Dispense complete del corso di

Reti

Professori **Elena Pagani** e **Gian Paolo Rossi**
Dipartimento di Scienze dell'Informazione – Università degli Studi di Milano

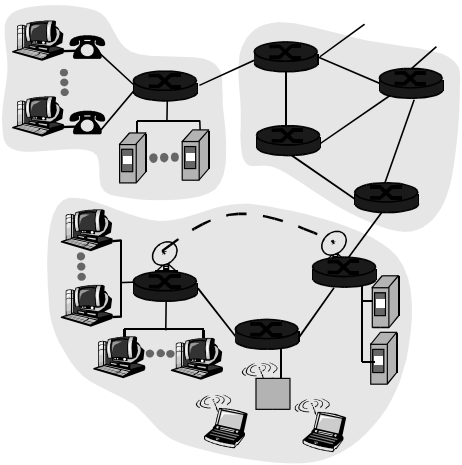
Pagina originale: http://homes.dsi.unimi.it/~pagae/
A cura di **dsy.it** – **staff@dsy.it**

Organizzazione del corso

- Contenuti: comprensione di architetture e protocolli di rete
  - Esercitazioni: Java socket, esercizi
- Testi di riferimento:
  - A. Tanenbaum, "Computer Networks – 3rd Edition", Prentice−Hall, 1996
  - J. Kurose, K. Ross, "Computer Networking: a Top−Down Approach", Addison−Wesley, 2001
- web: http://homes.dsi.unimi.it/~pagae/Reti
- modalità di esame:
  - scritto + orale sugli argomenti del corso
  - 2 verifiche in corso d'anno per chi frequenta

Reti di calcolatori

- collezione interconnessa di calcolatori autonomi
  - eterogeneità hw, sistemi operativi…
  - scambio informazioni attraverso media (doppino, fibra, rete telefonia cellulare, satellite…)
  - esistenza macchine remote esplicita
- sistema distribuito: esistenza molteplici calcolatori è trasparente



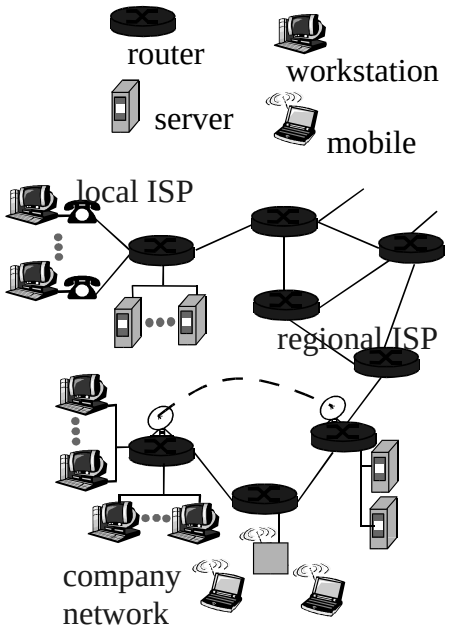
Reti: usi e applicazioni

- comunicazione: e−mail, Voice over IP, lavoro cooperativo, videoconferenza
- condivisione risorse: modello client−server
  - condivisione informazioni: web
- alta affidabilità: replicazione
- alta potenza di calcolo a basso costo: computazione distribuita (grid)
- problema: sicurezza (privacy, controllo accessi, integrità dati)

Reti: concetti fondamentali

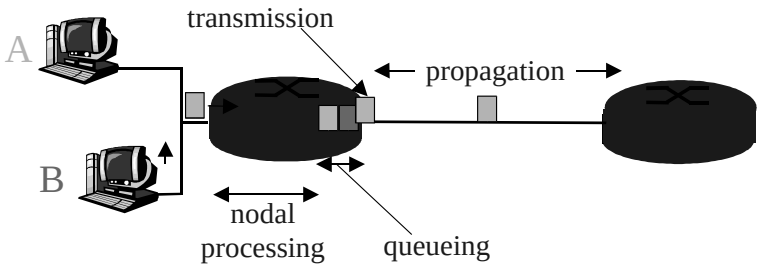
- millions of connected computing devices: *hosts, end−systems*
  - pc's workstations, servers
  - PDA's phones, toasters

  running *network apps*
- *communication links*
  - fiber, copper, radio, satellite
- *routers:* forward data through network
  - data addressing



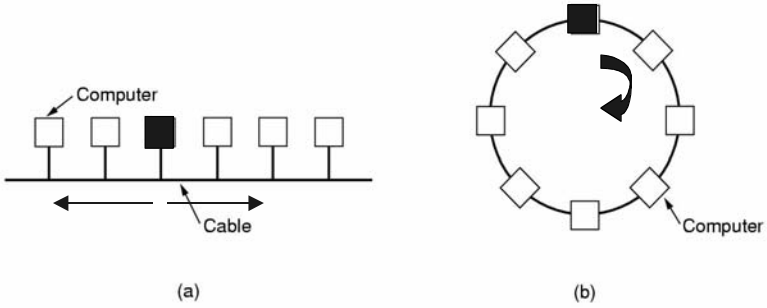
Parametri caratterizzanti

- link bandwidth: quantità dati che può essere trasmessa per unità di tempo
  - LAN: 10−100 Mbps; wireless 1−2 Mbps
- transmission delay
- propagation delay (~ 2/3 velocità luce)
- processing delay (e.g. come instradare?)
- queuing delay



