

## Il livello Trasporto

### Livello 4 (trasporto): cosa vedremo

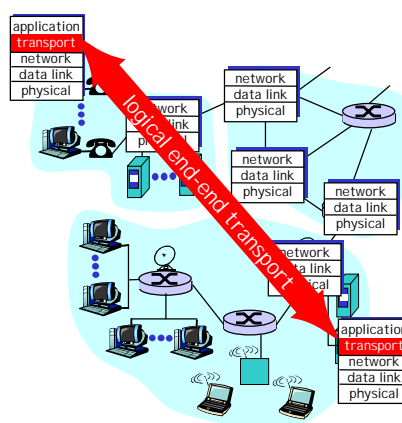
- ❑ Servizi di livello trasporto
- ❑ multiplexing/demultiplexing
- ❑ Protocollo non orientato alla connessione: UDP
- ❑ Principi di trasferimento end-to-end affidabile
- ❑ Protocollo orientato alla connessione: TCP
  - Trasferimento affidabile end-to-end
  - Controllo di flusso e controllo della congestione
  - Gestione della connessione
- ❑ Principi di controllo della congestione
- ❑ TCP

Copyright © Luciano Bononi 2004 (some figure credits to Kurose, Ross, Internet e reti di calcolatori)

3a-1

## Servizi e protocolli di livello trasporto

- ❑ fornire *comunicazione logica* tra processi in esecuzione su host remoti
- ❑ I protocolli di livello trasporto sono eseguiti solo dagli host agli estremi del cammino
- ❑ **Servizi di livello trasporto vs. servizi di livello rete:**
- ❑ *Livello rete:* trasferimento dati da host a host
- ❑ *Livello trasporto:* trasferimento dati tra processi agli estremi
  - basato su livello rete



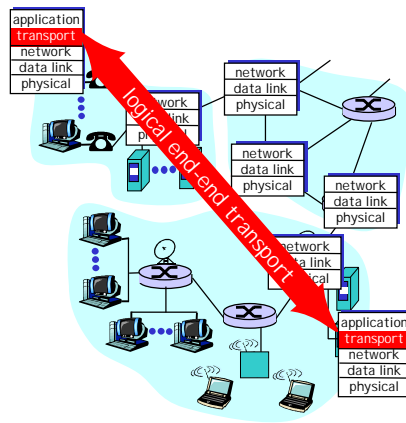
Copyright © Luciano Bononi 2004 (some figure credits to Kurose, Ross, Internet e reti di calcolatori)

3a-2

## Protocolli di livello trasporto

### Servizi di livello trasporto di Internet:

- Consegna ordinata e affidabile (TCP)
  - Controllo di congestione
  - Controllo di flusso
  - Setup della connessione
- Consegna inaffidabile e disordinata ("best-effort"): UDP
- Servizi non realizzati:
  - real-time
  - Qualità del servizio garantita
  - Multicast affidabile



Copyright © Luciano Bononi 2004 (some figure credits to Kurose, Ross, Internet e reti di calcolatori)

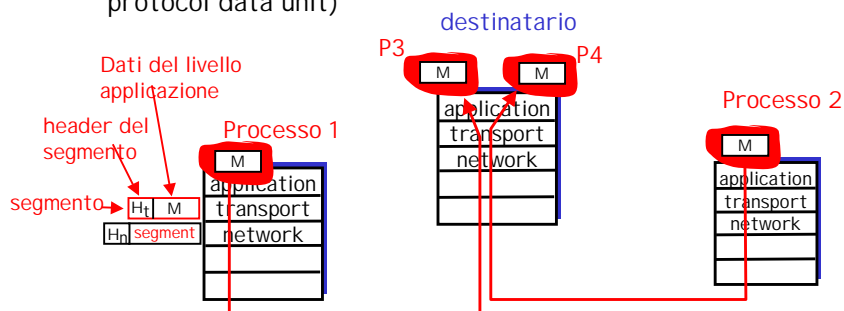
3a-3

## Multiplexing/demultiplexing

N.B.: *segmento* - unità di dati scambiata tra entità di livello trasporto

- TPDU: (transport protocol data unit)

**Demultiplexing:** la fase di consegna dei dati ricevuti all'applicazione superiore



Copyright © Luciano Bononi 2004 (some figure credits to Kurose, Ross, Internet e reti di calcolatori)

3a-4

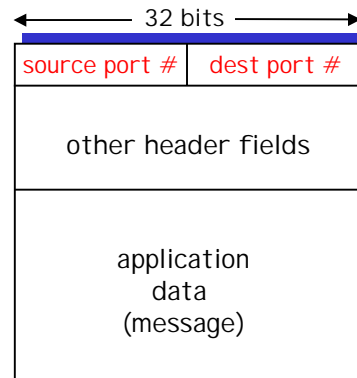
## Multiplexing/demultiplexing

### Multiplexing:

I dati ricevuti dalle applicazioni vengono incapsulati in segmenti con le informazioni che serviranno al demultiplexing

