

PARTE 9

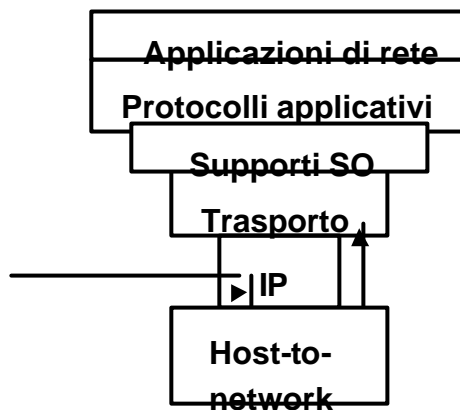
LIVELLO TRASPORTO

- Fondamenti teorici

Dove ci troviamo?

- Abbiamo introdotto i termini ed i concetti fondamentali del corso
- Abbiamo trattato il livello IP ed il routing
- Abbiamo analizzato il DNS
- Abbiamo accennato al livello h2n

DA FARE:



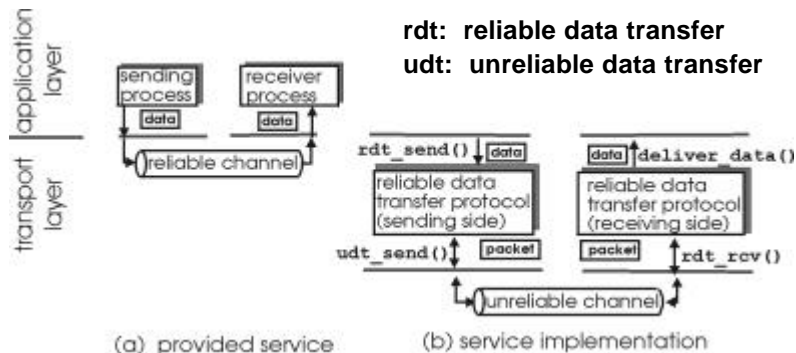
Livello 4 (*transport*)

- Il livello trasporto estende il servizio di consegna con impegno proprio del protocollo IP tra due host terminali ad un servizio di consegna a due processi applicativi in esecuzione sugli host
- Esempi di protocolli *transport*
 - UDP (*User Datagram Protocol*)
 - TCP (*Transmission Control Protocol*): offre un livello di trasporto affidabile e orientato alla connessione su di un canale trasmissivo inaffidabile quale Internet

Problema del trasferimento affidabile

Analisi del **trasferimento affidabile di dati su infrastruttura inaffidabile** (→ contesto più generale del TCP)

- il livello di inaffidabilità del canale determina la complessità del protocollo
- problema comune anche a livello 2 e livello 7 (applicazioni)
- uno dei problemi principali nel networking!



Diversi protocolli per diversi canali

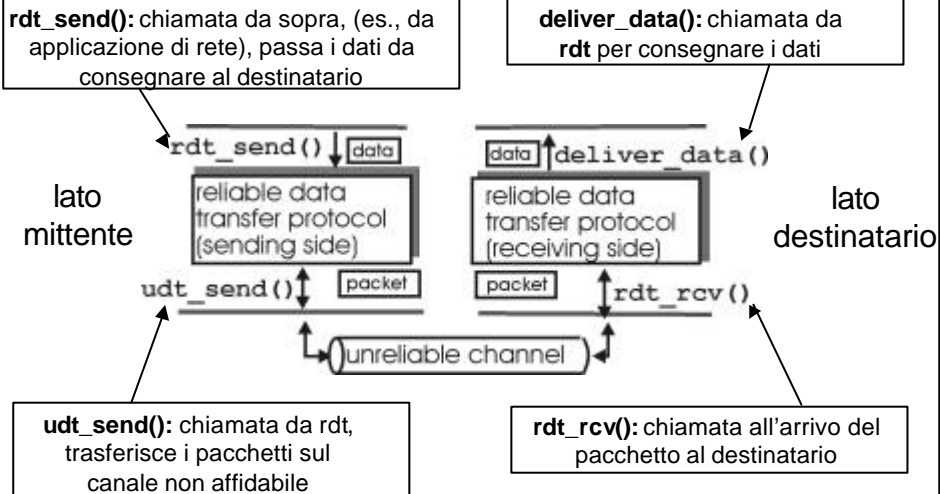
1. Protocollo *rdt1.0*: Trasferimento su canale completamente affidabile
2. Protocollo *rdt2.x*: Trasferimento su un canale con errore sui bit
 - 2.0 versione base
 - 2.1 risolve i problemi di duplicazione
 - 2.2 evita due tipi di ACK
3. Protocollo *rdt3.0*: Trasferimento su un canale con errore sui bit e perdita di pacchetti