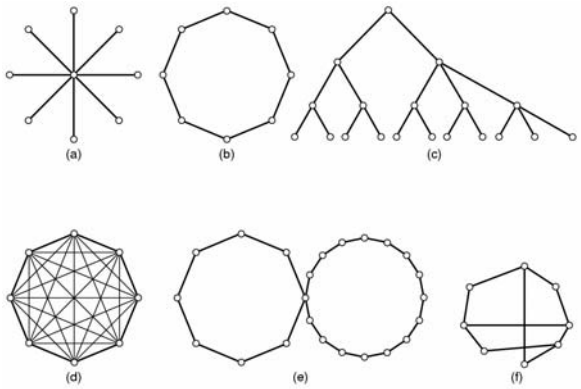
Reti broadcast

- tutti ricevono, solo destinatario processa
- broadcast / multicast addressing (definizione gruppi)
- meccanismo arbitraggio per risolvere contesa
  - un solo host alla volta deve “parlare”, altrimenti collisioni

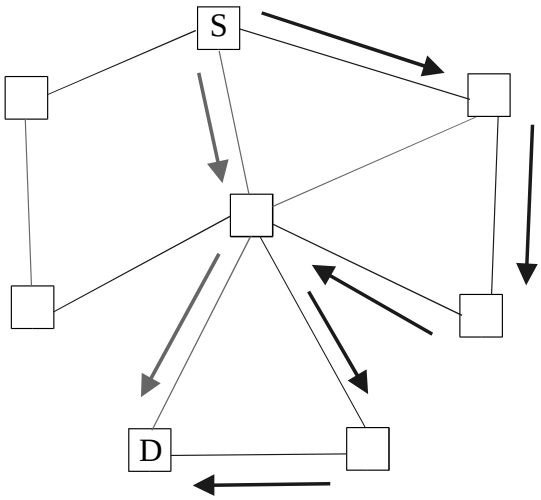


Reti point−to−point

- routing: ricerca path ottimale (e.g. in #hop)
- in generale: reti “piccole” sono broadcast, reti “grandi” sono p2p



Routing



linee da 56 Kbps

linee da 100 Mbps

trade−off overhead canali e computazione

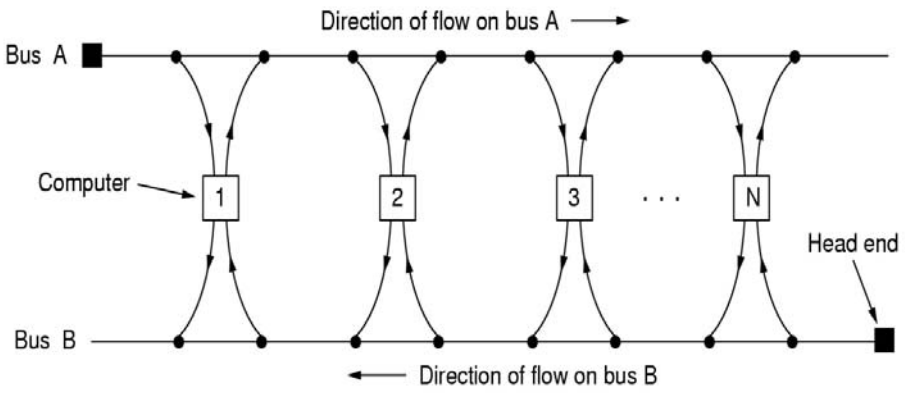
- *perché reti piccole bcast e grandi p2p?*

Tassonomia reti

tipo	taglia	caratteristiche
LAN	10 m. (room) – 1 Km (campus)	unico amministratore rete broadcast noto e limitato max tempo trasmissione
MAN	~10 Km (città)	rete broadcast (no router)
WAN	10−100 Km (nazione)	rete p2p
Internet	pianeta	interconnessione di reti eterogenee

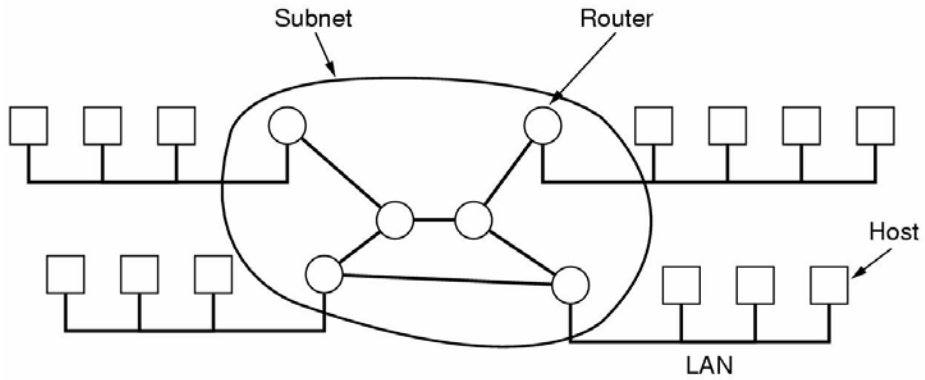
Esempio di MAN

- DQDB (Distributed Queue Dual Bus)