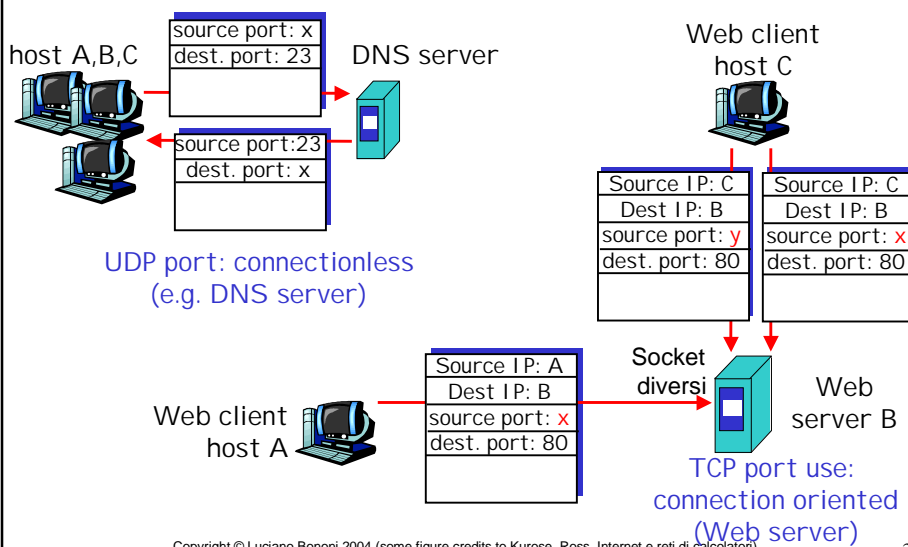
multiplexing/demultiplexing:

- Si basa sui Numeri di Porta e indirizzi IP del mittente e destinatario finale
  - Inserite in ogni segmento
  - N.B. well-known port numbers sono usati per applicazioni famose



TCP/UDP segment format

## Multiplexing/demultiplexing: esempi



## UDP: User Datagram Protocol [RFC 768]

- ❑ Semplice e pratico
- ❑ Servizio "best effort", i segmenti UDP possono
  - perdersi
  - essere disordinati
- ❑ *Servizio connectionless*:
  - Non c'è instaurazione preliminare della connessione tra sender e receiver
  - Ogni segmento UDP è gestito in modo indipendente (non c'è stato della connessione)

### Perchè è utile UDP?

- ❑ Non c'è instaurazione preliminare di connessione (trasferimento immediato e più veloce dei dati)
- ❑ Semplicità: non serve mantenere lo stato della connessione
- ❑ header del segmento ridotto (8 B)
- ❑ Non c'è controllo della congestione: UDP può essere spinto alla massima velocità (non sempre...)

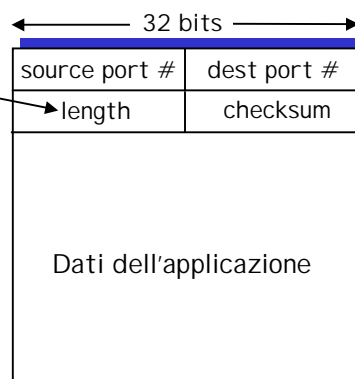
Copyright © Luciano Bononi 2004 (some figure credits to Kurose, Ross, Internet e reti di calcolatori)

3a-7

## UDP e suo utilizzo

- ❑ Si usa spesso per lo streaming multimediale
  - Tolleranza perdita di pacchetti
  - Sensibile al ritmo di invio dei pacchetti
- ❑ UDP usato anche in
  - DNS
  - SNMP
- ❑ R-UDP: UDP affidabile, si ottiene garantendo controlli a livello di ricezione al di sopra del livello trasporto
  - Soluzioni di gestione specifiche del livello applicativo

Dimensione in byte del segmento UDP incluso l'header



Formato del segmento UDP

Copyright © Luciano Bononi 2004 (some figure credits to Kurose, Ross, Internet e reti di calcolatori)

3a-8

## UDP checksum

**obiettivo:** rilevare errori sui bit del segmento

### Sender:

- ❑ Considera i dati come interi a 16 bit
- ❑ checksum: somma gli interi (senza riporto) e fa il complemento a 1
- ❑ Inserisce il valore della somma nel campo checksum del segmento

### Receiver:

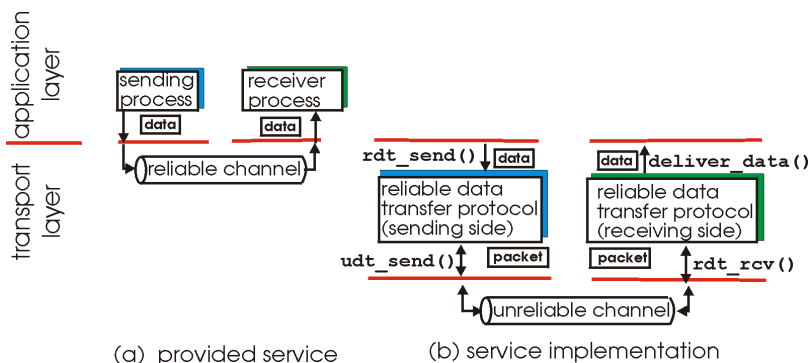
- ❑ Calcola la somma di controllo dei dati ricevuti
- ❑ Verifica la presenza di errori
  - Non ci sono errori: passa i dati all'applicazione
  - Ci sono errori: elimina i dati

Copyright © Luciano Bononi 2004 (some figure credits to Kurose, Ross, Internet e reti di calcolatori)

3a-9

## Principi di trasferimento dati affidabile

- ❑ Vogliamo affidabilità al livello applicativo attraverso servizi del livello rete inaffidabili: come si fa?



- ❑ Le caratteristiche del canale di comunicazione (e dei servizi di rete forniti) determinano il grado di complessità del livello trasporto per determinare un livello affidabile.

