

Servizio di Trasporto

L'utente, attraverso chiamate a funzioni, richiede il servizio di trasporto *con connessione* o *senza connessione*.

- Il trasporto *con connessione* fornisce un *canale affidabile* su cui scrivere e leggere dati

Modello *client-server*:

Il *server* attiva un processo che rimane in attesa di una richiesta di connessione (**LISTEN**)

Il *client* invia una richiesta di connessione (**CONNECT**)

Aperta la connessione, *Server* e *Client* si scambiano dati con le primitive **SEND** (invia i dati) e **RECEIVE** (attende dati)

Si chiude la connessione con la procedura **DISCONNECT**

I protocolli di Trasporto devono

- *definire la modalità di **indirizzamento** a livello trasporto (su uno stesso host possono essere disponibili più connessioni)*
- *Gestire il **controllo degli errori**, i **numeri di sequenza** e il **controllo di flusso** tra end system collegati attraverso una rete*

I problemi del livello trasporto sono simili a quelli del livello data link, ma il canale fisico è in questo caso l'intera sottorete di comunicazione.

Lungo la sottorete di comunicazione i pacchetti

**possono essere trattenuti nei router e consegnati dopo diversi secondi,*

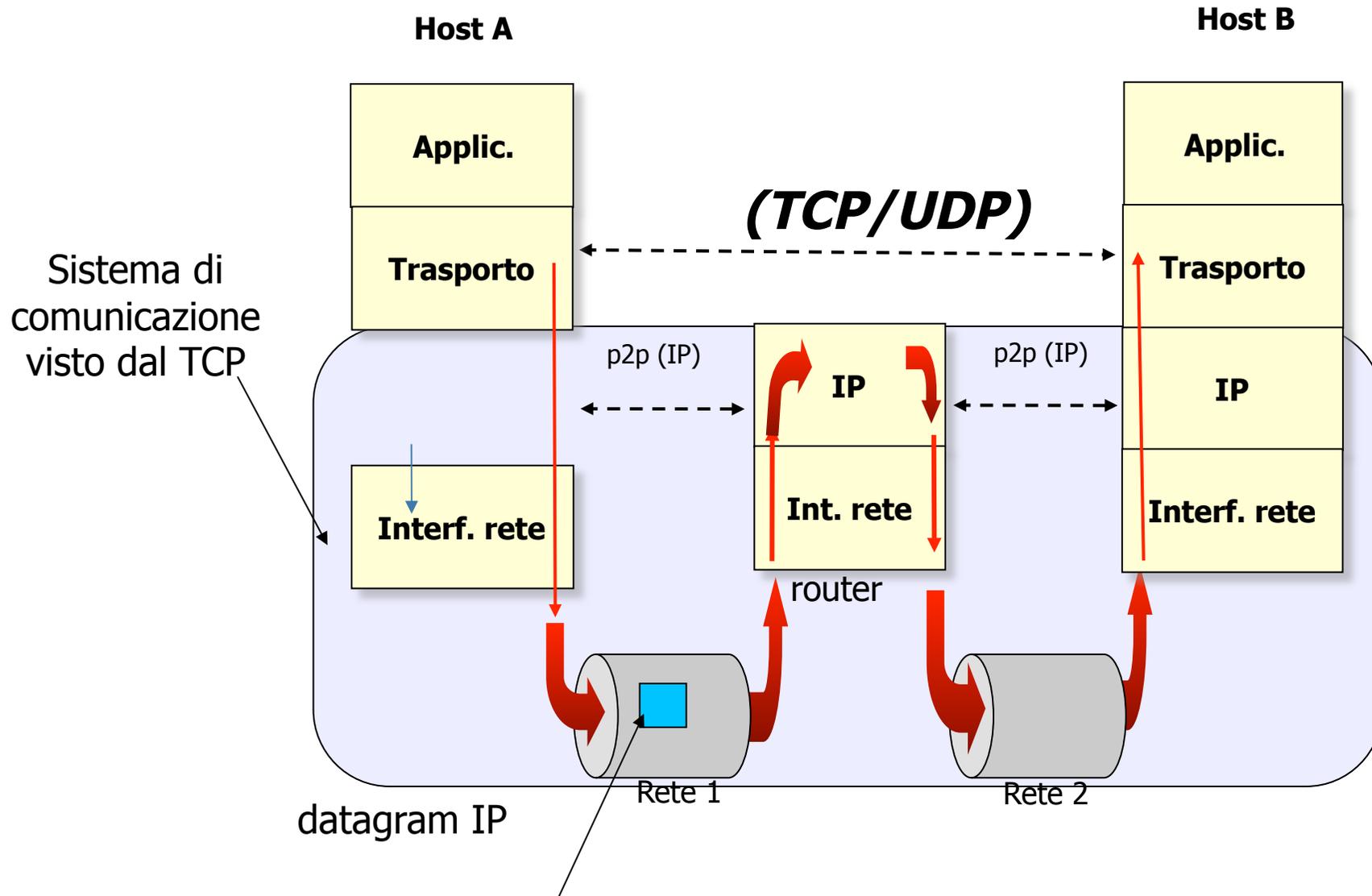
**possono seguire strade diverse ed arrivare in ordine diverso da quello di trasmissione)*

TCP e *UDP*

TCP e *UDP* sono due Protocolli di trasporto definiti su IP

- *Transmission Control Protocol (TCP)*: protocollo di trasporto orientato alla connessione
 - definito in RFC 793, RFC 1122 e RFC 1323
 - progettato per fornire un flusso affidabile end-to-end su una internet inaffidabile
- *User Data Protocol (UDP)*: protocollo senza connessione
 - descritto in RFC 768
 - permette di inviare datagram IP senza stabilire una connessione

TCP/IP



Funzionalità del TCP

Trasmissione

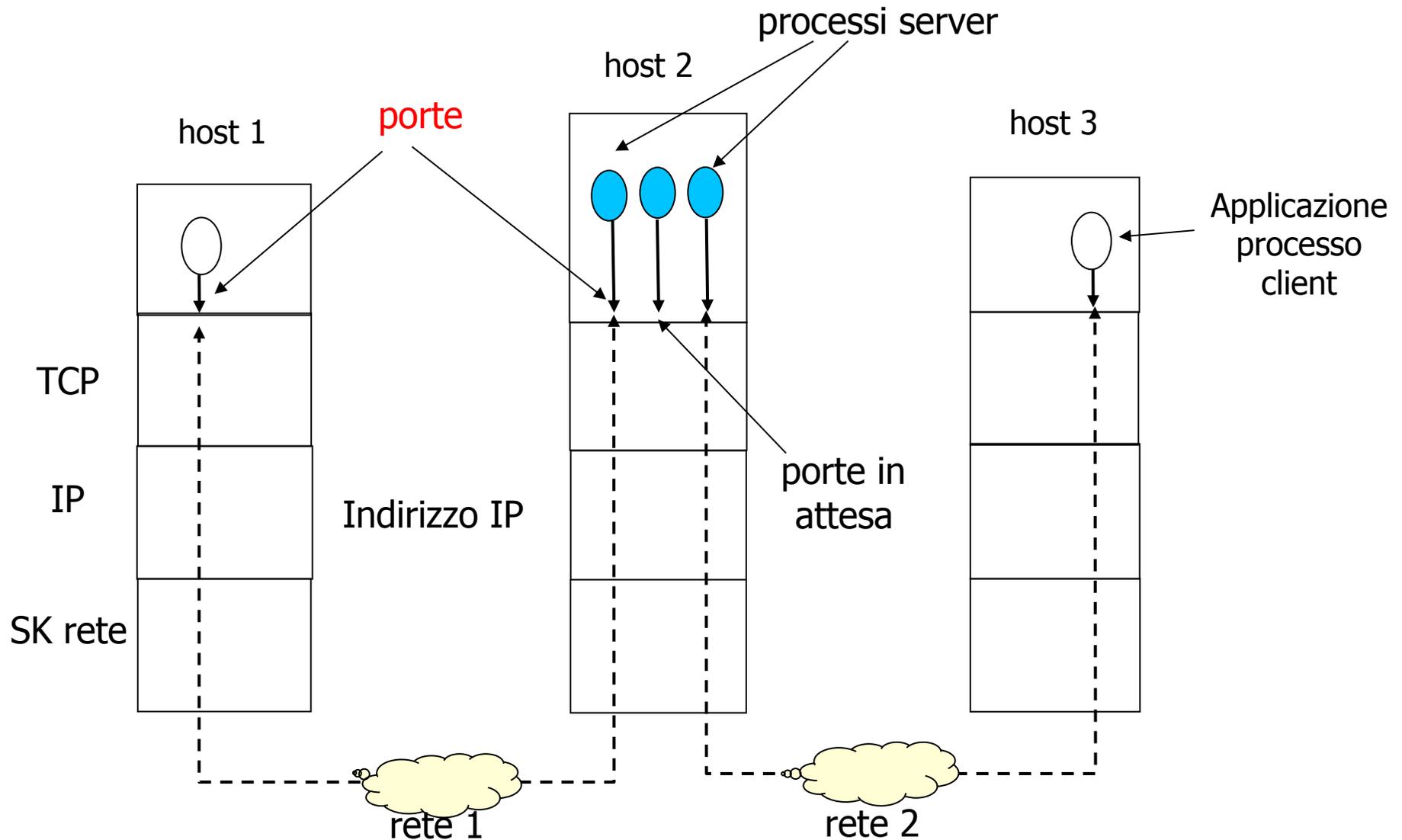
- Riceve un flusso di dati dall'applicazione
- Li organizza in unità lunghe al massimo 64Kb
 - Eventualmente bufferizza i dati prima di spedire il pacchetto (es. input da tastiera)
- Spedisce le unità di dati come datagram IP

Ricezione

- Riceve i datagram IP
- Ricostruisce e consegna all'applicazione la sequenza corretta del flusso di byte originale

Il TCP deve ritrasmettere i datagram non ricevuti e riordinare quelli arrivati in ordine sbagliato.