Esempio di WAN

- p2p, anche via satellite o radio terrestre
- separazione comunicazione (subnet) e applicativi (host)

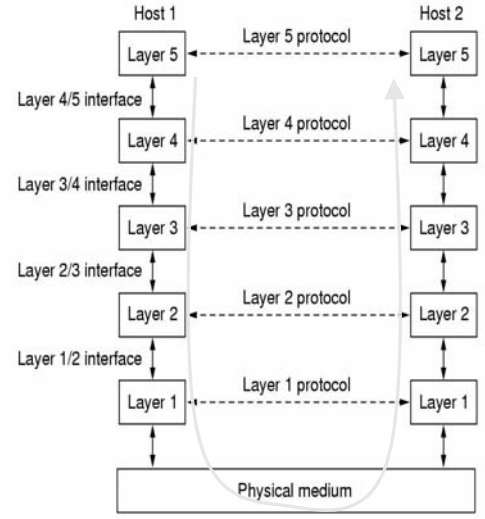


Reti wireless

- wireless: media=onde radio, infrarosso
- mobile: punto connessione a Internet variabile nel tempo
- nomadic = wireless network + mobile user
- single−hop vs. multi−hop (W−LAN)

Wireless	Mobile	Application
No	No	stationary workstation in office
No	Yes	using a portable in a hotel
Yes	No	W−LAN in unwired building
Yes	Yes	portable office

servizi+protocolli+algoritmi=architettura

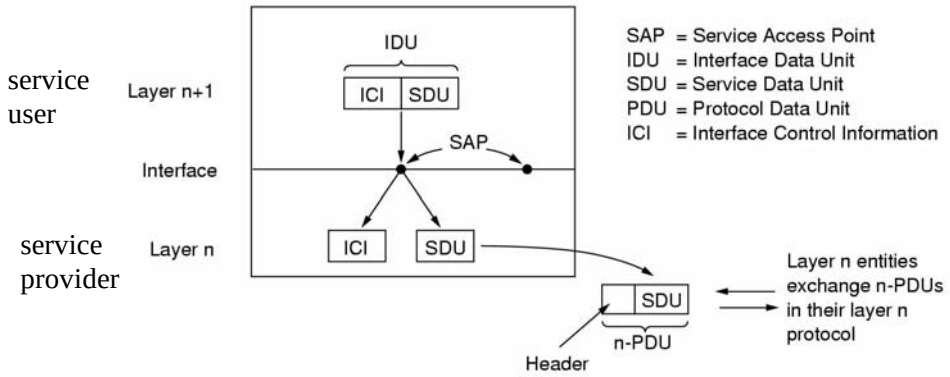


stack di protocolli

- servizi offerti dal livello *n* al livello *n+1*
- protocollo di livello *n*: insieme di regole che definiscono formato e ordine dei msg mandati e ricevuti tra peer, e le azioni intraprese a fronte di trasmissione e ricezione msg
- algoritmi implementano servizi e protocolli in modo *trasparente*

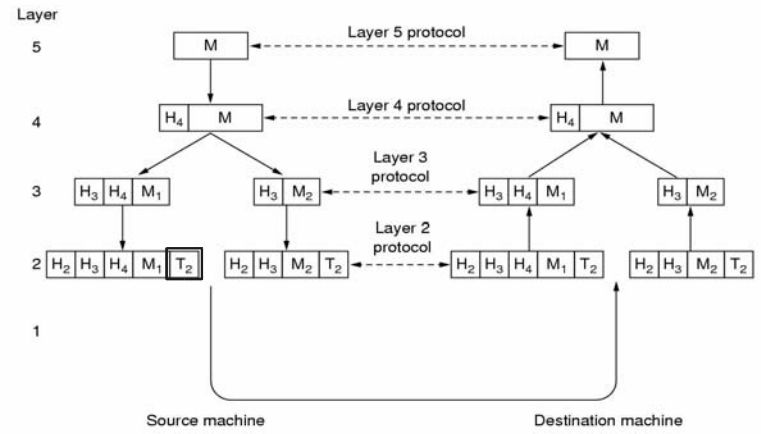
Interfacce tra livelli

- servizio: cosa il livello fa; interfaccia: come accedere ai servizi
- scambio info tra livelli → accordo su interfaccia
  - a SAP primitive servizio disponibili
  - IDU tra livelli adiacenti; PDU tra peer



Architetture di rete

- header: info controllo per implementazione servizio a peer
- comunicazione verticale; implementazione come se fosse orizzontale



Funzionalità dei livelli

- convenzioni naming e addressing
- regole trasferimento dati
  - simplex, half−duplex, full−duplex
- controllo degli errori, reliability
- flow control, congestion control
- frammentazione, ordinamento dati
- (de)multiplexing
- routing
- creazione connessioni