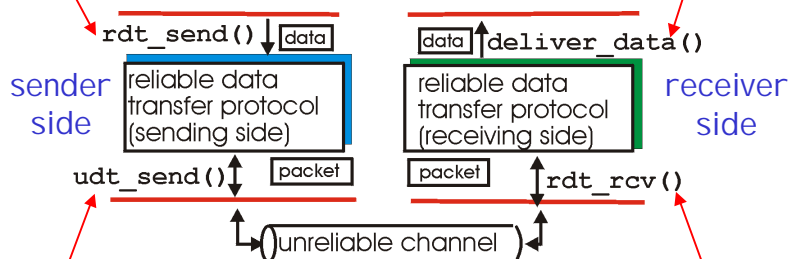
Copyright © Luciano Bononi 2004 (some figure credits to Kurose, Ross, Internet e reti di calcolatori)

3a-10

## Livello trasporto affidabile: come si fa?

**rdt\_send()**: chiamata da sopra (applicazione), passa i dati da inoltrare all'applicazione ricevente

**deliver\_data()**: chiamata dal livello trasporto per consegnare i dati ricevuti all'applicazione sopra



**udt\_send()**: chiamata del servizio inaffidabile di spedizione dei dati sulla rete

**rdt\_rcv()**: chiamata dal livello rete quando ci sono dati ricevuti per il livello trasporto

Copyright © Luciano Bononi 2004 (some figure credits to Kurose, Ross, Internet e reti di calcolatori)

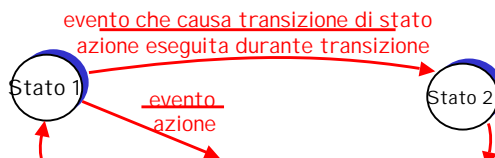
3a-11

## Livello trasporto affidabile: come si fa?

### Assunzioni:

- Partiamo da esempi semplici e poi via via arriviamo al caso reale
- Consideriamo solo trasferimento in una direzione
  - Anche se le info viaggiano nei due sensi
- Usiamo la notazione delle macchine a stati finiti per descrivere i protocolli

**stato**: dato uno stato abbiamo tante transizioni verso altri stati quanti sono i possibili eventi

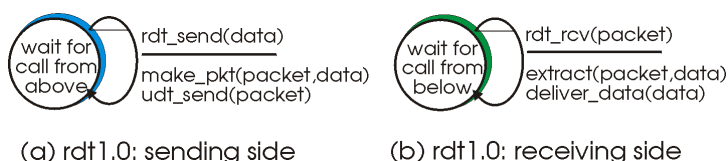


Copyright © Luciano Bononi 2004 (some figure credits to Kurose, Ross, Internet e reti di calcolatori)

3a-12

## Rdt1.0: trasferimento affidabile su rete affidabile

- È un caso quasi banale: vogliamo creare un livello trasporto affidabile (RDT) avendo già un livello rete affidabile
  - La rete non crea bit errati e non si perdono pacchetti!!!
- Vediamo come funzionano il sender e il receiver:
  - Sender spedisce dati sulla rete
  - Receiver riceve dati dalla rete



Copyright © Luciano Bononi 2004 (some figure credits to Kurose, Ross, Internet e reti di calcolatori)

3a-13

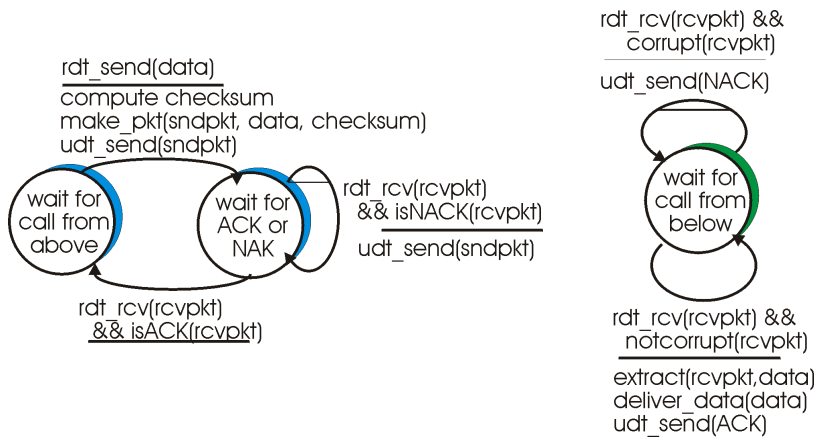
## Rdt2.0: rete con bit errati, senza perdita

- Assunzione: la rete altera i bit ma non perde pacchetti
  - N.B.: UDP checksum rileva i bit errati
- *Come si recuperano le situazioni con bit errati?*
  - *acknowledgements (ACKs)*: il receiver manda un pacchetto ACK al sender per dire che i dati sono OK
  - *negative acknowledgements (NAKs o NACKs)*: il receiver manda un pacchetto NAK al sender per dire che i dati sono errati
  - sender ritrasmette i dati se riceve NAK
- Nuovi meccanismi in **rdt2.0** (aggiunti a **rdt1.0**):
  - Rilevazione di errori sui bit (es. checksum)
  - Messaggi di controllo da parte del receiver (ACK,NAK)

Copyright © Luciano Bononi 2004 (some figure credits to Kurose, Ross, Internet e reti di calcolatori)

