
La rete Internet e l'architettura di protocolli TCP/IP

Il protocollo TCP (Transmission Control Protocol)

Definizione: RFC 793

Identificazione di bug: RFC 1122

Estensioni : RFC 1323

Transmission Control Protocol (TCP)

- TCP è un protocollo “con connessione” che fornisce un servizio affidabile “end-to-end” tra coppie di processi su host remoti
- TCP effettua funzioni di:
 - × indirizzamento di uno specifico utente all'interno di un host (multiplazione e de-multiplazione delle informazioni)
 - × trasferimento di un flusso informativo continuo e bi-direzionale (byte stream), ma non strutturato, di dati tra host remoti (funzione di segmentation & reassembly)
 - × gestione delle connessioni
 - × controllo e recupero di errore
 - × controllo di flusso
 - × controllo di congestione
 - × riordinamento delle unità informative

Indirizzamento TCP

- **Multiplexing:**
per permettere a molti processi in un Host di usare contemporaneamente le facilità di comunicazione del TCP, il TCP fornisce un set di indirizzi o **porte** in ogni host
- L'indirizzo completo TCP/IP è costituito dall'insieme di indirizzo IP e numero di porta e identifica univocamente un processo in esecuzione su un host
- Tale indirizzo viene spesso indicato con il nome di "**socket**" ed è costituito da:

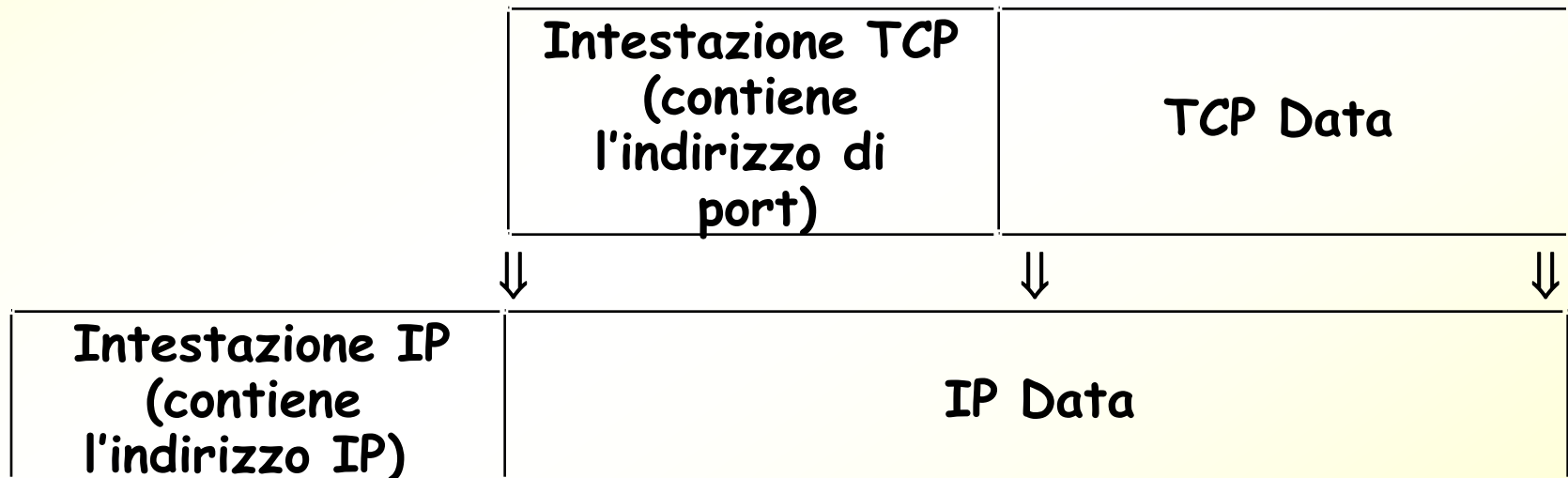
`port@IP_Address = port@Host_Id.Net_Id`

Es.: `molinaro@telecom.deis.unical.it`

`molinaro@160.97.25.95`

Indirizzamento TCP

- La componente "port" è contenuta nell'intestazione dell'unità dati di TCP, mentre la componente IP_Address è contenuta nell'intestazione dell'unità dati di IP

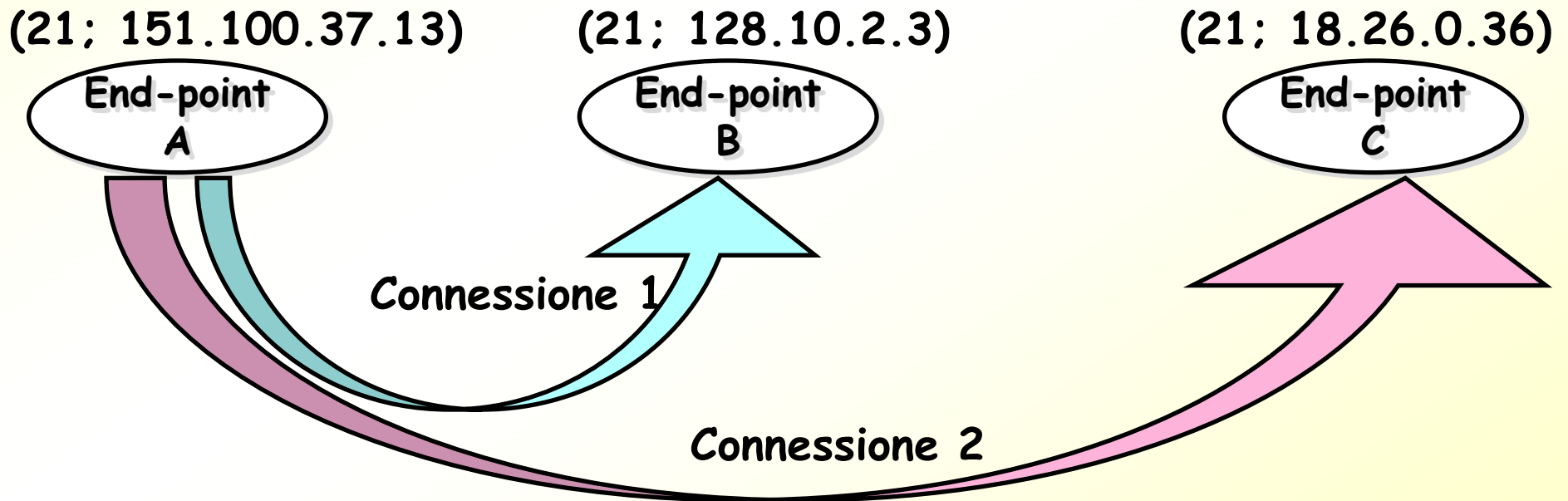


Indirizzamento TCP

- Questo significa che tutte le connessioni in atto tra due specifici host useranno lo stesso indirizzo IP di sorgente e lo stesso indirizzo IP di destinazione. Saranno perciò distinte solo a livello TCP
- Ne segue che queste connessioni possono essere viste come multiplate su un unico indirizzo IP, ovvero su un unico "canale" IP di comunicazione (non su una connessione IP, dato che IP è connectionless)
- Una **connessione TCP** è identificata dalla coppia di socket (sorgente e destinazione) associata ai due processi (end-point) che hanno stabilito la connessione

Indirizzamento TCP

- Un numero di porta può essere usato per più connessioni, ma l'insieme dei socket di sorgente e destinazione identifica univocamente una connessione. Così, ad esempio, un server web può avere più connessioni contemporaneamente attive sulla porta 80, ma riuscire a distinguere le diverse connessioni sulla base dell'indirizzo IP e del numero di porta dei client
- Un end-point può essere impegnato allo stesso tempo in più connessioni TCP



Indirizzamento TCP

- L'assegnazione del numero di porta può essere:
 - × statico
 - l'identificativo è staticamente associato all'applicazione
 - sono utilizzati identificativi inferiori a 255