Connection−oriented service

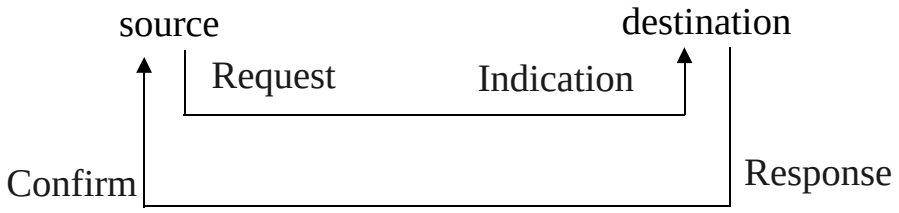
- simile a servizio telefonico: 3 fasi
  - apertura connessione (handshaking)
  - scambio dati
  - chiusura connessione
- *handshaking:* setup (preparazione per) data transfer
- dati ricevuti in ordine
  - orientato a msg vs. orientato a byte
  - può garantire anche affidabilità
- alti tempi, non sempre desiderabile
  - trasferimento pochi dati, multimedia

Connectionless service

- nessuna garanzia ordine → servizio postale
- ogni msg instradato indipendentemente
- più rapido, minore overhead
- anche affidabile (acknowledged); request−reply

	Service	Example
Connection-oriented	Reliable message stream	Sequence of pages
	Reliable byte stream	Remote login
	Unreliable connection	Digitized voice
Connection-less	Unreliable datagram	Electronic junk mail
	Acknowledged datagram	Registered mail
	Request-reply	Database query

Primitive di servizio



- per accedere al servizio o notificare evento
- possono avere parametri (e.g. handshaking)
- *NB: protocolli e servizi sono indipendenti*
  - protocolli contribuiscono a implementaz. servizi
- ES: `CONNECT` confirmed, `DATA` unconfirmed

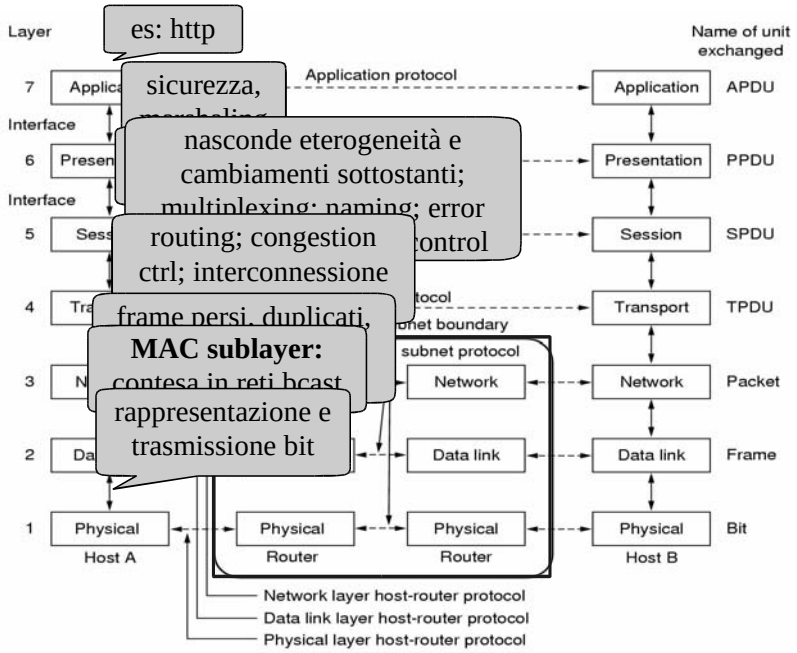
Who's who

- specifica standard di protocolli
  - *perché non di servizi?*
- ISO (International Standard Organization)
- ITU−T (International Telecommunication Union – Telecommunications)
- IETF (Internet Engineering Task Force)
  - Internet Draft, RFC (Request for Comments)
- IEEE (Institute of Electrical and Electronics Engineers)
  - DQDB = IEEE 802.6 standard
  - Ethernet = IEEE 802.3 standard
  - Wi−Fi = IEEE 802.11 standard