3a-14

## rdt2.0: definizione protocollo



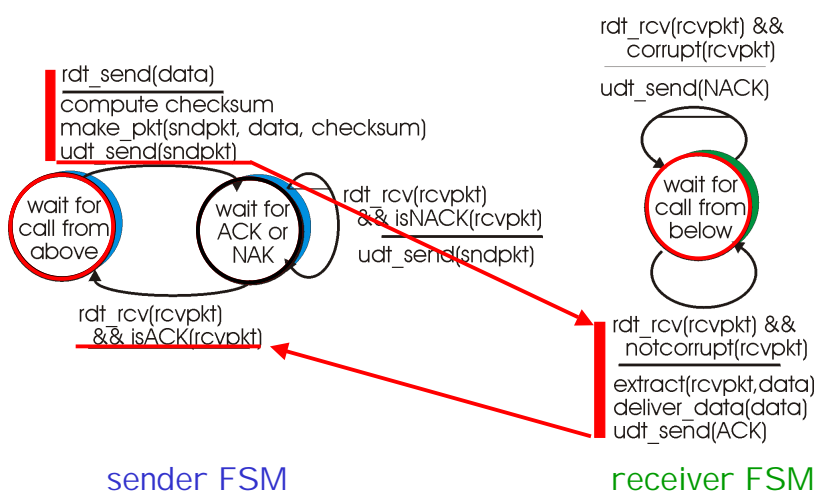
Macchina a stati finiti del sender

Macchina a stati finiti del receiver

Copyright © Luciano Bononi 2004 (some figure credits to Kurose, Ross, Internet e reti di calcolatori)

3a-15

## rdt2.0: caso1: spedizione senza errore

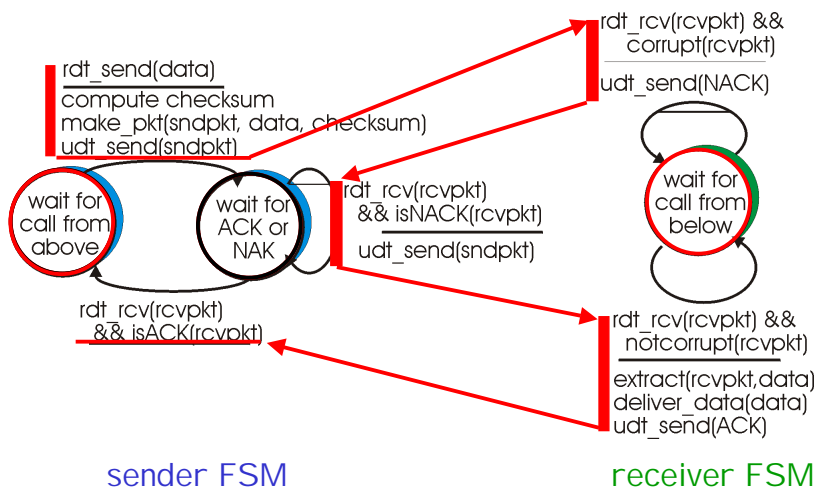


Copyright © Luciano Bononi 2004 (some figure credits to Kurose, Ross, Internet e reti di calcolatori)

3a-16



## rdt2.0: caso 2: spedizione con errore e ritrasmissione



Copyright © Luciano Bononi 2004 (some figure credits to Kurose, Ross, Internet e reti di calcolatori)

3a-17

## rdt2.0 ha un problema grave!

### Cosa succede se ci sono bit errati su Ack e NAK?

- Il sender non capisce cosa è accaduto sul receiver
- Se il sender decide di ritrasmettere si originano duplicati del segmento

### Come si fa?

- Inviare ACK di ACK?
  - No. Stesso problema
- Il sender ritrasmette!
  - Ma possono nascere duplicati del segmento!

### Gestione dei duplicati

- Il sender aggiunge *numero di sequenza (SN)* ad ogni segmento
- Il sender ritrasmette l'ultimo pacchetto finchè non riceve ACK
- Il ricevente elimina i segmenti duplicati (con stesso SN)

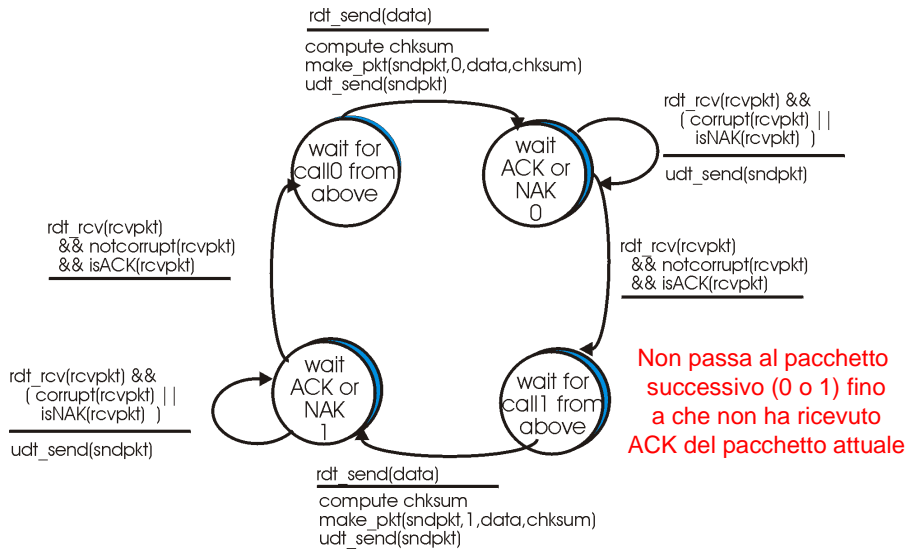
### Protocollo Stop and wait

Questa tecnica si basa sull'invio iterato di uno stesso segmento fino alla corretta ricezione (senza passare al successivo fino ad allora)