Un indirizzo di trasporto deve identificare l'host ed il numero di port sull'host



Socket: concetto introdotto in UNIX BSD

IP address dell'host + **numero di porta** (a 16 bit)

Le comunicazioni TCP richiedono una esplicita **connessione** fra un socket della macchina mittente ed un socket della macchina ricevente

*Le **connessioni** sono identificate con gli **identificatori dei socket dei due lati (socket1, socket2)***

Una volta attivato, un socket è utilizzato come un file

Sono disponibili primitive nei linguaggi di programmazione per aprire e usare socket (C, Java...)

Le porte attive definiscono i servizi TCP disponibili

Per connettersi ad un *servizio* specifico occorre conoscere il *numero di porta* che il sistema *server* ha assegnato al processo (*server*) attivato per quel *servizio*

Le porte inferiori a 256 sono dette *porte ben note (well-known ports)* e sono normalmente utilizzate per servizi standard. In Unix la lista dei servizi e delle porte è nel file */etc/services*

Ad esempio

-
-

Un servizio “standard” può anche essere attivato su una porta diversa

Porte Standard TCP/UDP

```
File Edit Setup Control Window Help
#ident "@(<#>services 1.9 93/09/10 SMI" /* SVr4.0 1.8 */
#
# Network services, Internet style
#
tcpmux          1/tcp
echo            7/tcp
echo            7/udp
discard         9/tcp          sink null
discard         9/udp          sink null
systat          11/tcp         users
daytime         13/tcp
daytime         13/udp
netstat         15/tcp
chargen         19/tcp         ttytst source
chargen         19/udp         ttytst source
ftp-data        20/tcp
ftp             21/tcp
telnet          23/tcp
smtp            25/tcp         mail
time            37/tcp         timserver
time            37/udp         timserver
name            42/udp         nameserver
whois           43/tcp         nickname          # usually to sri-nic
domain          53/udp
domain          53/tcp
hostnames       101/tcp        hostname          # usually to sri-nic
sunrpc          111/udp        rpcbind
sunrpc          111/tcp        rpcbind
ident           113/tcp        auth tap
# Host specific functions
#
tftp            69/udp
--More--<33%>
```

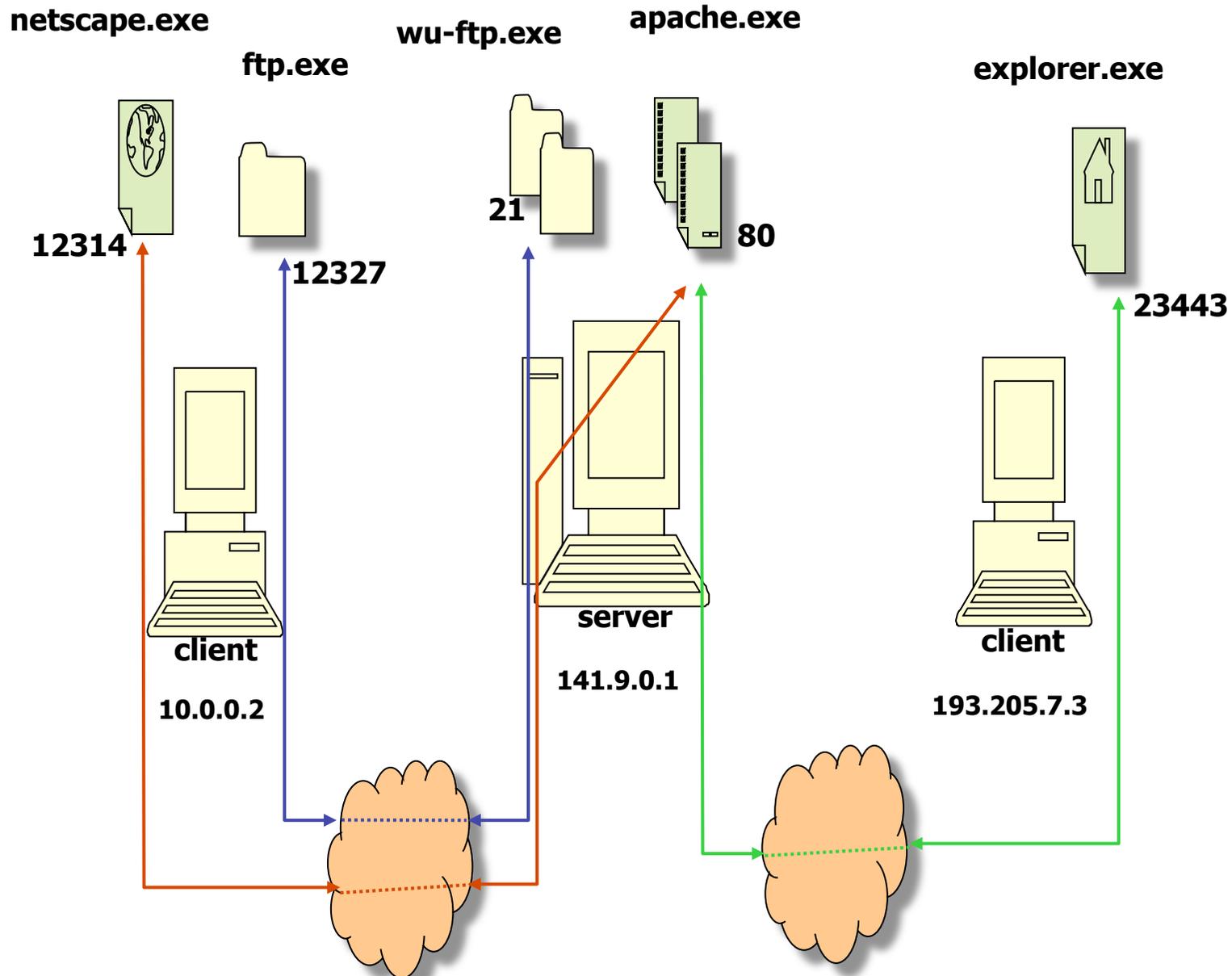
Numero di porta effimero: il client definisce la porta di ogni sua connessione utilizzando numeri in genere elevati e scelti in modo da essere unici sull'host

Ad esempio la coppia di porte assegnate ad una connessione HTTP potrebbe essere:

client port 23443

server port 80

Le connessioni sono punto-punto e full duplex



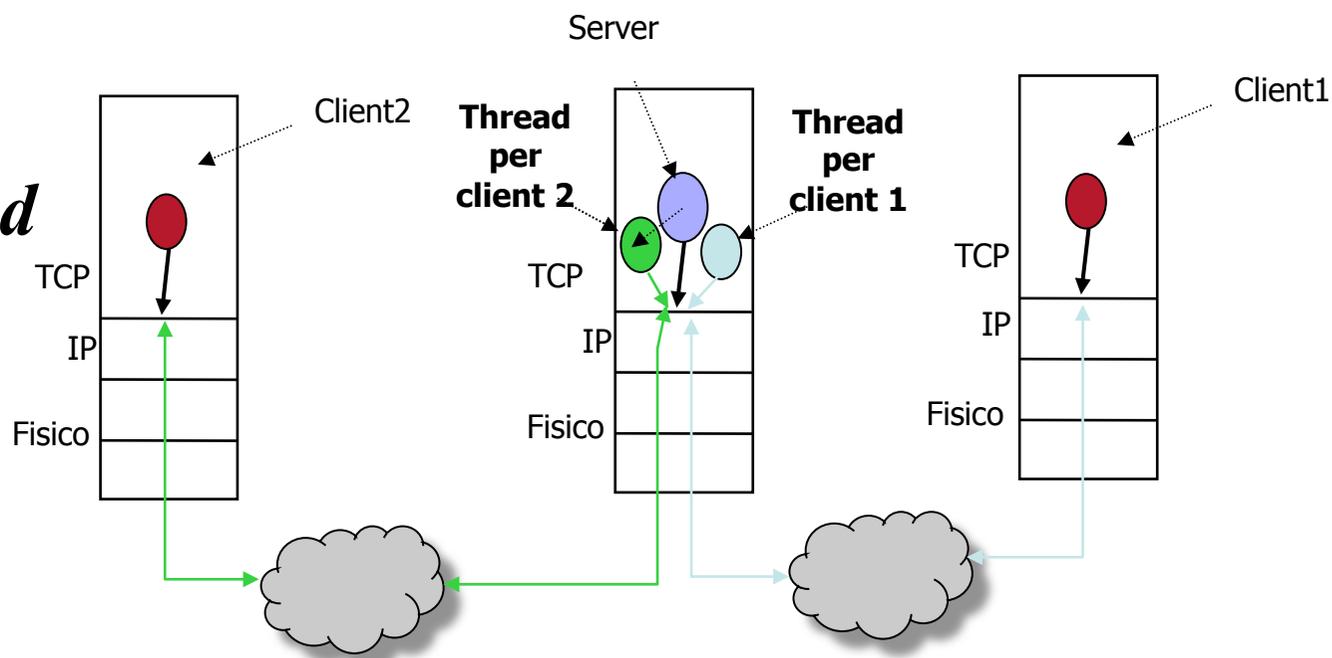
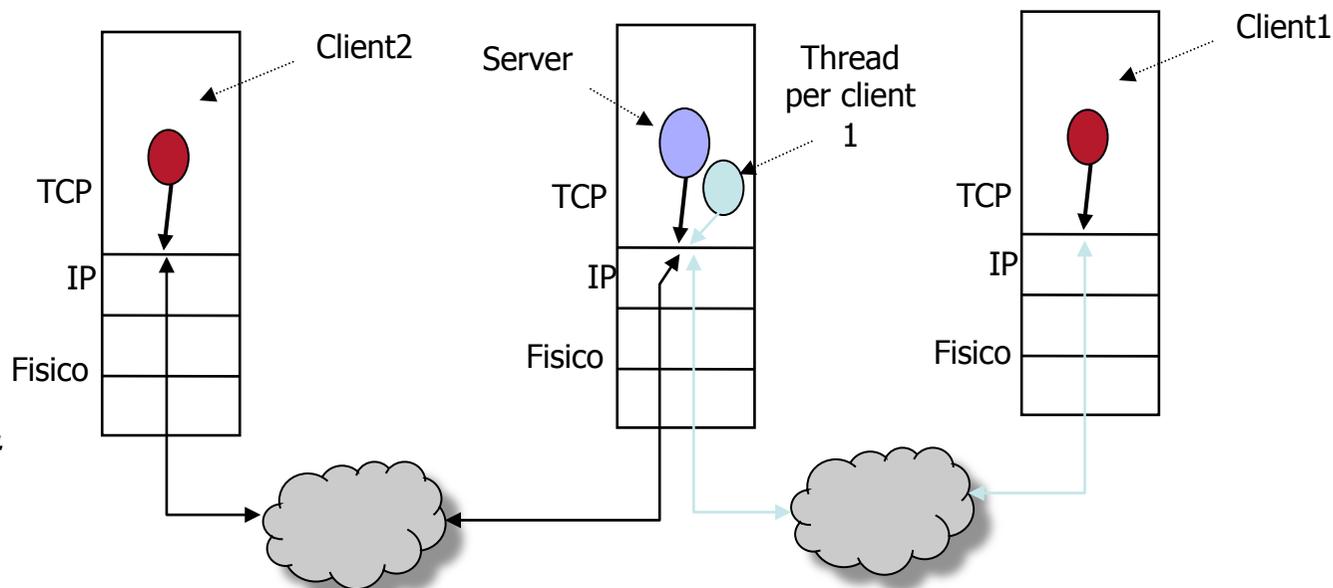
10.0.0.2,12314 – 141.9.0.1,80

10.0.0.2,12327 – 141.9.0.1,21

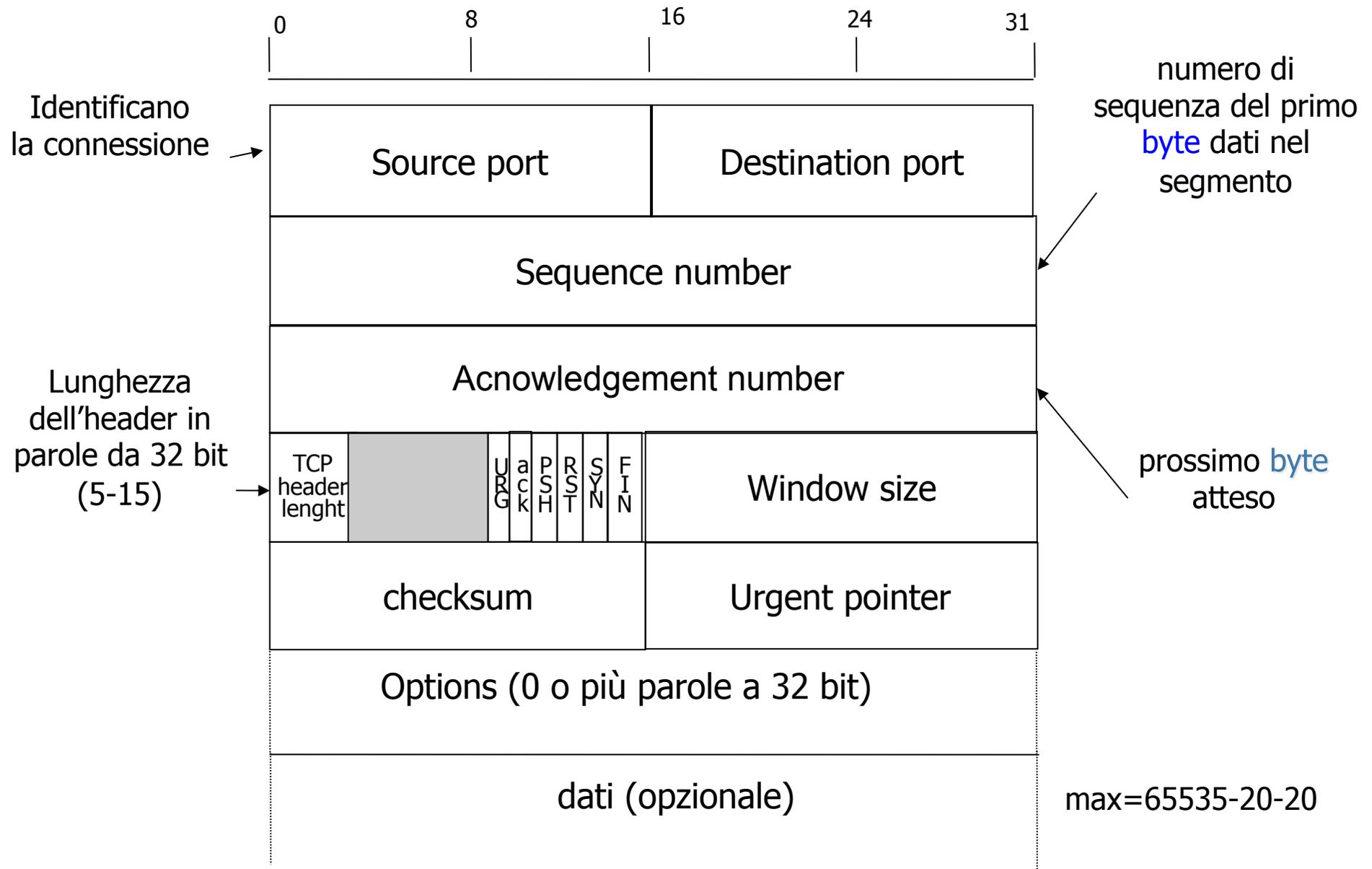
193.205.7.3,23443 – 141.9.0.1,80

Connessioni multiple

Per ogni client che fa una richiesta di connessione, il server genera uno nuovo thread



Ogni *segmento TCP* ha un header di 20 byte più eventuali parti opzionali seguito da 0 o più byte di dati



I flag TCP

Nel segmento TCP sono presenti 6 bit di flag

URG

Se URG assume valore 1, *l'Urgent Pointer indica la posizione, a partire dal numero di sequenza attuale, di dati urgenti* (es. pressione di CTRL-C per interrompere il programma remoto)

ACK

Indica se il campo Acknowledgement number è valido

PSH

Indica dati di tipo PUSH ovvero si richiede di consegnare subito i dati senza bufferizzarli

I flag TCP [2]

○ *RST*

Richiesta di reinizializzazione di una connessione diventata instabile. Viene anche usato per rifiutare un segmento non valido o l'apertura di una connessione

