Rete: livello trasporto (TCP/IP)
Parte 2 - Indice

1. Modello di servizio **TCP**
2. Il protocollo **TCP**
3. Il segmento **TCP**
4. Politica di trasmissione **TCP**
5. Intervallo di *time out*

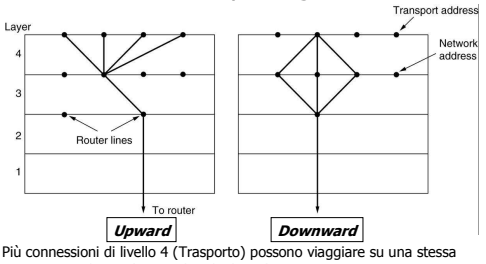
Rete: livello trasporto (TCP/IP)
Modello di servizio TCP - 1

- 2 tipi di servizio a livello trasporto
  - **TCP** (***Transmission Control Protocol***)
    - Garantisce comunicazione affidabile agli utenti (*end points*) anche in presenza di una rete inaffidabile
      - ***Connection-oriented***
  - **UDP** (***User Datagram Protocol***)
    - Fornisce agli utenti esclusivamente la qualità di comunicazione offerta dal livello **IP** sottostante
      - ***Connectionless***
      - Migliore velocità di trasporto, minore affidabilità

Rete: livello trasporto (TCP/IP)
Modello di servizio TCP - 2

- Il servizio **TCP** richiede la creazione di **socket** ai 2 estremi della connessione
  - La coppia di **socket** denota la connessione, la quale può convogliare più conversazioni simultanee tra quei punti terminali
  - Un singolo **socket** può essere punto terminale di più connessioni (*server*)
  - Ogni **socket** ha identità unica, determinata dall'indirizzo **IP** del nodo ospite (32 *bit*) e dal numero di porta locale (16 *bit*)

Rete: livello trasporto (TCP/IP)
Multiplexing



Più connessioni di livello 4 (Trasporto) possono viaggiare su una stessa connessione di rete (*upward multiplexing*)
Una connessione di livello 4 può aggirare limiti di traffico imposti da circuiti virtuali di livello 3 (Rete) attivando più connessioni di rete (*downward*)

Rete: livello trasporto (TCP/IP)
Modello di servizio TCP - 3

- I numeri di porta nell'intervallo 0..255 (***well-known ports***) sono preassegnati ad alcuni servizi standard, alcuni dei quali usano **TCP** altri **UDP**
  - Alcuni servizi standard che usano **TCP**:
    - 25 **SMTP** *Simple Mail Transfer Protocol*
    - 80 **HTTP** *HyperText Transfer Protocol*
    - 110 **POP3** *Post Office Protocol* versione 3

Rete: livello trasporto (TCP/IP)
Modello di servizio TCP - 4

- La connessione **TCP** è bidirezionale (***full-duplex***) e punto a punto
  - Non fornisce supporto per comunicazioni a diffusione (***broadcast***) e neppure per gruppo (***multicast***)
- La connessione **TCP** è a flusso di dati (*byte stream*)
  - Le unità di trasporto **non** corrispondono alle unità dati riconosciute dall'utente applicativo
    - Inviati x B → Trasportati ($x/2 + x/2$) B → Ricevuti x B
- Vi è memorizzazione (e dunque ritardo) tra ricezione da **M** ed emissione su rete e tra ricezione da rete ed emissione verso **D**
  - Le prime versioni di **TCP** prevedevano un'opzione per effettuare consegna immediata (modalità "PUSH")

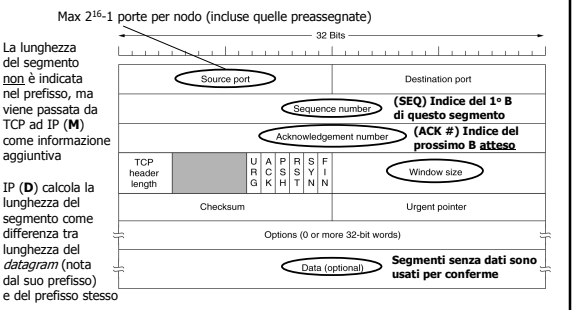
Rete: livello trasporto (TCP/IP)
Il protocollo TCP - 1

