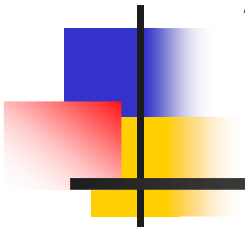
Reti di Calcolatori

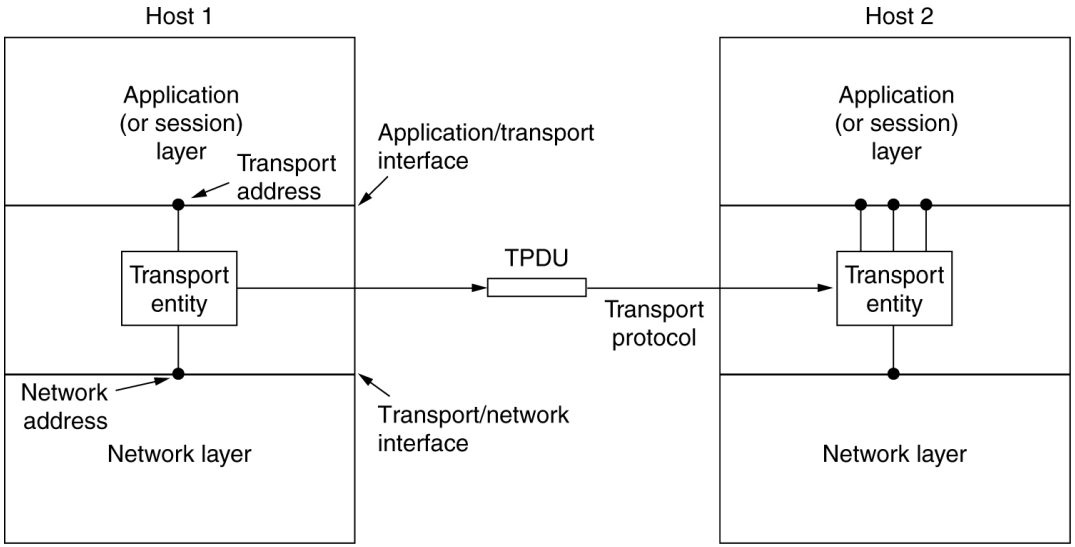


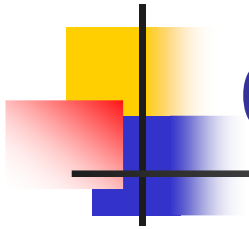
Alessandro Longheu
http://www.diit.unict.it/users/alongheu
alessandro.longheu@diit.unict.it

Livello di Trasporto
(Transport Layer)

Compiti del Transport Layer – 1

- **Il compito del livello di trasporto** è fornire un servizio affidabile ed efficiente a due utenti (di solito processi a livello di applicazione), nascondendo i dettagli della rete sottostante, sia in termini di visibilità che di errori:
  - il TL rimedia agli errori non risolti dal NL (ad esempio, servizio inadeguato)
  - Il TL è in esecuzione sugli host, il NL è sui router, di proprietà dei provider
  - Il TL può garantire un servizio indipendente dal tipo di rete sottostante
- **Il compito viene svolto dalle entità del livello di trasporto**, che possono essere situate nel kernel del sistema operativo, in un processo utente separato, nel software o hardware di rete
- Anche qui sono previste le modalità connection-less (datagram) e connection-oriented (circuito virtuale)





Compiti del Transport Layer – 2

- I **compiti del TL** sono simili a quelli del DLL (controllo di errori e flusso), ma vi sono delle differenze:
  - Nel DLL talvolta non occorre risolvere il **problema di indirizzamento**, mentre nel TL è sempre richiesto l'indirizzamento esplicito delle connessioni
  - Nel TL **stabilire una connessione** è più complesso
  - Nel TL è possibile avere una **capacità di memorizzazione** della rete, all'interno della quale pacchetti potrebbero girovagare per molto tempo e riapparire nel futuro
  - **Il buffering ed il controllo di flusso** del TL sono più complessi perché il numero di connessioni varia dinamicamente
  - Occorre una gestione del **multiplexing delle connessioni**
  - Occorre anche gestire i casi di **crash e di recupero delle connessioni**

Primitive del Transport Layer – 1

Le **primitive generiche del TL**, usate per assolverne i compiti, sono:

Primitive	Packet sent	Meaning
LISTEN	(none)	Block until some process tries to connect
CONNECT	CONNECTION REQ.	Actively attempt to establish a connection
SEND	DATA	Send information
RECEIVE	(none)	Block until a DATA packet arrives
DISCONNECT	DISCONNECTION REQ.	This side wants to release the connection

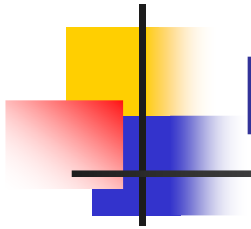
- Il server esegue una primitiva LISTEN, che lo blocca in attesa di una richiesta di connessione
- Il client esegue una CONNECT, provocando il blocco del chiamante
- Il server, ricevuta la CONNECT, viene sbloccato e la connessione è attiva
- A questo punto, client e server possono scambiarsi dati con SEND e RECEIVE (la seconda bloccante in attesa della prima)
- Per la disconnessione, sono previste due politiche in generale:
  - Asimmetrica, in cui uno dei due invia una DISCONNECT, provocando la chiusura della connessione
  - Simmetrica, in cui entrambi devono inviare una DISCONNECT (se uno solo la manda, può ancora ricevere dati dalla controparte)

Primitive del Transport Layer – 2

Primitive	Meaning
SOCKET	Create a new communication end point
BIND	Attach a local address to a socket
LISTEN	Announce willingness to accept connections; give queue size
ACCEPT	Block the caller until a connection attempt arrives
CONNECT	Actively attempt to establish a connection
SEND	Send some data over the connection
RECEIVE	Receive some data from the connection
CLOSE	Release the connection

Le **primitive Berkeley del servizio di trasporto** sono:

- SOCKET, se eseguita con successo, restituisce un descrittore di file (puntatore), analogamente ad una open() del C;
- un **socket** è un oggetto software che connette un'applicazione con un protocollo di rete; il socket può anche essere definito come ognuna delle due estremità (end-point) di un canale di comunicazione bidirezionale fra due host; normalmente è identificato dalla coppia <indirizzo IP dell'host, porta di connessione>



Primitive del Transport Layer – 3

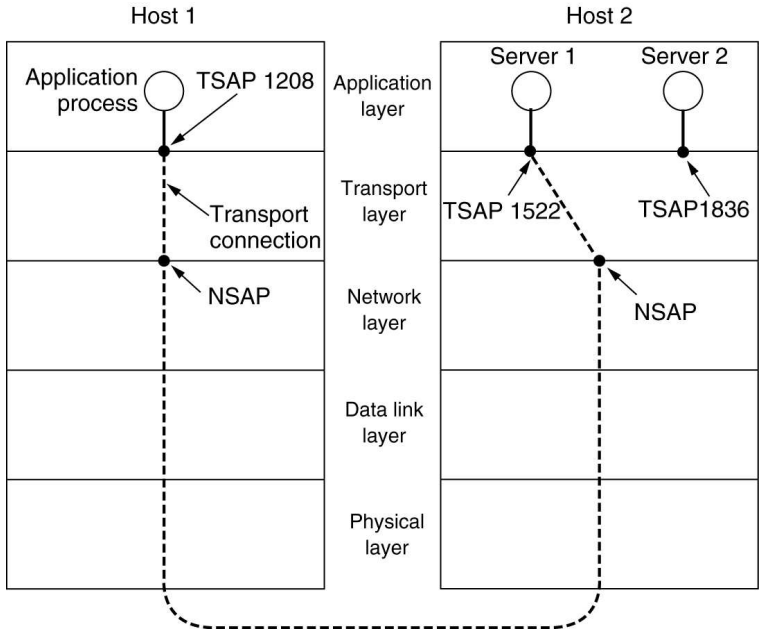
- **BIND** associa una porta ed un'interfaccia ad un socket; il disaccoppiamento fra la creazione del socket e l'associazione con la porta si rende necessario perché alcuni socket sono associati a porte standard (per esempio la 80 per HTTP), altri usano invece porte temporanee (sono prima creati e poi si associa una delle porte correntemente disponibili)
- **LISTEN** annuncia la capacità di accettare connessioni e le accoda
- **CONNECT** viene inviato richiedere una connessione ad un host in stato di LISTEN
- **ACCEPT** rappresenta la risposta che un host in stato di LISTEN fornisce a chi ha precedentemente inviato una CONNECT, semprechè si voglia accettare la richiesta di connessione
- **SEND** e **RECEIVE** sono usate dalle parti per comunicare
- **CLOSE** è usata per la chiusura; è simmetrica, per cui entrambi devono invocarla

- Solitamente, le SOCKET e BIND sono usate da entrambi gli host, client e server, poi il server si mette in LISTEN, e se il client manda una CONNECT, il server risponde con ACCEPT, comunicano con SEND e RECEIVE (utilizzabili da entrambi) e rilasciano con CLOSE

Indirizzamento – 1

- Esistono **indirizzi anche a livello di trasporto**, usati per distinguere i vari processi che girano sullo stesso host (gli indirizzi di rete individuano un host, quello del TL individuano i processi dell'host)
- La **terminologia** usata è TSAP (transport service access point), di fatto sono le porte; gli indirizzi di rete sono denominati NSAP (in pratica, gli IP)