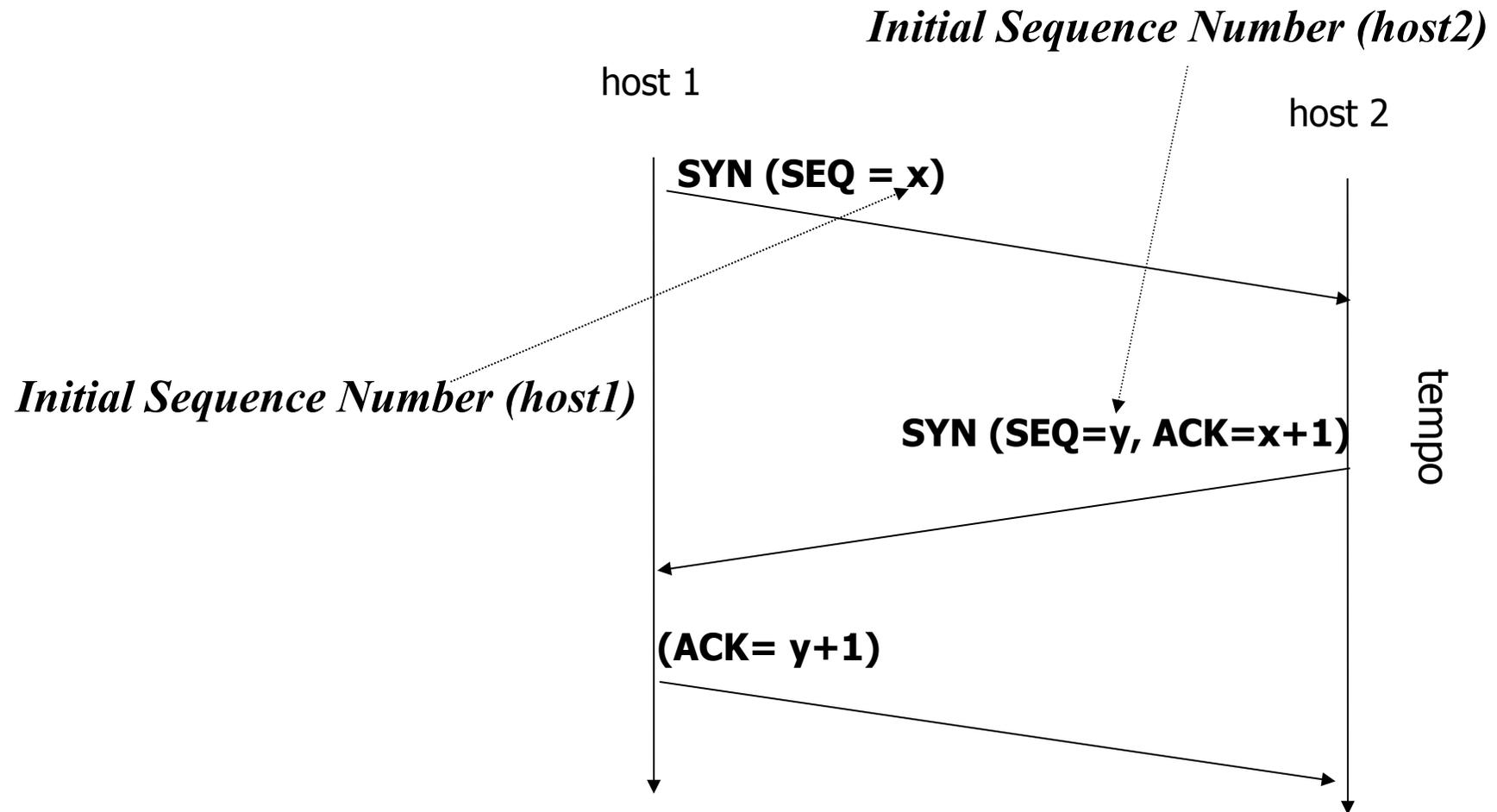
○ *SYN*

Viene utilizzato per creare connessioni. La richiesta di connessione è caratterizzata da SYN=1 e ACK=0 (**CONNECTION REQUEST**). La risposta di connessione contiene un ack e quindi ha SYN=1 e ACK=1 (**CONNECTION ACCEPTED**)

○ *FIN*

Viene utilizzato per chiudere una connessione (il mittente non ha altri dati da spedire)

Per l'apertura di una connessione si utilizza un protocollo *3-way handshake*



Se il TCP riscontra l'assenza del processo che deve essere in attesa sulla porta destinazione, manda un segmento di rifiuto della connessione (RST)

Un esempio di connessione

Connessione Telnet fra *10.6.1.9* e *10.6.1.2* catturata con **tcpdump**
porta client **4548** - porta server **23** (telnet)

```
10.6.1.9.4548 > 10.6.1.2.23: S 2115515278:2115515278 (0)  
win 32120 <mss 1460, nop, nop, sackOK, nop, wscale 0> (DF)
```

*opzioni da negoziare
(es. max segment size mss)*

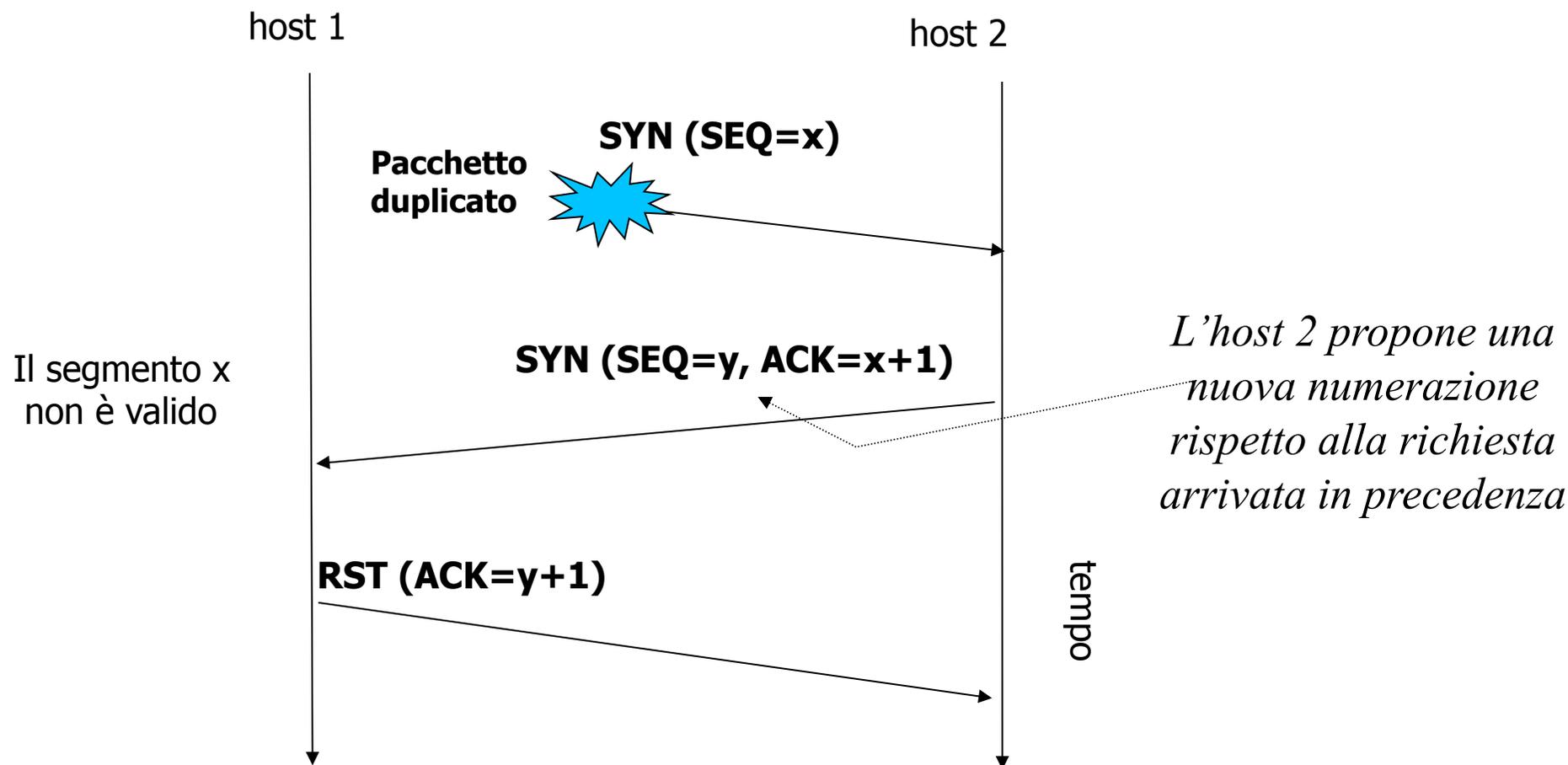
pacchetti senza dati

```
10.6.1.2.23 > 10.6.1.9.4548: S 1220480853:1220480853 (0)  
ack 2115515279 win 32120 <mss 1460, nop, nop, sackOK, nop, wscale  
0> (DF)
```

```
10.6.1.9.4548 > 10.6.1.2.23: . ack 1220480854 win 32120  
(DF)
```