Parte 9

Modulo 1: Protocolli su canale affidabile

Trasferimento affidabile su canale affidabile

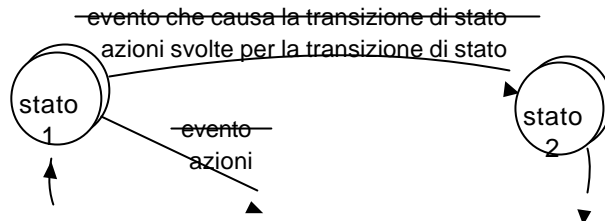
1) Trasferimento su canale completamente affidabile (prot. rdt1.0)



Formalismo: FSM

Uso del formalismo di *macchine a stati finiti* (FSM) per modellare il comportamento del mittente e del destinatario

stato: lo stato successivo è determinato in modo unico dallo stato attuale e dall'evento che occorre

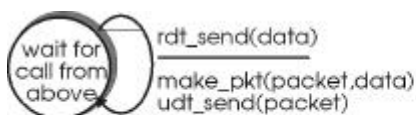


– Sviluppo incrementale delle funzioni svolte dal mittente e dal destinatario

– Ipotesi: trasferimento dei dati unidirezionale, ma informazioni viaggiano in modo bidirezionale

Automi FSM per protocollo *rdt1.0*

- **Canale affidabile:** non ci sono errori di bit, non c'è perdita di pacchetti
- Un automa per il mittente, uno per il destinatario
 - Il mittente invia dati sul canale sottostante
 - Il destinatario legge dati dal canale sottostante



(a) rdt1.0: mittente



(b) rdt1.0: destinatario

Parte 9

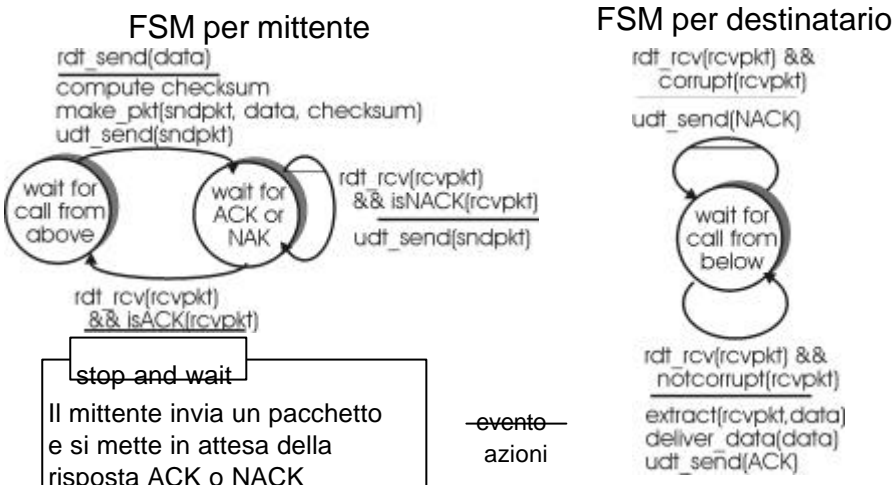
Modulo 2: Protocolli su canale con errore sui bit

Trasferimento affidabile su canale con errore sui bit

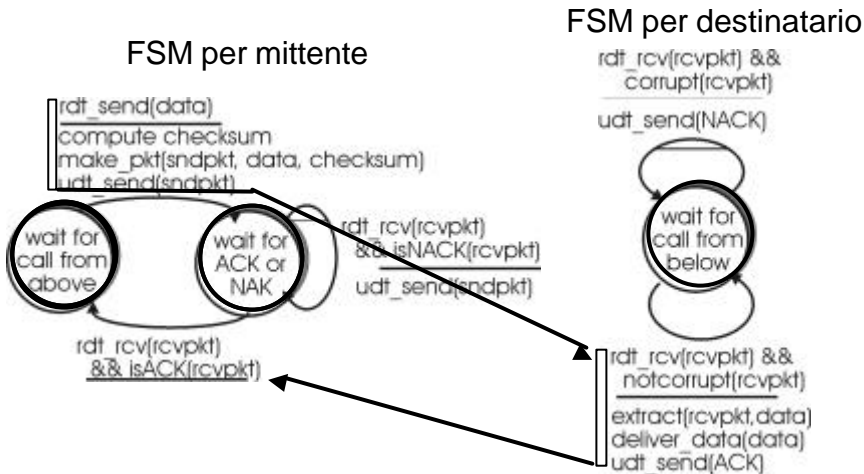
2) Trasferimento su un canale con errore sui bit (protocollo *rdt2.0*)

- Il canale trasmissivo può modificare il valore di un bit
- Possibile modalità per recuperare l'errore
 - *acknowledgement (ACK)*: il destinatario comunica esplicitamente al mittente che il pacchetto ricevuto è OK
 - *acknowledgement negativo (NACK)*: il destinatario comunica esplicitamente al mittente che il pacchetto ricevuto ha un errore
 - il mittente ritrasmette il pacchetto se riceve un NACK
- Nuovi meccanismi in *rdt2.0* rispetto a *rdt1.0*:
 - rilevamento dell'errore (*checksum*)
 - feedback del destinatario: messaggio di controllo (ACK,NACK) inviato al mittente