- Unità di trasporto **`TCP`** (segmento) composta da un prefisso obbligatorio e da un campo dati utente di ampiezza variabile
- Ampiezza massima di segmento determinata dall'esigenza di:
  - Contenere l'intero segmento entro il campo dati di 1 **`datagram`** = (64k – 1) – 20 B
    - Limite determinato dal livello **`IP`**
  - Tener conto della massima dimensione delle unità di trasporto su rete per evitare costose frammentazioni
    - Limite determinato dalle caratteristiche (di rami) della rete
    - In pratica, la dimensione del campo dati della trama **`Ethernet`** (= 1500 B)

Rete: livello trasporto (TCP/IP)
Il protocollo TCP - 2

- Un segmento troppo largo per le capacità di un tratto della rete viene frammentato in più **`datagram`**, ciascuno con proprio prefisso ed indice di frammento
  - Maggior costo trasmissivo
  - Onere di ricostituzione del segmento, con maggiore complessità nella gestione delle conferme
    - I frammenti vengono confermati nel corretto ordine di sequenza e non nel loro ordine di arrivo (SWP)
      - Arrivo: 1 | 3 | 0 | 4 | 2 → Conferma: - | - | 1 | - | 4

Rete: livello trasporto (TCP/IP)
Il segmento TCP - 1



Rete: livello trasporto (TCP/IP)
Il segmento TCP - 2

- Così come il **`datagram`**, anche il prefisso di segmento deve indicare la propria lunghezza (in unità di 32 *bit*), vista la presenza di un campo opzionale
- Il successivo campo da 6 *bit* era previsto per sviluppi futuri, ma al momento è ancora inutilizzato
- Seguono 6 *flag* da 1 *bit* ciascuno
  - **URG** : se 1, il campo **`Urgent Pointer`** indica, in B, la posizione del segmento corrente alla quale si trovano dati **urgenti**
    - Nel caso, a **D** viene inviata notifica immediata di interruzione, con invio dei dati corrispondenti
  - **ACK** : se 1, il campo **ACK #** è valido, altrimenti va ignorato

Rete: livello trasporto (TCP/IP)
Il segmento TCP - 3

- Gli altri *flag* da 1 *bit* ciascuno …
  - **PSH** : se 1, richiede all'entità **`TCP`** di lato **D** di consegnare il segmento a **D** senza introdurre ritardo (= memorizzazione al livello **`TCP`**)
  - **RST** : se 1, richiede *reset* di connessione
  - **SYN** : usato per stabilire una connessione
    - SYN=1 ACK=0 : **M** → **D** : richiesta di connessione
    - SYN=1 ACK=1 : **D** → **M** : richiesta accettata
  - **FIN** : se 1, notifica la richiesta di fine connessione
    - **Rilascio simmetrico**: un processo può continuare a ricevere dati ben oltre il proprio invio di richiesta di fine connessione

Rete: livello trasporto (TCP/IP)
Il segmento TCP - 4

- Nel campo `Window size` (16 *bit*) l'entità TCP di lato **D** indica l'ampiezza della propria finestra di ricezione per la connessione
  - Il valore 0 richiede ad **M** di sospendere ogni nuovo invio, confermando però ricezione fino al B di indice (**ACK#** – 1)
  - **M** potrà riprendere l'invio solo dopo l'emissione da parte di **D** di un segmento con lo stesso **ACK #** precedente, ma indicante ampiezza di finestra > 0
  - La finestra minima deve avere capienza sufficiente per un segmento di massima ampiezza (< 64 kB)

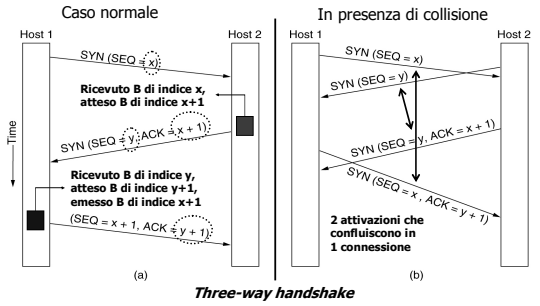
Rete: livello trasporto (TCP/IP)
Il segmento TCP - 5

