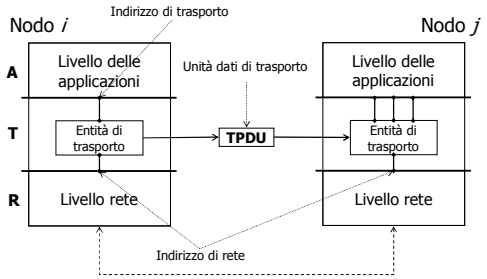
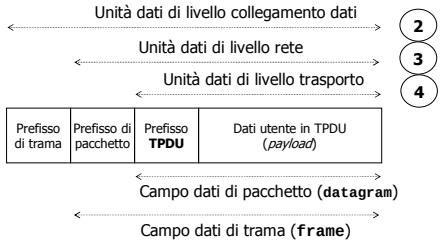
Posizionamento nella gerarchia – 1



Posizionamento nella gerarchia – 2

- Posto tra il livello delle applicazioni **A** (o sessione, 5 OSI, se presente) e il livello rete **R**
  - I suoi servizi sono resi da "entità di trasporto" che nascondono completamente al cliente l'esecuzione dei propri protocolli
  - I clienti di livello **A** in comunicazione tra loro vedono semplicemente un flusso di dati
- Offre ad **A** più punti di accesso a servizi di trasporto
  - Mappandoli su un unico indirizzo di rete nello spazio di **R**
- Contempera l'inaffidabilità del livello rete
  - Qualità del Servizio (QoS) è lo sforzo impegnato dal livello trasporto per raggiungere il livello di affidabilità richiesto dai clienti applicativi

Posizionamento nella gerarchia – 3



Posizionamento nella gerarchia – 4

- Il livello rete è eterogeneo
  - Dunque inevitabilmente inaffidabile rispetto alla corretta gestione di
    - Flusso (tra nodi *host*)
    - Congestione (di *router* quindi all'interno di *subnet*)
    - Errori trasmissivi
- Il livello 2 interconnette due nodi *router* agli estremi di un collegamento punto a punto
- Il livello 4 interconnette due nodi *host* agli estremi di una connessione virtuale realizzata su una rete senza connessioni

Socket – 1

- Basati sul modello UNIX BSD
- **`Socket`** = terminale (***end point***) di comunicazione di livello trasporto
  - **`Socket`** diversi per comunicazioni con o senza connessione
    - TCP o UDP
- Al **`socket`** di destinazione corrisponde un indirizzo locale (**porta**) associato all'indirizzo **`IP`** del nodo **D**
- A livello realizzativo un **`socket`** è una coda di ricezione con uno stato controllato da una componente di S/O
- Il processo utente associato al **`socket`** si blocca in attesa di comunicazioni in ingresso (se **D**) o di conferma di ricezione (**M**, se con connessione)

Socket – 2

- API di UNIX BSD (1/2)
  1. **`socket`**(…) : crea un **`socket`** restituendo al chiamante l'**`ID`** del descrittore del *file* speciale corrispondente
  2. **`bind`**(…) : associa una porta al **`socket`** creando così la sua identità di rete
     - Il processo *server* può così pubblicare la propria identità
     - Il processo cliente usa la propria identità per stabilire il collegamento con il *server*
  3. **`connect`**(…) : blocca il chiamante fino allo stabilirsi della connessione richiesta
     - Il cliente vi indica l'indirizzo del *server*
     - Il **`socket`** locale del cliente viene creato automaticamente
       - Senza invocazione esplicita di **`bind`**

Socket – 3

- API di UNIX BSD (2/2)
  4. **`listen`**(…) : dimensiona la coda associata al **`socket`** di lato *server* e pone il chiamante in attesa di richieste
     - Lo stesso *server* può servire più connessioni simultanee
  5. **`accept`**(…) : blocca il chiamante fino a una richiesta di connessione ricevuta la quale le associa un **`socket`** identico a ma distinto da quello sul quale la richiesta è pervenuta
     - Possibilità di più connessione simultanee
     - Un agente del *server* ascolta sul **`socket`** iniziale mentre varie istanze di *server* (anche diverse tra loro) lavorano su **`socket`** creati dinamicamente
  6. **`send`**(…) / **`recv`**(…) : invia / riceve su un **`socket`**
  7. **`close`**(…) : **M** e **D** rilasciano il proprio **`socket`** e dunque la relativa connessione