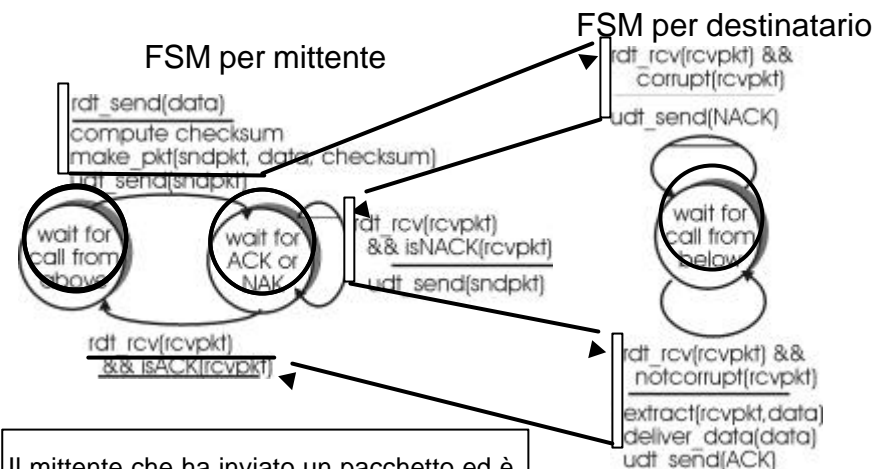
Automi FSM per protocollo *rdt2.0*



Protocollo *rdt2.0* nel caso di trasferimento corretto



Protocollo *rdt2.0* nel caso di trasferimento con errore di bit



Il mittente che ha inviato un pacchetto ed è in attesa della risposta ACK o NACK del destinatario, cicla in questo stato rinviando il pacchetto finché non riceve un ACK

Problemi del protocollo *rdt2.0*

Problema grave del protocollo *rdt2.0*:

- cosa avviene se ad essere danneggiati sono i messaggi di ACK/NACK?
- il mittente non sa se il destinatario ha ricevuto o meno il pacchetto

Prima soluzione: ritrasmettere il pacchetto → non funziona perché vi è rischio di duplicazione del pacchetto

- il protocollo *rdt2.0* non gestisce la duplicazione di pacchetti

Altra soluzione: inviare ACK/NACK in risposta a ACK/NACK del destinatario → non funziona perché se un ACK/NACK si perde, c'è ritrasmissione anche di pacchetti ricevuti correttamente

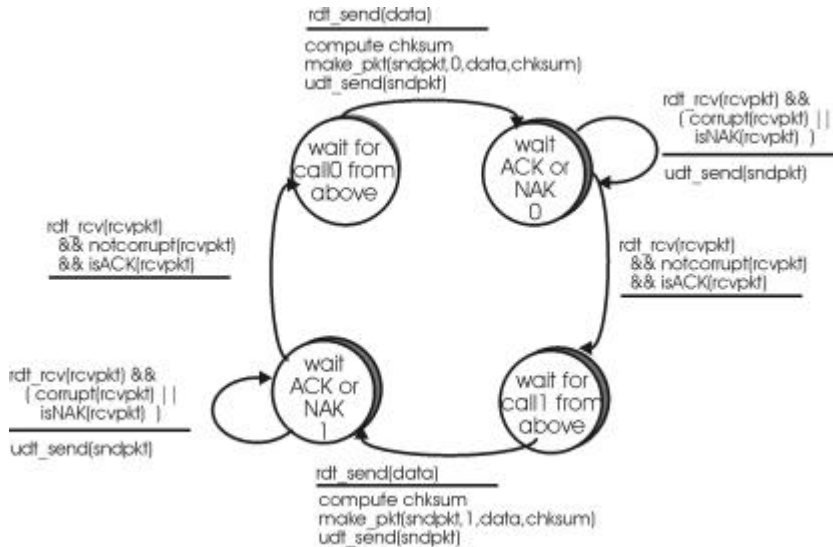
Protocollo *rdt2.1*: possibile miglioramento del protocollo *rdt2.0*

Protocollo *rdt2.1*: Trasferimento su un canale con errore sui bit e gestione dei numeri di sequenza

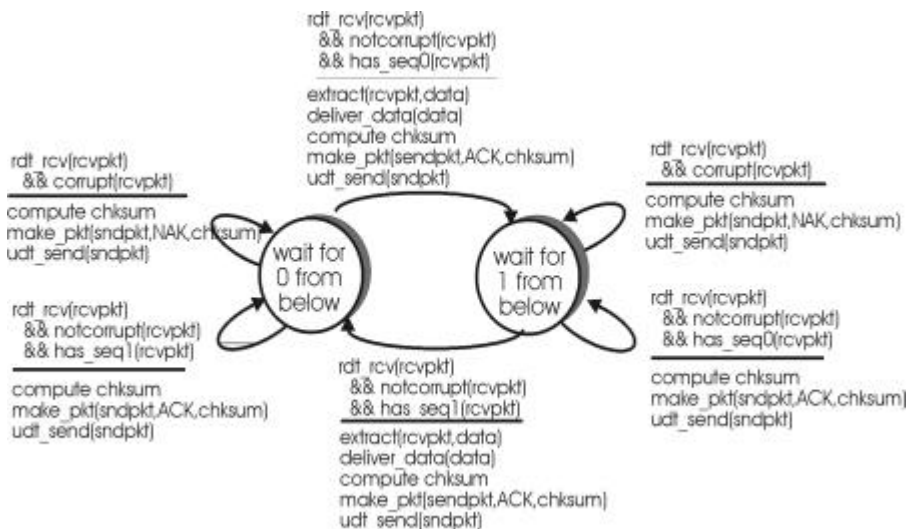
Nuovi meccanismi in *rdt2.1*: numeri di sequenza