* **tcpdump -S -n -t \((dst 10.6.1.2 and src 10.6.1.9\) or \((dst 10.6.1.9 and src 10.6.1.2)\)**

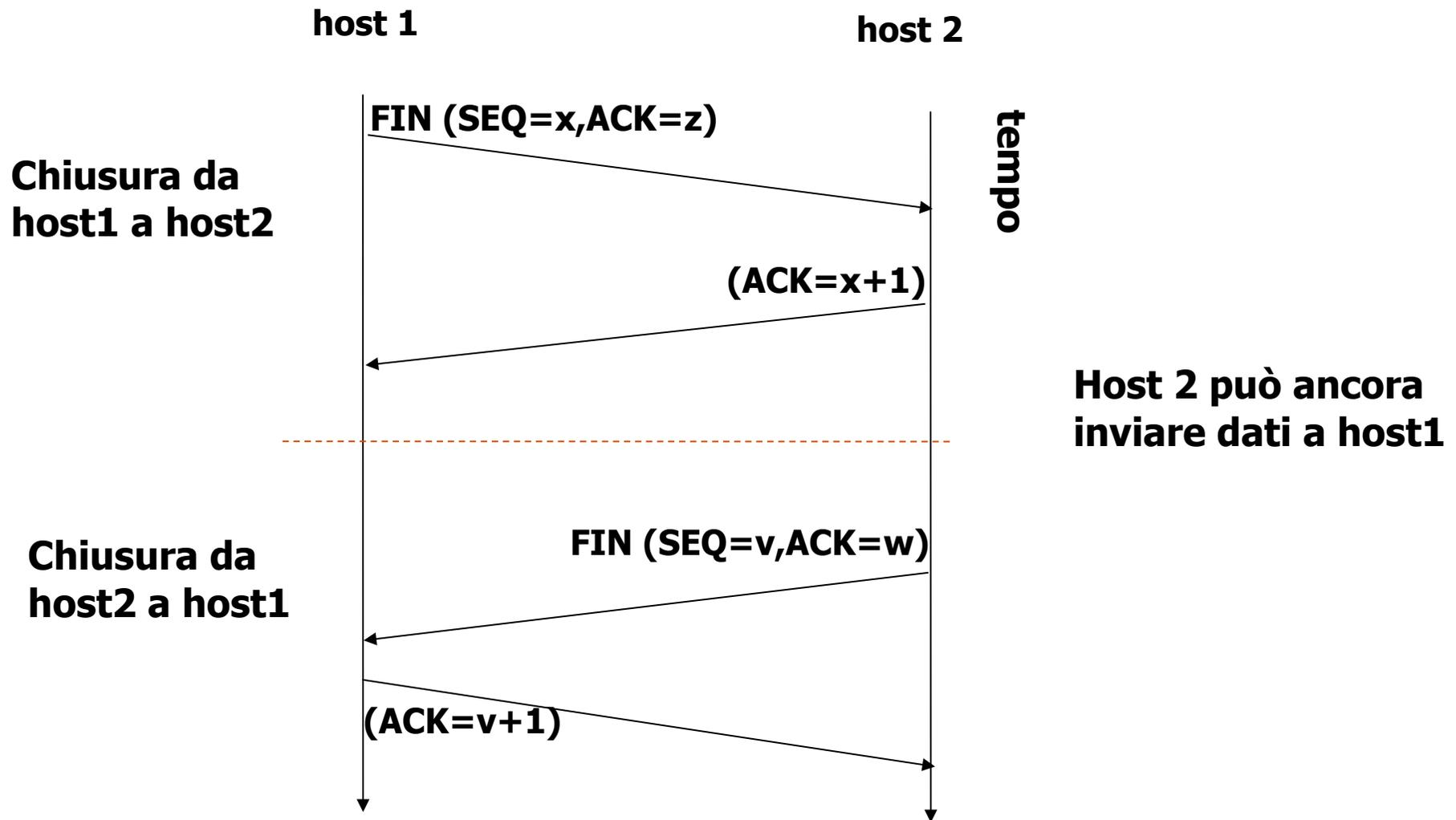
Pacchetti duplicati: I pacchetti possono essere memorizzati e ricomparire nella rete



Il numero di sequenza, espresso su 32 bit, corrisponde al valore di un orologio locale con tick ogni 4 μ s. Conseguentemente lo stesso numero di sequenza (32 bit) non può essere rigenerato prima di ~ 4 ore

A causa del **time to live** dei pacchetti IP, segmenti con lo stesso numero di sequenza non possono coesistere sulla rete

Chiusura della connessione: La connessione è full-duplex e le due direzioni devono essere chiuse indipendentemente

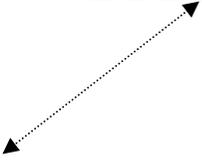


Se l'ack di un messaggio FIN si perde l'host mittente chiude comunque la connessione dopo un timeout

Chiusura Telnet da 10.6.1.9 porta 4548 – a 10.6.1.2 porta 23

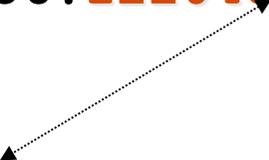
```
10.6.1.9.4548 > 10.6.1.2.23: F 2115515449:2115515449 (0)  
ack 1220480986 win 32120 (DF) ↓
```

```
10.6.1.2.23 > 10.6.1.9.4548: . ack 2115515450 win 32120  
(DF)
```



```
10.6.1.2.23 > 10.6.1.9.4548: F 1220480986:1220480986 (0)  
ack 2115515450 win 32120 (DF)
```

```
10.6.1.9.4548 > 10.6.1.2.23: . ack 1220480987 win 32120  
(DF)
```



Flusso di dati interattivi

Nel caso di connessioni interattive (es. telnet) non si possono accumulare i dati ma occorre inviare segmenti anche se piccoli

Il 90% dei segmenti telnet porta circa 10 byte

Nel caso limite si ha un segmento per ogni carattere battuto

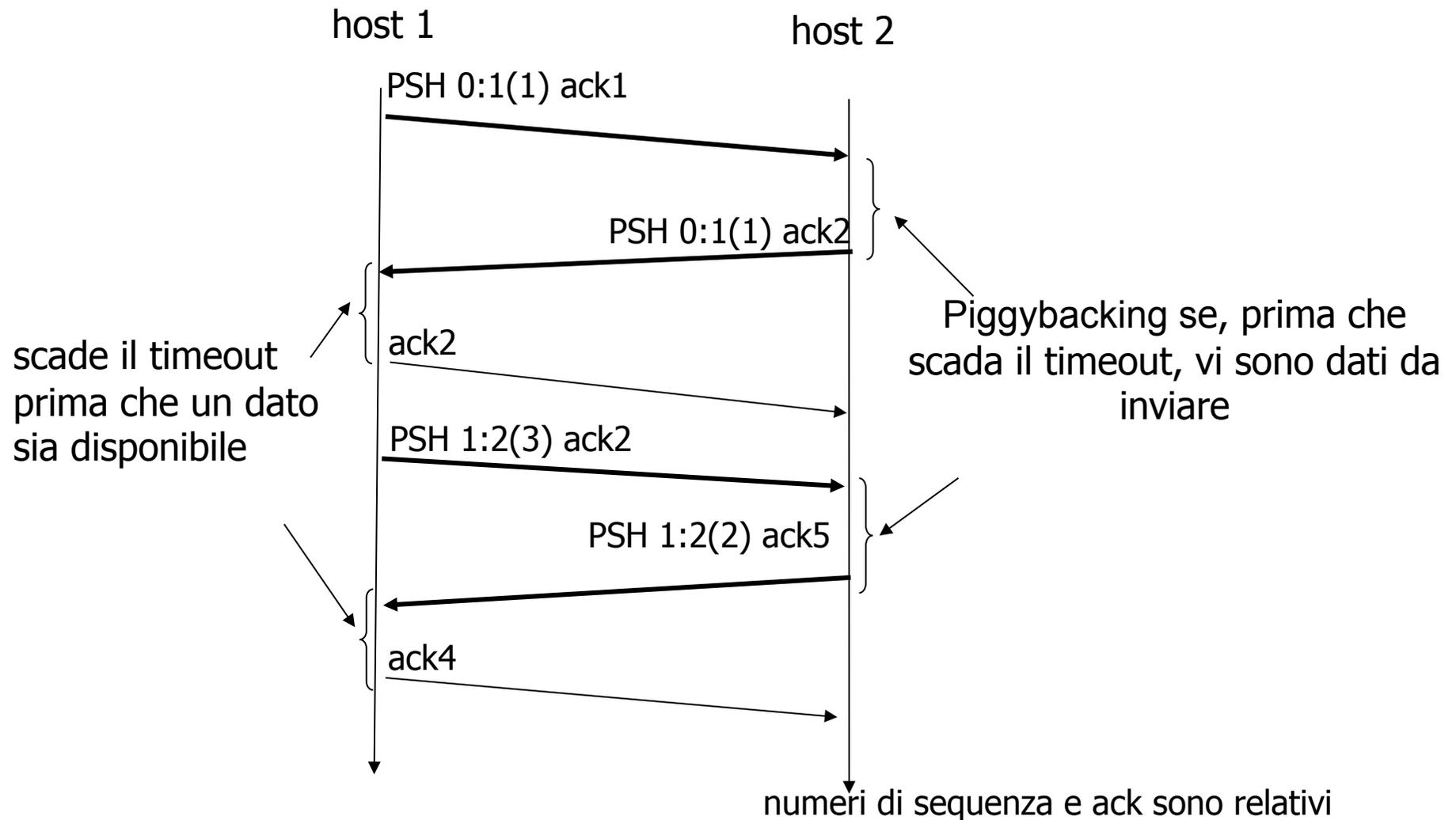
Il ricevente server in genere fa un echo del carattere battuto