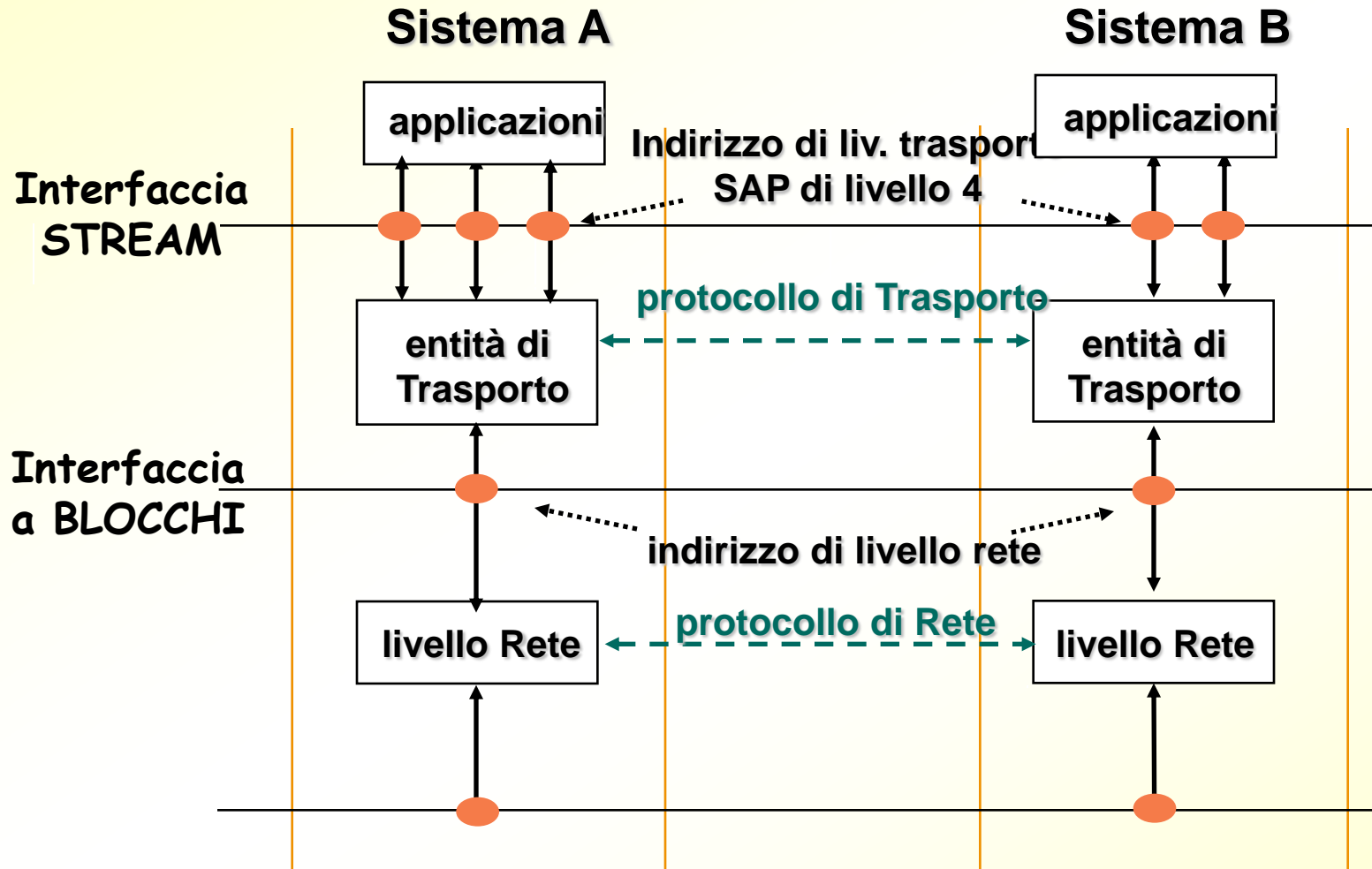
Numero	Applicazione	Numero	Applicazione
7	Echo	37	Time
21	FTP	53	Domain Name Server
23	TELNET	103	X400 Mail Service
25	SMTP	119	NNTP (USENET New Transfer Prot.)

- × dinamico
 - l'identificativo è assegnato direttamente dal sistema operativo al momento dell'apertura della connessione

Indirizzamento TCP

- Per rendere user-friendly l'utilizzo delle interfacce utente, alcuni processi applicativi effettuano l'indirizzamento automatico delle porte
 - se si vuole attivare una sessione telnet sarà l'applicativo ad indirizzare automaticamente i dati alla porta 23

Trasferimento dati TCP



Trasferimento dati TCP

- TCP accetta dal livello applicativo un flusso continuo di dati non strutturati (byte stream), li frammenta e li invia in unità dati distinte, detti **segmenti**
- La lunghezza massima dei segmenti viene negoziata durante la fase di apertura della connessione (**MSS** - Maximum Segment Size) e comunque è **inferiore a 64 kbyte**, dato che il segmento deve poter utilizzare il payload di un unico pacchetto IP, e dipende dall'MTU
- Il TCP utilizza dei **buffer** software per immagazzinare i byte di dati finché si costruisce il segmento
- Tipicamente, è il TCP che decide quando un segmento va trasmesso, anche se l'applicazione ha dei mezzi (la funzione **push**) per forzare la trasmissione di segmenti prima che il TCP intervenga in modo automatico

Trasferimento dati TCP

- Anche dal lato ricevente il TCP raccoglie i byte in un buffer, che successivamente (quando il buffer si riempie o un **URG** flag viene ricevuto) trasmette al livello superiore sottoforma di un flusso continuo di byte; l'applicazione ricevente assorbe i byte dal buffer in base alla sua velocità di elaborazione dell'informazione
- TCP è un protocollo **orientato al byte** e non al messaggio, quindi i confini tra i messaggi non devono essere mantenuti in ricezione
 - Cioè, se il processo sorgente fa un'operazione di scrittura di 4 blocchi di 512 byte ciascuno su un flusso TCP, il processo ricevente può ricevere i dati in 4 blocchi di 512 byte, o in 2 blocchi di 1024 byte o in 1 blocco da 2048 byte, o in altri modi
 - Ogni byte è numerato con un numero di sequenza di 32 bit

Trasferimento dati TCP

- In generale, il riempimento dei buffer dei TCP di trasmissione e ricezione determina il trasferimento dei dati, però il livello applicazione ha modo di intervenire per forzare l'invio di alcuni dati

Sender Application -> Sender TCP (PUSH)

- l'applicazione sorgente può richiedere al TCP sorgente l'invio immediato dei dati verso il TCP destinazione, in questo caso l'applicazione usa il flag PUSH
 - Es. se un utente è collegato in telnet su un terminale remoto, dopo aver digitato una linea e premuto il tasto di Carriage Return, è necessario che la linea venga immediatamente inviata all'host remoto senza bufferizzarla in attesa della linea successiva

Dati urgenti

Sending Application/TCP - Receiving TCP/Application (URG)

- A volte è necessario inviare dei dati urgenti ("out of band") senza aspettare che l'entità ricevente finisca di elaborare i dati precedentemente trasmessi (molti dati possono essere in viaggio, nei router sul path e nella coda di input dell'host remoto)
 - Ad esempio, in una sessione Telnet, un utente potrebbe inviare un segnale di "interrupt" (DEL o CTRL-C) che termini immediatamente l'applicazione remota
 - Il segnale di interrupt deve poter essere inviato senza aspettare che l'host remoto elabori tutti i dati già inviati