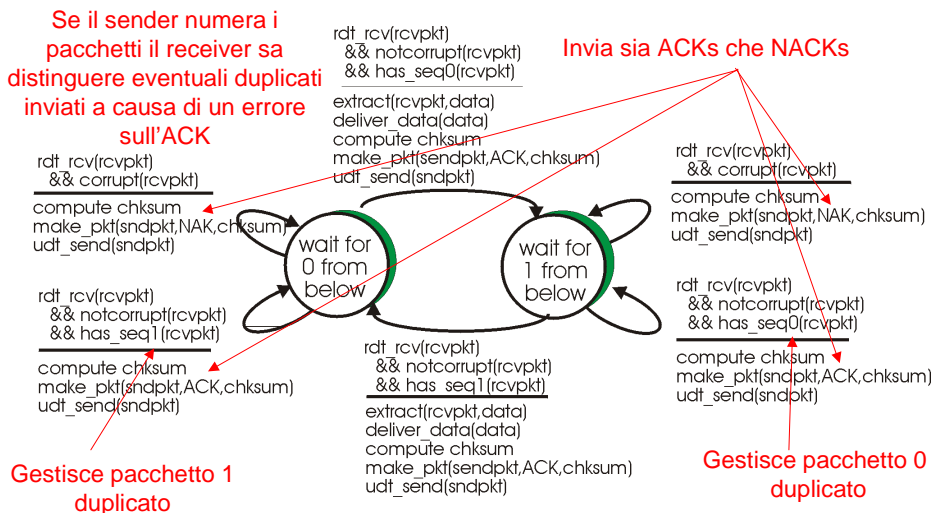
Copyright © Luciano Bononi 2004 (some figure credits to Kurose, Ross, Internet e reti di calcolatori)

3a-18

## rdt2.1: lato sender, ora gestisce ACK e NAK errati



## rdt2.1: lato receiver, gestisce segmenti duplicati



## rdt2.1: punto della situazione

### Sender:

- ❑ Aggiunge numero di sequenza ai pacchetti
- ❑ Bastano numeri 0 e 1 perchè è stop & wait
- ❑ Verifica se ACK e NAK sono corretti
- ❑ Ha ora il doppio degli stati
  - Lo stato serve a ricordare se l'ultimo segmento era 0 o 1

### Receiver:

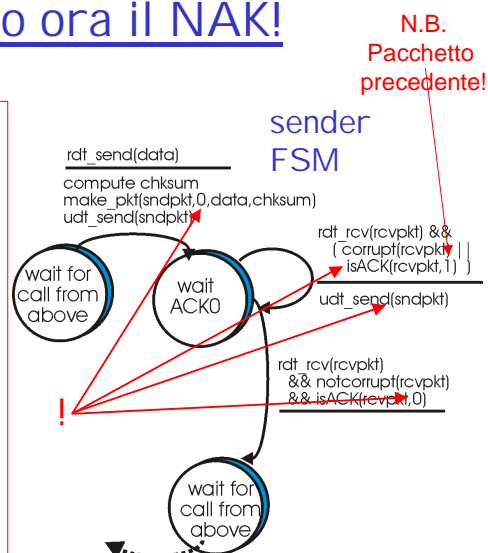
- ❑ Deve verificare se riceve segmenti duplicati
  - Lo stato dice quale pacchetto si aspetta (0 o 1)
- ❑ N.B. il receiver non sa se il ACK o NAK è stato ricevuto dal sender...

Copyright © Luciano Bononi 2004 (some figure credits to Kurose, Ross, Internet e reti di calcolatori)

3a-21

## rdt2.2: eliminiamo ora il NAK!

- ❑ Invece di NAK, receiver **spedisce sempre ACK per l'ultimo pacchetto ricevuto corretto!**
  - receiver **inserisce il numero del pacchetto corretto nell'ACK**
- ❑ Se il sender riceve ACK duplicati li interpreta come NAK! ...*E ritrasmette il pacchetto successivo a quello a cui si riferisce l'ACK*



Copyright © Luciano Bononi 2004 (some figure credits to Kurose, Ross, Internet e reti di calcolatori)

3a-22

## rdt3.0: rete con errori sui bit *e perdita dei pacchetti*

**Nuova assunzione:** la rete può anche perdere i pacchetti (sia dati che ACK)

- o checksum, seq. #, ACKs, e ritrasmissione non bastano più...

**Q:** come si gestisce la perdita di pacchetti?

- o Il sender rischia di bloccarsi all'infinito nell'attesa di un ACK.

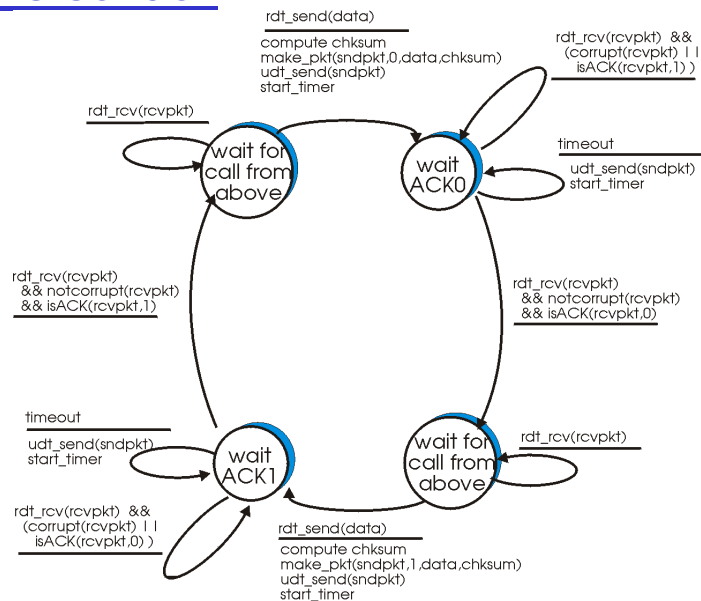
**Soluzione:** sender attende l'ACK per un tempo ragionevole (timeout)

- o ...e ritrasmette se il Timeout scade senza avere ricevuto ACK
- o Ma se il pacchetto dati (o l'ack) arrivassero dopo il timeout?
  - o Allora si è trasmesso un duplicato, ma il numero di sequenza risolve già il problema sul receiver
  - o receiver deve specificare il numero di pacchetto del quale invia l'ACK
- o Ma ora serve un timer!!!