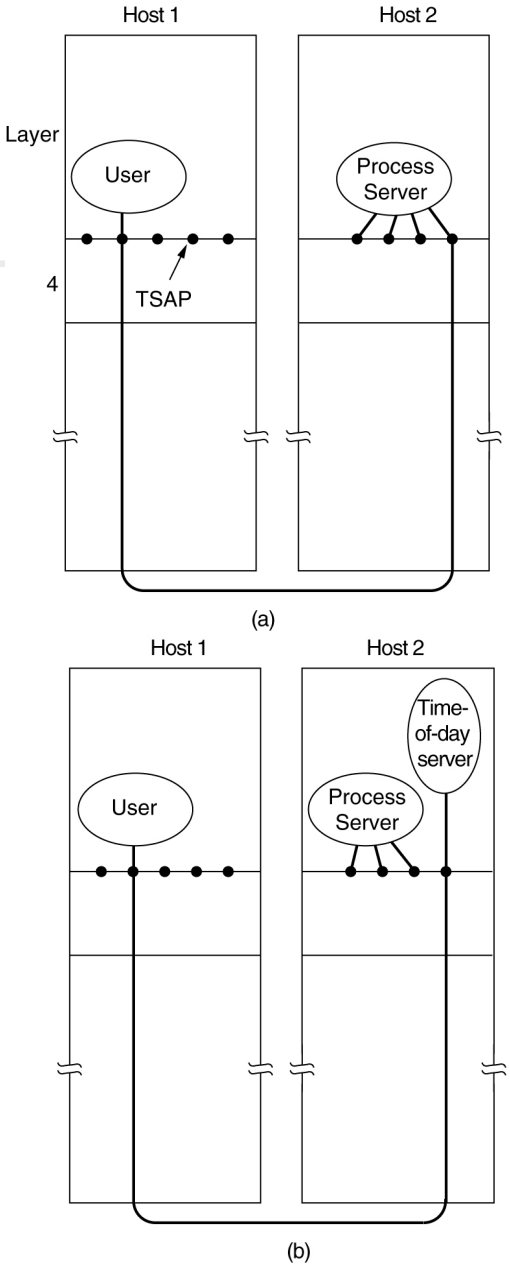
- L'associazione fra NSAP e TSAP è gestita dall'entità di trasporto
- I TSAP possono talvolta essere noti perché standard (la porta HTTP è 80, ad esempio), ma talvolta la connessione è occasionale, ed in tal caso si utilizza una delle porte attualmente libere

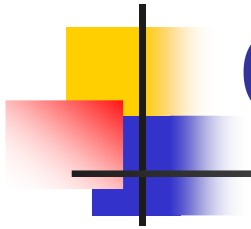


Indirizzamento – 2

Per **gestire l'associazione**, sono previste due soluzioni:

- **Uso di un proxy process server**, unico processo che ascolta su tutte le porte non standard; tale processo intercetta le richieste di connessione, crea il processo server associato alla TSAP richiesta, e ridirige il collegamento del richiedente al processo appena creato, sganciandosi e tornando pronto per altre nuove richieste; questo evita di mantenere in vita un processo per ogni TSAP possibile (dispendioso, specie se il TSAP è usato raramente)
- **Uso di un name (directory) server**, che detiene l'associazione tipo di processo – porta, e al quale tutti si rivolgono quando desiderano stabilire connessioni, potendo quindi conoscere il TSAP da adottare per le connessioni con altri server; il sistema è centralizzato e richiede la preventiva registrazione dei servizi



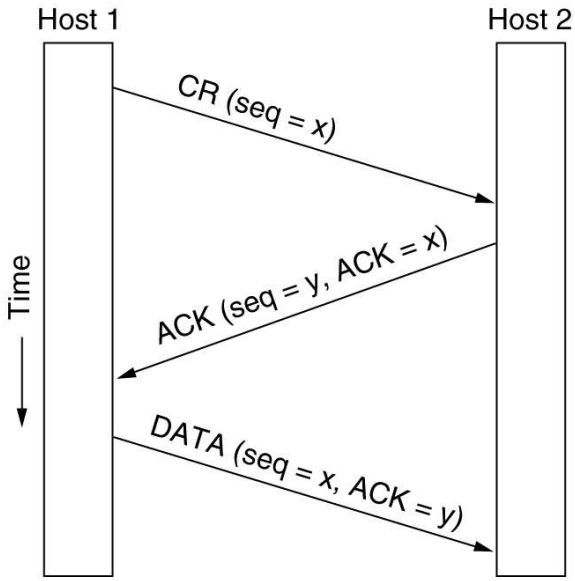


Gestione della connessione – 1

- La gestione della connessione non è semplice, solitamente si analizzano separatamente l'instaurazione ed il rilascio
- **Stabilire una connessione** presenta il problema di eventuali **duplicati ritardati** (pacchetti che ricompaiono perché restituiti dai router); per risolvere il problema:
  - si potrebbero usare TSAP monouso, ma così i TSAP non potrebbero essere gestiti efficacemente,
  - assegnare ad ogni connessione un ID opportuno, ma questo costringerebbe gli host a mantenere lo storico in tabelle locali
  - di fatto, anziché addossare il problema agli host, si cerca di garantire che nella rete i **pacchetti obsoleti possano essere distrutti con certezza entro un certo tempo**; l'intervallo effettivo che deve trascorrere per considerare eliminato un pacchetto deve poi essere più lungo del suo TTL (di solito un opportuno multiplo) per avere la garanzia che anche tutti gli ACK associati siano stati eliminati; ogni pacchetto utilizza un **numero di sequenza** per potere distinguere quelli obsoleti dai nuovi (riconoscimento duplicati); la scelta dei numeri è regolata in base al multiplo del TTL, che li vieta per un certo intervallo

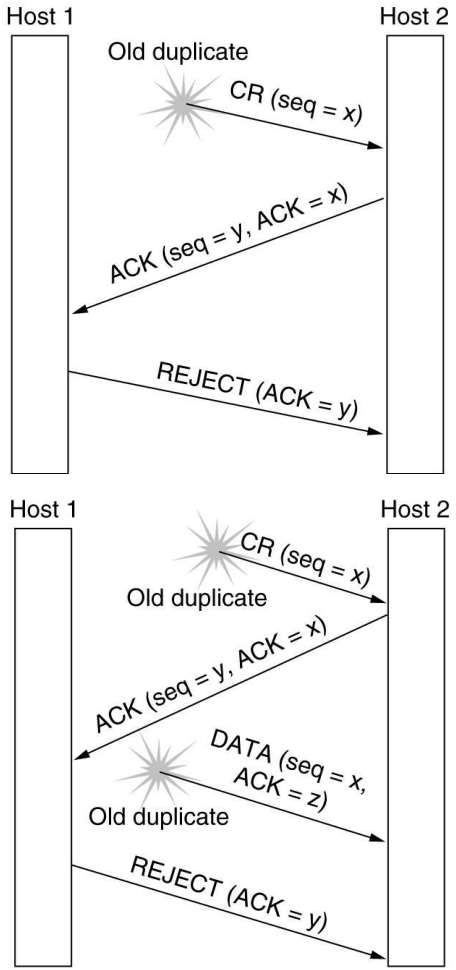
Gestione della connessione – 2

- In particolare nella **prima fase di instaurazione**, si adotta il **protocollo handshake a tre vie**:
  - nella prima fase, una CONNECTION REQUEST (CR) viene inviata all'host 2 indicando il numero di sequenza scelto dall'host 1
  - nella seconda, l'host 2 comunica l'ack a 1 e contemporaneamente anche il numero di sequenza scelto per da 2 stesso (in generale diverso da 1, così possono essere gestiti localmente e separatamente)
  - nella terza fase, che coincide con il primo invio di dati, 1 comunica l'ack a 2



Gestione della connessione – 3

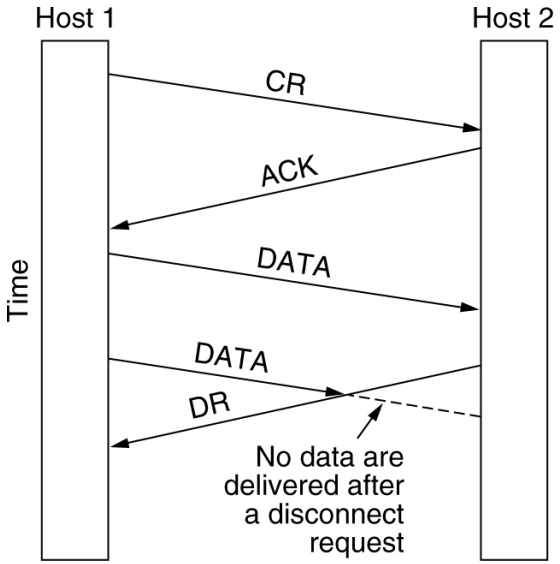
- il **protocollo handshake a tre vie** funziona anche in presenza di duplicati ritardati:
  - se arriva una CR obsoleta di una vecchia connessione, 2 risponderà con il suo ack, ma riceverà da 1 un rifiuto, in quanto 1 non ha mandato una CR recentemente, in questo modo 2 ed 1 non instaureranno una connessione
  - se arriva una CR ed anche un DATA-ACK, entrambi obsoleti, 2 manda l'ack a 1 con il numero z, che non coinciderà con il numero del pacchetto DATA-ACK (z), il che provocherà da parte di 2 il rifiuto della connessione; 1 rifiuta comunque perché non aveva inviato nessun CR recente
  - in generale, il protocollo garantisce che nessuna combinazione di vecchie TPDU può provocare l'instaurarsi accidentale di una connessione indesiderata



Gestione della connessione – 4

Disconnessione:

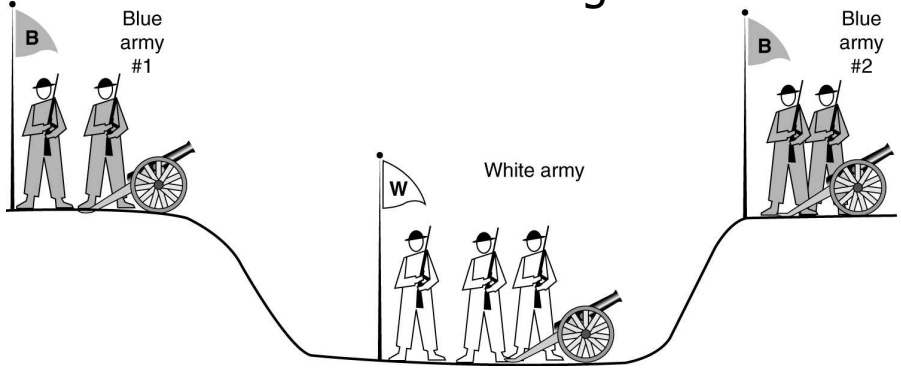
- La **disconnessione asimmetrica non viene utilizzata**, in quanto può generare perdita a causa della non conoscenza della controparte della avviata chiusura (host 1 riceve una DR ma ha inviato dei dati che saranno persi)
- la **disconnessione simmetrica** tout-court non è comunque ancora in grado di assicurare un buon funzionamento, in quanto entrambe le parti devono essere d'accordo sul momento di inizio della disconnessione, ossia sul fatto che per entrambe lo scambio di dati è completo; se una delle due non ha questa coscienza, la simmetrica è ancora assimilabile ad una asimmetrica, almeno nelle conseguenze; il problema è descritto nell'esempio del problema dei due eserciti