Dati urgenti

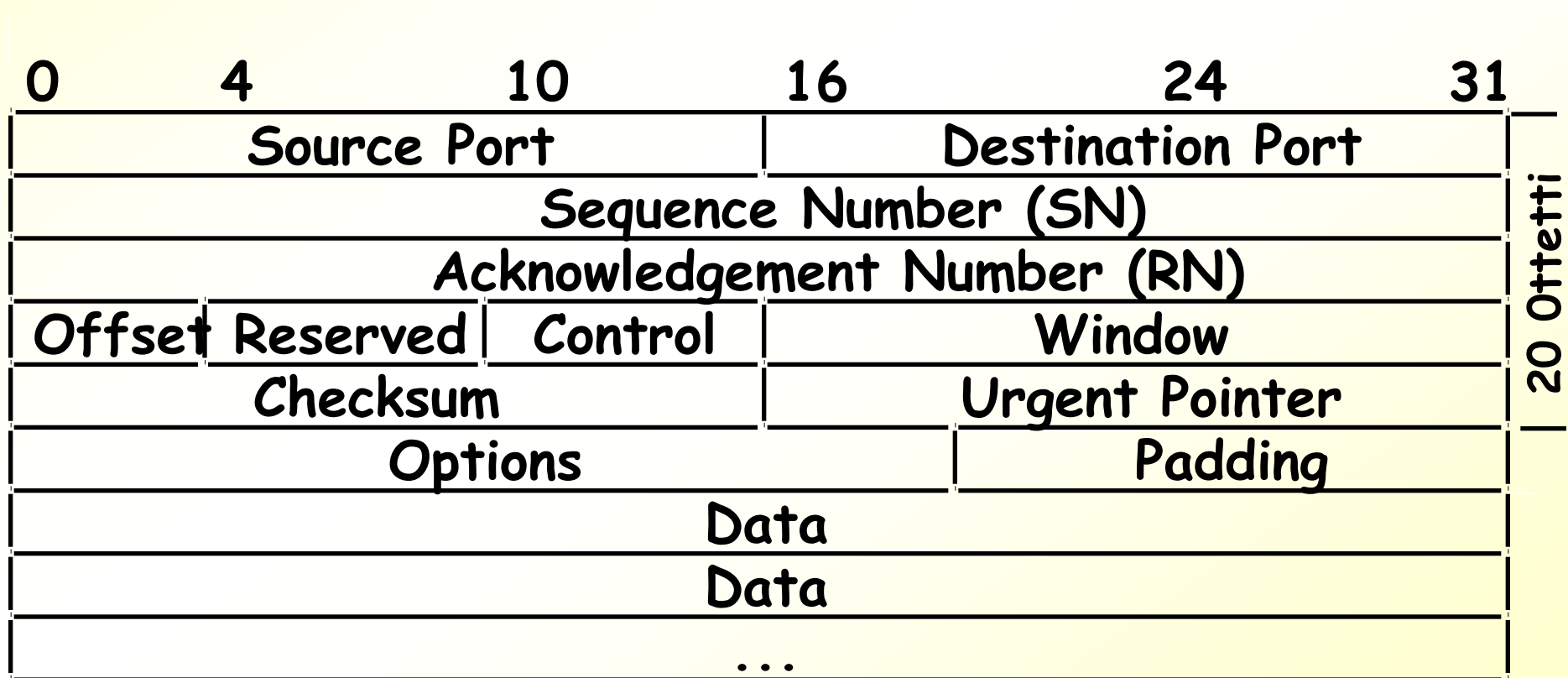
- A tal fine TCP prevede l'invio di "dati urgenti" che hanno priorità su tutti gli altri dati già inviati e vengono trasmessi al processo remoto immediatamente; quando i dati urgenti sono stati elaborati, il processo remoto riprende in esame i dati "normali"
- Il meccanismo usato dal TCP per i dati urgenti consiste:
 - L'applicazione sorgente "setta" il bit **URG** del segmento per avvertire il TCP sender di smettere di accumulare i dati e inviare "immediatamente" tutto quello che ha bufferizzato verso il TCP destinazione
 - Appena i dati urgenti sono ricevuti dal TCP remoto, l'applicazione ricevente viene interrotta (riceve un "signal" in terminologia UNIX) in modo che smetta qualsiasi operazione e legga i dati urgenti
 - La fine dei dati urgenti è marcata dall'applicazione sender che inserisce nel campo **Urgent Pointer** il numero di sequenza dell'ultimo byte di dati urgenti

Trasferimento dati TCP

- Le operazioni di segmentazione e riassemblamento possono avvenire in parallelo, cioè il TCP è un protocollo **full-duplex punto-punto**
- Il trasferimento dati TCP avviene secondo un meccanismo di **sliding window**: all'invio di un segmento il sender fa partire un timer, quando il segmento arriva a destinazione il receiver invia un segmento con il riscontro del segmento ricevuto; se il timer del sender scade prima della ricezione del riscontro il sender ritrasmette il segmento
- Il TCP utilizza un unico formato di trame sia per la trasmissione di informazione d'utente che per l'informazione di servizio (apertura e chiusura della connessione, messaggi per il controllo d'errore e quello di flusso)

Formato dell'unità dati TCP

- L'header è lungo 20 byte se il campo opzioni non viene utilizzato. Il campo dati può essere vuoto (trame di acknowledgment, connessione, ecc.).



Formato dell'unità dati

- **Source Port** (16 bit): definisce l'indirizzo logico del processo sorgente dei dati
- **Destination Port** (16 bit): definisce l'indirizzo logico del processo destinatario dei dati
- **Sequence Number** (32 bit): numero di sequenza in trasmissione (SN); contiene il numero di sequenza del primo **byte** di dati contenuti nel segmento a partire dall'inizio della sessione (se $SN=m$ ed il segmento contiene n byte il prossimo SN sarà pari a $m+n$)
- **Acknowledgement Number** (32 bit): numero di sequenza in ricezione (RN); nei segmenti in cui il bit ACK è 1, contiene il numero di sequenza del prossimo byte che il ricevitore si aspetta di ricevere. Il meccanismo di acknowledgment usato è cumulativo, così l'ack di un SN X indica che sono stati ricevuti tutti i byte fino a X escluso X . In caso di connessioni interattive bidirezionali gli ack sono inviati in piggybacking (nei segmenti di risposta contenenti dati di utente). I numeri SN e RN vengono utilizzati per il controllo d'errore e di flusso

Formato dell'unità dati: SN e Ack

- I numeri di sequenza e gli ack servono a rendere affidabile la trasmissione TCP. Concettualmente, ad ogni byte di dati si assegna un numero di sequenza. In ogni segmento TCP si inserisce il numero di sequenza del primo byte di dati contenuto in quel segmento (cioè il SN)
- I segmenti in direzione opposta portano anche un numero di acknowledgment che è il numero di sequenza del successivo byte di dati da trasmettere atteso dal ricevitore
- Quando il TCP trasmette un segmento contenente dati, ne conserva una copia in una coda di ritrasmissione e fa partire un timer; quando il TCP riceve l'acknowledgment per quei dati, allora cancella il segmento dalla coda. Se l'acknowledgment non è ricevuto prima della scadenza del time-out, il segmento viene ritrasmesso
- Così il TCP mantiene la corretta sequenza dei segmenti in ricezione; cioè prima di inviare una nuova sequenza di byte aspetta che la sequenza precedente venga riscontrata

Formato dell'unità dati

- **Offset** (4 bit): contiene il numero di parole di 32 bit contenute nell'intestazione TCP (da Source port a Padding). L'intestazione TCP non supera i 64 byte ed è un numero di bit multiplo di 32
- **Reserved** (6 bit): riservato per usi futuri, contiene zeri
- **Control bit** (6 bit): i bit di controllo sono 6:
 - × **URG**: vale 1 quando il campo Urgent Pointer contiene un valore significativo; è usato per indicare dati urgenti che vengono trasmessi al di fuori del controllo di flusso con un meccanismo di segnalazione end-to-end tra processi remoti
 - × **ACK**: vale 1 quando il campo Acknowledgement Number contiene un valore significativo
 - × **PSH**: vale 1 quando l'applicazione esige che i dati forniti vengano trasmessi e consegnati all'applicazione ricevente prescindendo dal riempimento dei buffer allocati fra applicazione e TCP e viceversa (di solito il riempimento dei buffer scandisce la trasmissione e la consegna dei dati)

Formato dell'unità dati

- **RST**: vale 1 quando un malfunzionamento impone il reset della connessione
- **SYN**: indica l'inizio della connessione; vale 1 solo nel primo segmento inviato durante la fase di sincronizzazione fra le entità TCP (3-way handshaking)
- **FIN**: indica la fine della connessione; vale 1 quando la sorgente ha esaurito i dati da trasmettere
- **Window** (16 bit): larghezza della finestra per il controllo di flusso; il TCP ricevente riporta al TCP trasmittente il valore di una "window", che contiene il numero di byte che, a cominciare dal valore del campo Acknowledgement Number, il TCP ricevente è disposto a ricevere. Il controllo di flusso è orientato al byte
- **Checksum** (16 bit): contiene la sequenza che permette al TCP ricevente di verificare la correttezza del segmento; protegge l'intero segmento più alcuni campi dell'header IP, cioè uno pseudo-header di 96 bit

