

- Per quanto riguarda la ri-inizializzazione dei contatori di *timeout*, nella versione New Reno l'inizializzazione avviene solo dopo:
  - l'eventuale ricezione del primo ACK parziale (*Impatient Variant*).

    In questo caso, se fosse perso un numero considerevole di segmenti nella stessa finestra di trasmissione, è molto probabile che scada un *timeout* e quindi che l'agente TCP dia inizio ad una fase di *Slow Start*.
  - Ad ogni ACK parziale ricevuto (*Slow-but-Steady Variant*).

    Con questa opzione, in presenza di un numero significativo di segmenti persi nella stessa finestra, si ha la ritrasmissione di almeno un segmento per RTT.

5.66

Controllo di congestione (TCP)

Sack

- Una perdita di più pacchetti nella stessa finestra di trasmissione può abbassare drasticamente il *throughput* di una connessione TCP.
- Il *Selective Acknowledgment* (SACK) è una strategia che permette di ovviare a questi problema.
- Con l'opzione SACK, l'agente TCP di ricezione può comunicare a quello di trasmissione quali segmenti sono stati ricevuti correttamente, così da ovviare ad eventuali inutili ritrasmissioni.
- Il SACK non costituisce di per sé una versione del TCP, ma è un'opzione aggiuntiva.
- Attualmente la versione TCP di *default* implementata nei S.O. MS Windows e Linux è il New Reno con opzione SACK attiva.

5.67

Controllo di congestione (TCP)

Sack

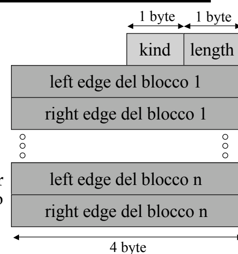
- L'estensione *selective acknowledgment* utilizza fondamentalmente due opzioni:
  - "*SACK-permitted*" (2 byte), trasmessa nel segmento di SYN dall'agente TCP in ricezione per indicare il supporto all'estensione SACK.
  - "*SACK-option*", trasmessa solo a connessione già stabilita tra due agenti TCP; trasferisce le informazioni necessarie all'*aknoledgement* esteso.

5.68

Controllo di congestione (TCP)

Sack

- *kind* (1byte): campo identificativo dell'opzione SACK, il valore di default è 5.
- *length* (1 byte): lunghezza dell'opzione.
- *left edge* (4 byte): sequence number del primo segmento del blocco non contiguo ricevuto correttamente
- *right edge* (4 byte): sequence number del primo segmento non riscontrato successivo al blocco non contiguo ricevuto correttamente



5.69

Controllo di congestione (TCP)

Sack

- L'opzione SACK deve essere inviata dall'agente TCP in ricezione per informare quello in trasmissione sugli eventuali blocchi di dati non contigui ricevuti correttamente.
- L'agente in ricezione attende di ricevere i segmenti mancanti per riempire i *gap* tra i blocchi di dati ricevuti correttamente.
- Quando i segmenti mancanti sono finalmente ricevuti, l'agente TCP di ricezione riscontra i segmenti "normalmente" mettendo nell'*Acknowledgement Number Field* dell'*header* TCP il valore della *left window edge*.

5.70

Controllo di congestione (TCP)

Sack

- Il TCP SACK nel recupero di perdita di uno o più segmenti utilizza un meccanismo di *Fast-recovery* modificato.
- Viene usata una variabile *pipe* al posto di *Flightsize:*
  - Viene incrementata per ogni segmento trasmesso e decrementata per ogni ACK duplicato ricevuto
- L'agente TCP di inoltro invia i soli segmenti che l'ultimo ACK esteso ricevuto segnala come persi.
- Una volta riscontrati i segmenti inizialmente persi si esce dalla fase di *Fast-recovery* in modo analogo a come succede nel New Reno.

5.71