Gestione della connessione – 5

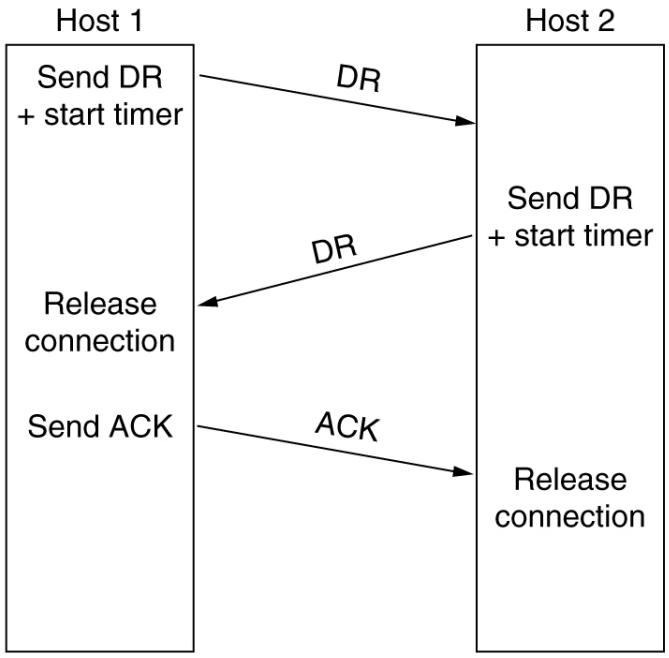
Disconnessione:

- **problema dei due eserciti:** i due eserciti blu, se attaccassero insieme, vincerebbero contro quello bianco, ma occorre l'accordo, e questo si concretizza in messaggi inviati da un messaggero nella valle dell'esercito bianco (canale soggetto ad errori)
  - se 1 manda a 2 l'informazione su quando attaccare, 2 potrebbe non ricevere il messaggio oppure si e mandare una risposta che si perde; entrambi i casi sono indistinguibili per 1, che quindi nel dubbio non attacca;
  - se nessuno dei due casi di prima si verifica, ossia 2 riceve l'informazione e manda l'ack, non sarà comunque sicuro se l'ack arriva ad 1 oppure no perché 1 non manda un ulteriore ack, quindi 2 non attacca perché nel dubbio;
  - l'aggiunta di ack ulteriori **non risolverà mai il problema** perché chi invia l'ack non potrà mai essere sicuro che venga ricevuto



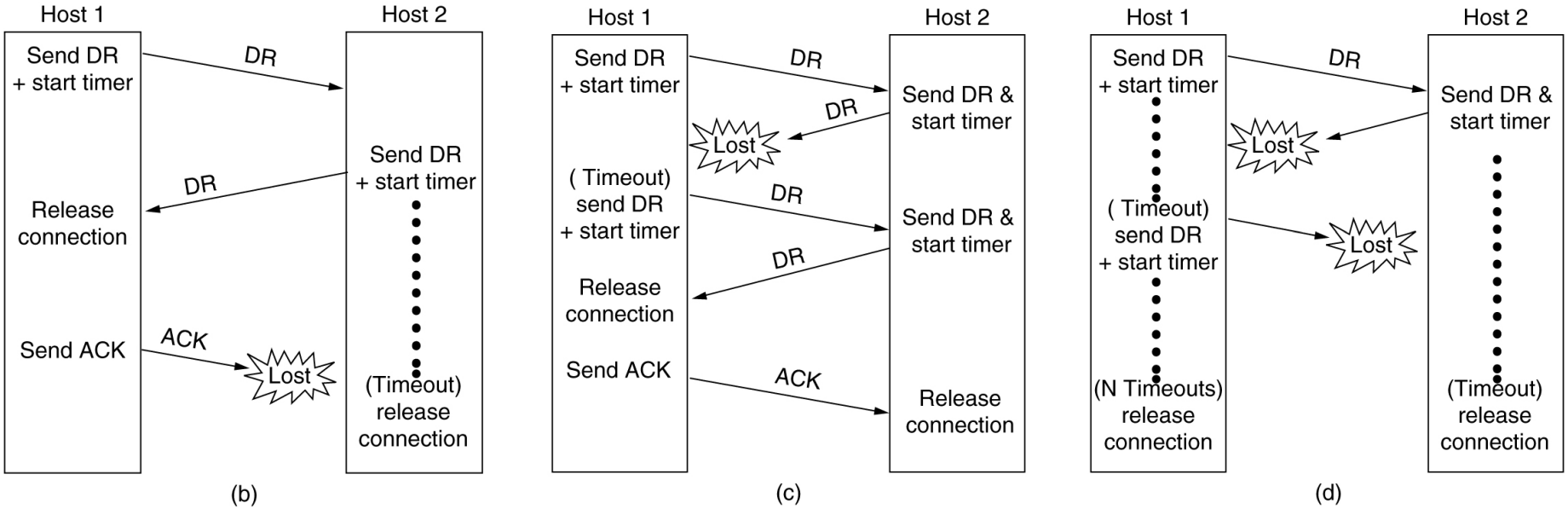
Gestione della connessione – 6

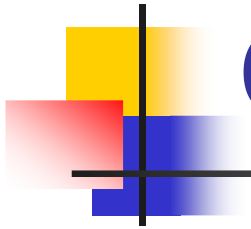
- il **problema della disconnessione non è risolvibile**; **si accetta un protocollo più debole**, che non da garanzie assolute ma è accettabile nella maggior parte dei casi
- **si utilizza un handshake a tre vie con timeout**, che permettono di rilasciare le connessioni in caso di perdita di messaggi, anche se questo potrebbe provocare un rilascio asimmetrico



Gestione della connessione – 7

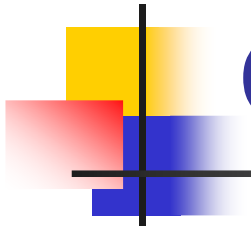
- **il protocollo di disconnessione funziona in vari casi**: il caso (b) illustra la perdita di un ACK; comunque il timeout permette il rilascio; il caso (c) illustra la perdita della DR della seconda fase, evento che fa ripartire tutto dall'inizio; il caso (d) illustra perdite ripetute, che provocano alla fine l'intervento del timeout che rilascia le connessioni





Gestione della connessione – 8

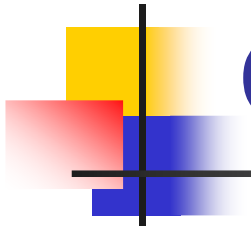
- **il fallimento del protocollo di disconnessione** si verifica ad esempio quando la prima DR viene sempre persa; in tal caso, l'host 1 alla fine rilascerà la connessione tramite timeout (quindi entro un numero finito di tentativi di invio di DR), mentre l'host 2 non avrà mai conosciuto l'intenzione di lasciarla, verificandosi la asimmetria; inviare DR fino ad ottenere risposta (che è come dire non usare i timeout) provoca livelock quindi il male minore è la asimmetria
- per non lasciare in sospeso le connessioni, l'host 2 che non ha mai ricevuto il DR potrebbe attivare una regola secondo cui **una connessione inattiva per troppo tempo viene considerata morta** anche in assenza di DR; in tal modo, 2 chiuderà la connessione anche senza avere ricevuto DR da 1; questo però pone il problema che se la connessione è inattiva per lungo tempo ma nessuno desidera chiuderla, la regola ne provocherebbe la chiusura indesiderata; allora è possibile mandare periodicamente una TPDU fittizia per tutte le connessioni che, pur inattive per lunghi periodi, si desidera mantenere, ma se le TPDU fittizie sono tutte perse si ritorna al punto di partenza, come quando si perdevano tutte le DR
- tutte le soluzioni in definitiva non sono ottimali (possono fallire) ma accettabili (il fallimento si ha in presenza di più fattori poco probabili)



Controllo di flusso e buffering - 1

- **Il buffering ed il controllo di flusso** del TL sono più complessi perché il numero di connessioni varia dinamicamente
- per affrontare la **bufferizzazione** occorre considerare diversi aspetti:
  - servizio datagram vs connection-oriented
  - dimensione del buffer
  - buffer su mittente, ricevente o entrambi
  - meccanismo di gestione dinamica dei buffer
- il **controllo di flusso** viene garantito agendo su due fattori:
  - buffer di mittente e ricevente
  - capacità di trasporto della sottorete

  possono diventare collo di bottiglia uno dei due fattori o entrambi



Controllo di flusso e buffering - 2

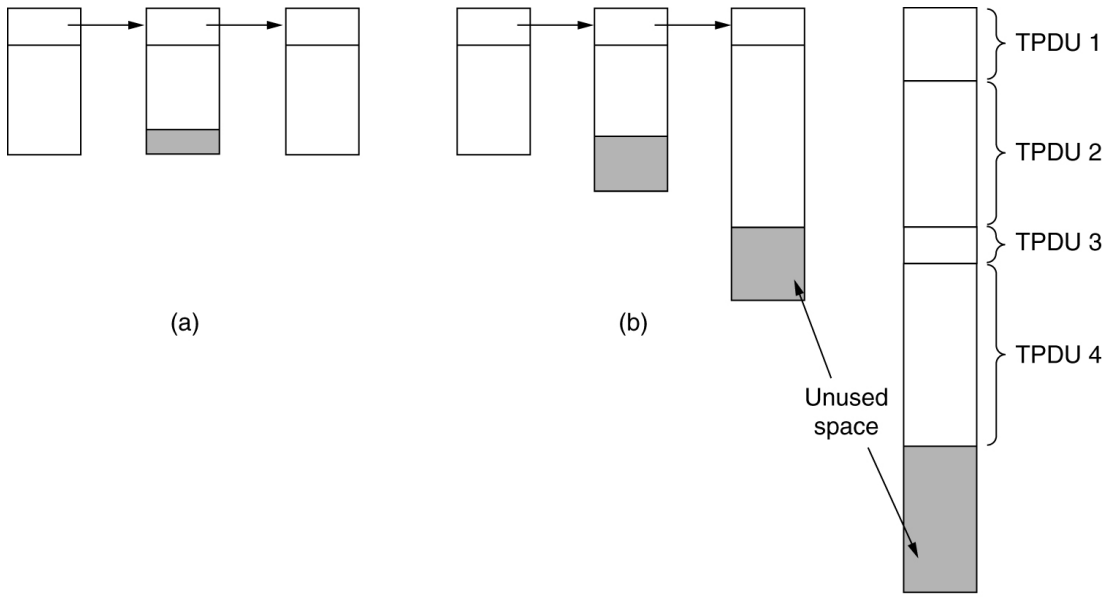
Bufferizzazione - Datagram vs connection-oriented:

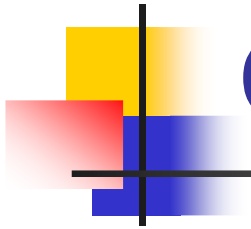
- nel **servizio datagram** è infatti opportuno che il mittente bufferizzi tutto quello che invia (il servizio datagram è infatti inaffidabile)
- in **presenza di connessioni**, il mittente potrebbe non bufferizzare se sa che il ricevente ha sempre buffer, dovrebbe invece bufferizzare se i buffer, pur presenti, potrebbero non soddisfare le esigenze (in tal caso la perdita di pacchetti non sarebbe dovuta all'inaffidabilità del canale ma all'esaurimento dello spazio da parte del ricevente)
- i buffer possono poi essere **raggruppati** e gestiti per connessione, con un ulteriore gruppo per il servizio datagram o appartenere ad un unico pool di buffer usati sia per le connessioni che per i servizi datagram, o ancora essere divisi in due soli gruppi: servizi connection-oriented e datagram

Controllo di flusso e buffering - 3

Bufferizzazione - dimensione del buffer:

- si possono avere **buffer concatenati a dimensione fissa** (a), generalmente pari alla dimensione di una TPDU, tuttavia le TPDU possono essere di dimensione diversa (il payload può differire), quindi la dimensione fissa può comportare spreco (se scelta mediamente superiore rispetto alla dimensione media delle TPDU), o inefficenza (se troppo piccola, situazione che costringe a usare più buffer per la stessa TPDU)
- alternativamente, si possono usare **buffer concatenati di dimensione variabile** (b) o all'estremo **un solo buffer circolare** (c), in cui il confine di ogni buffer non è definito (anche se la gestione è più complessa, quindi si rivela utile solo in caso di utilizzo pesante)

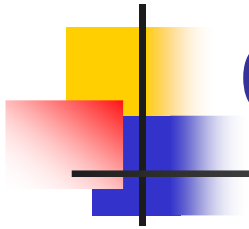




Controllo di flusso e buffering - 4

Bufferizzazione - mittente o destinatario:

- per **traffico irregolare con poca bandwidth richiesta** (ad esempio accesso su terminale), conviene avere buffer dinamici per mittente e ricevente, e poiché questo implica che il ricevente potrebbe non avere buffer in un dato istante, il mittente deve bufferizzare (si fa carico del problema)
- **per traffico uniforme ad elevata larghezza di banda** (tipo trasferimento file) è fondamentale che sia il ricevente a garantire la bufferizzazione, in modo da potere ricevere tutto e sfruttare al meglio la banda
- le due casistiche suggeriscono chi deve principalmente bufferizzare, senza escludere l'altro (è quindi in generale possibile ed auspicabile bufferizzare su entrambe le estremità)



Controllo di flusso e buffering - 5

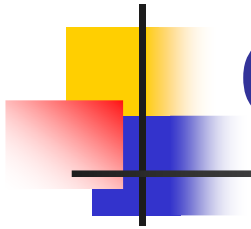
Bufferizzazione – algoritmo per la gestione dinamica

- un modo generico per gestire dinamicamente i buffer consiste nel **separare il buffering dall'ack**, a differenza del DLL dove si bufferizzavano tutti e soli i frame di cui l'ack non era pervenuto (sliding window protocols)
- **l'algoritmo a livello di trasporto** opera come segue:
  - il mittente A richiede un certo numero di buffer (in base alle proprie esigenze) al ricevente B, che può o meno garantirglieli (in base alla propria disponibilità);
  - durante il funzionamento, A manda pacchetti, decrementando il numero dei buffer disponibili su B, fermandosi se arriva a zero senza che B nel frattempo ne abbia allocati di nuovi;
  - B dal canto suo manda (in piggybacking o meno) informazioni sugli ack dei pacchetti e sui buffer dinamicamente allocati

Controllo di flusso e buffering - 6

Bufferizzazione – algoritmo per la gestione dinamica: esempio

	A	Message	B	Comments
1	→	< request 8 buffers>	→	A wants 8 buffers
2	←	<ack = 15, buf = 4>	←	B grants messages 0-3 only
3	→	<seq = 0, data = m0>	→	A has 3 buffers left now
4	→	<seq = 1, data = m1>	→	A has 2 buffers left now
5	→	<seq = 2, data = m2>	• • •	Message lost but A thinks it has 1 left
6	←	<ack = 1, buf = 3>	←	B acknowledges 0 and 1, permits 2-4
7	→	<seq = 3, data = m3>	→	A has 1 buffer left
8	→	<seq = 4, data = m4>	→	A has 0 buffers left, and must stop
9	→	<seq = 2, data = m2>	→	A times out and retransmits
10	←	<ack = 4, buf = 0>	←	Everything acknowledged, but A still blocked
11	←	<ack = 4, buf = 1>	←	A may now send 5
12	←	<ack = 4, buf = 2>	←	B found a new buffer somewhere
13	→	<seq = 5, data = m5>	→	A has 1 buffer left
14	→	<seq = 6, data = m6>	→	A is now blocked again
15	←	<ack = 6, buf = 0>	←	A is still blocked
16	• • •	<ack = 6, buf = 4>	←	Potential deadlock

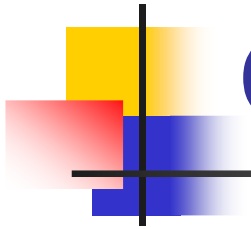


Controllo di flusso e buffering - 7

Bufferizzazione – algoritmo per la gestione dinamica

l'esempio mette in evidenza che:

- le informazioni sugli ack dei pacchetti e sui buffer dinamicamente allocati non sono legate: mandare ack non significa che i buffer siano nuovamente liberi, il loro numero è comunicato separatamente, quindi si potrebbero verificare situazioni di **stallo** (riga 15 dell'esempio)
- in **reti datagram** si potrebbero anche perdere dati (riga 5 dell'esempio) o peggio informazioni di controllo (riga 16 dell'esempio)
- per evitare questi problemi, A e B si scambiano periodicamente informazioni di controllo sullo stato degli ack e dei buffer



Controllo di flusso e buffering - 8

Controllo di flusso

il limite imposto alla velocità dei dati può derivare da due fattori:

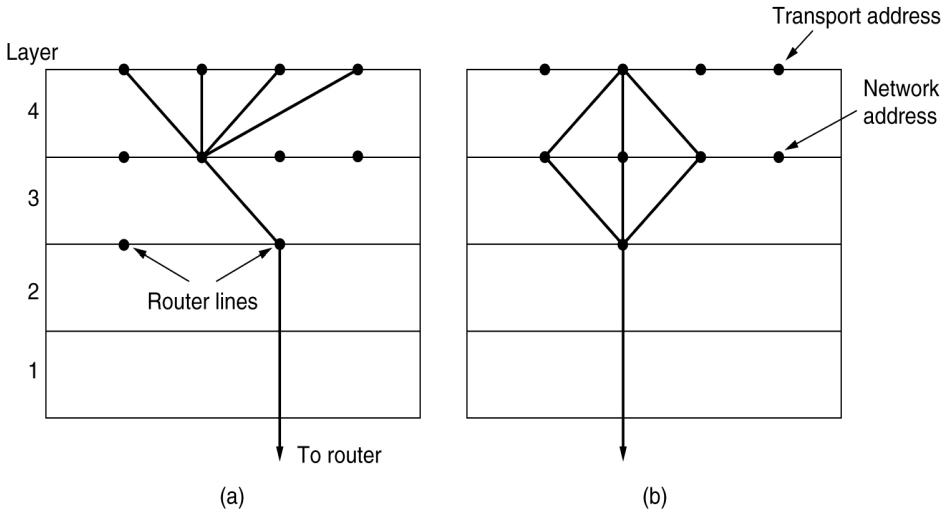
- buffer su mittente o ricevente
- capacità di trasporto della rete
- storicamente, l'hardware dei buffer è a costo decrescente, per cui il collo di bottiglia tende a diventare il secondo fattore, anche se in generale entrambi devono essere considerati; in definitiva occorre controllare il flusso agendo sui buffer ma anche tenendo conto della capacità di trasporto della rete
- un **meccanismo di controllo di flusso che tiene conto della capacità della rete** va applicato al mittente, per evitare che provochi congestione nella rete stessa, e prevede che la dimensione della finestra di trasmissione dipenda dalla capacità della rete, in particolare dal numero di TPDU gestibili al secondo (c) e dal tempo necessario per gestirne una (r); il protocollo deve anche periodicamente rilevare questi dati per aggiornare la dimensione della finestra

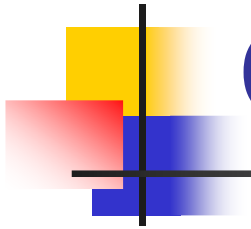
Multiplexing

Multiplexing

Il multiplexing al livello di trasporto prevede **due possibilità**:

- **upward multiplexing**, che si verifica quando più connessioni del livello di trasporto sono mappate su una di livello 3; tutte le connessioni attivate simultaneamente dallo stesso host (browsing, posta elettronica, rete windows) sono multiplexate ad opera del software di trasporto sull'unica connessione di livello 3 disponibile
- **downward multiplexing**, che invece si opera quando una connessione di livello 4 (trasporto) necessita di molta banda, e allora viene multiplexata su più connessioni di livello 3; l'uso delle due linee da 64kbps dell'ISDN per ottenere una connessione da 128kbps è un esempio di tale multiplexing, che richiede un intervento a livello 3 (il set di linee deve essere disponibile a livello di rete, non basta il software di trasporto a risolvere il problema)





Crash Recovery – 1

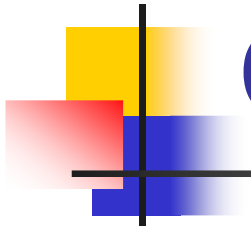
Crash recovery

si distinguono **due situazioni**:

- (a): **crash della rete** fra due host (rete o router è irrilevante)
- (b): **crash di un host**

Nel caso (a) gli host si trovano di fronte a **due sottocasi**:

- **servizio network di tipo datagram**, in cui le entità di trasporto prevedono la perdita di una TPDU e sanno come gestirla
- **servizio network di tipo connection-oriented**, caso in cui le entità di trasporto creano una nuova connessione e si scambiano informazioni per sapere dove erano arrivate e da lì proseguire

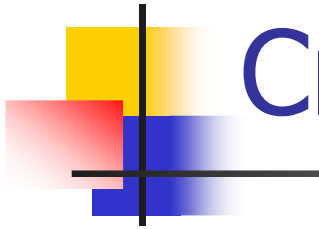


Crash Recovery – 2

Crash recovery

Nel caso (b), ossia crash di un host:

- **chi stava trasmettendo** può adottare quattro strategie, indicate nella tabella che segue (S1 indica che l'host aveva TPDU in attesa di ack, S0 che non ne aveva)
- **chi stava ricevendo**, deve distinguere gli eventi, in generale temporalmente disgiunti, di invio dell'ack (A) all'entità di trasporto dell'altro host, e di inoltro locale (W) della TPDU all'entità di livello superiore nello stesso host; il crash (C) può avvenire prima, durante o dopo rispetto ad A e W, a loro volta in uno dei due ordini AW o WA, a seconda dell'implementazione del software di trasporto
- la tabella riassume tutte le possibili configurazioni, evidenziando come **in ogni caso ci sia sempre una situazione di fallimento del protocollo**, tutto a causa della non contemporaneità dei due eventi A e W

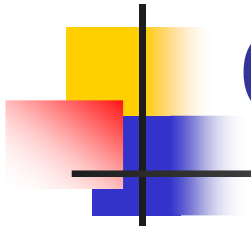


Crash Recovery – 3

Crash di un host: possibili configurazioni

Strategy used by sending host	Strategy used by receiving host: First ACK, then write			Strategy used by receiving host: First write, then ACK		
	AC(W)	AWC	C(AW)	C(WA)	W AC	WC(A)
Always retransmit	OK	DUP	OK	OK	DUP	DUP
Never retransmit	LOST	OK	LOST	LOST	OK	OK
Retransmit in S0	OK	DUP	LOST	LOST	DUP	OK
Retransmit in S1	LOST	OK	OK	OK	OK	DUP

OK = Protocol functions correctly
DUP = Protocol generates a duplicate message
LOST = Protocol loses a message



Crash Recovery – 4

Crash recovery

- la non contemporaneità dei due eventi A e W rende il crash ed il successivo recovery due eventi **non più trasparenti per il livello sovrastante** (in conseguenza del fatto che il livello corrente di trasporto fallisce in talune configurazioni); questo implica che è il livello superiore (applicativo) a dovere risolvere il problema, compito assolvibile comunque solo se si dispone di sufficienti informazioni
- la problematica del crash è complessa, e porta a quella della **incertezza dell'ack**: un ack certo end-to-end fra due host, ossia la cui ricezione è garanzia di successo totale e la cui non ricezione è garanzia di insuccesso totale è di fatto impossibile da raggiungere

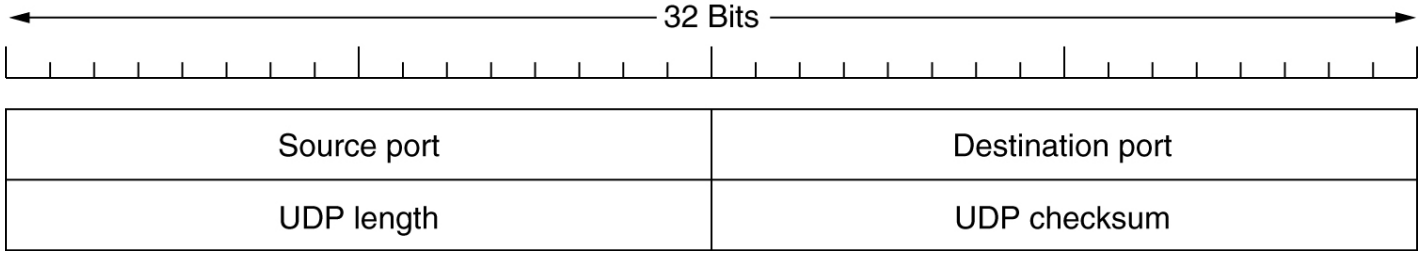
Internet Transport Protocols: UDP – 1

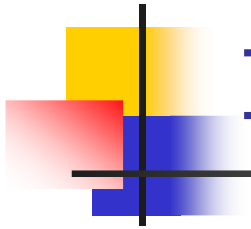
In Internet il livello di trasporto offre due protocolli:

- **UDP** (User Datagram Protocol), senza connessione
- **TCP** (Transport Control Protocol) connection-oriented

UDP

- Offre un modo per potere mandare datagrammi IP senza dovere stabilire una connessione; **di fatto rispetto a IP aggiunge solo le porte**
- UDP trasmette **segmenti** costituiti da un'intestazione di 8 byte seguita dal payload
  - Le porte sorgente e destinazione sono gli indirizzi di livello 4
  - La UDP length è totale, la checksum serve per controllare l'integrità del pacchetto

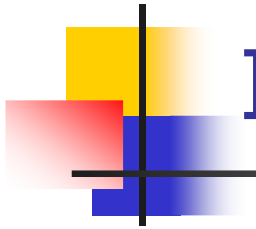




Internet Transport Protocols: UDP – 2

UDP

- **UDP non supporta controllo di flusso o di errore** (ritrasmissione), azioni che devono essere tutte implementate dall'applicazione; si limita pertanto a fornire il demultiplexing tramite l'introduzione delle porte
- **UDP è spesso usato nelle applicazioni di tipo request-reply** fra un client ed un server, situazione in cui l'affidabilità della trasmissione non è determinante (in caso di perdita, il client di solito va in timeout e ripete il tutto) ed in cui si mandano il numero minimo di pacchetti
- Diverse **applicazioni** utilizzano UDP:
  - **RPC** (Remote Procedure Call)
  - **RTP** (Real-Time Protocol)
  - **DNS** (Domain Name System)
  - **TFTP** (Trivial FTP)
  - **NCS** (Network Computing System)
  - **SNMP** (Simple Network Management Protocol)



Internet Transport Protocols: UDP – 3

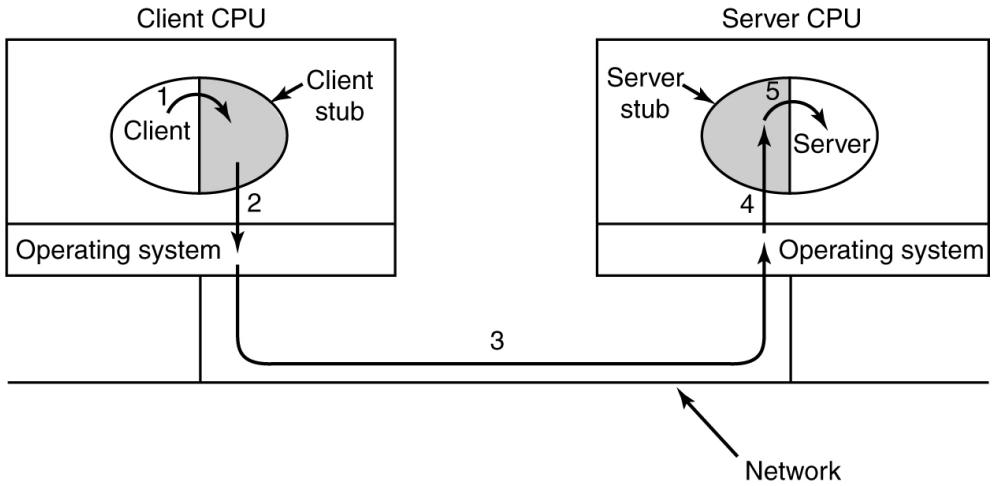
RPC

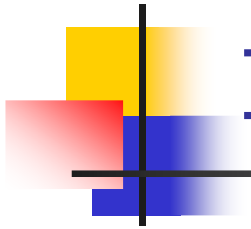
- La chiamata di procedura remota consente a due programmi in esecuzione su host differenti di cooperare, potendo l'uno chiamare una funzione dell'altro (opportunamente dichiarata)
- l'RPC è alla base di molte applicazioni di rete, visto che modella di fatto la comunicazione di tipo request-reply spesso presente fra un client ed un server
- Per consentire al meccanismo di funzionare ed al contempo garantire la trasparenza ai programmi dei due host, esistono due procedure stub (una client ed una server) con le quali si interfacciano i programmi per effettuare l'RPC

Internet Transport Protocols: UDP – 4

RPC

- Il client effettua una chiamata locale al proprio stub, a cui passa i parametri; lo stub si interfaccia con il SO, inserisce i parametri in un pacchetto (di solito UDP), effettuando il cosiddetto marshalling, e richiede l'invio del pacchetto al SO
- Nella controparte, il SO contatta lo stub server, che estrae i parametri dal pacchetto (unmarshalling), e li passa tramite chiamata locale alla procedura server, che elabora i parametri, e fornisce il risultato
- Il risultato viene inviato indietro al client usando (al contrario) lo stesso meccanismo illustrato finora)





Internet Transport Protocols: UDP – 5

RPC

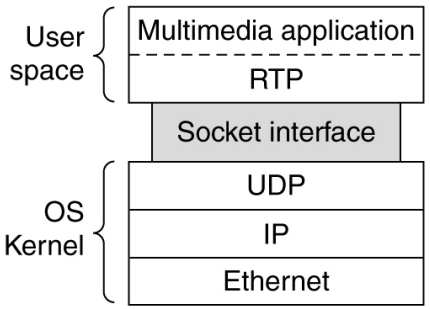
- RPC cerca di garantire la trasparenza ai programmi dei due host, tuttavia esistono **alcune limitazioni**:
  - **I puntatori non possono essere passati** (i due host non condividono memoria), problema che viene bypassato mandando una copia del valore, allocando spazio sul server, e passando allo stub del server il puntatore locale alla variabile il cui valore coincide con quello della variabile locale del client; eventuali cambiamenti al valore tramite puntatore locale del server devono essere propagati in maniera trasparente al client
  - **La tipizzazione debole di alcuni linguaggi** (C, C++) consente ad esempio di non dichiarare la lunghezza di un array, impedendo di fatto il marshalling (il pacchetto deve avere una dimensione certa)
  - **Non è possibile usare variabili globali**, perché client e server hanno memoria non condivisa

  le limitazioni mostrano che nelle situazioni reali l'RPC non è trasparente
- RPC usa di solito UDP, ma per operazioni non idempotenti si appoggia su TCP (esiste il TCP transazionale in versione sperimentale)