Formato dell'unità dati

- La checksum è il complemento a 1 del complemento a 1 della somma di tutte le word da 16 bit nell'header e nel testo; se un segmento contiene un numero dispari di byte, l'ultimo byte è riempito di zeri a destra (padding) per formare una word di 16 bit. Il pad non è trasmesso come parte del segmento
- TCP Length è la lunghezza del TCP header più la lunghezza dei dati in byte (questa quantità non viene esplicitamente trasmessa, ma è calcolata), e non tiene conto dei 12 byte dello pseudo-header

0	8	16	24	31
Source IP address				
Destination IP address				
Padding		Protocol	TCP length	

Formato dell'unità dati

- **Urgent Pointer** (16 bit): indica i dati urgenti all'interno del segmento; contiene il numero di sequenza dell'ultimo byte di dati che devono essere consegnati con urgenza al processo ricevente. Tipicamente sono messaggi di controllo che esulano dalla comunicazione in senso stretto. A tale traffico ci si riferisce di solito con il nome di out-of-band
- **Options** (lunghezza variabile): sono presenti solo raramente; le più note sono End of Option List, No-operation e Maximum Segment Size (di default è 536 byte)
 - × MSS quando è usata, serve per comunicare al TCP trasmittente la dimensione massima del segmento accettabile in ricezione; il campo è inviato soltanto nella richiesta iniziale di connessione (cioè, nei segmenti col bit SYN settato); se non si usa questa opzione è ammessa una qualsiasi dimensione del segmento
- **Padding** (lunghezza variabile): contiene sempre degli zeri. Serve come riempitivo per far sì che l'intestazione abbia una lunghezza multipla di 32 bit

Instaurazione di una connessione

- TCP è un protocollo **orientato alla connessione**; sono presenti le fasi di instaurazione della connessione, trasferimento dati ed abbattimento della connessione
- Prima di iniziare il trasferimento di informazioni verso un utente remoto, deve essere instaurata una connessione, tramite un meccanismo di sincronizzazione noto come **three-way handshake** (stretta di mano in 3 fasi)
- La **sincronizzazione** serve a risolvere potenziali situazioni anomale, dovute al fatto che IP non è affidabile e quindi i datagrammi possono essere persi, ritardati, duplicati o consegnati fuori sequenza e TCP ritrasmette i segmenti persi dopo un certo timeout:
 - × un segmento appartenente ad una “vecchia” connessione può arrivare ad un host dopo che tra i processi relativi a quel segmento sia stata instaurata una nuova connessione
 - × un host “cade” e perde traccia delle connessioni ancora in atto

Instaurazione di una connessione

Obiettivo:

- Evitare la presenza di duplicati ritardati (stesso numero di sequenza); assicurarsi che un pacchetto sia "morto" e che siano "morti" anche tutti i riscontri relativi a quel pacchetto dopo un tempo T (\rightarrow MSL)

Soluzione:

- Ogni host usa un clock (non sincronizzato con gli altri) che incrementa un contatore binario a passi regolari (la cadenza dipende dalla velocità della rete; a 2 Mbit/s è di 4 μ s, cioè $1/(2M/8)$)
- Il numero di bit del contatore deve essere maggiore o uguale del numero di bit dei numeri di sequenza (32 bit)
- Il clock deve continuare a funzionare anche quando l'host va in "crash"

Instaurazione di una connessione

Requisiti:

- Quando si attiva una connessione, il valore del clock è usato come numero di sequenza iniziale (ISN)
- Gli ottetti sono numerati sequenzialmente a partire dal numero (ISN) scelto durante il 3-way handshake
- Lo spazio dei numeri di sequenza (2^{32}) deve essere abbastanza ampio in modo che quando i numeri di sequenza si ripetono i vecchi datagrammi con lo stesso numero di sequenza siano scomparsi
- E' necessario evitare che i numeri di sequenza vengano usati (cioè assegnati a nuovi segmenti) per un tempo T dopo il loro potenziale utilizzo come ISN

Instaurazione di una connessione

- Il numero di sequenza in trasmissione non inizia da un valore fisso; ma ogni volta che si instaura una nuova connessione l'host trasmittente sceglie in modo pseudo-casuale l'ISN di 32 bit da cui partire, ponendolo uguale al valore del suo contatore binario, sincronizzato al clock locale
- Il numero di sequenza si sceglie in modo pseudo-casuale tra 1 e $2^{32} = 4.294.967.296$. La cadenza del clock dipende dalla velocità della rete; a 2 Mbit/s è di 4 μ s, cioè $1/(2M/8)$. Sono necessarie circa 4 ore ($4\mu s \times 2^{32} = 4.7$ ore) per compiere un ciclo completo di valori di ISN
- Dato che si assume che i segmenti restino in rete per non più del **Maximum Segment Lifetime** (MSL) e dato che il MSL è tipicamente minore di 4.7 ore, si può ragionevolmente assumere che il valore di ISN sia unico

Instaurazione di una connessione

- Poiché il contatore viene incrementato ad una velocità molto superiore a quella relativa al numero di sequenza di una qualsiasi connessione (almeno alle attuali velocità disponibili in rete), questo meccanismo risolve i problemi visti prima
 - anche a velocità di 100Mbit/s, il tempo di ciclo diventa 5,4 min, cioè ancora maggiore dei valori del **MSL che è dell'ordine di decine di sec (120sec)**
- Riassumendo: ogni segmento occupa uno o più numeri di sequenza nello spazio da 1 a 2^{32} ; i numeri usati da un segmento sono considerati "occupati" finché passano **MSL sec (quite time)** dopo un crash per accertarsi che quei segmenti siano scomparsi dalla rete

Instaurazione di una connessione

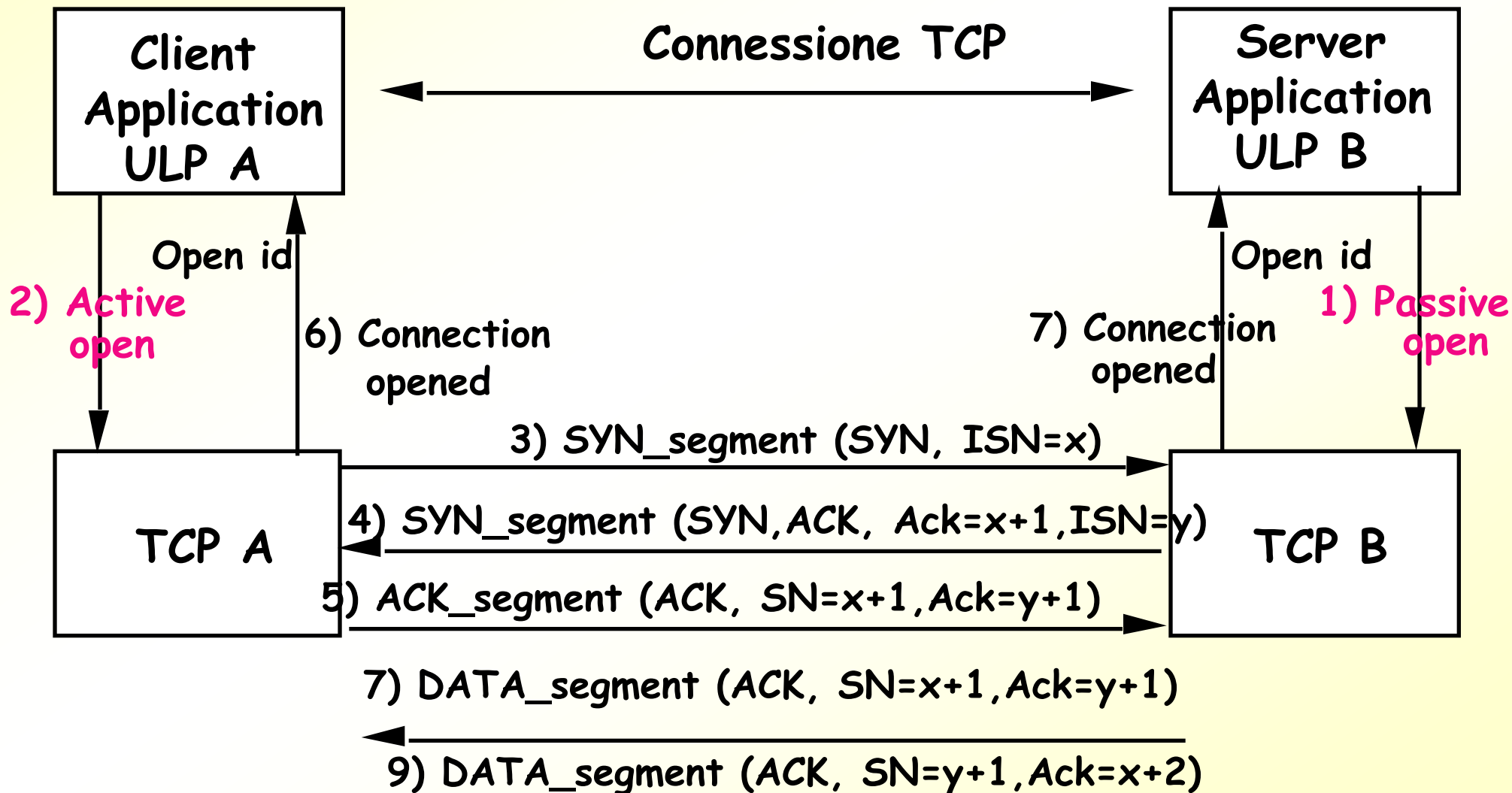
- Un meccanismo di tipo three way handshake è necessario perché i numeri di sequenza non sono collegati a un clock globale nella rete. Il ricevitore del primo ISN non ha modo di sapere se il segmento è un vecchio segmento ritardato o no, a meno che ricordi l'ultimo numero di sequenza usato nella connessione (non sempre possibile), e quindi deve chiedere al sender di verificare questo ISN

