

II Livello Trasporto

Obiettivi:

- r Comprendere i principi costitutivi dei servizi del livello trasporto:
 - m multiplexing/demultiplexing
 - m Trasf. dati affidabile
 - m controllo flusso
 - m controllo congestione
- r Realizzazione in Internet

Panoramica:

- r Servizi del livello trasporto
- r multiplexing/demultiplexing
- r trasporto connectionless : UDP
- r principi del trasferimento dati affidabile
- r Trasporto connection-oriented: TCP
 - m trasferimento affidabile
 - m controllo del flusso
 - m Management della connessione
- r principi controllo congestione
- r controllo congestione del TCP

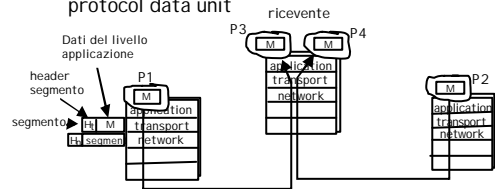
LivelloTrasporto 3a-1

Multiplexing/demultiplexing

Segmento - unità di dati scambiati tra entità di livello trasporto

m TPDU: transport protocol data unit

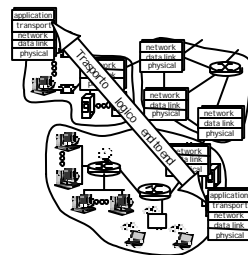
Demultiplexing: consegna dei segmenti ricevuti ad opportuni processi di livello applicazione



LivelloTrasporto 3a-4

Servizi e protocolli di Trasporto

- r Forniscono una *comunicazione logica* tra i processi applicativi in esecuzione su host differenti
- r I protocolli di trasporto agiscono sugli end systems
- r Servizi di trasporto e di rete:
- r **Livello network:** trasferimento dati tra end systems
- r **Livello trasporto:** trasferimento dati tra processi
 - m Si appoggia su, e migliora, i servizi di livello network



LivelloTrasporto 3a-2

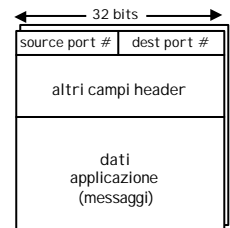
Multiplexing/demultiplexing

Multiplexing:

raccolta dati dai processi di applicazione, imbustamento dati con header (poi usati per il demultiplexing)

multiplexing/demultiplexing:

- r Basati sui numeri di porta e gli indirizzi IP del mittente e del destinatario
 - m porte di source, dest in ogni segmento
 - m Porte well-known per applicazioni specifiche



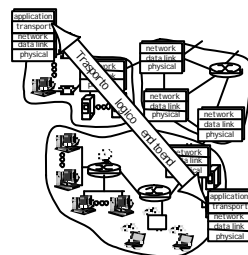
Formato del segmento TCP/UDP

LivelloTrasporto 3a-5

Protocolli del Livello Trasporto

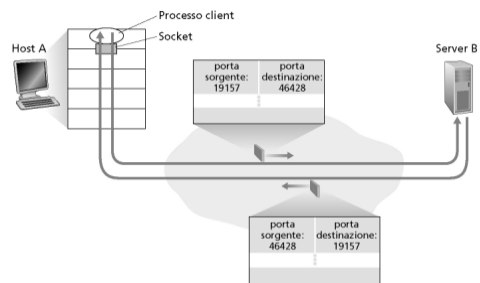
Servizi di trasporto Internet:

- r Consegna affidabile e ordinata di tipo unicast (TCP)
 - m congestione
 - m Controllo del flusso
 - m setup della connessione
- r Consegna inaffidabile ("best-effort"), disordinata di tipo unicast o multicast: UDP
- r servizi non disponibili:
 - m real-time
 - m garanzia sulla banda
 - m multicast affidabile