Copyright © Luciano Bononi 2004 (some figure credits to Kurose, Ross, Internet e reti di calcolatori)

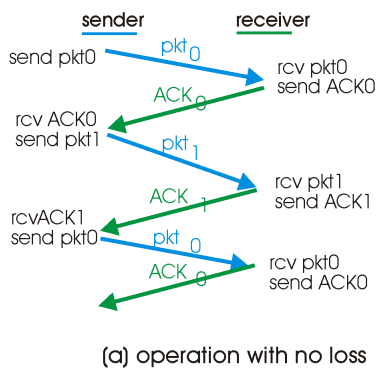
3a-23

## rdt3.0 sender

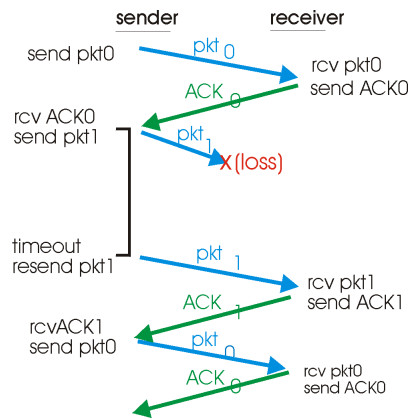


3a-24

## rdt3.0: esempio 1



(a) operation with no loss

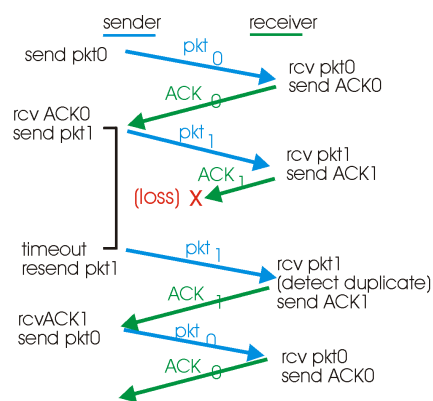


(b) lost packet

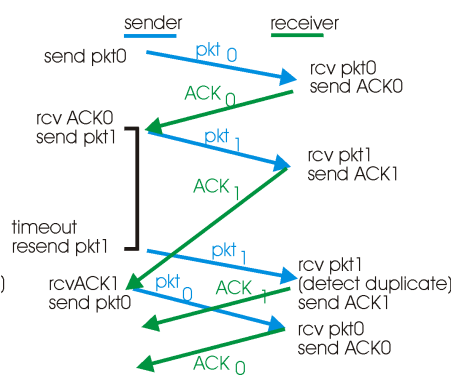
Copyright © Luciano Bononi 2004 (some figure credits to Kurose, Ross, Internet e reti di calcolatori)

3a-25

## rdt3.0: esempio 2



(c) lost ACK



(d) premature timeout

Copyright © Luciano Bononi 2004 (some figure credits to Kurose, Ross, Internet e reti di calcolatori)

3a-26

## Prestazioni del protocollo nella versione: rdt3.0

- ❑ rdt3.0 è un livello di trasporto affidabile, e in pratica racconta come funziona TCP, ma le prestazioni sono inaccettabili!
- ❑ esempio: data rete a 1 Gbps, 15 ms ritardo da sender a receiver, e voglio spedire pacchetti da 1KB:

$$T_{\text{transmit}} = \frac{8\text{kb/pkt}}{10^{**9} \text{ b/sec}} = 8 \text{ microsec}$$

$$\text{Utilizzo} = U = \frac{\text{Frazione di tempo in cui il canale è occupato}}{\text{}} = \frac{8 \text{ microsec}}{30.016 \text{ msec}} = 0.00015$$

- 1KB ogni 30 msec -> 33kB/sec su una rete a 1 Gbps!!!
- Il protocollo di livello trasporto limita troppo le prestazioni del sistema. Occorre trovare una soluzione più valida, e pensare ad eliminare il principio Stop & Wait in favore dei protocolli a Pipeline.

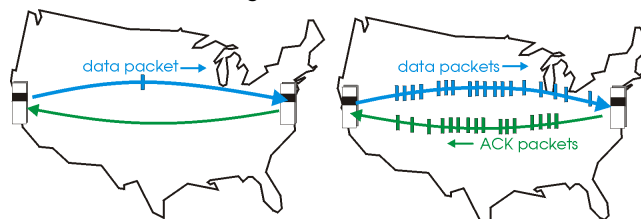
Copyright © Luciano Bononi 2004 (some figure credits to Kurose, Ross, Internet e reti di calcolatori)

3a-27

## Protocolli a Pipeline

**Pipelining:** sender trasmette più di un segmento prima di ricevere il riscontro del primo di essi...

- Quindi non basta più numerare 0 e 1 i pacchetti
- Serve anche buffering su sender e receiver



(a) a stop-and-wait protocol in operation

(b) a pipelined protocol in operation

- ❑ Esistono due classi di protocolli a pipeline: *go-Back-N*, *selective repeat*

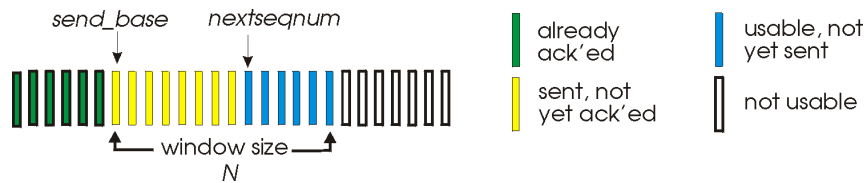
Copyright © Luciano Bononi 2004 (some figure credits to Kurose, Ross, Internet e reti di calcolatori)