- Segmento dal client col carattere battuto (20IP+20TCP+1byte=41byte)
 - Segmento di ack dal server al client (40 byte)
 - Segmento di echo dal server (41 byte)
 - Segmento di ack dal client (40 byte)
- } si possono unire...

In totale si userebbero 162 (122) byte in 4 (3) segmenti TCP per 1 carattere!!

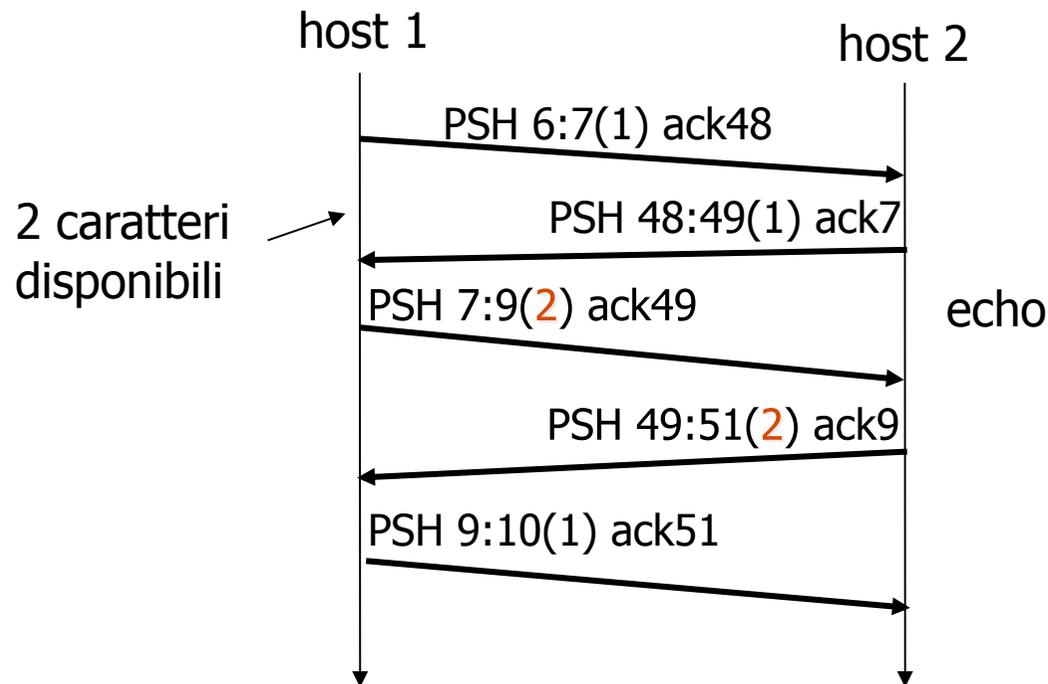
Ack ritardati

- Normalmente il TCP non invia un ack istantaneamente ma ritarda l'invio sperando di avere dati da spedire (**piggyback**)
- Molte implementazioni usano un ritardo di 200ms

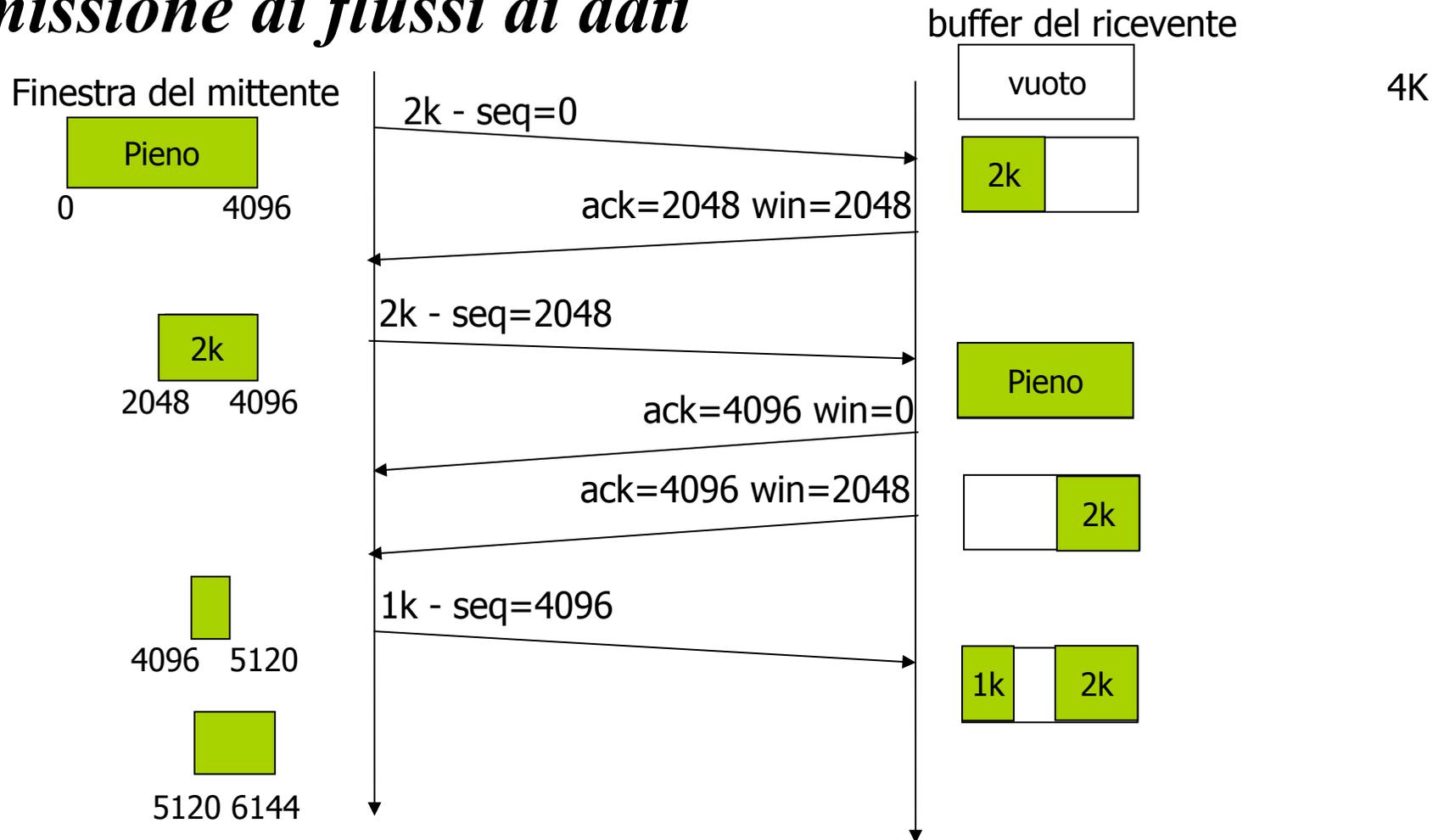


L'algoritmo di Nagle (1984)

- Ha effetto per connessioni lente (es. WAN)
- Si accumulano i dati fino a che non si riceve l'ack per il segmento inviato in precedenza
- In alcuni casi deve essere disabilitato (es. mouse in Xwindows)



Trasmissione di flussi di dati



- se il ricevente indica una finestra 0 il mittente non può trasmettere dati
- se si perdono i pacchetti, per non rimanere in attesa infinita, il mittente può inviare un segmento di un byte per forzare il destinatario a indicare il prossimo byte atteso e l'ampiezza della finestra - *timer di persistenza*