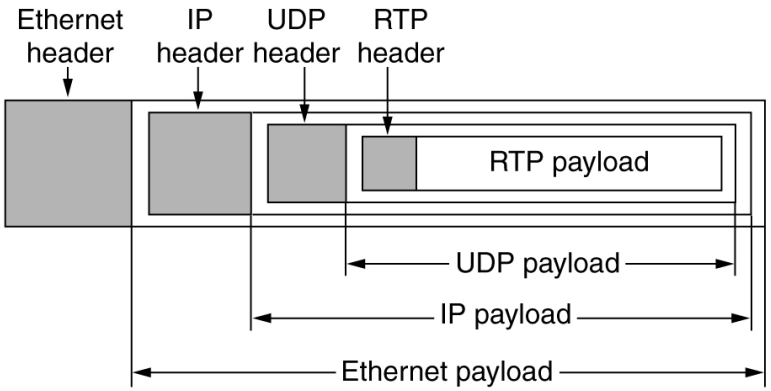
Internet Transport Protocols: UDP – 6

RTP

- Con il crescere delle applicazioni multimediali (videoconferenza, radio via internet, video on demand), si è reso necessario un **protocollo generico per la trasmissione in tempo reale**; RTP è la risposta
- RTP normalmente riceve da un'applicazioni più flussi (video, audio) che compongono la trasmissione multimediale; **la libreria RTP esegue il multiplexing dei flussi**, codificandoli in pacchetti RTP, che sono poi inseriti in un socket ed inviati come pacchetti UDP
- **La posizione di RTP è ambigua**, in quanto essendo eseguito nello spazio utente dovrebbe essere un protocollo dello strato 5 (applicazione), ma essendo generico (indipendente dall'applicazione che ne fa uso) dovrebbe essere di livello 4; di fatto **RTP è un protocollo di trasporto implementato a livello applicativo**



(a)



(b)

Internet Transport Protocols: UDP – 7

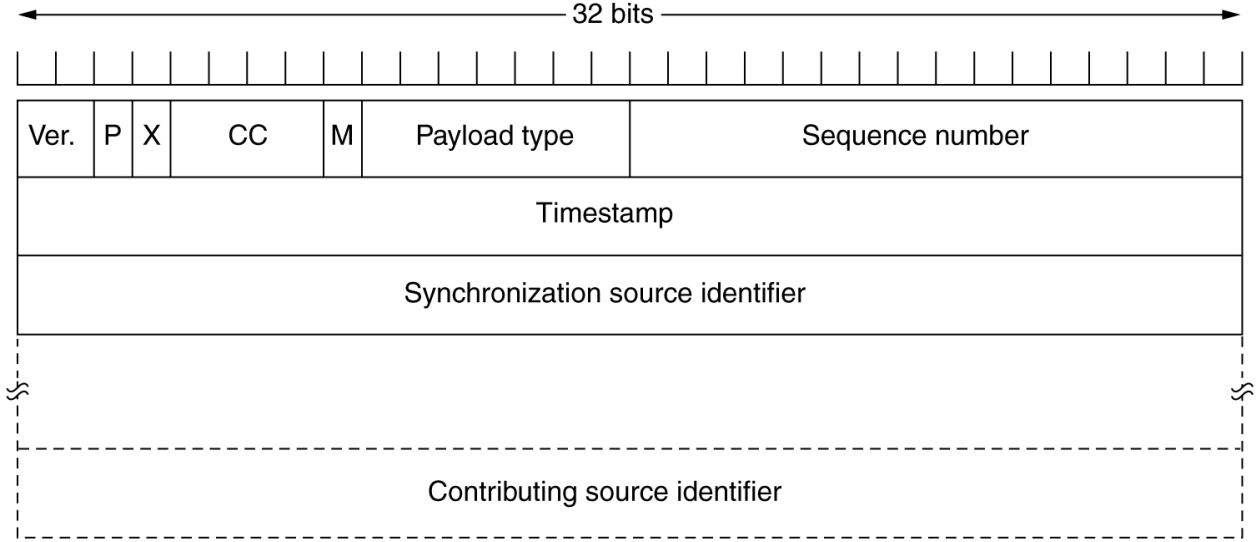
RTP

- **RTP si appoggia su UDP**, pertanto non è trattato particolarmente dai router (anche se incapsula traffico multimediale), a meno che non si attivi una qualche QoS
- **Ogni pacchetto RTP prevede un numero di sequenza**, usato per rilevare perdite di pacchetti, caso in cui non si prevede la ritrasmissione (il pacchetto ritrasmesso arriverebbe comunque troppo tardi), ma piuttosto un'interpolazione
- **Ogni pacchetto RTP può incapsulare più campioni**, RTP supporta anche diverse codifiche
- Il primo campione di ogni pacchetto viene solitamente contrassegnato con un **timestamp temporale**, potendo quindi effettuare un minimo di buffering sui campioni alla destinazione, che può riprodurli nell'istante opportuno (a prescindere dal loro reale arrivo, in quanto fa fede il timestamp e le sue variazioni, anch'esse associate ai campioni successivi); il timestamp consente anche la sincronizzazione dei flussi per la ricostruzione dell'applicazione multimediale alla destinazione

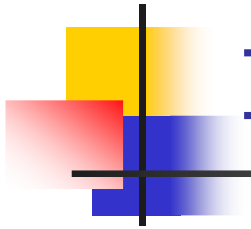
Internet Transport Protocols: UDP – 8

RTP

- **L'intestazione** prevede il campo version, il campo X per estendere l'intestazione in futuro, il campo payload type per indicare la codifica, il numero di sequenza, il timestamp (prodotto da chi genera il flusso), il sync source id indica a quale flusso appartiene il pacchetto



- **RTP si accompagna a RTCP**, che fornisce feedback (per adattare velocità e tipo di codifica), sincronizzazione fra flussi e interfaccia utente (ad esempio, trasporto di informazioni testuali); RTCP non trasporta dati



Internet Transport Protocols: TCP – 1

TCP

- Per la maggior parte delle applicazioni UDP è inadatto, **TCP fornisce invece un servizio affidabile** per un flusso di byte end-to-end su una internetwork inaffidabile
- Un host che implementa TCP possiede **un'entità di trasporto TCP**, che può essere una libreria, un processo utente o una parte del kernel;
- **Funzionamento**:
  - Per effettuare invio di dati, le controparti devono avere instaurato una **connessione tramite socket**, identificato dalla coppia (IP, porta); un socket su ogni host deve essere attivo; lo stesso socket può supportare più connessioni, ma più connessioni non possono usare la stessa coppia di socket sugli host (almeno uno deve essere diverso)
  - l'entità TCP accetta dai processi locali i **flussi dati utente**, li suddivide in segmenti di non più di 64KB (in pratica, usa 1460 byte come dimensione, che si adatta ad un singolo frame Ethernet incluse le intestazioni IP e TCP, valore indicato dalla **MTU**, maximum transfer unit) e li incapsula in **datagrammi IP**; da notare che in quanto datagrammi, non è garantito nulla, per cui tutto il lavoro viene svolto dal TCP, d'altra parte si usano datagrammi perche' si considera inaffidabile la rete sottostante

Internet Transport Protocols: TCP – 2

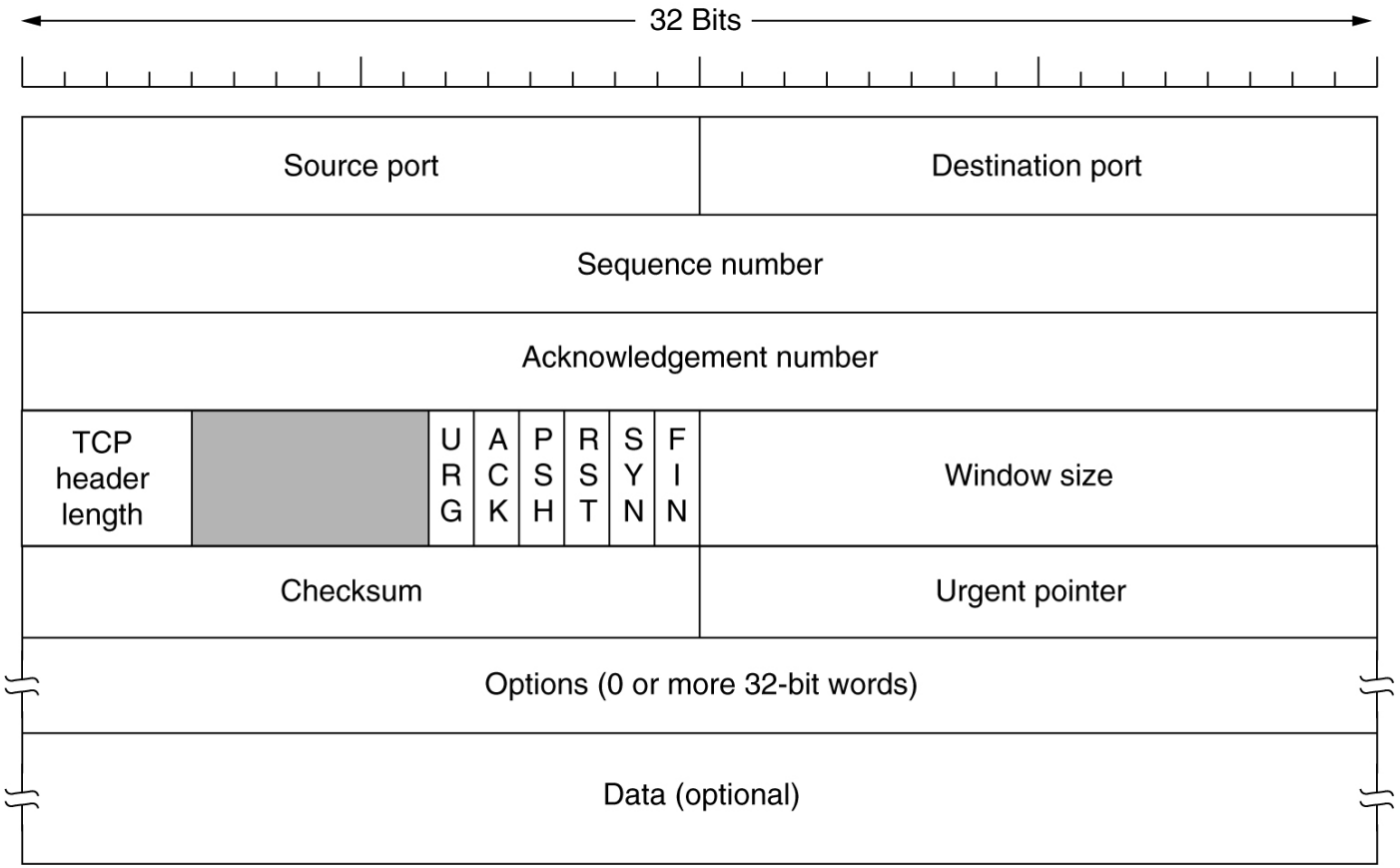
- **Gestione delle porte**: le porte inferiori a 1024 sono note, e per esse di solito si attivano processi dedicati, per le altre, anziché mantenere attivo un server per ognuna di esse, si attiva un solo processo, inetd, che intercetta le richieste di connessione su una porta >1024, crea il processo relativo e redirige la connessione da se stesso al processo appena creato, liberandosene per intercettare future richieste