3a-28

## Go-Back-N (o Sliding Window)

### Sender:

- Nell'header di ogni pacchetto c'è un numero di sequenza su  $k$  bit
- "finestra" di max  $N$  (per controllo di flusso), consecutivi pacchetti possono essere spediti



- Se ricevo ACK( $n$ ): vale per tutti i pacchetti fino a  $n$  - "ACK cumulativo"
- C'è un timer per ogni pacchetto in sospeso (timeout)
  - Se scatta *timeout*( $n$ ): ritrasmetto tutti i pacchetti da  $n$  in poi appartenenti alla finestra scorrevole (sliding window)

Copyright © Luciano Bononi 2004 (some figure credits to Kurose, Ross, Internet e reti di calcolatori)

3a-29

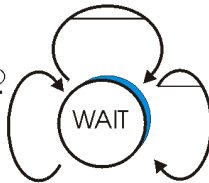
## GoBackN: lato sender

```

rdt_send(data)
if (nextseqnum < base+N) {
    compute chksum
    make_pkt(sndpkt(nextseqnum),nextseqnum,data,chksum)
    udt_send(sndpkt(nextseqnum))
    if (base == nextseqnum)
        start_timer
    nextseqnum = nextseqnum + 1
}
else
    refuse_data(data)
    
```

```

rdt_rcv(rcv_pkt) && notcorrupt(rcvpkt)
base = getacknum(rcvpkt)+1
if (base == nextseqnum)
    stop_timer
else
    start_timer
    
```



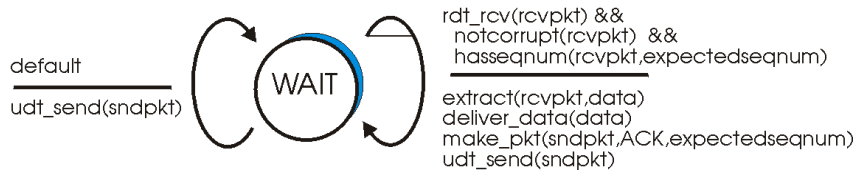
```

timeout
start_timer
udt_send(sndpkt(base))
udt_send(sndpkt(base+1))
.....
udt_send(sndpkt(nextseqnum-1))
    
```

Copyright © Luciano Bononi 2004 (some figure credits to Kurose, Ross, Internet e reti di calcolatori)

3a-30

## GoBackN: lato receiver



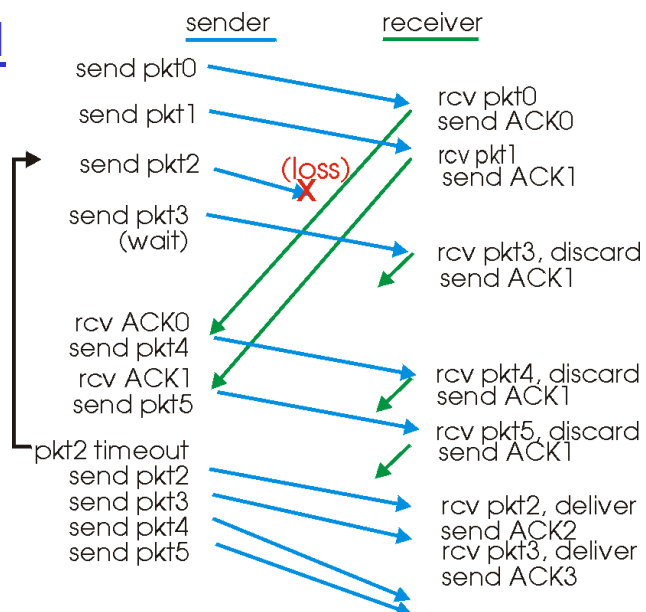
### Il receiver è semplice:

- ACK: spedisce sempre ACK relativo al pacchetto con numero di sequenza maggiore, prima del quale tutti i pacchetti sono stati ricevuti
  - Può spedire in modo testardo ack duplicati per ribadire che sta ricevendo pacchetti successivi a quello mancante
  - Deve ricordare solo il numero di sequenza del prossimo pacchetto che gli manca, in ordine = **expectedseqnum**
- I pacchetti ricevuti correttamente, ma fuori ordine
  - Sono eliminati (non bufferizzato) -> **perchè no?**
  - Invia sempre e solo Ack del pacchetto più in alto nella sequenza **completa dei pacchetti ricevuti**

Copyright © Luciano Bononi 2004 (some figure credits to Kurose, Ross, Internet e reti di calcolatori)

3a-31

## GoBackN in azione



Copyright © Luciano Bononi 2004 (some figure credits to Kurose, Ross, Internet e reti di calcolatori)