Architetture standard

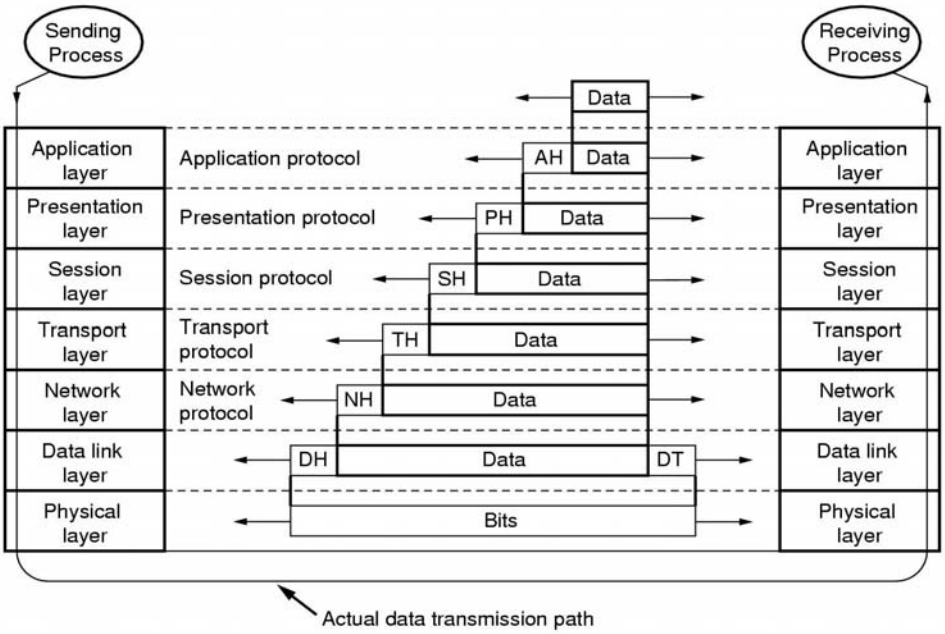
- un livello per ogni differente grado di astrazione
- per ogni livello una funzione ben definita
- interfacce tra livelli t.c. è minimizzata la quantità di dati da scambiare
- #livelli
  - grande abbastanza da separare funzioni distinte
  - piccolo abbastanza perché modello resti maneggevole

Modello ISO/OSI

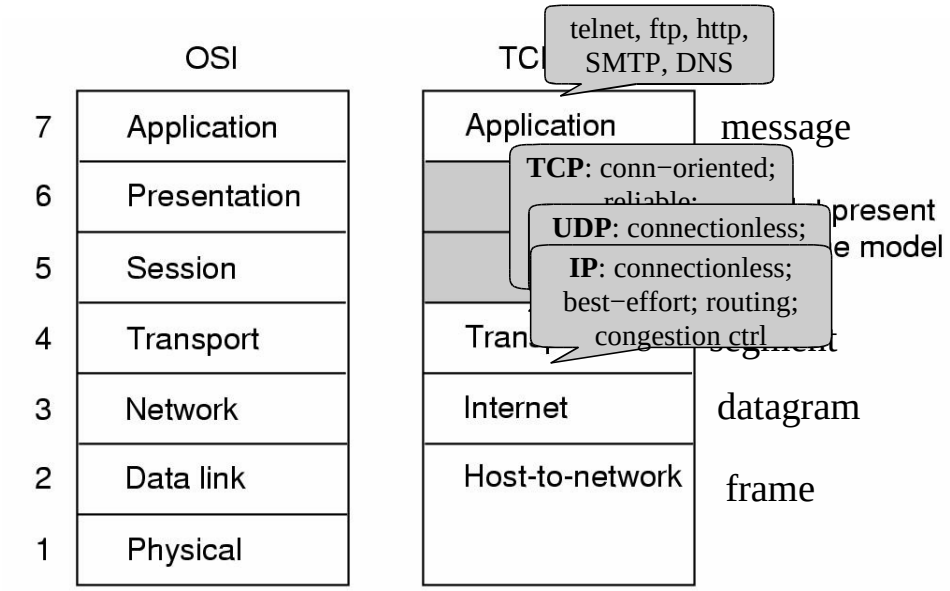


NB: esistono duplicazioni di servizi

Data transmission

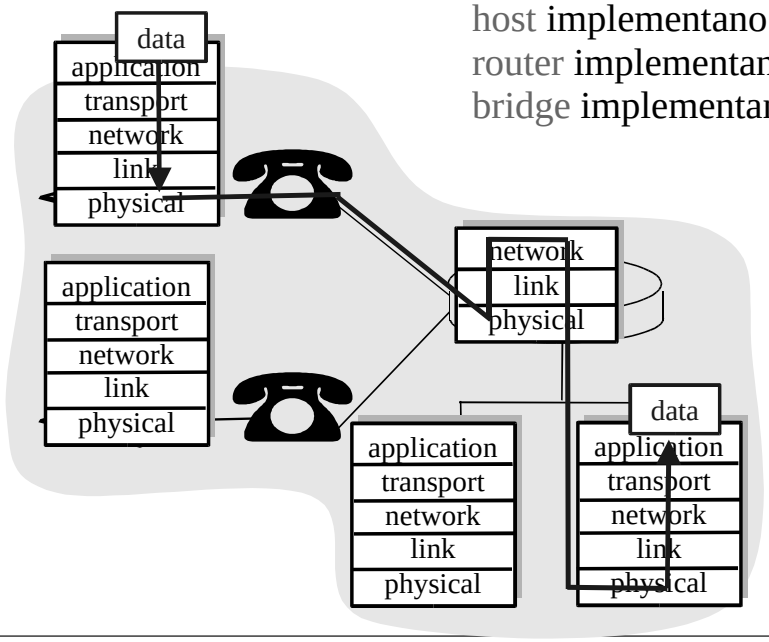


Modello TCP/IP (Internet)



