Corsi di Laurea in Ingegneria Informatica
Ingegneria delle Telecomunicazioni
Ingegneria dell'Automazione

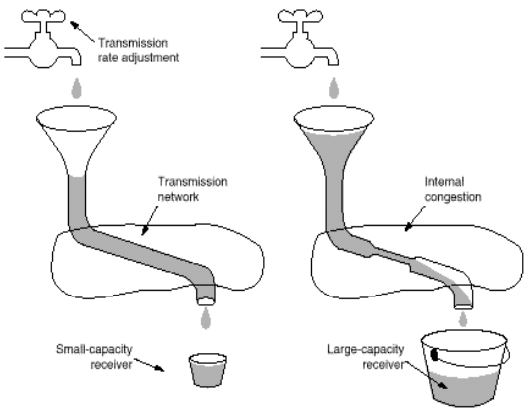
Corso di Reti di Calcolatori

Simon Pietro Romano (spromano@unina.it)
Antonio Pescapè (pescape@unina.it)
Giorgio Ventre (giorgio@unina.it)

Il livello trasporto:

TCP: controllo di flusso e controllo della congestione

TCP: Controllo di Flusso e di Congestione



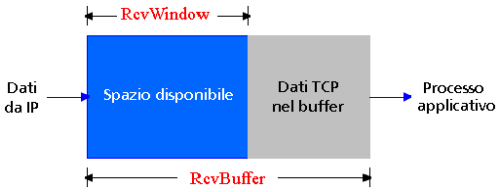
Come gestire entrambi i tipi di controllo?

- *Receiver window*: dipende dalla dimensione del buffer di ricezione
- *Congestion window*: basata su una stima della capacità della rete

I byte trasmessi corrispondono alla dimensione della finestra più piccola

TCP: controllo di flusso

- Il lato ricevente della connessione TCP ha un buffer di ricezione:



- Il processo applicativo potrebbe essere rallentato dalla lettura nel buffer

Controllo di flusso

- Servizio di corrispondenza delle velocità: la frequenza d'invio deve corrispondere alla frequenza di lettura dell'applicazione ricevente

(supponiamo che il destinatario TCP scarti i segmenti fuori sequenza)

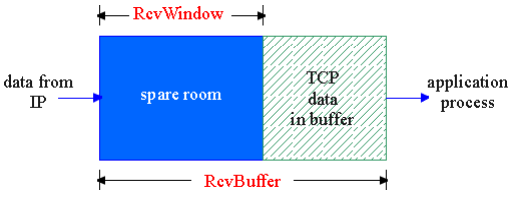
TCP Flow Control

flow control

Il mittente non dovrà sovraccaricare il ricevente inviando dati ad una velocità troppo elevata

`RcvBuffer` = size of TCP Receive Buffer

`RcvWindow` = amount of spare room in Buffer



receiver buffering

ricevente: comunica dinamicamente al mittente la dimensione corrente del buffer

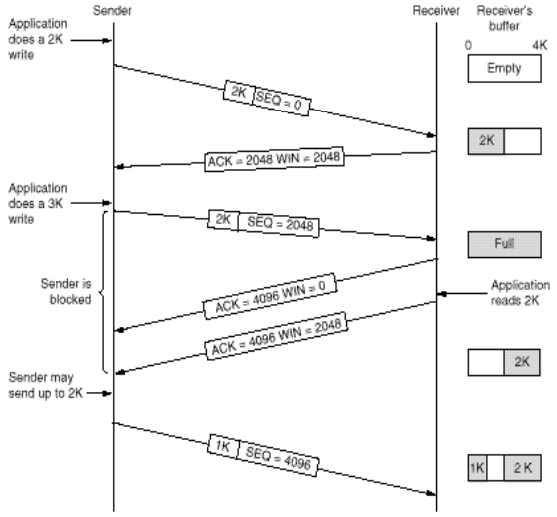
- campo **`RcvWindow`** nel segmento TCP:

```
RcvWindow = RcvBuffer -
   [LastByteRcvd -
   LastByteRead]
```

mittente: conserva i dati già trasmessi ma non riscontrati e limita tale quantità all'ultima **`RcvWindow`** ricevuta:

```
LastByteSent - LastByteAcked
   ≤ RcvWindow
```