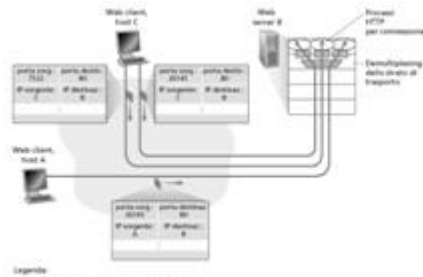
LivelloTrasporto 3a-3

Multiplexing/demultiplexing: esempi



LivelloTrasporto 3a-6

Multiplexing/demultiplexing: esempi



Livello Trasporto 3a-7

UDP checksum

Obiettivo: rilevare "errori" (per es., bit invertiti) nei segmenti trasmessi

Mittente:

- r contenuti del segmento trattati come sequenze di interi a 16-bit
- r checksum: somma (in complemento a 1) dei contenuti del segmento
- r Il mittente inserisce il valore del checksum nel campo checksum del pacchetto UDP

Destinatario:

- r Calcola il checksum del segmento ricevuto
- r Verifica se il checksum calcolato è uguale al valore del campo checksum:
 - m NO - errore rilevato
 - m YES - nessun errore rilevato (non vuol dire che non ci siano errori ...)

LivelloTrasporto3a-10

UDP: User Datagram Protocol [RFC 768]

- r Il protocollo di trasporto di Internet "senza fronzoli"
- r Servizio "best effort", i segm. UDP possono essere:
 - m persi
 - m Consegnati all'appl. fuori ordine
- r *connectionless*:
 - m non c'è handshaking tra mittente e destinatario UDP
 - m ogni segmento UDP viene trattato separatamente dagli altri

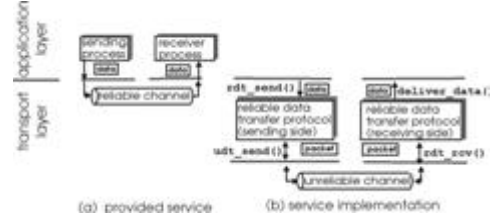
Perchè esiste UDP?

- r Non occorre stabilire una connessione (meno ritardi)
- r semplicità: non occorre gestire la connessione
- r L'header del segmento è piccolo
- r Non c'è controllo della congestione: l'UDP può spedire dati tanto velocemente quanto lo si desidera

LivelloTrasporto 3a-8

Principi di trasferimento affidabile

- r importante per i livelli applicazione, trasporto, data link
- r È nella top-10 delle problematiche di rete importanti!

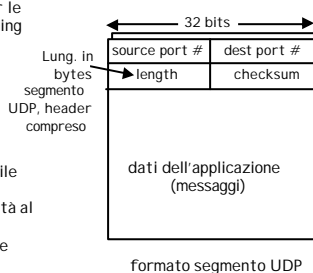


- r Le caratteristiche di un canale inaffidabile determinano la complessità di un protocollo per il trasferimento affidabile dei dati (rdt - reliable data transfer)

LivelloTrasporto3a-11

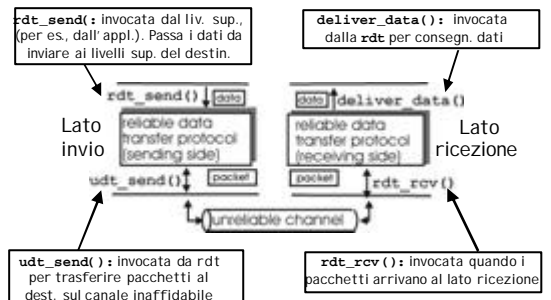
UDP (2)

- r Spesso è utilizzato per le applicazioni con streaming multimedia che sono
 - m loss tolerant
 - m rate sensitive
- r Altri usi UDP:
 - m DNS
 - m SNMP
- r Trasferimento affidabile con UDP: aggiungere il controllo dell'affidabilità al livello applicazione
 - m Recupero dell'errore specifico per l'applicazione!