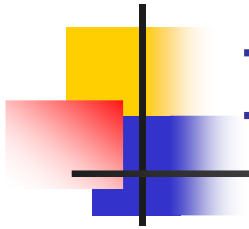
Port	Protocol	Use
21	FTP	File transfer
23	Telnet	Remote login
25	SMTP	E-mail
69	TFTP	Trivial File Transfer Protocol
79	Finger	Lookup info about a user
80	HTTP	World Wide Web
110	POP-3	Remote e-mail access
119	NNTP	USENET news

- Le connessioni TCP sono full-duplex punto-punto (no broadcast né multicast nativi)
- Una connessione TCP è un flusso di byte, qualora rappresentino messaggi, il TCP non ne ha coscienza (inizio e fine di ogni messaggio sono risolti dall'applicazione)
- TCP di solito bufferizza in uscita, ma il flag URGENT si può forzare l'immediato invio
- Il flag PUSH obbliga invece l'entità ricevente di mandare immediatamente i dati all'applicazione ricevente senza bufferizzare in ricezione

Internet Transport Protocols: TCP – 3

Segmento (Pacchetto) TCP

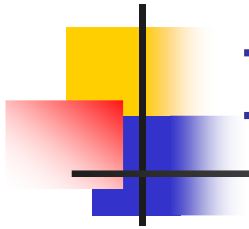




Internet Transport Protocols: TCP – 4

Segmento (Pacchetto) TCP

- L'intestazione di un segmento TCP è lunga 20 byte, più altre eventuali opzioni e i dati, fino a 65535 byte in totale; nell'intestazione, figurano:
  - **porta sorgente e destinazione**
  - il **numero di sequenza** del primo byte dati del segmento
  - L'**ack**, come numero di sequenza del successivo segmento da ricevere (non indica il numero dell'ultimo ricevuto)
  - Il **campo di lunghezza** serve a causa della parte opzionale (variabile)
  - **6 bit sono rimasti liberi** (rivelano la robustezza del TCP, visto che non è sinora sorta l'esigenza di sfruttarli)
  - **6 bit di flag** indicano: URG posto a 1 indica la presenza di dati urgenti la cui posizione è indicata nell'offset urgent pointer posto successivamente; ACK posto a 1 indica un segmento di ack, PSH posto a 1 segnala la presenza di dati push, RST indica di resettare la connessione o esprime un rifiuto ad aprirla (l'applicazione deve individuare il problema), SYN=1 e ACK=0 equivale ad un Connection Request, SYN=1 e ACK=1 indicano Connection Accepted, FIN indica la disconnessione, che è però simmetrica (quindi il processo che manda FIN può ancora ricevere dati dall'altro, che dovrà anch'esso mandare FIN per chiudere definitivamente)



Internet Transport Protocols: TCP – 5

Segmento (Pacchetto) TCP

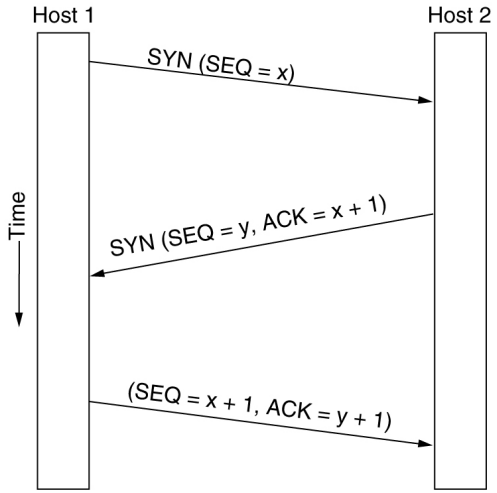
- La **window size** evidenzia la gestione sliding window a dimensione variabile go-back-n adottata dal TCP; la dimensione variabile consente di regolare il flusso indipendentemente dall'ack (anche se sono stati ricevuti tutti i segmenti, viene mandato l'ack relativo ma non è detto che sia disposto ad accettare dati)
- Il campo **checksum** viene calcolato coinvolgendo anche informazioni IP (pseudointestazione con indirizzi souce e destination IP), il che permette anche di rilevare pacchetti IP consegnati in modo errato, pur costituendo una violazione dell'indipendenza dei protocolli
- Le **opzion**i possono essere diverse, tre le più usate:
  - Una che consente ad ogni host di specificare il **massimo carico utile TCP**; un host potrebbe infatti avere limitazioni locali, ed in tal modo i due negoziano il valore accettabile per entrambi
  - L'opzione **window scale** consente di shiftare di 14 bit a sinistra il campo window size, per incrementare la dimensione della finestra (nelle reti odierne a larga banda, la dimensione della finestra può essere elevata per ridurre i tempi di attesa ma il campo previsto è di soli 16 bit
  - Una terza opzione implementa la **ripetizione selettiva**

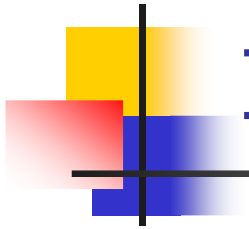
Internet Transport Protocols: TCP – 6

Connessione TCP

Per gestire la connessione, TCP usa **l'handshake a tre vie**

- Solitamente un host (ad esempio un server) invoca LISTEN ed ACCEPT, restando in ascolto per una connessione
- L'altro host (client) esegue una CONNECT con l'IP, la porta e la dimensione massima del segmento TCP ammessa; la primitiva invia il segmento TCP con SYN=1 e ACK=0
- L'entità TCP remota controlla se su quella porta c'è un processo in ascolto, e procede con l'ack, che riceve una ulteriore conferma con il primo segmento dati; se non ci sono processi in ascolto su quella porta, viene inviato un RST
- Il rilascio è simmetrico e richiede il FIN da ambo i lati; si usano dei timer per evitare il problema dei due eserciti

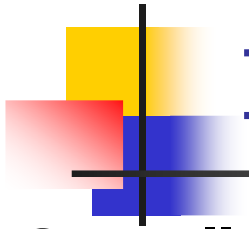




Internet Transport Protocols: TCP – 7

Gestione Finestra

- TCP varia dinamicamente la dimensione della finestra, utilizzando due algoritmi per ottimizzarne la gestione
  - **l'algoritmo di Nangle** gestisce la finestra per i casi in cui l'applicazione di invio consegna troppo lentamente i dati all'entità TCP locale, situazione che spesso richiede molti byte di intestazione e controllo per mandare pochi byte di dati; l'approccio prevede di mandare il primo byte, e bufferizzare il resto fino alla ricezione del primo ack, momento in cui i dati sono tutti inviati con un solo segmento (bufferizzazione a tratti)
  - **l'algoritmo di Clark** regola la dimensione della finestra quando è il ricevente a prelevare troppo lentamente i dati, che provoca il problema di buffer di ricezione quasi sempre pieno; l'approccio prevede di consentire al mittente di inviare dati quando il buffer di ricezione è "abbastanza" vuoto (ad esempio, quando è in grado di ricevere la dimensione massima del segmento)