TCP: Transmission Policy



5

Silly Window Syndrome

- Silly Window Syndrome (ricevitore): il ricevitore svuota lentamente il buffer di ricezione e invia segmenti di ack con dimensione della finestra molto piccola, quindi il trasmettitore invia segmenti corti con molto overhead. Soluzione con l'algoritmo di Clark: il ricevitore indica una finestra nulla finchè il buffer di ricezione non si è svuotato per metà o per una porzione uguale a MSS.
- Silly Window Syndrome (trasmettitore): l'applicazione genera dati lentamente, invia segmenti molto piccoli così come vengono prodotti. Soluzione algoritmo di Nagle: il TCP sorgente invia la prima porzione di dati anche se corta e gli altri vengono inviati solo se o il buffer di uscita contiene almeno MSS byte, oppure se si riceve un ack per il segmento precedente.
- Una volta vengono inviati segmenti corti con molto overhead perché la finestra al ricevitore è grande un byte, un'altra volta perché è l'applicazione che genera un byte alla volta molto lentamente.

6

L’algoritmo di Nagle

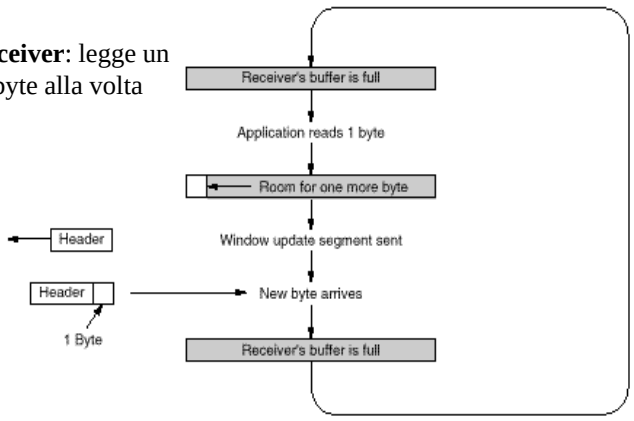
- Per applicazioni che inviano dati un byte alla volta (es: TELNET):
  - invia il primo byte e bufferizza il resto finché non giunge l’ACK
  - in seguito:
    - invia, in un unico segmento, tutti i caratteri bufferizzati
    - ricomincia a bufferizzare finché non giunge l’ACK per ognuno di essi
  - Elimina la sindrome della Silly Window al trasmettitore

7

La sindrome della Silly Window

Sender: invia blocchi grandi

Receiver: legge un byte alla volta



Soluzione di Clark: Impedisce al receiver di aggiornare la finestra un byte alla volta

Il ricevitore indica una finestra nulla finchè il buffer di ricezione non si è svuotato per metà o per una porzione uguale a MSS.

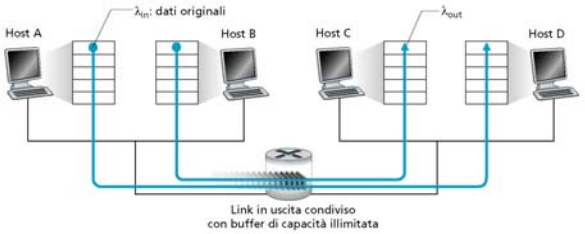
8

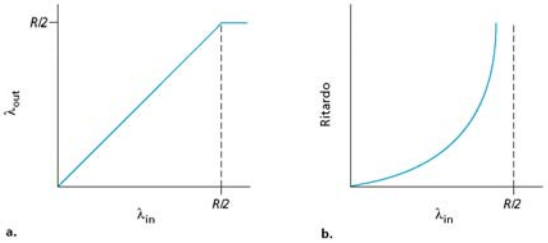
Controllo della congestione

- Congestione nella rete
  - Tecnicamente dovuta a:
    - un numero elevato di sorgenti di traffico
    - sorgenti di traffico che inviano troppi dati
    - traffico inviato ad una frequenza troppo elevata
  - In presenza di questi fenomeni, singoli o concomitanti, la rete è sovraccarica
    - Effetti:
      - perdita di pacchetti:
        - » buffer overflow nei router
      - ritardi nell'inoltro dei pacchetti:
        - » accodamenti nei buffer dei router
      - scarso utilizzo delle risorse di rete

Effetti della congestione: esempi (1/2)