LivelloTrasporto 3a-9

Reliable data transfer (1)



LivelloTrasporto3a-12

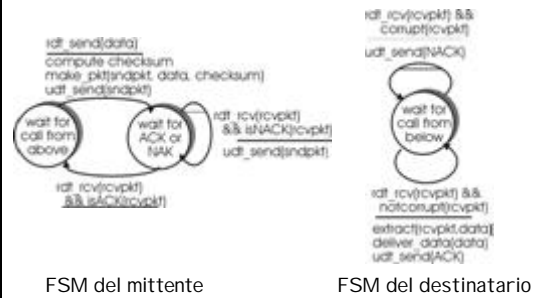
Reliable data transfer: Introduzione

- r Sviluppiamo incrementalmente le parti mittente e destinatario di un protocollo rdt
- r Consideriamo solo trasferimenti monodirezionali
 - m ma le info di controllo vanno in ambedue le direzioni!
- r mittente e destinatario come macchine a stati finiti



LivelloTrasporto3a-13

rdt2.0: specifica delle FSM



FSM del mittente

FSM del destinatario

LivelloTrasporto3a-16

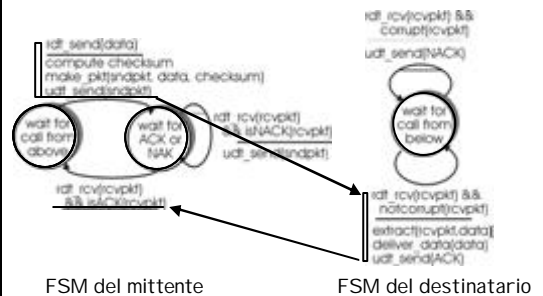
Rdt1.0: trasferim. affidabile su canale affidabile

- r Il canale sottostante è perfettamente affidabile
 - m Non ci sono errori nei bit
 - m Non si perdono pacchetti
- r FSMs separate per mittente e destinatario:
 - m Il mittente invia dati nel canale sottostante
 - m Il destinatario legge dati dal canale sottostante



LivelloTrasporto3a-14

rdt2.0: in azione (nessun errore)



FSM del mittente

FSM del destinatario

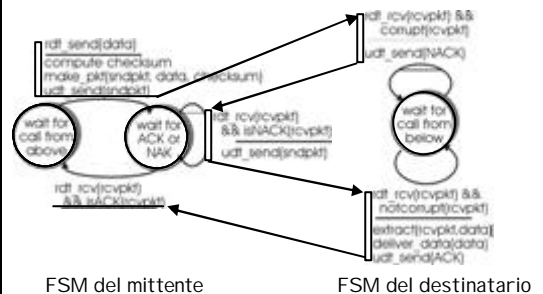
LivelloTrasporto3a-17

Rdt2.0: canale con errori sui bit

- r Il canale può alterare i bits del pacchetto
 - m il checksum UDP usato per rilevare gli errori sui bit
- r la questione: come recuperare gli errori:
 - m *acknowledgements (ACKs)*: il destinatario dice esplicitamente al mittente che il pkt ricevuto è OK
 - m *negative acknowledgements (NAKs)*: il destinatario dice esplicitamente al mittente che il pkt ricevuto ha errori
 - m Il mittente ritrasmette i pkt se riceve un NAK
 - m Un esempio umano che usa ACK e NAK?
- r Nuovi meccanismi nel **rdt2.0** (rispetto a **rdt1.0**):
 - m Rilevamento errori
 - m Feedback del destinatario: msg di controllo (ACK, NAK) dal dest->mitt

LivelloTrasporto3a-15

rdt2.0: in azione (scenario errori)



FSM del mittente

FSM del destinatario

LivelloTrasporto3a-18

rdt2.0

Che succede se si altera un ACK/NAK?

- r Il mitt. non sa cosa è accaduto al dest.!
- r Non può ritrasmettere e basta: possibili duplicati

Che fare?