- il mittente aggiunge un numero di sequenza a ciascun pacchetto
- il mittente ritrasmette il pacchetto se il messaggio di ACK/NACK viene scartato
- il destinatario scarta (nel senso che non consegna al livello superiore) i pacchetti duplicati

Automa FSM per protocollo *rdt2.1- mittente*



Automa FSM per protocollo *rdt2.1- destinatario*



Analisi del protocollo *rdt2.1*

- **Mittente**

- Numero di sequenza aggiunto a ciascun pacchetto
- Due sequenze di numeri (0,1) sono sufficienti nel caso di ACK/NACK per singolo pacchetto prima di (ri)trasmissione
- Deve verificare se anche ACK/NACK è danneggiato
- Serve il doppio di stati rispetto a *rdt2.0* in quanto lo stato corrente deve “ricordarsi” se il pacchetto corrente ha un numero di sequenza pari a 0 o 1

- **Destinatario**

- Deve controllare se il pacchetto ricevuto è duplicato (lo stato consente di verificare se il numero di sequenza del pacchetto è 0 o 1)
- Il destinatario può non sapere se il suo ultimo ACK/NACK è stato ricevuto correttamente dal mittente

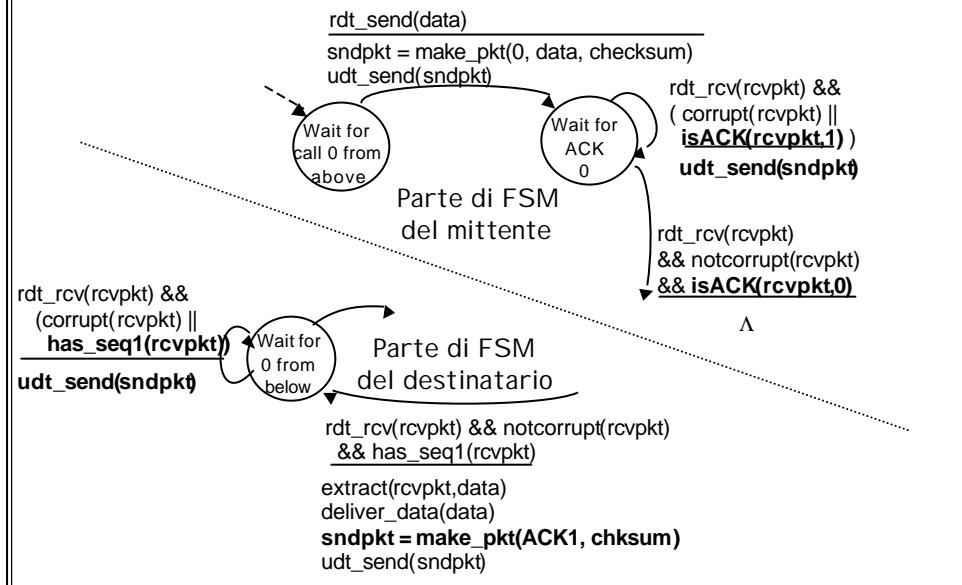
Protocollo *rdt2.2*: possibile miglioramento del protocollo *rdt2.1*

- In *rdt2.1* il destinatario invia sia ACK che NACK al mittente
- NACK viene inviato sia se è stato ricevuto un pacchetto danneggiato sia se è fuori sequenza