- 2 mittenti
- 2 riceventi
- 1 router con buffer (coda) ∞:
  - non ci sono ritrasmissioni



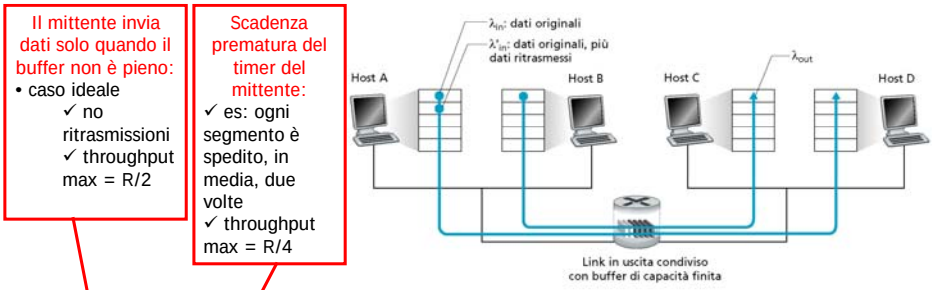


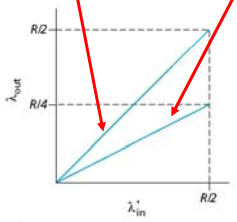
- I ritardi aumentano all'avvicinarsi del limite di capacità del canale
- Non si può superare il max throughput

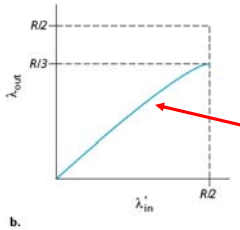
Effetti della congestione: esempi (2/2)

Il mittente invia dati solo quando il buffer non è pieno:
- caso ideale
  - ✓ no ritrasmissioni
  - ✓ throughput max = R/2

Scadenza prematura del timer del mittente:
- ✓ es: ogni segmento è spedito, in media, due volte
- ✓ throughput max = R/4







Il mittente rispedisce un segmento solo quando è sicuro che sia andato perso:

- ✓ il throughput effettivo è inferiore al carico offerto (trasmissioni dati originali + ritrasmissioni)
- ✓ es: curva in figura

11

Tecniche di Controllo della Congestione

Approccio end-to-end:

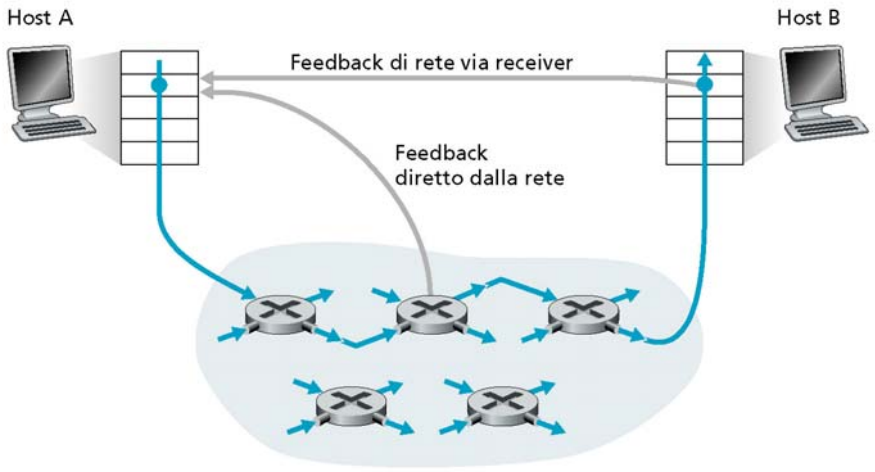
- Nessuna segnalazione esplicita dalla rete
- A partire dall'osservazione di ritardi e perdite di pacchetti gli end-system deducono uno stato di congestione nella rete
- Approccio utilizzato da TCP

Approccio in base a segnalazione della rete:

- I router forniscono informazioni circa lo stato della rete agli end-system:
  - l'invio di un singolo bit indica lo stato di congestione
    - SNA, DECbit, TCP/IP ECN, ATM
  - in alternativa, il sender è informato circa la massima frequenza alla quale può trasmettere