Instaurazione di una connessione

- Le due entità TCP interagenti si sincronizzano scambiandosi il proprio numero di sequenza iniziale (**ISN**), che rappresenta il numero a partire dal quale tutti i byte trasmessi, una volta instaurata la connessione, saranno sequenzialmente numerati
- La sincronizzazione consiste nell'invio, da parte di ogni sistema coinvolto nella connessione, del proprio ISN e nella ricezione di una conferma dall'altra parte in un messaggio di acknowledgment:
 - 1) A --> B SYN (Synchronize) my sequence number is X
 - 2) A <-- B ACK your sequence number is X
 - 3) A <-- B SYN (Synchronize) my sequence number is Y
 - 4) A --> B ACK your sequence number is Y
- I passi 2 e 3 sono combinati in un unico messaggio, perciò la procedura è nota come three way (o three message) handshake

Three-way handshake: instaurazione

ULP=Upper Layer Protocol



Instaurazione di una connessione

- **1-2:** Affinché dal lato server di un servizio applicativo vi sia un processo in ascolto su una porta deve avvenire l'attivazione del TCP mediante una primitiva detta di **PASSIVE_OPEN**, che serve a predisporre il server alla ricezione di richieste di connessione; dal lato client, quando si vuole effettivamente aprire una connessione deve essere passata al TCP locale una primitiva di **ACTIVE_OPEN**, che contiene il socket di destinazione
- **3.** La prima trama inviata dal TCP del lato client è una trama di **SYN** (synchronize) caratterizzata dal bit di SYN posto a 1. Il numero di sequenza contenuto nel campo ISN è il numero iniziale scelto dal TCP del client per la nuova connessione

Instaurazione di una connessione

- **4.** In risposta, il TCP del server nell'host destinatario invia una trama di **SYN-ACK** con i bit di SYN e ACK posti a 1. ISN contiene il numero di sequenza iniziale scelto dal TCP del server per la nuova connessione dal valore del suo contatore. Il campo AckN contiene il riscontro della corretta ricezione della trama precedente (valore di ISN ricevuto più 1)
- **5-7.** L'apertura della connessione viene completata con l'invio di un segmento vuoto con l'**ACK** finale contenente il riscontro del TCP client della corretta ricezione della trama di SYN-ACK del TCP server; con il numero di sequenza riscontrato pari al numero di sequenza in trasmissione dell'host destinatario + 1. Quindi il TCP sender comincia a trasmettere dati, dopo aver inviato la primitiva di **Connection Opened** al livello applicativo. Si noti che il SN dei messaggi 5 e 7 è lo stesso perché l'invio degli ACK non contribuisce a incrementare il valore di SN
- **8-9.** L'host destinatario comincia a trasmettere solo dopo aver ricevuto quest'ultimo terzo segmento e aver inviato al proprio livello applicativo la primitiva di **Connection Opened**

Instaurazione di una connessione: anomalie

Instaurazione simultanea

- Ogni TCP cicla tra SYN e ACK finchè si sincronizza o invia un RST

TCP A		TCP B	
1. CLOSED		CLOSED	
2. SYN	--> <SEQ=100><SYN>	...	
3. SYN-REC	<-- <SEQ=300><SYN>	<-- SYN	
4.	... <SEQ=100><SYN>	--> SYN-REC	
5. ACK	--> <SEQ=100><ACK=301><SYN,ACK>	...	
6. OK	<-- <SEQ=300><ACK=101><SYN,ACK>	<-- ACK	
7. ACK	... <SEQ=100><ACK=301><SYN,ACK>	--> SYN-REC	

Instaurazione di una connessione: anomalie

Recovery da un vecchio SYN duplicato

- Il TCP A si accorge del duplicato perché il campo ACK è errato e invia un RST (reset) col campo SEQ aggiornato in modo da rendere credibile il segmento; il TCP B ricevuto RST torna nello stato LISTEN

TCP A

TCP B

1. CLOSED

LISTEN

2. SYN --> <SEQ=100><SYN>

...

3. (duplicato) ... <SEQ=90><SYN>

--> SYN-REC

4. SYN <-- <SEQ=300><ACK=91><SYN,ACK> <-- ACK

5. SYN --> <SEQ=91><RST>

--> LISTEN

6. ... <SEQ=100><SYN>

--> SYN-REC

7. SYN <-- <SEQ=400><ACK=101><SYN,ACK> <-- ACK

8. OK --> <SEQ=101><ACK=401><ACK> --> OK

Instaurazione di una connessione: anomalie

Half-Open Connection Discovery

- Due processi A e B comunicano tra loro e avviene un crash che causa la perdita di memoria del TCP di A. Quando il TCP si riattiva, A proverà ad APRIRE la connessione di nuovo o ad INVIARE dati sulla connessione che crede aperta. Nel secondo caso, riceve il messaggio di errore "connection not open" dal TCP locale di A. Nel primo caso, il TCP di A invia un segmento SYN (in figura), il TCP B, intanto, pensa che la connessione sia aperta

TCP A

TCP B

1. (CRASH)

(send 300, receive 100)

2. CLOSED

ESTABLISHED

3. SYN-SENT --> <SEQ=400><SYN>

--> (??)

Instaurazione di una connessione: anomalie

- Quando il TCP B riceve il SYN da A, dato che il segmento arrivato è fuori dalla finestra, B risponde con un ACK in cui ribadisce quale numero di sequenza si aspetta di ricevere (ACK 100). Il TCP A si accorge che il segmento non riscontra nulla che sia stato inviato e invia un RST perché scopre che la connessione è half-open. Il TCP B abortisce la connessione. Il TCP A continua a tentare di stabilire la connessione e il caso diventa come quello di base del 3-way handshake

TCP A

TCP B

- | | | |
|-------------|-----------------------------|---------------|
| 4. (!!) | <-- <SEQ=300><ACK=100><ACK> | <-- OK |
| 5. SYN-SENT | --> <SEQ=100><RST> | --> (Abort!!) |
| 6. SYN-SENT | | CLOSED |
| 7. SYN-SENT | --> <SEQ=400><SYN> | --> |

Maximum Segment Size

- Quando il TCP trasmittente invia il primo segmento (SYN) per instaurare una connessione, può inserire nel segmento un'informazione che rappresenta la massima dimensione del campo dati di utente di un segmento (**MSS**) che è in grado di trattare
- Il TCP ricevente risponde comunicando la propria MSS; con questo scambio di informazioni le due entità TCP stabiliscono il valore di MSS comune (il minore tra i due)
- La scelta della MSS dipende da due fattori:
 - × la dimensione della memoria a disposizione delle entità TCP
 - × la dimensione della Maximum Transfer Unit (MTU) della sottorete a cui è connesso l'end-system; l'MTU è resa nota all'entità IP e TCP dal software che interfaccia TCP/IP alla sottorete (detto driver di rete)
- Il calcolo dell'MSS avviene sottraendo all'MTU la dimensione degli header IP e TCP (in genere le opzioni non si usano e gli header hanno dimensione fissa di 20 byte)