Socket – 4

```
1  socket_descr = socket(…, …, …)
   …
2  net_addr = bind(socket_descr, …);
   …
3  conn = connect(socket_descr, …); // client side
   …
4  out = listen(socket_descr, …)    // server side
   …
5  sock = accept(socket_descr, …, …) // server side
   …
   // send (~ write) / receive (~ read)
   …
7  end = close(socket_descr); // (sock) server side
```


Socket – 4

- I **`socket`** BSD hanno un tipo che specifica
  - Il dominio di indirizzamento
    - **`AF_INET`** denota l'indirizzamento **`IP`** → uso di porta
  - Lo stile di comunicazione → l'unità dati
    - **`SOCK_DGRAM`** denota l'uso di **`datagram`**
  - Il tipo di comunicazione → il protocollo
    - **`IPPROTO_TCP`** / **`_UDP`** denota i protocolli a noi noti
- I **`socket`** (**`datagram`**) inviano a **`sendto`** (**D**) e ricevono da **`recvfrom`** (**M**)
- **`send`** e **`recv`** operano su qualunque connessione aperta

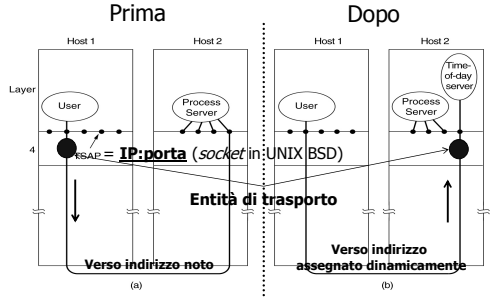
Indirizzamento – 1

- La comunicazione a livello trasporto richiede conoscenza dell'indirizzo locale di **D**
  - Indirizzo espresso come (**`IP`** : porta)
  - Forma di indirizzamento gerarchico
- L'indirizzo di **D** può essere noto a priori
  - Alcune applicazioni ascoltano su una porta standard fissata dunque nota a priori ad ogni utente
- Altrimenti un ***name server*** apposito associa il nome logico dell'applicazione richiesta al suo indirizzo di livello trasporto

Indirizzamento – 2

- **M** e **D** sono associati dalle rispettive entità di trasporto a uno specifico punto di accesso locale
  - Unico nel nodo
- Alcuni processi **D** (***server***) si pongono in attesa permanente dietro il loro specifico punto di accesso
- Altri processi usano invece un agente (***proxy***) che ascolta il loro vece e assegna loro dinamicamente un punto di accesso all'arrivo di una richiesta di connessione

Indirizzamento – 3



Connessione – 1

- La rete ha memoria poiché non sa cancellare automaticamente tutti i duplicati di pacchetti consegnati con successo né le loro conferme
  - Pacchetti duplicati arrivati tardi a destinazione possono esser trattati come nuova conversazione
  - Ogni connessione dovrebbe invece avere inizio e fine univoci così da invalidare flussi duplicati
- Problema reso ancor più complesso dal fatto che i nodi possono perdere traccia del loro stato di comunicazione

Connessione – 2

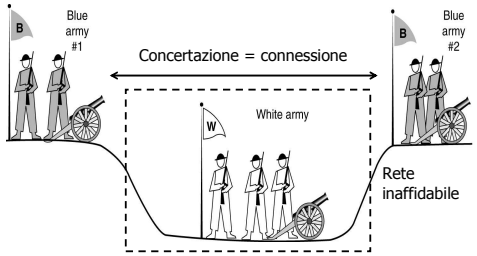
- I pacchetti in viaggio sulla rete accumulano ritardi variabili
  - Pacchetti dati, conferme, pacchetti di controllo
- Servono tecniche che offrano garanzie di inizializzazione consistente a **M** e **D**
- ***Three-way handshake***
  - **M** → **D** (`rc,x`) : richiesta di connessione con emissione di TPDU numerati a partire da x
  - **D** → **M** (`cc,y,x`) : conferma di connessione (`rc,x`) con emissione di TPDU numerati a partire da y
  - **M** → **D** (`dati,y`) : invio del primo TPDU dati numerato x con conferma dell'indice y di **D**