12

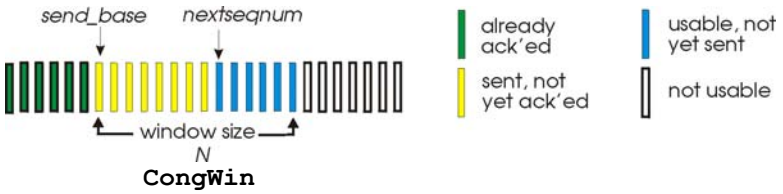
Feedback di rete: tecniche alternative



13

Controllo della congestione in TCP

- Controllo end-to-end: nessun feedback dalla rete
- Frequenza di trasmissione variabile:
  - dipendente dalla cosiddetta *finestra di congestione (CongWin)*



Considerando controllo di flusso e controllo di congestione insieme, si ha, dunque:

LastByteSent – LastByteAcked ≤ min{RcvWindow, CongWin}

14

Controllo della congestione: idea di base

- Si procede per tentativi, per stabilire quanto si può trasmettere:
  - obiettivo:
    - trasmettere alla massima velocità possibile (`Congwin` quanto più grande possibile) senza perdite
  - approccio utilizzato:
    - incrementare `Congwin` finché non si verifica la perdita di un segmento (interpretata come il sopraggiungere dello stato di congestione)
    - in seguito alla perdita di un segmento:
      - decrementare `Congwin`
      - ricominciare daccapo

15

Controllo della congestione: fasi

- *Slow Start*
  - Partenza lenta (per modo di dire!)
- Congestion Avoidance:
  - Additive Increase, Multiplicative Decrease (AIMD)
    - incremento additivo, decremento moltiplicativo

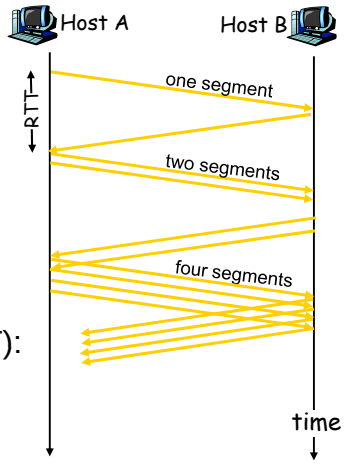
16

Lo Slow Start in TCP

Algoritmo Slowstart

```
//initialization
Congwin = 1 MSS

for (each segment ACKed)
      Congwin=Congwin+1MSS
until (loss event OR
        CongWin > threshold)
```

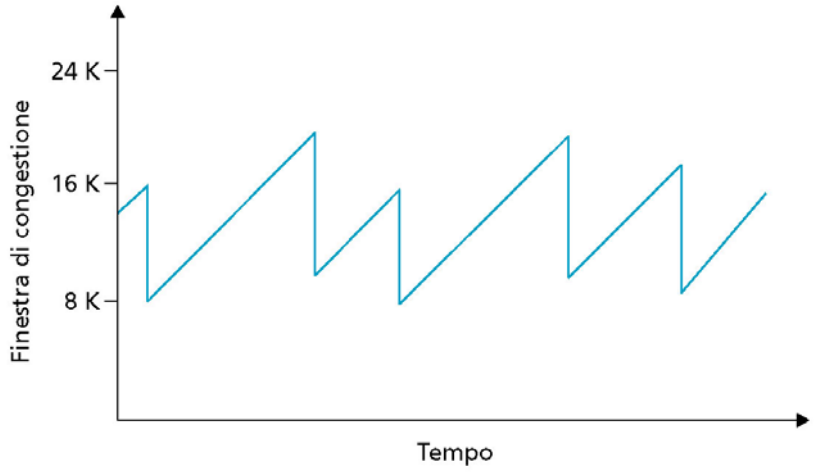


- Crescita esponenziale della dimensione della finestra (ogni RTT):
  - "Slow start" → termine improprio!
- Evento di *perdita*:
  - timeout
  - tre ACK duplicati consecutivi

Dopo lo slow start: AIMD

- *Additive Increase*
  - Una volta raggiunta la soglia:
    - ci si avvicina con cautela al valore della banda disponibile tra le due estremità della connessione
    - meccanismo adottato:
      - incremento di `CongWin` di `MSS·(MSS/Congwin)` alla ricezione di un ACK
    - questa fase è nota come fase di *congestion avoidance*
- *Multiplicative Decrease:*
  - Al sopraggiungere della congestione (scadenza di un timeout o ricezione di tre ACK duplicati consecutivi):
    - la finestra di congestione viene dimezzata