Maximum Segment Size

- Se questo non succede, cioè se il TCP trasmittente non specifica nel primo segmento (SYN) il valore di MSS, si usa il valore di **default di MSS pari a 536 byte**
- Un valore di MSS di 536 byte, sommata ai 40 byte di header IP + TCP, dà luogo a un datagramma IP di dimensione pari a **576 byte**, ovvero un datagramma che tutti i sistemi di Internet devono necessariamente accettare e gestire
- Oggi è diffusa la tendenza a usare valori molto più grandi dell'MSS per aumentare l'efficienza e la velocità del collegamento (con IPv6 si arriva ad una MTU di 1280 byte)

Rilascio di una connessione

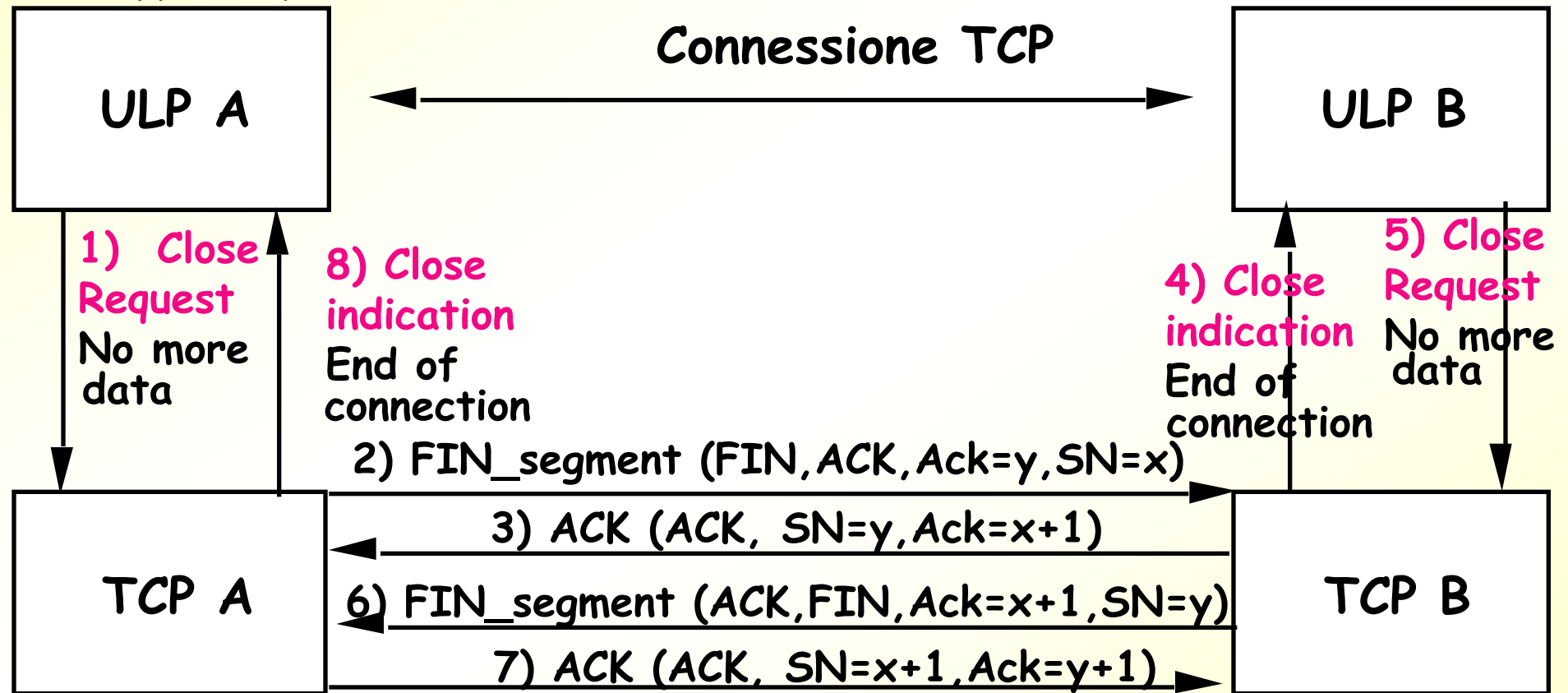
- Il rilascio di una connessione avviene mediante un meccanismo analogo di **three-way handshake modificato**
- Una connessione TCP è bi-direzionale e può essere vista come due flussi di dati indipendenti per ciascuna direzione; nel rilascio, le 2 vie sono chiuse indipendentemente
- Quando un programma applicativo non ha più dati da trasmettere ordina al TCP di chiudere la connessione "in una direzione". Il TCP finisce di trasmettere gli eventuali dati rimasti nel buffer e ne aspetta il riscontro, quindi invia un messaggio di **FIN** (bit di FIN settato a 1) al TCP ricevente
- Il TCP ricevente invia un messaggio **ACK** di avvenuta ricezione al TCP trasmittente e poi avvisa il suo livello applicativo che non ha più dati da trasferirgli (questa fase può richiedere tempo per l'interazione umana)

Rilascio di una connessione

- Intanto i dati possono continuare a fluire nella direzione ancora aperta della connessione, e i riscontri devono continuare a essere inviati dal TCP che aveva chiesto la chiusura della connessione. Questo finchè anche l'altra direzione della connessione verrà chiusa
- Normalmente per il rilascio di una connessione servono 4 segmenti: 2 FIN e 2 ACK, però può succedere che il primo ACK e il secondo FIN siano contenuti nello stesso segmento e il numero totale di segmenti necessari si riduce così a 3

Three-way handshake: rilascio

ULP=Upper Layer Protocol



Rilascio di una connessione

Chiusura contemporanea

TCP A

TCP B

1. ESTABLISHED

ESTABLISHED

2. (Close)

(Close)

FIN-WAIT-1 --> <SEQ=100><ACK=300><FIN,ACK> ...FIN-WAIT-1

<-- <SEQ=300><ACK=100><FIN,ACK> <--

... <SEQ=100><ACK=300><FIN,ACK> -->

3. CLOSING --> <SEQ=101><ACK=301><ACK>

... CLOSING

<-- <SEQ=301><ACK=101><ACK>

<--

... <SEQ=101><ACK=301><ACK>

-->

4. TIME-WAIT
(2 MSL)
CLOSED

TIME-WAIT
(2 MSL)
CLOSED

Rilascio di una connessione

- Il rilascio di una connessione è più complesso della sua instaurazione; infatti informare un'applicazione che è pervenuta una richiesta di chiudere una direzione della connessione ed ottenere una risposta potrebbe richiedere un tempo considerevole (se implica, ad es., un'interazione umana)
- Il meccanismo usato non garantisce la corretta chiusura in ogni caso e il TCP ovvia con dei time out (pari a 2 MSL). Se il pacchetto di FIN non è seguito da un ACK, dopo un tempo sufficiente la connessione full duplex viene chiusa. L'altra parte può non accorgersi della chiusura immediatamente ma prima o poi si renderà conto che nessuno risponde e a sua volta chiude.