- Il campo `Checksum` consente a **D** di effettuare controllo di integrità di segmento
  - Complemento ad 1 della somma in complemento ad 1 di tutte le parole a 16 *bit* del segmento
    - Aggiungendo al segmento uno **pseudo-prefisso** che include: (i) gli indirizzi **`IP`** di **M** e **D**, (ii) 1 B a 0, (iii) 1 B identificatore di protocollo (TCP=6), (iv) 2 B che indicano l'ampiezza in B dell'intero segmento (non presente nel prefisso reale!)
      - **Violazione di astrazione**! **`TCP`** conosce gli indirizzi **`IP`**
      - Ampiezza massima di segmento < (2^{16} – 1 – 20) B
    - Rendendo pari l'ampiezza del campo dati con l'aggiunta di 1 B di 0 (se necessario)
    - Considerando 0 il campo `Checksum` ricevuto

Rete: livello trasporto (TCP/IP)
Il segmento TCP - 6

- **D** può usare il campo opzionale per indicare la massima dimensione accettabile dell'**area dati** di segmento
- **M** può usarlo per fare la sua proposta
  - Il valore minore tra i 2 viene selezionato
  - Altrimenti si assume semplicemente il valore 536 B, dunque segmenti di dimensione massima 556 B
- Il campo opzionale può anche servire per fissare una dimensione di finestra di ricezione su 32 *bit* (più ampia di quella standard)
  - Quando la rete permette emissione veloce di unità ampie 64 kB (= 1 segmento di massima dimensione), consente ad **M** di evitare frequenti periodi di attesa forzata

Rete: livello trasporto (TCP/IP)
Attivazione di connessione - 1



Three-way handshake

Rete: livello trasporto (TCP/IP)
Attivazione di connessione - 2

- Caso normale (singola attivazione)
  - **M** → **D** (SEQ = x, SYN = 1, ACK = 0)
  - **D** → **M** (SEQ = y, SYN = 1, ACK# = x+1)
  - **M** → **D** (SEQ = x+1, SYN = 1, ACK# = y + 1)
- 2 attivazioni sulla stessa connessione
  - **M**n1 → **D**n2 (SEQ = x, SYN = 1, ACK = 0)
  - **M**n2 → **D**n1 (SEQ = y, SYN = 1, ACK = 0)
  - **D**n2 → **M**n1 (SEQ = y, SYN = 1, ACK# = x+1)
  - **D**n1 → **M**n2 (SEQ = x, SYN = 1, ACK# = y + 1)
    - Se entrambe le attivazioni hanno successo, esse danno luogo ad 1 sola connessione con indici iniziali (**x**, **y**)

Rete: livello trasporto (TCP/IP)
Politica di trasmissione TCP - 1

- **Controllo di flusso** concerne la disponibilità di memoria in **D**, non l'emissione di conferme di ricezione
  - Con ogni conferma emessa, **D** aggiorna la propria disponibilità residua (***window size***)
  - **M** usa questo dato per disciplinare i propri invii
- Lo stile di comunicazione tra **M** e **D** può dar luogo a situazioni di estrema inefficienza
  - **M** potrebbe inviare 1 B di dati alla volta → 1 segmento da 21 B → 1 **`datagram`** da 41 B
  - **D** potrebbe consegnare al suo processo utente 1 B alla volta → **M** alterna attese ad invii di 1 B

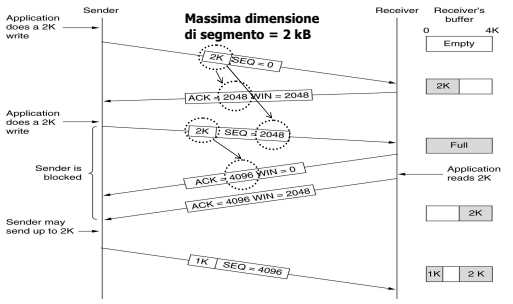
Rete: livello trasporto (TCP/IP)
Politica di trasmissione TCP - 2

- Ove possibile, conviene raggruppare la conferma e l'aggiornamento di disponibilità con la 1ª emissione di dati nella stessa direzione sulla stessa connessione
  - Per ridurre il numero di segmenti inviati senza carico utile
- Quando TCP (**M**) riceve dal suo processo utente 1 B alla volta:
  - Invia il 1° B memorizzando ogni altro B, ritardandone l'invio sino all'arrivo della 1ª conferma
  - A quel punto invia tutti i B memorizzati in 1 singolo segmento, memorizzando ogni altro B fino alla conferma di ricezione del segmento
  - **Algoritmo di Nagle** → inadatto per sessioni grafiche interattive su rete perché corrompe la tempistica degli eventi mostrati all'utente