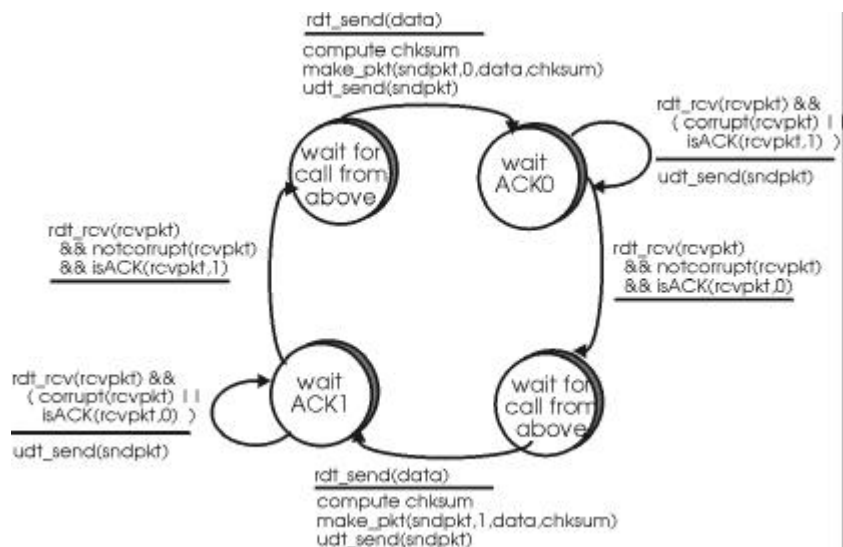
Possibile miglioramento: invece di inviare NACK, inviare un ACK per l'ultimo pacchetto ricevuto correttamente (il destinatario deve includere il numero di sequenza del pacchetto a cui si riferisce l'ACK)

® Se il mittente riceve due ACK per lo stesso pacchetto, sa che il destinatario non ha ricevuto correttamente il pacchetto successivo a quello per il quale è stato ricevuto il doppio ACK e quindi deve agire come un NACK: **ritrasmettere il pacchetto**

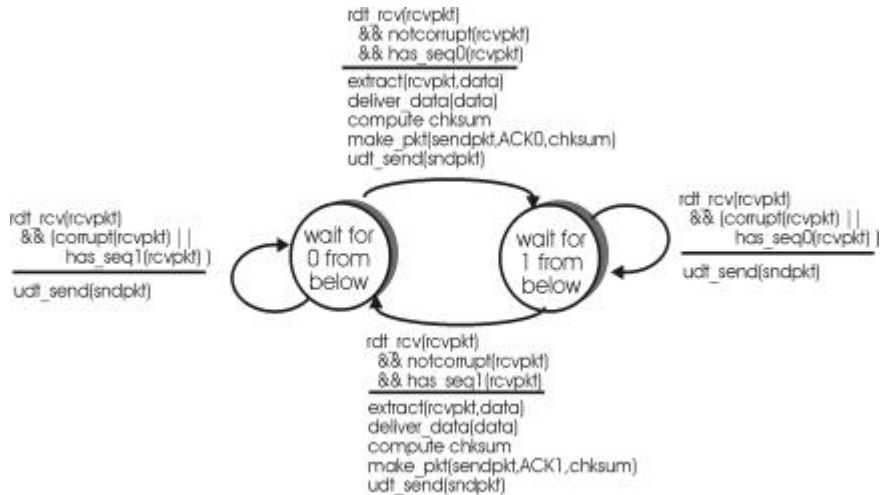
Parti di automi FSM per *rdt2.2*



Automa FSM per protocollo *rdt2.2- mittente*



Automa FSM per protocollo *rdt2.2- destinatario*



Parte 9

Modulo 3: Protocolli su canale con errore sui bit e perdita pacchetti

Nuove ipotesi

- Il canale di trasmissione può causare errori sui bit e perdita di pacchetti (dati, ACK)

Come realizzare una trasmissione affidabile?

- L'uso di checksum, numeri di sequenza dei pacchetti, ACK, ritrasmissioni sono utili, ma non sufficienti
- **SOLUZIONE:** il mittente attende l'ACK per un intervallo di tempo "ragionevole", poi ritrasmette il pacchetto se non ha ricevuto l'ACK

Trasferimento affidabile su canale con errore sui bit e perdita

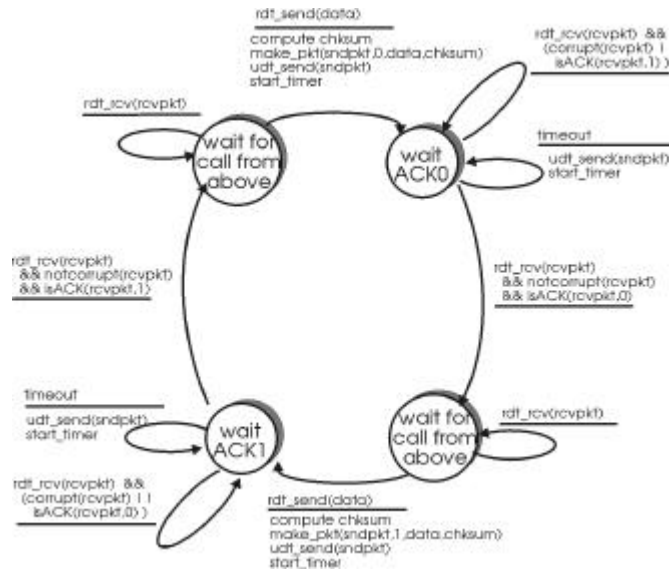
3.0) Trasferimento su un canale con errore sui bit e perdita di pacchetti (protocollo *rdt3.0*)

Soluzione → il mittente attende l'ACK per un intervallo di tempo "ragionevole", poi ritrasmette