- r ACK/NAK del mittente e ACK/NAK del dest? Che succedesse si perde un ACK/NAK del mittente?
- r ritrasmettere, ma si possono ritrasmettere pacchetti ricevuti correttamente

Trattamento duplicati :

- r il mitt. aggiunge *sequence number* a ciascun pkt
- r il mitt. ritrasmette il pkt corrente se ACK/NAK corrotti
- r dest. scarta (non consegna ai livelli superiori) i pkt duplicati

stop and wait

il mittente invia un solo pacchetto e poi attende la risposta del destinatario

LivelloTrasporto3a-19

rdt2.1: discussione

Mittente:

- r aggiunta di seq# al pkt
- r bastano due seq# (0,1)
- Perchè?
- r deve controllare se vengono ricevuti ACK/NAK corrotti
- r Il doppio degli stati
 - m lo stato deve "ricordare" se il pkt "corrente" ha seq# 0/1

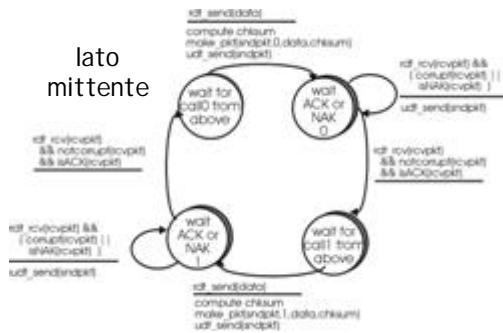
Destinatario:

- r Deve controllare se il pacchetto ricevuto è un duplicato
 - m lo stato indica se il seq # del pacchetto atteso deve essere 0 o 1
- r nota: il dest. non può sapere se il suo ultimo ACK/NAK è stato ricevuto correttamente dal mittente

LivelloTrasporto3a-22

rdt2.1: gestione ACK/NAK corrotti

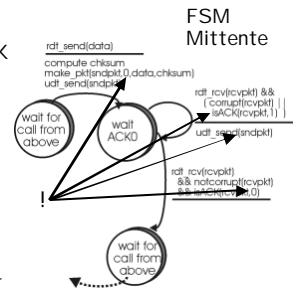
lato mittente



LivelloTrasporto3a-20

rdt2.2: un protocollo NAK-free

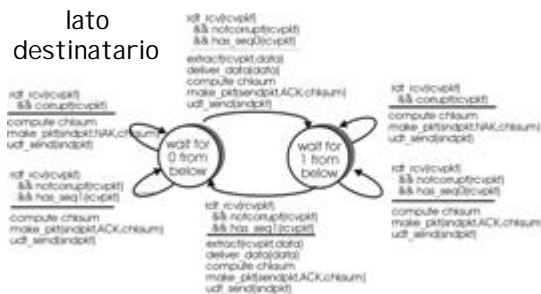
- r stesse funzionalità di rdt2.1, usando solo ACK
- r invece di NAK, il dest. invia ACK per l'ultimo pkt ricevuto OK
 - m il dest. deve includere *esplicitamente* il seq # del pkt di cui fa l'ACK
- r un ACK duplicato provoca dal mittente la stessa azione di un NAK: *ritrasmetti il pkt*



LivelloTrasporto3a-23

rdt2.1: gestione ACK/NAK corrotti

lato destinatario



LivelloTrasporto3a-21

rdt3.0: canali con errori e perdite

Nuova ipotesi: il canale sottostante può anche smarrire pacchetti (sia dati che ACK)

- m checksum, seq #, ACK, ritrasmissioni sono di aiuto ma non bastano

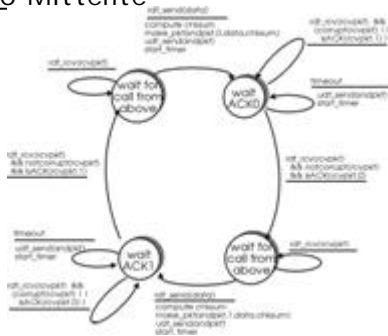
D: come gestire gli smarrimenti?