Rete: livello trasporto (TCP/IP)
Politica di trasmissione TCP - 3

- Quando TCP (**M**) ha molti dati da inviare ma TCP (**D**) può riceverli ed inviarli al suo processo destinatario 1 solo B alla volta:
  - TCP (**D**) ritarda l'emissione verso **M** della propria disponibilità di ricezione fino a quando non ne abbia accumulata una quantità decente
    - Tipicamente almeno metà del proprio *buffer* oppure la dimensione massima di segmento dichiarata in fase di inizializzazione di connessione
  - **Algoritmo di Clark** (*silly window problem*)

Rete: livello trasporto (TCP/IP)
Politica di trasmissione TCP - 4



Rete: livello trasporto (TCP/IP)
Politica di trasmissione TCP - 5

- Vediamo come TCP (**M**) si adatta alla capacità ricettiva di TCP (**D**) con finestra ampia 4 kB
  - **M** → **D** (SEQ = 0, dati = 2 kB)
  - **D** → **M** (ACK# = 2048, WIN = 2048)
  - **M** → **D** (SEQ = 2048, dati = 2 kB)
    - **M** → deve sospendere ogni nuovo invio perché ha per ora esaurito la capacità ricettiva di **D**
  - **D** → **M** (ACK# = 4096, WIN = 0)
  - **D** → **M** (ACK# = 4096, WIN = 2048)
    - **M** → può riprendere gli invii

Rete: livello trasporto (TCP/IP)
Controllo di congestione - 1

- In presenza di reti fortemente inaffidabili è difficile determinare la causa della perdita di segmento segnalata da un *time out* di ricezione
  - **Congestione di rete** → ritardo sulla linea, con o senza distruzione di pacchetto
  - Errori trasmissivi → distruzione di pacchetto
- Non conviene fare assunzioni se non si conosce la topologia del tratto di rete da attraversare
  - Le connessioni che usano linee fisiche hanno buona affidabilità
    - L'arrivo del *time out* è sintomo di congestione
  - Le connessioni *wireless* sono invece molto inaffidabili
    - L'arrivo del *time out* può essere sintomo di interferenza

Rete: livello trasporto (TCP/IP)
Controllo di congestione - 2

- Il controllo di flusso lega l'emissione di segmenti da parte di **M** alla capacità ricettiva di **D**, assumendo una rete con sufficiente capacità trasmissiva
- Il controllo di congestione aiuta ad alleviare la perdita di pacchetti in presenza di reti inaffidabili
  - Può essere necessario indipendentemente dalla capacità di ricezione di **D**

Rete: livello trasporto (TCP/IP)
Controllo di congestione - 3

- **M** mantiene 2 finestre concettuali d'invio
  - **`RW`** = capacità dichiarata da **D**
  - **`CW`** = dimensione di segmento che si ritiene eviti congestione della rete
    - **`Congestion window`**
- **M** invia segmenti di dimensione pari al minimo tra **`RW`** e **`CW`**
- **M** usa un'euristica per approssimare l'ampiezza migliore di **`CW`**

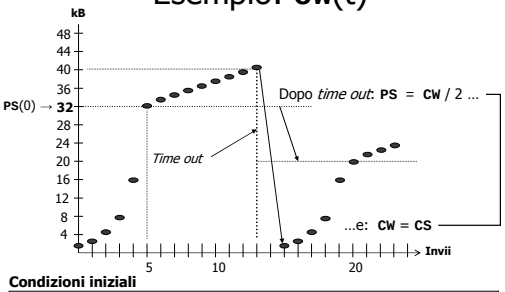
Rete: livello trasporto (TCP/IP)
Controllo di congestione - 4

- **M** inizializza **CW** alla massima dimensione di segmento permesso sulla connessione (**CS**)
- **M** emette una sequenza (*burst*) di **CW**/**CS** segmenti ampi **CS**, verificando per ciascuno di essi l'arrivo della conferma di ricezione prima del *time out*
- Per ogni singola conferma ricevuta, **M** incrementa l'ampiezza di **CW** di 1 **CS** e ripete l'operazione finché non si verifichi un *time out* o **CW** diventi > **RW**
  - In questo modo l'ampiezza di **CW** raddoppia ad ogni conferma completa di un'intera sequenza
- **Algoritmo slow start** → adottato da tutte le realizzazione concrete di **TCP**