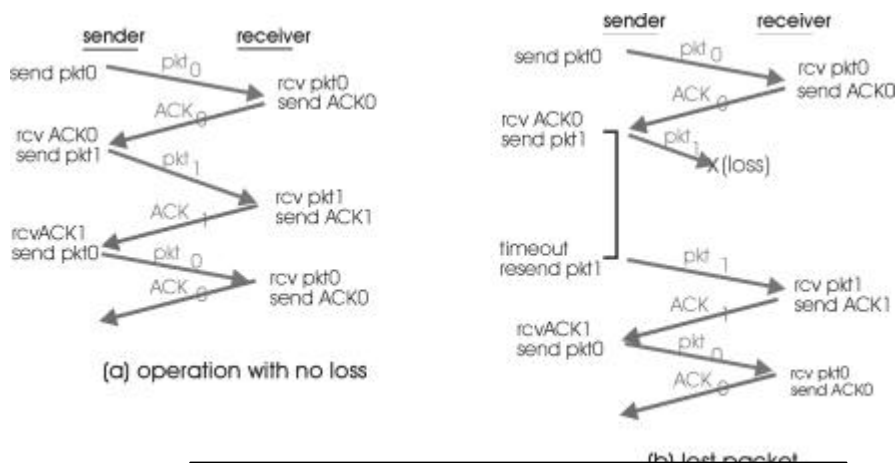
Possibili conseguenze

- 1) Se il pacchetto (o l'ACK) è in ritardo (ma non perso), il pacchetto ritrasmesso sarà duplicato → questo caso è già risolto dall'uso dei numeri di sequenza (per poter funzionare, anche il destinatario deve specificare il numero di sequenza dei pacchetti per cui ha inviato ACK)
- 2) richiede l'uso di un *countdown timer*

Automa FSM per protocollo *rdt3.0 - mittente*



Sequenza operazioni in *rdt3.0*



Il meccanismo send-ack per singolo pacchetto è molto semplice ed affidabile, ma INEFFICIENTE.

Prestazioni

Protocollo rdt3.0: funziona correttamente, ma la prestazione del meccanismo *stop-and-wait* è scadente.

Esempio

- Canale fisico di capacità 1 Gbps (= 10^9 b/sec)
- Ritardo di propagazione (RTT/2) = 15 msec $R = \text{transmission rate}$
- Pacchetti da 1KB (=8 Kb) $L(\text{pkt}) = \text{lunghezza pacchetto}$

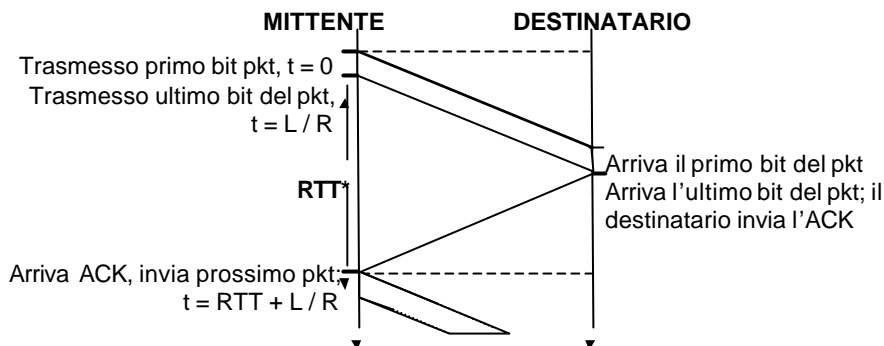
$$T_{\text{trasm/pkt}} = \frac{L(\text{pkt})}{R} = \frac{8\text{Kb}}{10^9 \text{ b/sec}} = 8 \mu\text{sec}$$

$$\text{Utilizzazione} = \frac{L/R}{RTT + L/R} = \frac{0.008}{30 + 0.008} = \frac{0.008 \text{ msec}}{30.008 \text{ msec}} = 0.00027$$

- 1 KB ogni 30 msec → throughput 33KB/sec su un link da 1 Gbps

→ **Evidente uso limitato delle risorse fisiche di rete**

Motivazione delle prestazioni



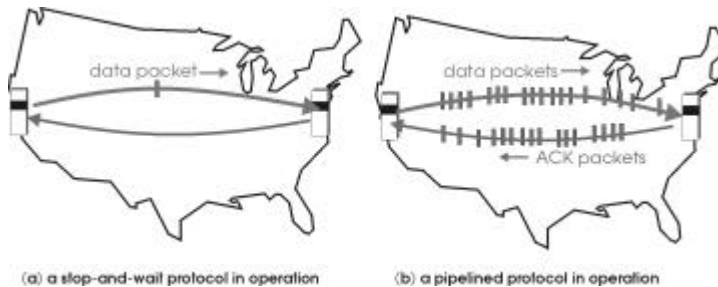
$$U_{\text{sender}} = \frac{L / R}{RTT + L / R} = \frac{.008}{30.008} = 0.00027$$

RTT* = Round Trip Time (se destinatario non rallentato da altri processi)