Rilascio

- La terminazione di una connessione può avvenire in modo **a**simmetrico o simmetrico
  - Asimmetrico: come nella conversazione telefonica
    - Il rilascio di una parte distrugge la connessione
      - Può comportare **perdita di dati** → non desiderabile
  - Simmetrico: il rilascio deve avvenire da ambo i lati della connessione (anche in modo asincrono)
    - Il **`socket`** richiede rilascio **simmetrico**
    - Può lasciare la connessione **aperta** all'infinito
      - **Problema dei 2 eserciti**
    - I lati delle connessioni che non portino dati validi entro intervalli fissati sono autonomamente rilasciati dal rispettivo possessore (approccio collaborativo)

Problema senza soluzione

Attacco coordinato = rilascio simmetrico



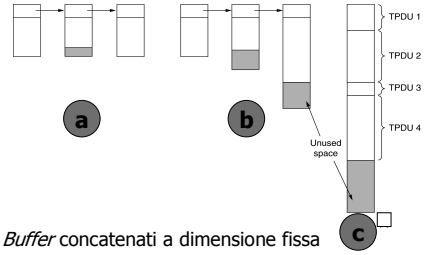
Controllo di flusso – 1

- Problematica analoga a quella incontrata al livello 2 (collegamento dati)
  - Protocolli come **SWP** e **SRP** richiedono ai 2 lati della connessione di trattenere trame in memoria
    - I nodi *router* interconnettono relativamente poche linee
      - Ciò richiede *buffer* di ampiezza non eccessiva
    - Nodi **M** e **D** possono ospitare numerosissimi flussi di conversazione
      - Ciò richiede molta più memoria
- Se la rete è inaffidabile (caso **`Internet`**) **M** e **D** devono salvare i pacchetti inviati
  - Secondo le regole del protocollo impiegato

Controllo di flusso – 2

- Allocazione **statica** di memoria pacchetti
  - Un insieme di *buffer* di dimensione fissa
    - Ciascun *buffer* assegnato a 1 TPDU ?
      - Enorme spreco di memoria causato dall'esigenza che i *buffer* siano dimensionali alla massima ampiezza di TPDU (~ frammentazione interna)
- Allocazione **dinamica**
  - 1 solo *buffer* circolare ∀ connessione
    - Spreca memoria se la connessione ha poco traffico
    - Ha buona resa altrimenti

Controllo di flusso – 3



a. *Buffer* concatenati a dimensione fissa
b. *Buffer* concatenati a dimensione variabile
c. Singolo *buffer* circolare per singola connessione

Controllo di flusso – 4

- **Allocazione dinamica**
  - **SWP** e **SRP** legano la gestione dei propri *buffer* all'arrivo di conferme di ricezione
    - Ciò non conviene in ambiente di rete
  - Meglio che **M** richieda a **D** un'ampiezza di *buffer* dimensionata alle sue aspettative di traffico e **D** indichi la propria disponibilità corrente all'atto di attivazione della connessione
    - Lo spazio reso disponibile da **D** si riduce progressivamente a ogni TPDU emesso da **M**
    - In questo modo **M** può adattare il suo flusso in uscita alle capacità ricettive di **D**

Controllo di flusso – 5

- **Allocazione dinamica**
  - Un meccanismo di controllo che si basi solo sulla capacità di ricezione di **D** assume (sbagliando!) che la rete abbia capacità trasmissiva infinita
  - Per **M** non è conveniente emettere troppi pacchetti senza conferma
    - Capacità di assorbimento della rete → ***n*** pacchetti / *sec.*
    - Tempo medio tra emissione di TPDU e arrivo di conferma → ***p*** *sec.*
    - Con 1 *buffer* ampio ***n*** × ***p*** pacchetti **M** può operare a massima velocità (rispetto alla rete) avendo però il *buffer* sempre mediamente pieno