Internet Transport Protocols: TCP – 8

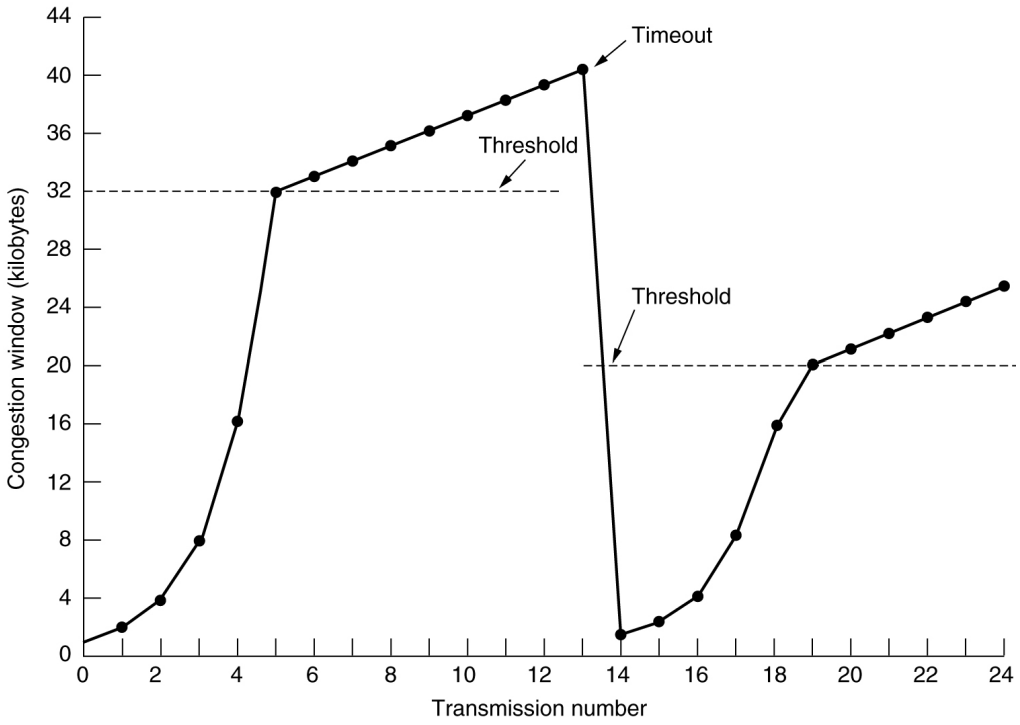
Controllo congestione

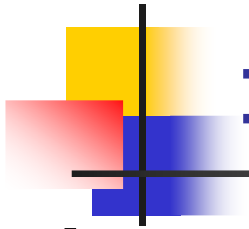
- anche se lo strato di network effettua il controllo della congestione, lo strato di trasporto è anch'esso protagonista perché spesso l'unico vero rimedio è la riduzione della velocità di trasmissione, gestibile in locale dalle entità di trasporto; **TCP gestisce la congestione** combinando **tre strategie**: prevenzione, rilevamento, cura
  - La **prevenzione** viene attuata tenendo presenti due fattori, ossia la capacità della rete e il ricevitore lento, il primo tramite la **finestra di congestione** ed il secondo con quella già utilizzata per inviare dati; la quantità di dati inviata viene regolata al minimo delle due dimensioni; la finestra di congestione viene inizialmente posta dall'entità di trasporto pari alla dimensione del segmento massimo usato sulla connessione, e successivamente l'entità prova ad inviare il segmento; se arriva correttamente (ack entro un timeout), raddoppia la dimensione, reiterando invio ed incremento in caso di successo; **l'algoritmo viene denominato avvio lento**
  - il **rilevamento della congestione** avviene tramite timeout, in quanto TCP suppone che la perdita di pacchetti entro un certo tempo sia imputabile solo alla congestione e quasi mai alla rete (così non è per il wireless)

Internet Transport Protocols: TCP – 9

Controllo congestione

- La **cura della congestione** prevede una **soglia**, inizialmente pari a 64KB, che limita il raddoppio della dimensione della finestra di congestione (dopo la soglia, il raddoppio è bloccato, e l'aumento diventa lineare); in caso di timeout (rilevamento congestione), la finestra di congestione viene fatta ripartire da zero e la soglia viene dimezzata, azioni che, in sintesi, riducono il traffico

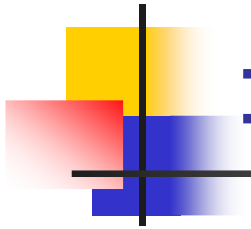




Internet Transport Protocols: TCP – 10

Timer TCP

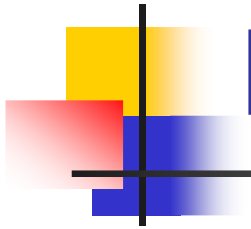
- TCP utilizza diversi timer, il più importante dei quali è il **timer di ritrasmissione**, quello che in caso di mancato ack provoca la ritrasmissione
- **la scelta del timer è complessa**, in quanto nel DLL il ritardo ha una bassa varianza, nel TCP la varianza è eccessiva, quindi si adotta un algoritmo dinamico che regola il timeout con continue misurazioni delle prestazioni di rete
- sono usati anche **altri timer**:
  - **timer di persistenza**, usati per prevenire situazioni di stallo
  - **timer keepalive**, usato per rilevare inattività su una connessione ed eventualmente rilasciarla
  - **timer di chiusura connessione**, che durante l'operazione lascia passare il tempo necessario alla rimozione di tutti i pacchetti generati dalla connessione



Internet Transport Protocols: TCP – 11

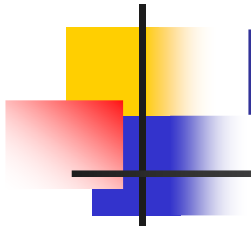
TCP Wireless

- Il TCP dovrebbe essere indipendente dai livelli inferiori, ma spesso le scelte progettuali tengono conto della rete sottostante
- Nelle reti wireless il problema peggiore è la perdita di dati intrinseca nella rete, che inganna il meccanismo di controllo della congestione TCP, che rallenta il traffico laddove sarebbe meglio insistere nella ritrasmissione, essendo nelle reti wireless la perdita dovuta non alla congestione ma principalmente alla inaffidabilità della trasmissione radio
- il TCP potrebbe cambiare comportamento, ma occorre conoscere la rete sottostante; a peggiorare le cose, spesso la rete potrebbe essere eterogenea (wired e wireless); in tali casi, si implementa il TCP indiretto, che anziché instaurare una connessione, ne crea diverse (una per ogni tratta wired o wireless), oppure si addestrano le stazioni base wireless ad operare per rimediare all'inefficacia del wireless, senza fare trasparire la cosa alla rete wired



Prestazioni – 1

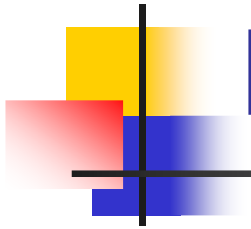
- Le prestazioni della rete sono essenziali per il suo funzionamento, anche se allo stato attuale **la teoria per affrontare il problema è piuttosto scarsa**, portando spesso reti ormai complesse a problemi di prestazioni la cui analisi è spesso un'arte
- il problema viene in parte anche affrontato dal NL, anche se esso tendenzialmente si occupa del routing e del controllo della congestione; peraltro spesso il problema risiede nel software di rete sugli host e/o dalle risorse disponibili per gli stessi, quindi il livello di trasporto è coinvolto
- gli **aspetti** da considerare sono:
  - elenco dei problemi di prestazioni
  - misurazione delle prestazioni
  - progettazione di una rete tenendo conto delle prestazioni
  - elaborazione rapida delle TPDU
  - protocolli nuovi per reti ad alte prestazioni



Prestazioni – 2

I problemi di prestazioni possono avere varie cause:

- **cause temporanee**, come la congestione dei router per elevato traffico
- **cause di tipo strutturale,** ad esempio una linea a 1Gbps interfacciata con un'altra a 56Kbps determina un collo di bottiglia permanente
- **cause di tipo sincrono**, per esempio una TPDU errata mandata in broadcast potrebbe provocare parecchi messaggi di errore (broadcast storm), o la semplice mancanza di energia elettrica e quindi il contemporaneo riavvio di macchine potrebbero innescare un calo delle prestazioni
- **le reti moderne con elevato data rate (Gbps) aumentano il problema,** evidenziando i limiti hardware degli host e/o i limiti del protocollo utilizzato; solitamente il parametro da considerare è il prodotto banda – ritardo, che misura la capacità del canale, e che i valori del protocollo e delle risorse disponibili dovrebbero supportare



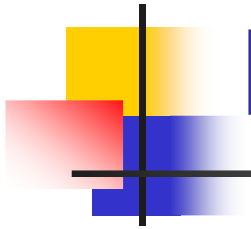
Prestazioni – 3

Il **ciclo** seguito per incrementare le prestazioni di solito prevede **tre fasi**:

- misurare le prestazioni
- analizzare le cause di eventuali problemi
- alterare un qualche parametro di rete e ripetere dall'inizio

Spesso però **diversi fattori costituiscono un ostacolo** a questo processo:

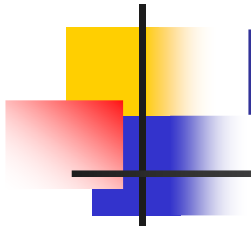
- assicurarsi che la quantità di campioni sia sufficiente
- assicurarsi che i campioni siano significativi
- utilizzare un clock preciso
- prevenire fenomeni anomali durante la misurazione (spesso per evitarli il traffico è fittiziamente generato)
- evitare l'intervento delle cache locali che limitando l'accesso in rete possono falsare le misurazioni



Prestazioni – 4

La fase di progettazione si fonda sui seguenti principi:

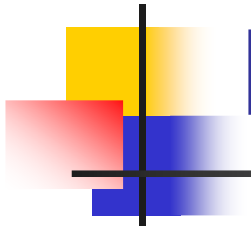
- la velocità di elaborazione è più critica della velocità di rete (ormai)
- in generale, è meglio ridurre il numero di pacchetti (incrementandone la dimensione) per ridurre l'overhead del generico livello
- minimizzare i context switch (fra kernel e modalità utente) nell'esecuzione del protocollo
- minimizzare le copie multiple che vengono generate in locale (dalla scheda di rete in un buffer del kernel, da qui in uno utente ecc.) durante l'esecuzione del protocollo
- è in generale possibile incrementare la banda (basta comprarla) ma non diminuire il ritardo (rihiede interventi più strutturali sul protocollo)
- prevenire è in genere meglio che curare
- evitare i timeout, che sono precursori di ritrasmissione, che potrebbe essere uno spreco di risorse



Prestazioni – 5

Elaborazione TPDU

- Le considerazioni precedenti mettono in evidenza che il limite principale è il software di protocollo, non la rete; occorre quindi **migliorare l'elaborazione delle TPDU**; di solito si interviene su **diversi fattori**:
  - Le intestazioni sono mediamente sempre le stesse una volta stabilita la connessione, per cui un'intestazione prototipo viene bufferizzata dall'entità di trasporto, che in esecuzione dovrà soltanto cambiare pochi dati
  - In ricezione, per individuare il record della connessione a cui la TPDU arrivata fa riferimento si usa spesso una tabella hash
  - I buffer devono essere ottimizzati
  - I timer non dovrebbero scadere mai (scelta giusta del timeout)



Prestazioni – 6

Reti ad alte prestazioni

- Le moderne reti veloci rendono ancor più evidente la limitazione introdotta dal protocollo, che dovrebbe adattarsi alle nuove tecnologie (IPv6 va in questa direzione)
- I problemi delle reti ad alte prestazioni sono:
  - Numeri di sequenza a 32 bit, che vengono riutilizzati entro soli 34 secondi
  - Le velocità di comunicazione sono incrementate più rapidamente delle capacità di calcolo
  - I protocolli tipo go-back-n sono troppo scadenti su reti veloci
  - Le linee veloci sono principalmente limitate dal ritardo e non dalla larghezza di banda, cosa da considerare nella progettazione del protocollo
  - Il jitter, specie per i flussi multimediali, è critico
- Le soluzioni adottabili sono:
  - Avere protocolli leggeri, inutile incrementare le risorse (si creano solo nuovi colli di bottiglia)
  - Evitare la retroazione, meglio concordare le risorse prima e scambiarsi poche informazioni di controllo a regime, in questo senso si va verso il connection-oriented
  - Incrementare la dimensione massima dei dati