Apparati di rete

host implementano tutti i livelli
router implementano liv. 1−3
bridge implementano liv. 1−2



OSI vs. TCP/IP

- OSI prima di implementazione, TCP/IP dopo
- OSI più generale, maggiore indipendenza da protocolli
  - alcuni livelli sovraccarichi, altri vuoti
  - funzioni duplicate (e.g. error ctrl)
- Netw.L. OSI connectionless o conn−oriented
  - TCP/IP: solo connectionless
- Transp.L. OSI solo connection−oriented
  - TCP/IP lascia scelta a applicazione
- OSI non è mai decollato
  - TCP/IP difficile da adattare a nuove tecnologie

Reti: cenni storici

- 1967: DoD ARPAnet
- 1970: ALOHAnet
- 1975: TCP/IP model per internetwork
- 1976: Ethernet
- 1983: TCP/UDP/IP standard per ARPAnet
- 1987: la rete cresce → DNS
- 1989−1991: web

guiding principles:

- autonomia reti
- servizio best effort
- router stateless
- controllo distribuito

Mezzi trasmissivi (1)

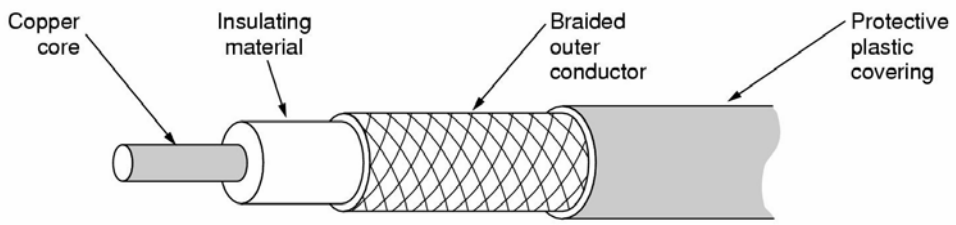
da sist. telefonico: Unshielded Twisted Pair

- per trasmissioni analogiche e digitali
- portata fino a 5Km senza ripetitori
- bwth dipendente da spessore
- Category 3: traditional phone wires, 10 Mbps Ethernet
- Category 5: 100Mbps Ethernet

Mezzi trasmissivi (2)

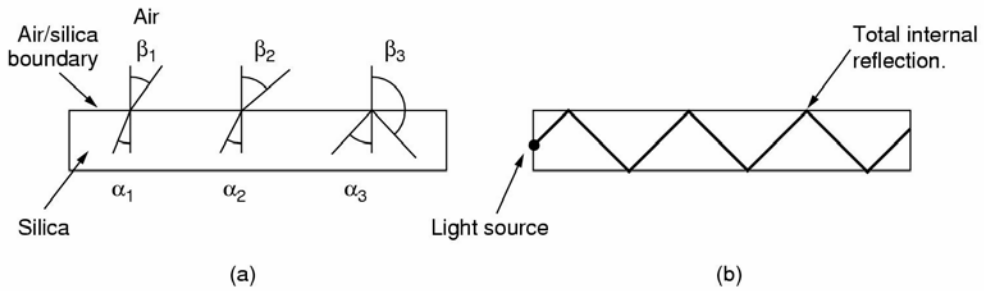
Cavo coassiale

- per trasmissioni digitali
- portata fino a 5Km senza ripetitori
- bwth 1−2 Gbps su 1 Km
  - per distanze maggiori, banda minore



Fibre ottiche (1)

- in funzione diametro fibra:
  (a) multi−mode; *(b)* single−mode (bwth Gbps x 30Km)
- immune da interferenze elettromagnetiche; bassa attenuazione
- usato spettro infrared (λ=0.85−1.55 μ)



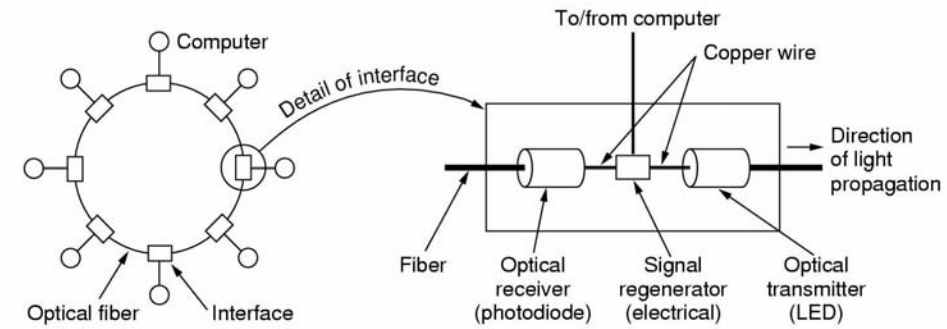
Fibre ottiche (2)

- fibre connesse a emettitore (LED o laser) e rilevatore
- connessione tra fibre difficile (> sicurezza)
  - connettori (10%−20% attenuazione); giunzione meccanica (10%); fusione

caratteristica	LED	laser
data rate	basso	alto
modo	multi	multi e single
distanza	breve	lunga
lifetime	lungo	breve
sensibilità temperatura	piccola	sostanziale
costo	basso	alto