Un approccio: il mittente attende un "ragionevole" ammontare di tempo per un ACK

- r Ritrasmette se non riceve ACK entro questo tempo
- r se il pkt (o l'ACK) era solo in ritardo (non smarrìto):
 - m la ritrasmissione genera un duplicato ma il seq. # sono già in grado di gestirlo
 - m il dest. Deve specificare il seq # del pkt di cui fa ACK
- r Ci vuole un countdown timer

LivelloTrasporto3a-24

rdt3.0 Mittente



LivelloTrasporto3a-25

Performance del rdt3.0

- rdt3.0 funziona, ma le prestazioni sono molto basse!
- esempio: link da 1 Gbps, ritardo prop. 15 ms e-e, pacchetto 8KB :

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

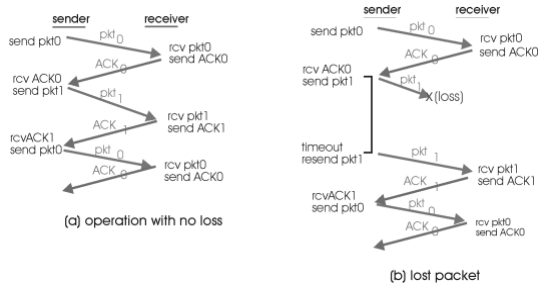
$$\text{Utilizzazione} = U = \frac{\text{frazione del tempo}}{\text{mitt. impegn. spedire}} = \frac{8 \text{ microsec}}{30.016 \text{ msec}} = 0.00015$$

- 1 KB pkt ogni 30 msec -> 33KB/sec throughput su un link 1 Gbps
- Il protocollo limita l'uso delle risorse fisiche (teoricamente potrei andare a 128MB/sec!!!).



LivelloTrasporto3a-28

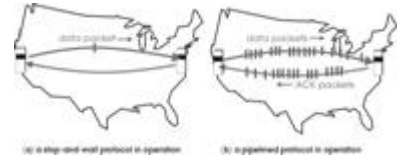
rdt3.0 in azione



LivelloTrasporto3a-26

Protocolli Pipelined

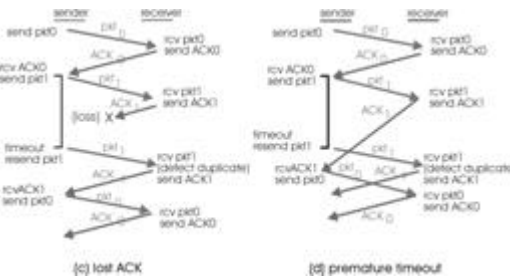
- Pipelining: il mittente accetta l'esistenza di molti pacchetti "in viaggio", non ancora riscontrati
- L'intervallo di numeri di sequenza si deve incrementare
- Bufferizzazione al mittente e/o al destinatario



- Due generiche forme di protocolli pipelined: *go-Back-N*, *selective repeat*

LivelloTrasporto3a-29

rdt3.0 in azione



LivelloTrasporto3a-27

Go-Back-N

Mittente:

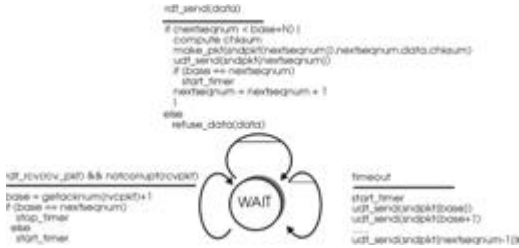
- k-bit seq # nell'header del pkt
- "finestra" consentita di (al massimo) N pacchetti consecutivi non riscontrati



- ACK(n): riscontra tutti i pkts fino a seq # n - "ACK cumulativo"
 - può nascondere ACK duplicati (vedi destinatario)
- Un solo timer
- timeout(n): ritrasmette il pkt n e tutti i pacchetti con seq # maggiori nella finestra

LivelloTrasporto3a-30

GBN: FSM estesa del mittente



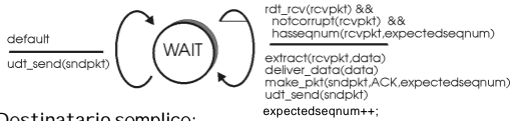
Livello Trasporto 3a-31

Selective Repeat

- r destinatario riscontra *individualmente tutti* i pkt correttamente ricevuti
 - m Bufferizza i pkt, come necessario, per poterli alla fine consegnare in ordine al livello superiore
- r mittente re-invia solo i pkt per i quali non riceve ACK
 - m timer del mittente per ciascun pkt non riscontrato
- r Finestra del mittente
 - m N seq # consecutivi
 - m Limita di nuovo i seq # dei pkt inviati e non riscontrati