Rilascio di una connessione

- Se una connessione non può essere chiusa secondo la procedura normale a causa di situazioni anomale, o se un programma applicativo è forzato a chiudere immediatamente una connessione, TCP prevede una procedura di **reset**
- Tale procedura consiste nell'inviare un segmento con il bit **RST** "settato". Alla ricezione di tale segmento la connessione è immediatamente terminata senza scambio di ulteriori messaggi e TCP ne informa i programmi applicativi

Trasferimento dati

- Una volta stabilita la connessione inizia lo scambio dati
- Il sender tiene traccia del successivo numero di sequenza da usare nella variabile **SND.NXT** e del più vecchio numero di sequenza non riscontrato in **SND.UNA**; il receiver tiene traccia del successivo numero di sequenza da accettare nella variabile **RCV.NXT**
- Se il flusso di dati è momentaneamente "idle" e tutti i dati inviati sono stati riscontrati le 3 variabili sono uguali
- Quando il sender crea un segmento e lo trasmette, fa avanzare **SND.NXT**; quando il ricevitore accetta un segmento fa avanzare **RCV.NXT** e invia un riscontro; quando il sender riceve un riscontro, fa avanzare **SND.UNA**

Trasferimento dati

- La differenza tra i valori di queste variabili è una misura del ritardo nella comunicazione; la quantità di cui le variabili sono fatte avanzare è la lunghezza dei dati nel segmento
- Si noti che una volta che la connessione è stabilita tutti i segmenti devono trasportare informazioni di riscontro
- La chiamata utente a `CLOSE` implica una funzione `PUSH`, come fa il flag `FIN` in un segmento entrante

Controllo e recupero di errore

- La strategia utilizzata dal TCP per il recupero di errore è simile a quella usata in X.25 (LAPB), di tipo **Go-back-n** con riscontri solo positivi (**PAR**, Positive Acknowledge with Retransmission), ed è basata sull'uso di finestre in trasmissione ed in ricezione
- TCP vede il flusso di dati in trasmissione come una sequenza di byte e quindi la finestra in trasmissione opera a livello di **byte**, invece che a livello di trama o pacchetto come in X.25
- La **finestra in trasmissione** specifica il numero di byte che possono essere inviati senza ricevere un riscontro (rispetto a meccanismi come Stop & Wait che prevedono un riscontro per ogni unità dati, i meccanismi a finestra permettono di incrementare la portata del collegamento)

Controllo e recupero di errore

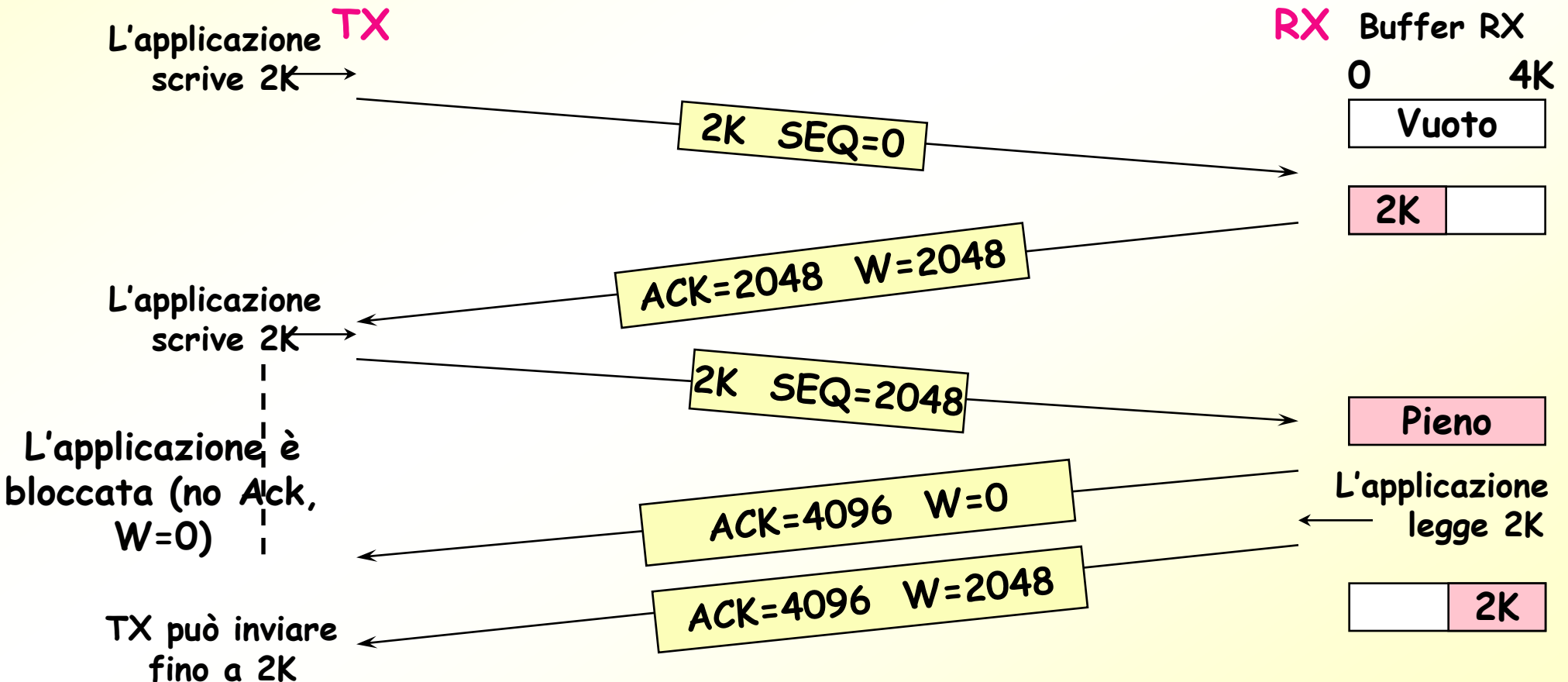
- Analogamente, la **finestra in ricezione** specifica il numero di byte che possono essere accettati fuori sequenza
- In TCP, la dimensione della finestra in ricezione coincide con quella in trasmissione; quindi la dimensione di finestra specifica sia il numero di byte che un'entità TCP può trasmettere senza ricevere riscontri, sia il numero di byte che un'entità TCP può ricevere fuori sequenza
- La dimensione della finestra di trasmissione non è decisa dal mittente, ma è comunicata al mittente dal destinatario (nel campo **Window** di 16 bit) "ogni volta" che questo emette un segmento; quindi la dimensione della finestra varia dinamicamente nel tempo
- Per questo motivo la finestra è chiamata in TCP **Advertised Window** (finestra "pubblicizzata" dal destinatario al mittente)

Controllo e recupero di errore

- All'inizio di una connessione, quando il mittente non ha ancora trasmesso dei dati, emette un numero di byte pari alla dimensione W della finestra; quindi si interrompe in attesa di un riscontro
- Il TCP prevede esclusivamente **riscontri (ACK) positivi**
- I riscontri possono essere "cumulativi", cioè il destinatario conferma la ricezione dell'ultimo byte di una sequenza di dati ricevuta "completamente" in modo corretto, ovvero senza errori e senza elementi mancanti
- In particolare, il destinatario comunica al mittente il "prossimo" byte che si aspetta di ricevere (nel campo Acknowledgement Number), significando così che "tutti" i byte precedenti sono stati ricevuti correttamente, e la nuova dimensione della finestra W

Controllo e recupero di errore

- Alla ricezione di un riscontro (con Acknowledgement Number pari a x) il mittente sposta in avanti la finestra ed è autorizzato a inviare, senza attesa di riscontro, i byte con numero di sequenza compreso tra x e $W+x$