Reti a fibre ottiche

- active repeater: rottura interrompe il ring
  - no limitazione taglia ring per attenuazione



- passive star: broadcast dell'insieme degli input
- comunicaz. full−duplex: 2 fibre o 2 bande di frequenza su una fibra

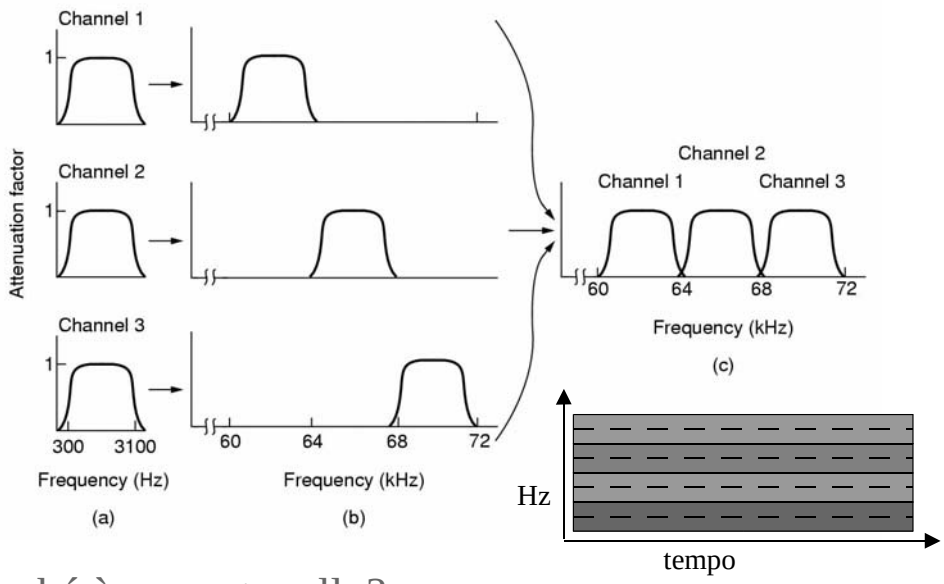
Accesso alla rete

- modem: trasforma segnale digitale in analogico; bwth~56 Kbps
- ISDN (integrated services digital network)
  - integrazione voce e dati, solo digitale
  - bwth~144 Kbps
- ADSL (asymmetric digital subscriber line)
  - 3 bande frequenza /* FDM */
    - telefono
    - downstream: 8 Mbps
    - upstream: < 1 Mbps

Protocolli di Physical L.

- multiplexing di molte conversazioni su un canale
- Frequency Division Multiplexing
  - e.g. radio → allocazione range di frequenze
  - realizzabile con apparecchi analogici (ADSL)
  - interferenza da sovrapposizione → guard bands
  - telefonia: 4 KHz per ogni canale,3 KHz per segnale (< 30 Kbps) + 2 guardie da 500 Hz
  - standard: 1 gruppo=12 canali da 4 KHz; 1 supergruppo=5 gruppi

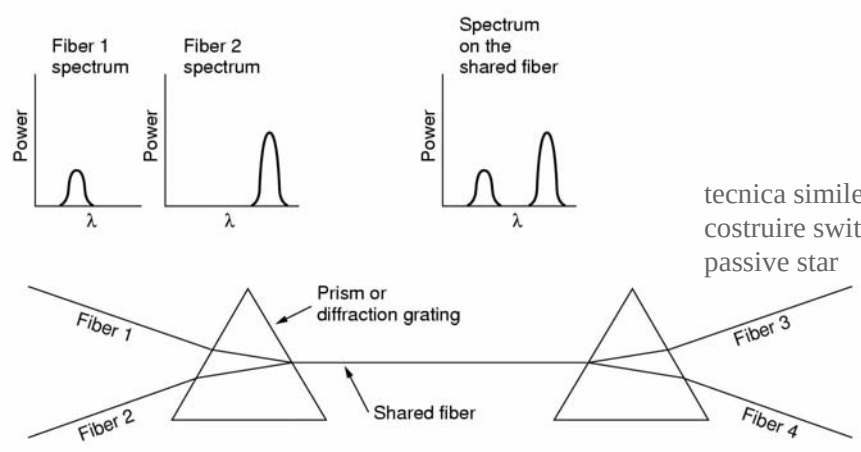
FDM



perché è un protocollo?