Livello Trasporto 3a-34

GBN: FSM estesa del destinatario

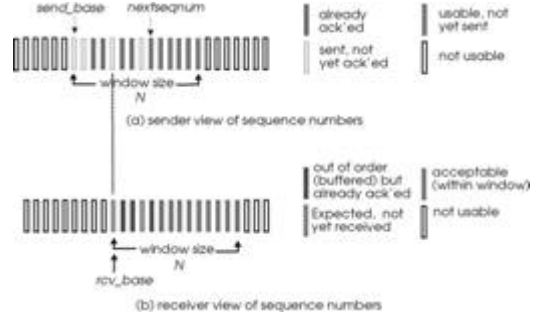


Destinatario semplice:

- r ACK-only: invia sempre un ACK per il pkt correttamente ricevuto col più alto seq # *in-ordine*
 - m può generare ACK duplicati
 - m deve solo ricordare **expectedseqnum**
- r Pkt fuori ordine :
 - m scarta (non bufferizza) -> non c'è buffer al destinatario!
 - m dai un ACK per il pkt con il più alto seq # in ordine

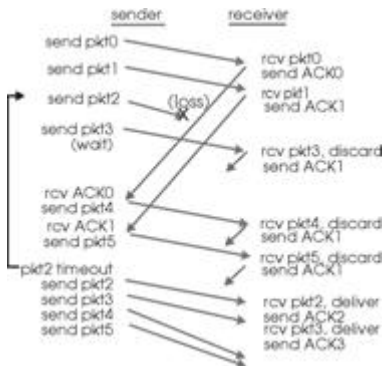
Livello Trasporto 3a-32

Selective repeat: finestre mittente/destin.



Livello Trasporto 3a-35

GBN in azione



Livello Trasporto 3a-33

Selective repeat

Sender

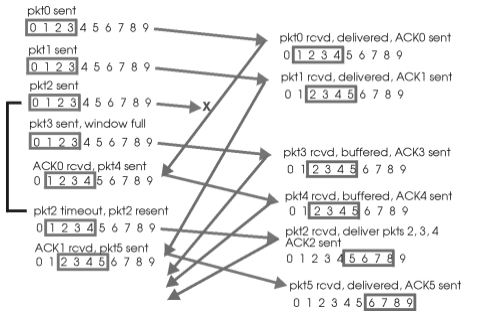
- Dati dall'alto:
 - r Se c'è un seq # disponibile nella finestra, invia pkt
- timeout(n):
 - r reinvia pkt n, riavvia timer
- ACK(n) in [sendbase, sendbase+N):
 - r segna pkt n come ricevuto
 - r if n è il più piccolo unACKed pkt, avanza window base al prossimo unACKed seq #

Receiver

- pkt n in [rcvbase, rcvbase+N-1)
 - r invia ACK(n)
 - r Fuori ordine: buffer
 - r In ordine: consegna, avanza la window al prossimo pkt non ancora ricevuto
- pkt n in [rcvbase-N, rcvbase-1)
 - r ACK(n)
- altrimenti:
 - r ignora

Livello Trasporto 3a-36

Selective repeat in azione



LivelloTrasporto3a-37

Selective repeat: un dilemma

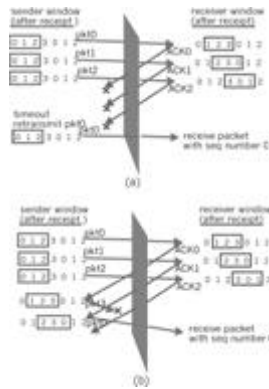
Esempio:

r seq #: 0, 1, 2, 3
r window size=3

- r Il destinatario non distingue le due situazioni!
- r in (a) sbaglia e considera come nuovi pkt i duplicati

Q: che relazione tra la dimensione dei seq # e quella della window?

R: $Win_size \leq \frac{MaxSeq_range}{2}$



LivelloTrasporto3a-38