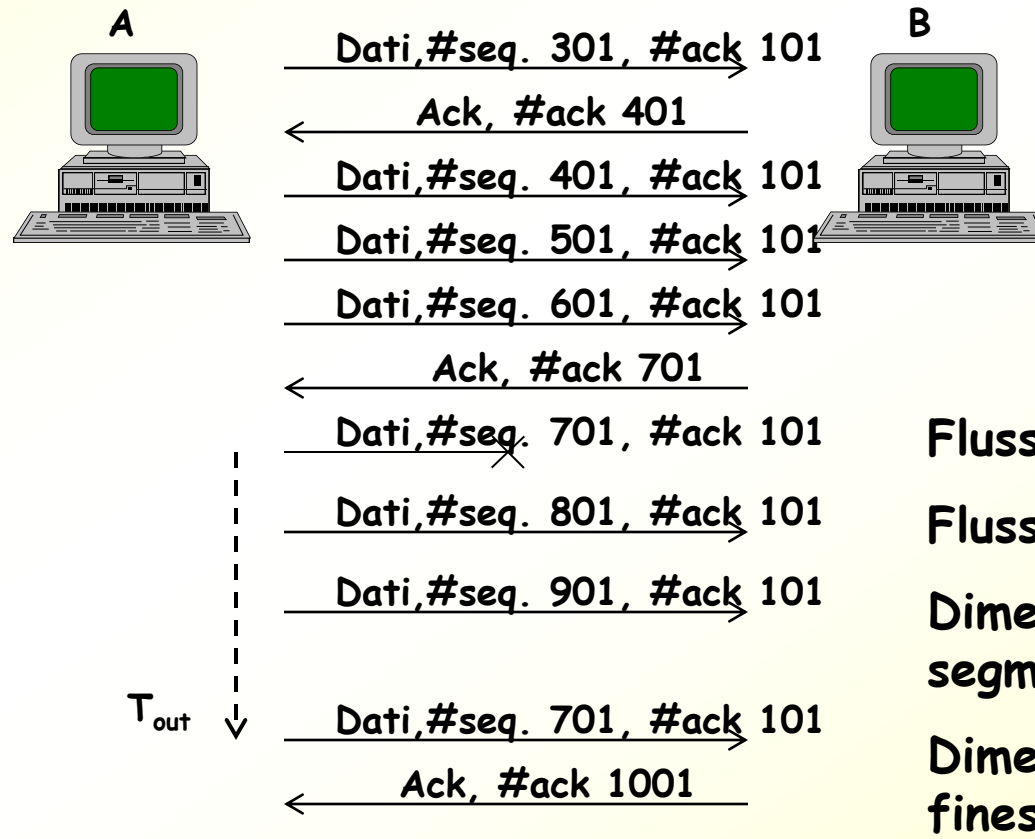


Controllo e recupero di errore

- Ogni segmento contiene una **checksum** che il ricevitore usa per verificare l'integrità dei dati. Se il segmento è corretto viene inviato un segmento di acknowledge. Se il segmento non è corretto viene semplicemente scartato, dopo un timeout il segmento verrà ritrasmesso
- Il meccanismo di recupero da errore è basato sull'uso di un timeout (fuori tempo massimo); ovvero la ritrasmissione di un segmento è innescata dalla mancata ricezione degli ACK entro un tempo predefinito (**RTO=retransmission timeout**)
- L'entità mittente, dopo avere inviato un segmento, aspetta un tempo limite prefissato (RTO) e, se non riceve un riscontro, assume che il segmento si sia perso

Controllo e recupero di errore

- Due eventi provocano la ritrasmissione di un segmento: il segmento si è perso e non arriva a destinazione, il segmento arriva a destinazione non corretto (checksum errata)



Flusso dati: da A a B

Flusso riscontri: da B ad A

Dimensione dei
segmenti=100 byte

Dimensione della
finestra=300 byte

Calcolo del RTO

- Il dimensionamento del timeout (RTO) è un aspetto critico per le prestazioni del TCP.
 - × minore è il timeout, in meno tempo si interrompe la trasmissione. Però se il suo valore è troppo piccolo, i segmenti in ritardo, a causa di congestione, potrebbero essere considerati persi e quindi ri-trasmessi, ciò aumenterebbe la congestione e di conseguenza le ri-trasmissioni finchè la portata tende a zero
 - × se il suo valore è troppo grande, la risposta ad un evento di perdita sarebbe troppo lenta con conseguente perdita di efficienza e minore velocità di trasferimento

Calcolo del RTO

- Il timeout è fissato con uno schema "adattativo" in base al valore stimato del **round-trip delay** (RTT)
- TCP misura il RTT, cioè il tempo di andata e ritorno tra sorgente e destinazione impiegato da un'unità dati, ovvero il tempo che passa tra l'invio di un segmento e la ricezione del relativo ACK
 - × RTT è somma di 3 contributi: ritardo di trasferimento in un verso della comunicazione; ritardo di trasferimento nell'altro verso; tempo necessario al destinatario per rispondere (tempo di risposta)

Calcolo del RTO

- In generale, la determinazione dei tempi di trasferimento nel RTT è complessa in Internet:
 - × una connessione può attraversare una LAN ad alta velocità o più sottoreti lente in diversi continenti; perciò il ritardo di trasferimento è molto variabile ed è influenzato dallo stato di congestione momentanea delle sottoreti attraversate
- Se si potesse effettuare il recupero di errore a strati inferiori al TCP, sottorete per sottorete, si avrebbero stime precise del ritardo di trasferimento, ma in Internet ciò non è possibile, visto che richiederebbe
 - × la cooperazione tra sottoreti diverse (impossibile)
 - × il controllo di errore effettuato da ogni sistema di interconnessione intermedio (da evitare)

Calcolo del RTO

- In base al valore di RTT misurato in maniera regolare, il TCP aggiorna “dinamicamente” il valore di RTO; TCP potrebbe scegliere un valore di RTO maggiore o uguale al “valore medio” osservato di RTT
- Tuttavia, la misura del RTT basata sulla “media” delle osservazioni può essere affetta da errori:
 - × l'emissione degli ACK da parte del TCP ricevente può essere non immediata (riscontri cumulativi non riferiti a uno specifico datagramma)
 - × se è stata effettuata una ritrasmissione è impossibile distinguere se l'ACK si riferisce alla trasmissione iniziale o alla ritrasmissione (l'RTT si calcola dal datagramma originale o da quello ritrasmesso?)
 - × lo stato di congestione della rete può cambiare molto rapidamente

Calcolo del RTT

- **Soluzione:** si usa una stima del RTT che fornisce una media pesata del RTT (media esponenziale), denominata **Smoothed Round Trip Estimate (SRTT)** (RFC793)
- L'entità TCP mittente inizializza un temporizzatore (timer) nell'istante in cui inizia la trasmissione di un segmento e ne memorizza l'istante di partenza in una memoria destinata a contenere le informazioni di gestione della connessione
- Alla ricezione di un riscontro valido (che include il numero di sequenza del segmento trasmesso), il mittente calcola il tempo trascorso, cioè l'RTT corrente, e aggiorna il valore stimato di SRTT

Calcolo del RTT

- L'SRTT privilegia i valori dei campioni di RTT più recenti:

$$SRTT(k+1) = \alpha SRTT(k) + (1-\alpha) RTT(k+1)$$

$0 \leq \alpha \leq 1$; tanto più il valore del peso (smoothing factor) α è vicino a 0, tanto maggiore sarà il peso dato all'ultima osservazione di RTT (normalmente $0.8 \leq \alpha \leq 0.9$)

- Valori minori del peso corrispondono ad un aggiornamento veloce del SRTT; valori maggiori rendono il SRTT insensibile a brevi variazioni del ritardo di trasferimento

Calcolo del RTT

- Usando un valore costante di α ($0 < \alpha < 1$), indipendentemente dal numero di osservazioni passate, si considerano comunque tutte le osservazioni ma si dà minor peso a quelle più distanti

- Infatti:

$$\begin{aligned} \text{SRTT}(k+1) = & (1-\alpha) \text{RTT}(k+1) + \alpha (1-\alpha) \text{RTT}(k) + \alpha^2(1-\alpha) \\ & \text{RTT}(k-1) + \dots + \alpha^k(1-\alpha) \text{RTT}(1) + \alpha^{k+1} \text{RTT}(0) \end{aligned}$$

dato che α e $(\alpha - 1)$ sono ≤ 1 ogni termine successivo è più piccolo; per es. per $\alpha = 0.8$ ($= 7/8$), si ha:

$$\text{SRTT}(k+1) = 0.2 \text{RTT}(k+1) + 0.16 \text{RTT}(k) + 0.128 \text{RTT}(k-1) + \dots$$

Calcolo del RTO

- L' espressione generale di RTO potrebbe essere

$$RTO(k+1) = SRTT(k+1) + \Delta$$

Δ è un termine non proporzionale a SRTT, però:

- se $SRTT \gg \Delta$ piccole fluttuazioni del ritardo possono causare ritrasmissioni inutili
- se $SRTT \ll \Delta$ si possono generare inutili attese per le ritrasmissioni

Calcolo del RTO

- La specifica TCP (RFC 793) prevede che il valore di RTO sia proporzionale a SRTT secondo la seguente espressione:

$$RTO(k+1) = \min[UBOUND, \max [LBOUND, \beta SRTT(k+1)]]$$

- × LBOUND (valore tipico 1 s) e UBOUND (valore tipico 1 m) sono i prefissati limiti inferiore e superiore di RTO
- × β è una costante il cui valore deve essere scelto $1.3 < \beta < 2.0$ (delay variance factor)
- × Il lower bound DOVREBBE essere misurato in frazioni di secondo (per accomodare LAN ad alta velocità) e l'upper bound dovrebbe essere $2 * MSL$, cioè 240 secondi