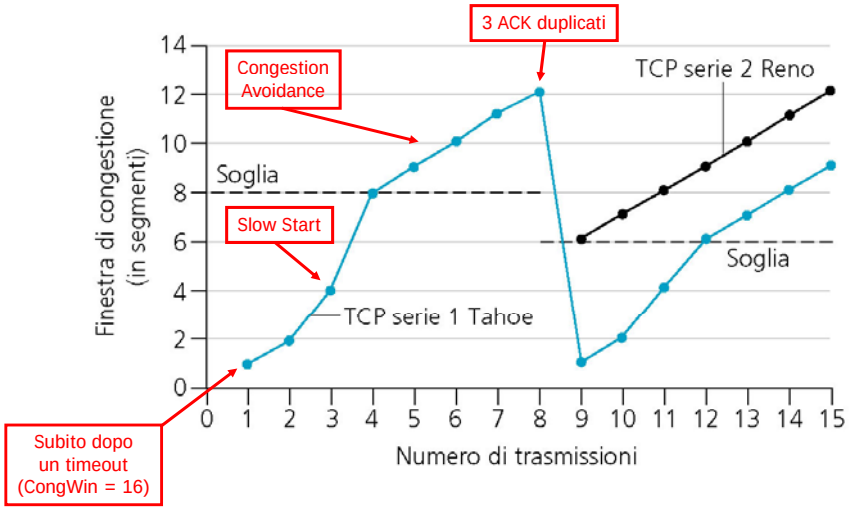
AIMD: andamento a “dente di sega”



19

Controllo della congestione in Internet

È presente un ulteriore parametro: soglia (threshold)



20

Ricapitolando…

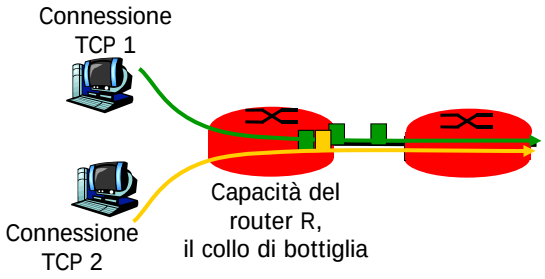
- Finestra di congestione sotto la soglia:
  - Slow start
  - Crescita esponenziale della finestra
- Finestra di congestione sopra la soglia:
  - Prevenzione della congestione
  - Crescita lineare della finestra
- Evento di perdita dedotto da ACK duplicato 3 volte:
  - Soglia posta alla metà del valore attuale della finestra
  - TCP Reno:
    - Finestra posta pari alla soglia
  - TCP Tahoe:
    - Finestra posta pari ad un segmento (MSS -- Maximum Segment Size)
- Evento di perdita dedotto da timeout:
  - Soglia posta alla metà del valore attuale della finestra
  - Finestra posta pari ad un segmento (MSS -- Maximum Segment Size)

TCP Reno: "*fast recovery*"

- TCP Reno elimina la fase di partenza lenta dopo un evento di perdita dedotto dalla ricezione di tre ACK duplicati:
  - tale evento indica che, nonostante si sia perso un pacchetto, almeno 3 segmenti successivi sono stati ricevuti dal destinatario:
    - a differenza del caso di timeout, la rete mostra di essere in grado di consegnare una certa quantità di dati
    - è possibile, quindi, evitare una nuova partenza lenta, ricominciando direttamente dalla fase di prevenzione della congestione

Equità tra le connessioni TCP (1/2)

Equità: se K sessioni TCP condividono lo stesso collegamento con ampiezza di banda R, che è un collo di bottiglia per il sistema, ogni sessione dovrà avere una frequenza trasmissiva media pari a R/K.



Si può dire che AIMD è equo?

Equità tra le connessioni TCP

- Ipotesi:
  - MSS e RTT uguali per le due connessioni:
    - a parità di dimensioni della finestra quindi il throughput è lo stesso
  - Entrambe le connessioni si trovano oltre lo slow start:
    - fase di prevenzione della congestione:
      - incremento additivo
      - decremento moltiplicativo