Un esempio di trasmissione

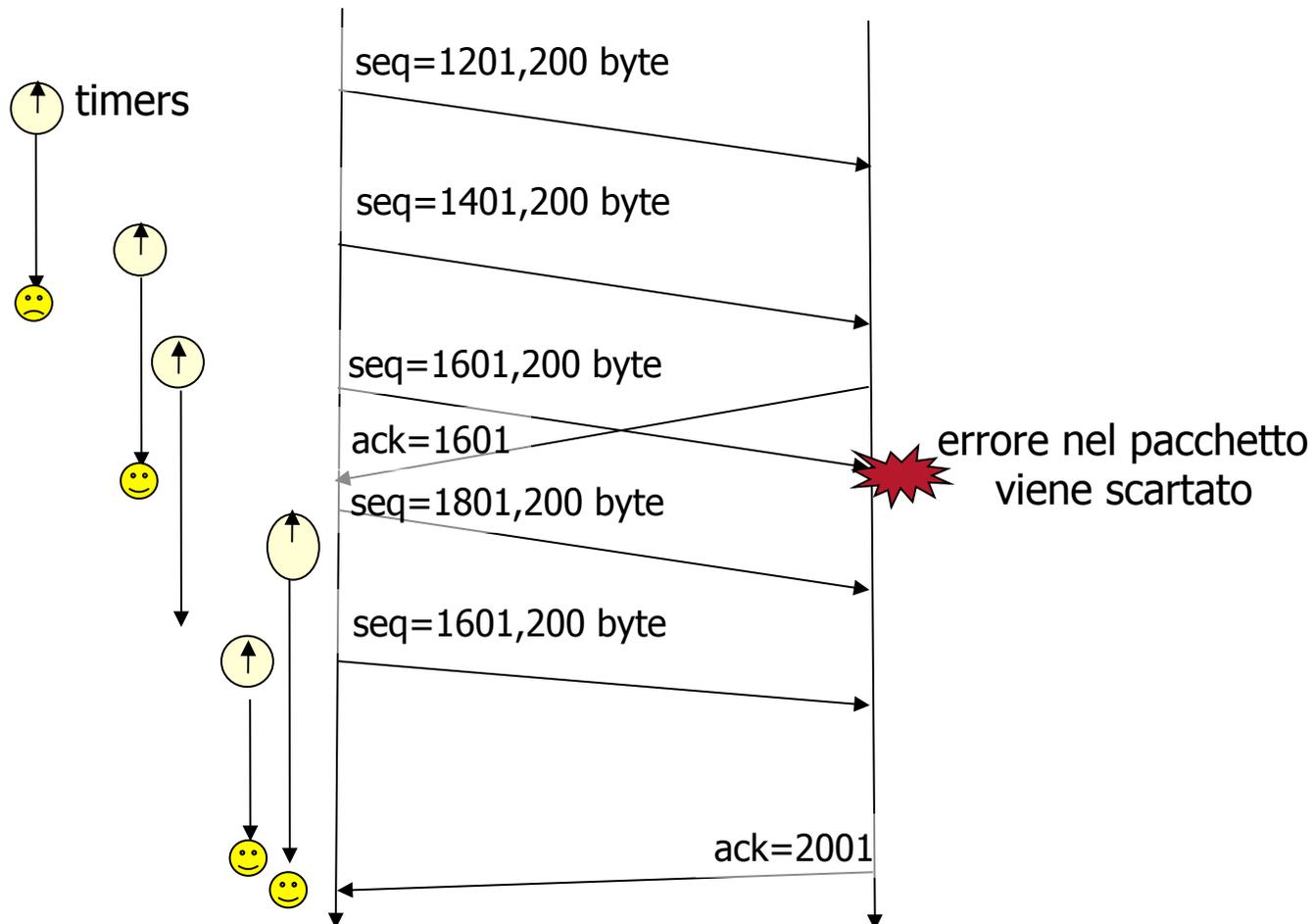
*Connessione Telnet fra da 10.6.1.9 a 10.6.1.2 catturata con tcpdump
porta client 4548 - porta server 23 (telnet)*

```
10.6.1.9.4548 > 10.6.1.2.23: P 2115515279:2115515306(27) ack 1220480854  
win 32120  
10.6.1.2.23 > 10.6.1.9.4548: . ack 2115515306 win 32120  
10.6.1.2.23 > 10.6.1.9.4548: P 1220480854:1220480866(12) ack 2115515306  
win 32120  
10.6.1.9.4548 > 10.6.1.2.23: . ack 1220480866 win 32120  
10.6.1.2.23 > 10.6.1.9.4548: P 1220480866:1220480905(39) ack 2115515306  
win 32120  
10.6.1.9.4548 > 10.6.1.2.23: P 2115515306:2115515443(137) ack 1220480905  
win 32120  
10.6.1.2.23 > 10.6.1.9.4548: P 1220480905:1220480908(3) ack 2115515443  
win 32120  
10.6.1.9.4548 > 10.6.1.2.23: P 2115515443:2115515446(3) ack 1220480908  
win 32120  
10.6.1.2.23 > 10.6.1.9.4548: . ack 2115515446 win 32120
```

Gestione degli errori

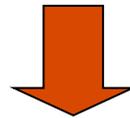
Il mittente ha un timeout di attesa dell'ack per i pacchetti spediti

- Se il timeout scade il pacchetto viene ritrasmesso
- Permette di gestire la perdita di pacchetti (non arrivati o arrivati con errori)



TCP Timeout e ritrasmissione

- TCP utilizza un timeout di attesa dell'ack dopo di che provvede alla ritrasmissione dei dati
- Il problema è determinare il valore del timeout migliore (i ritardi possono essere molto variabili nel tempo sulla rete)
 - Se il timeout è troppo piccolo si fanno ritrasmissioni inutili
 - Se il timeout è troppo elevato si avranno ritardi di trasmissione



- Si utilizza un algoritmi di stima del migliore timeout basato sulla misura del **Round-Trip Time** (RTT)

Controllo di congestione

Il TCP adatta la velocità di trasmissione alla capacità della rete

- pacchetti possono essere rifiutati dai router se si riempiono i buffer di trasmissione/ricezione
- Per i pacchetti distrutti nella rete non si riceve l'ack e quindi vengono ritramessi
- La ritrasmissione è un potenziale fattore per incrementare ulteriormente la congestione

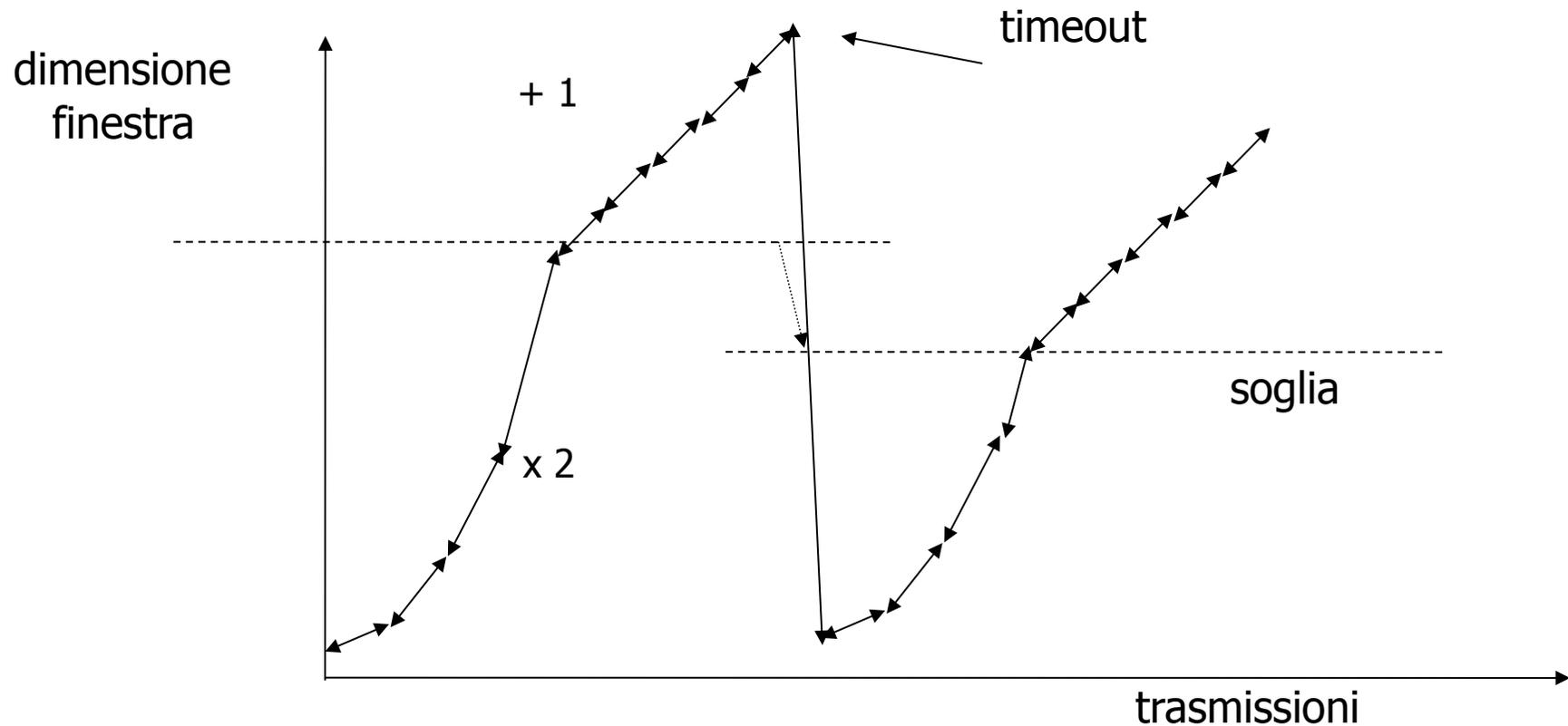
Si utilizza una finestra di congestione per il mittente