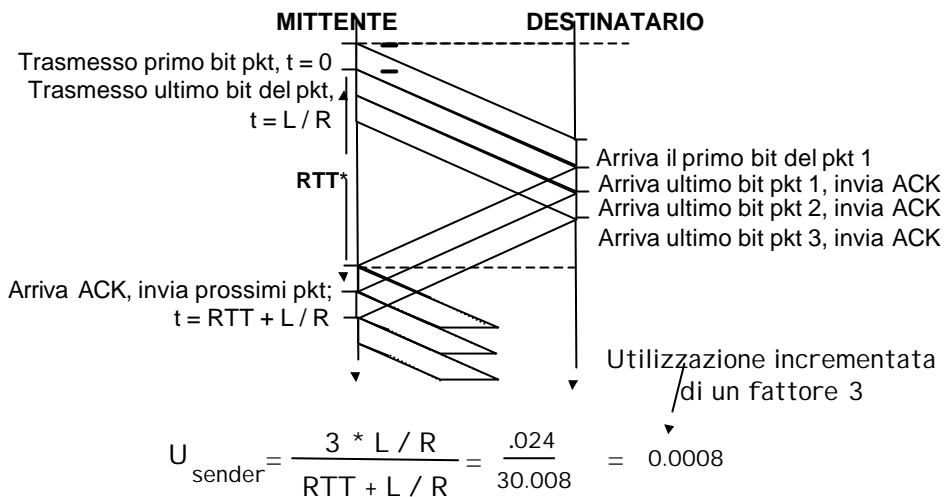
Come combinare affidabilità e efficienza

Pipelining ® il mittente invia un numero multiplo di segmenti prima di ricevere **ACK** ® Serve un buffer

L'intervallo della sequenza di segmenti viene gradualmente aumentato fino a raggiungere la dimensione massima del buffer del mittente e del destinatario



Incremento delle prestazioni



Algoritmi per affidabilità del pipelining

Vi sono due possibili meccanismi per l'affidabilità del pipelining (in caso di mancato ACK):

- **Go-Back-N**
- **Ritrasmissione selettiva**

Algoritmo Go-Back-N

Mittente:

- Numero di sequenza di k bit nell'header del pacchetto
- finestra (**window size**) di max N pacchetti consecutivi, inviabili senza ACK

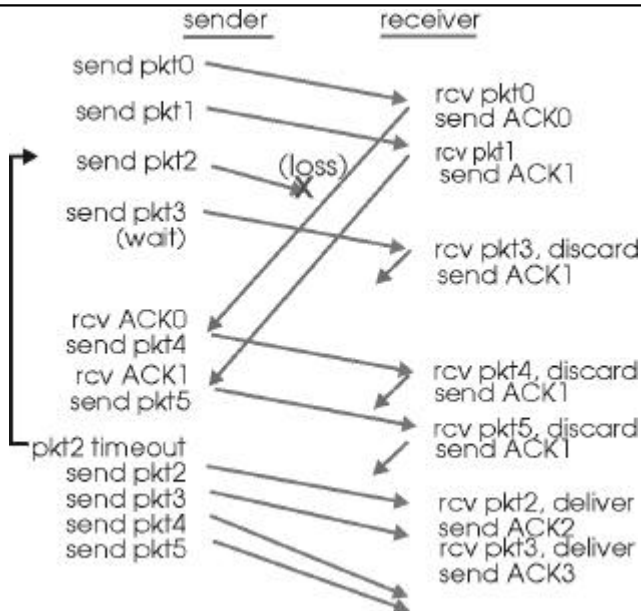


- ACK(n) cumulativo: **ACK per n pacchetti**
- **timeout per singolo segmento**
- **timeout(n): ritrasmetti il pacchetto n e tutti quelli che hanno un numero di sequenza superiore ad n**

Funzionamento algoritmo *Go-Back-N*

Ipotesi:

Finestra=4



Reti di Calcolatori 2003/2

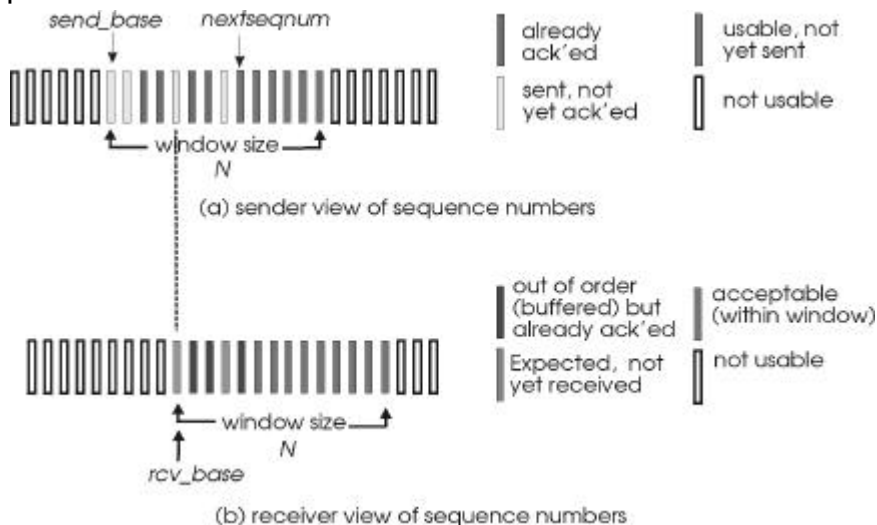
9.35

Algoritmo *Ritrasmissione selettiva*

- **Il destinatario invia ACK di tutti i pacchetti ricevuti correttamente**
 - Bufferizzazione dei pacchetti, per consegnarli secondo la sequenza ordinata al processo applicativo
- **Il mittente ritrasmette soltanto i pacchetti per i quali non ha ricevuto ACK dal destinatario**
 - Gestisce un timeout per ciascun pacchetto
- **Mittente e destinatario gestiscono due finestre di sequenze di N pacchetti consecutivi**
 - La finestra del destinatario avanza di 1 per ogni pacchetto ACK
 - La finestra del mittente avanza di 1 per ogni ACK ricevuto entro il timeout

Finestra mittente e destinatario

Ipotesi: finestra = 14



Algoritmo Ritrasmisione selettiva

mittente

Dati dall'applicazione:

- se c'è un numero di sequenza disponibile nella finestra, invia pacchetto

Timeout(n):

- ritrasmetti pacchetto n , inizializza nuovo timer per n

ACK(n) nella finestra [sendbase, sendbase+ N):

- segna pacchetto n ricevuto
- se n è il minimo pacchetto non ACK, incrementa la finestra fino al successivo pacchetto non ACK

destinatario

Pacchetto n ricevuto in [rcvbase, rcvbase+ N -1]

- invia ACK(n)
- *non ordinato*: metti in buffer
- *ordinato*: consegna alla applicazione; avanza la finestra al pacchetto successivo non ancora ricevuto

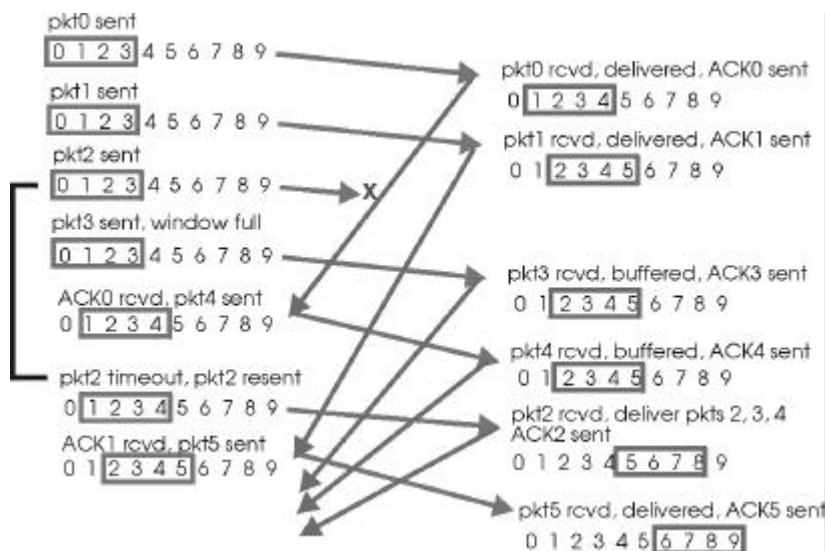
Pacchetto n ricevuto in [rcvbase- N , rcvbase-1]

- ACK(n)

Altrimenti:

- ignora

Funzionamento *Ritrasmissione selettiva*

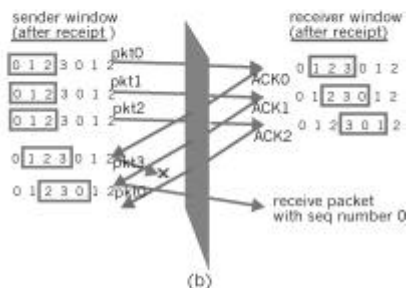
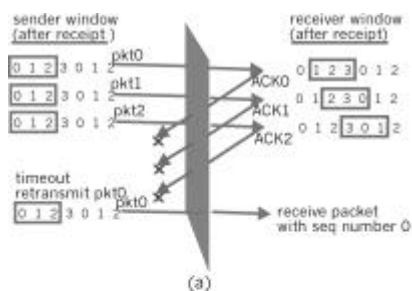


Attenzione al numero di sequenza!

Esempio

- Numeri di sequenza: 0, 1, 2, 3
- Dimensione finestra=3
- Il destinatario non vede differenze nei due scenari a fianco
- Di conseguenza nel caso (a) considera come nuovo un pacchetto duplicato

Ci deve essere una relazione tra dimensione finestra e numeri di sequenza



Prossimo obiettivo

- **Analizzare quali meccanismi utilizza il protocollo TCP per realizzare una “comunicazione affidabile” su di un “canale inaffidabile”**