3a-32



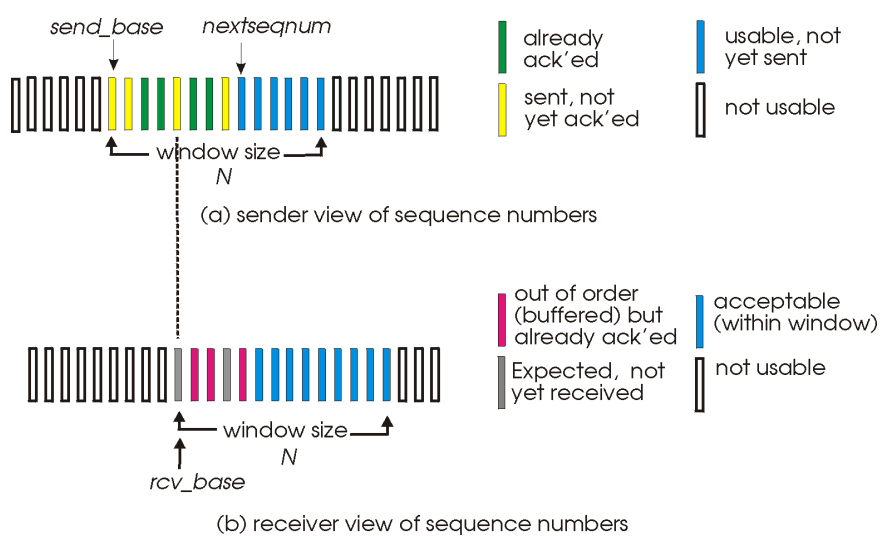
## Selective Repeat

- receiver *spedisce ACK individuali di ogni pacchetto ricevuto (no ack cumulativi)*
  - Bufferizza i pacchetti ricevuti fuori sequenza per poi mandare segmenti completi all'applicazione
- sender rispedisce solo i pacchetti per i quali non ha ricevuto ACK (e non tutta la finestra)
  - C'è un timer per ogni pacchetto in sospeso
- La finestra del sender esiste ancora
  - N numeri di sequenza consecutivi
  - Limita il numero massimo di pacchetti in sospeso (controllo di flusso)
- Esiste anche una finestra sul receiver

Copyright © Luciano Bononi 2004 (some figure credits to Kurose, Ross, Internet e reti di calcolatori)

3a-33

## Selective repeat: finestre di sender e receiver



Copyright © Luciano Bononi 2004 (some figure credits to Kurose, Ross, Internet e reti di calcolatori)

3a-34

## Selective repeat

—sender—

**Dati dall'applicazione :**

- Se la finestra non è tutta usata spedisce il pacchetto

**timeout(n):**

- Rispedisci pacchetto n e riavvia il timer(n)

**ACK(n) ricevuto**  
appartenente alla finestra  
[sendbase, sendbase+N]:

- Marca pacchetto n ricevuto
- se n era il primo pacchetto senza ACK, avanza la finestra fino al prossimo pacchetto senza ACK

—receiver—

**Se riceve pacchetto n appartenente alla finestra del receiver**  
[rcvbase, rcvbase+N-1]

- spedisce ACK(n)
- Se è fuori ordine: inserisci pacchetto in buffer
- Se la finestra contiene una serie di pacchetti in ordine, passali all'applicazione e avanza la finestra sul primo pacchetto mancante

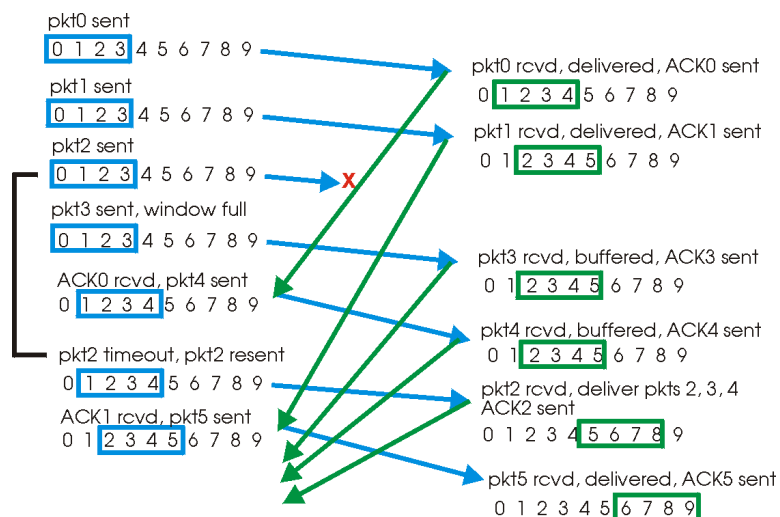
**Se pacchetto n duplicato**

- ACK(n)

Copyright © Luciano Bononi 2004 (some figure credits to Kurose, Ross, Internet e reti di calcolatori)

3a-35

## Selective repeat in action



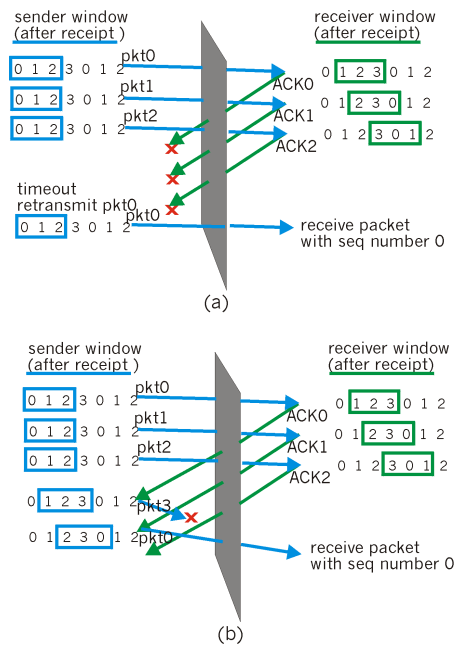
Copyright © Luciano Bononi 2004 (some figure credits to Kurose, Ross, Internet e reti di calcolatori)

3a-36

Selective repeat:  
problema con la  
dimensione della finestra

esempio

- seq num: 0, 1, 2, 3
- Dimensione finestra=3
  
- receiver non vede differenza nei due scenari! E quindi passa un duplicato all'applicazione come se fosse il dato atteso (errore!)
- La dimensione della finestra non deve essere multiplo o sottomultiplo del contatore di sequenza



Copyright © Luciano Bononi 2004 (some figure credits to Kurose, Ross, Internet e reti di calcolatori)

3a-37