Rete: livello trasporto (TCP/IP)
Controllo di congestione - 5

- L'algoritmo di controllo di congestione usa un ulteriore parametro soglia **PS**, inizializzato alla massima dimensione teorica di segmento (= 64 kB)
- Ad ogni *time out* si aggiornano i valori:
  - **PS** (i+1) = **CW** (i) / 2
  - **CW** (i+1) = **CS**
- Fino al successivo *time out* si usa l'algoritmo **slow start** per riportare **CW** ad un valore ottimale:
  - **CW** (i+1) = 2 × **CW** (i) finché **CW** ≤ **PS**
  - **CW** (i+1) = **CW** (i) + **CS** successivamente

Esempio: **cw**(t)



Condizioni iniziali

CS=1 kB **PS**(0)=32 kB **CW**(0) = **CS** **RW** >48 kB

Rete: livello trasporto (TCP/IP)
Intervallo di time out - 1

- **TCP** usa un'euristica per determinare la durata dell'intervallo di *time out* adatta alle prestazioni effettive della rete, che sottopone a monitoraggio continuo
  - La variabile **RTT** (***round-trip time***) denota la miglior stima corrente dell'intervallo tra un invio e la sua conferma di ricezione
  - Un orologio misura l'intervallo **T** per ogni invio
  - Il valore di **RTT** viene così aggiornato:
    - **RTT** = α **RTT** + (1 − α) **T**
      dove α ha valore prevalente (p.es. 7/8)

Rete: livello trasporto (TCP/IP)
Intervallo di time out - 2

- L'idea iniziale era fissare l'intervallo di *time out* come β**RTT** , p.es. β = 2, ma si è poi osservato sperimentalmente che non conviene usare un fattore β costante
- Attualmente l'intervallo di *time out* viene calcolato come **RTT** + 4 × **D**
  - Dove **D** viene costantemente aggiornata al valore **D** = α **D** + (1 − α) | **RTT** − **T** |

Rete: livello trasporto (TCP/IP)
Intervallo di time out - 3

- In presenza di reinvio a seguito di *time out* è difficile dire se una successiva conferma di ricezione concerne il 1º invio o la sua ritrasmissione
  - L'attribuzione erronea di **T** corrompe **RTT**
- **Algoritmo di Karn** prevede di non utilizzare per il calcolo di **RTT** i valori di **T** relativi a segmenti ritrasmessi

Rete: livello trasporto (TCP/IP)

Altre misurazioni

- Oltre ad **RTT**, l'entità **TCP** usa altri 2 valori temporali
  - Un orologio misura il tempo trascorso da quando **D** abbia notificato disponibilità **0** senza più aggiornarla
    - Contro un **limite di persistenza** prefissato
    - Trascorso tale limite, **M** chiede a **D** di aggiornare la sua disponibilità di ricezione
  - Un altro orologio misura il tempo dall'ultimo invio o ricezione sulla connessione
    - Contro un **limite di attività vitale** prefissato
    - Trascorso tale limite, il lato dove questo è avvenuto invia un **segmento di controllo** per verificare la presenza della controparte
    - In assenza di risposta, lo stesso lato **rilascia** la connessione

Livello trasporto (TCP/IP) Architettura degli Elaboratori 2 - T. Vardanega *Pagina 453*

Rete: livello trasporto (TCP/IP)

Modello di servizio UDP

- **UDP** fornisce all'utente semplicemente accesso diretto al servizio `datagram` di **IP**
- Ne segue che la sua entità di trasporto non effettua alcun controllo di correttezza di trasmissione
- L'entità **UDP** è dunque un normale `datagram` prefissato da:
  - 2 parole da 16 *bit* indicanti le porte di **M** e **D**
  - 1 parola da 16 *bit* indicante l'ampiezza del `datagram`
  - 1 parola da 16 *bit* con `checksum` calcolato con la stessa tecnica usata per il segmento **TCP**

Livello trasporto (TCP/IP) Architettura degli Elaboratori 2 - T. Vardanega *Pagina 454*