- La finestra effettiva è la minima fra quella di congestione (definita dalla rete) e quella di ricezione (definita dallo stato del buffer del ricevente)

Controllo di congestione

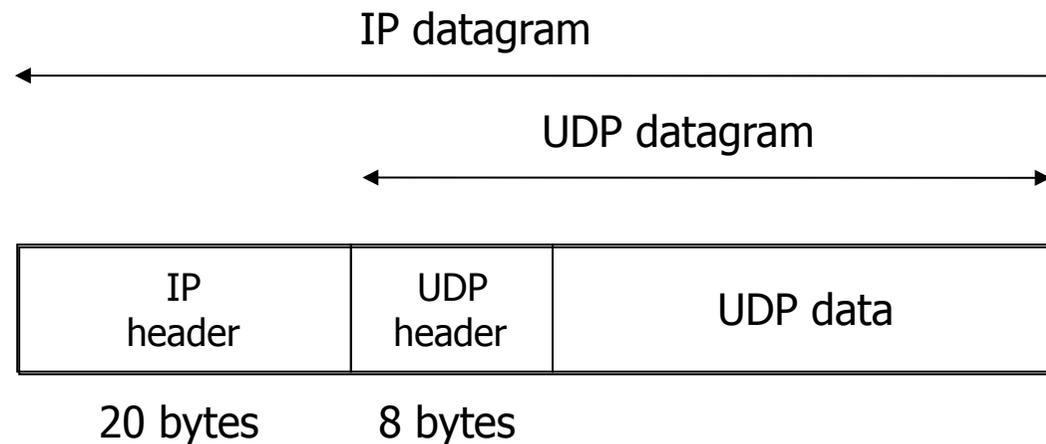
La dimensione della finestra di congestione è adattata dinamicamente

- Cresce se non scade il timeout di trasmissione
- Viene ridotta se scade il timeout di ritrasmissione



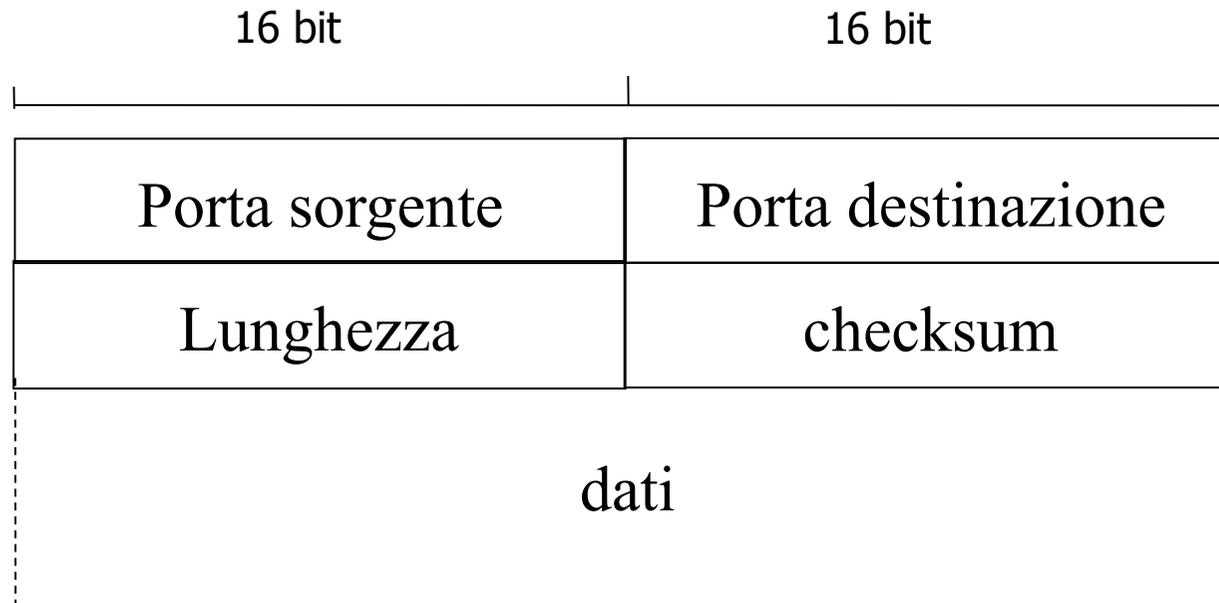
UDP

Ogni operazione di output produce esattamente un datagram UDP che comporta l'invio di un datagram IP



- *UDP non garantisce affidabilità di consegna*
- *Se il datagram eccede la **MTU** (**Maximum Transfer Unit**) della rete, esso viene frammentato*
- *Richiede meno overhead di una connessione TCP (header/connessione/ack)*

Header UDP



Le porte UDP sono indipendenti da quelle TCP

La lunghezza in byte comprende sia i dati che l'header

Un pacchetto UDP può contenere fino a 65507 byte = $65535 - 20(\text{IP}) - 8(\text{UDP})$

Il checksum comprende anche uno pseudo-header che contiene le informazioni IP (indirizzi IP, tipo di protocollo, dimensione)

Ulteriori caratteristiche dell'UDP

- Servizio di consegna con impegno (best effort): i segmenti UDP possono essere persi, duplicati, consegnati senza ordine.
- Servizio senza connessione: non vi è handshaking tra mittente e destinatario del segmento UDP.
- Ogni segmento UDP è trattato in modo indipendente dagli altri

A differenza di TCP, UDP

- non introduce ritardo per instaurare la connessione
- non mantiene lo stato di connessione (non deve gestire buffer di invio/ricezione, parametri per il controllo della congestione, numeri di sequenza, etc.)
- produce un minore overhead (header di 20 byte per TCP e 8 per UDP)
- non controlla la congestione (il controllo della congestione effettuato dal TCP può avere un severo impatto su applicazioni real-time)

<i>Applicazione</i>	<i>Prot. strato applicativo</i>	<i>Prot. trasporto sottostante</i>
Posta Elettronica	SMTP	TCP
Accesso Terminale Remoto	Telnet	TCP
Web	HTTP	TCP
Trasferimento File	FTP	TCP
File Server Remoto	NFS	solitamente UDP
Multimedia Streaming	proprietario	solitamente UDP
Telefonia Internet	proprietario	solitamente UDP
Gestione della rete	SNMP	solitamente UDP
Protocollo di Routing	RIP	solitamente UDP
Traduzione dei nomi	DNS	solitamente UDP

Posta elettronica, Telnet, Web, FTP: per questi servizi è necessario il servizio di trasferimento dati affidabile fornito da TCP

RIP per gli aggiornamenti periodici delle tabelle usa UDP, eventuali informazioni perse saranno sostituite da quelle più aggiornate

DNS usa UDP: si evitano i ritardi dovuti all'instaurazione della connessione

Telefonia su Internet usa l'UDP perché tollera la perdita di dati ma richiede un tasso di trasmissione costante (non controlla la congestione)

Applicazioni multimediali usano l'UDP perché il TCP non consente il multicast (problema della mancanza del controllo di congestione in UDP)

Il checksum UDP risulta indispensabile per la rilevazione di errori laddove il livello 2 adottato non fornisce tale servizio

Il checksum a livello IP estende il suo controllo al solo header IP, quindi non consente di rilevare errori nei dati

L'UDP non opera alcun recupero dell'errore (il segmento errato viene scartato o consegnato all'applicazione segnalando che presenta errori)

Checksum UDP:– calcolato usando il complemento ad 1 della somma di tutti i campi dello pseudo-header e del segmento UDP

Esempio: 3 parole da 16 bit l'una

0110011001100110	0110011001100110
0101010101010101	0101010101010101
0000111100001111	<u>0000111100001111</u>
	1100101011001010
complemento ad 1	0011010100110101

campo checksum nel segmento UDP = 0011010100110101

Il destinatario calcola il checksum del segmento UDP ricevuto (senza calcolare complemento a 1)

checksum dati + checksum UDP = 1111111111111111 assenza di errore

checksum dati + checksum UDP \neq 1111111111111111 errore