Wavelength Division Multiplexing

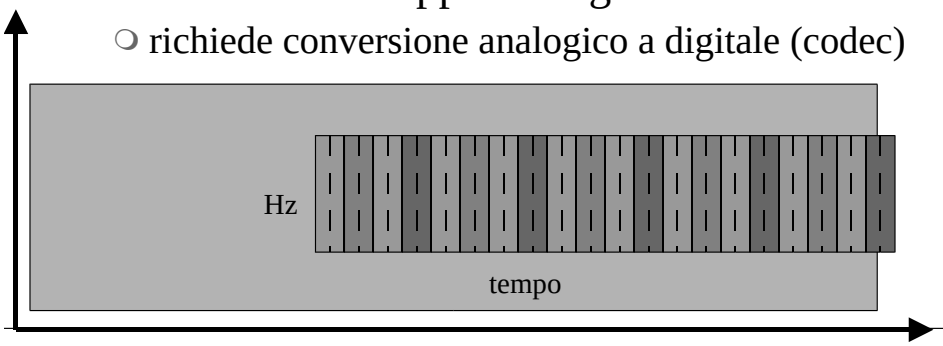
- FDM per fibre ottiche
- velocità conversione ottico ↔ elettrico limita frequenza (bwth) segnale; data bwth fibra, molti segnali multiplexed



tecnica simile per costruire switch con passive star

Time Division Multiplexing

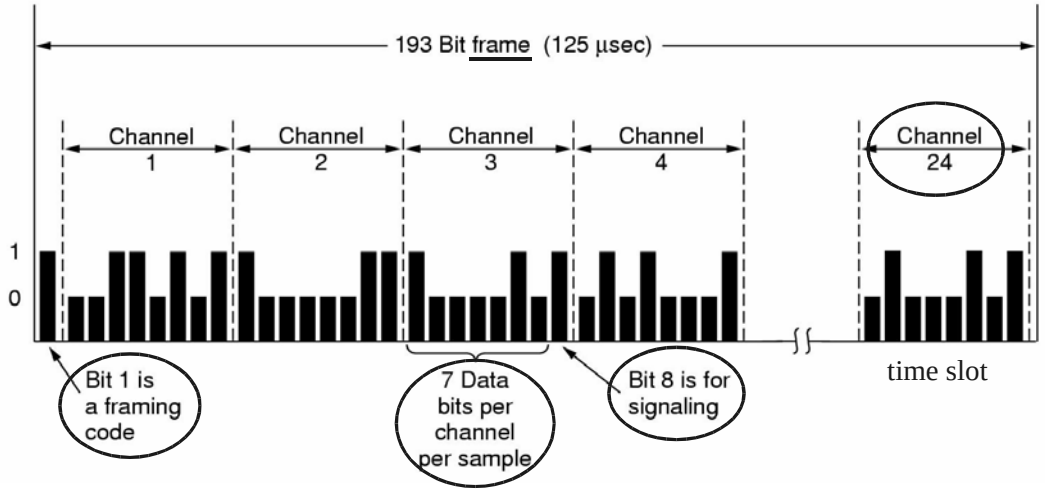
- 1 time slot ̃tempo necessario per inviare 1 (dll) frame
- 1 time frame = N tme slot t.c. N numero entità che condividono il canale
- realizzabile con apparati digitali
  - richiede conversione analogico a digitale (codec)



Digitalizzazione segnale

- Pulse Code Modulation (PCM)
  - per ogni canale da 4 KHz, 8000 campioni/sec. (125 µsec)
  - per ogni campione, valore da 7 (8) bit
- Differential PCM per ridurre ridondanza
  - valore = differenza tra 2 campioni consecutivi
- Delta modulation
  - valore = 1 bit indicante se segnale è maggiore o minore del campione precedente
- Predictive encoding
  - algo predizione valore successivo (x sender e receiver)
  - valore = differenza tra segnale attuale e predetto
- in b), c), d) prbl se segnale cambia velocemente

TDM carrier: T1

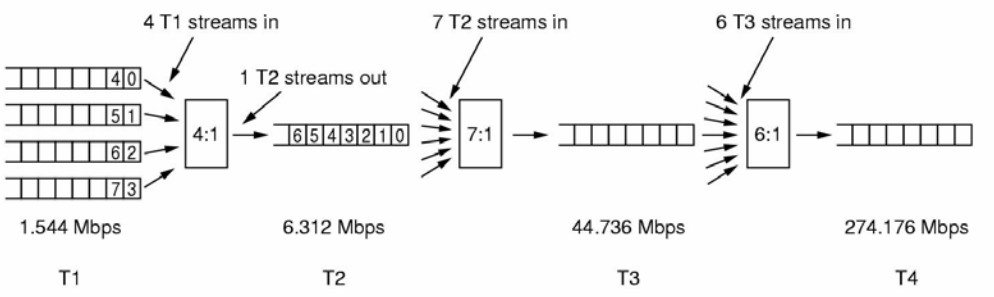


- per ogni canale, 8000 sample/sec, 7 bit
- 193 x 8000 = 1.544 Mbps; (7+1)x24x8000 = 1.344 Mbps dati
- bit 1 per synchro pattern: 1010101…

TDM carrier

- common channel signaling
  - 8 bit dati → 1.536 Mbps dati; 193º bit synchro
- channel associated signaling:
  - come precedente, ma ogni 6 frame dati di 7 bit + 1 bit ctrl
- E1 (usato in Europa)
  - 30 slot da 8 bit dati → 240
  - 2 slot da 8 bit ctrl → 16
  - (240+16)x8000 = 2.048 Mbps
- NB: standard differiscono nei vari paesi…

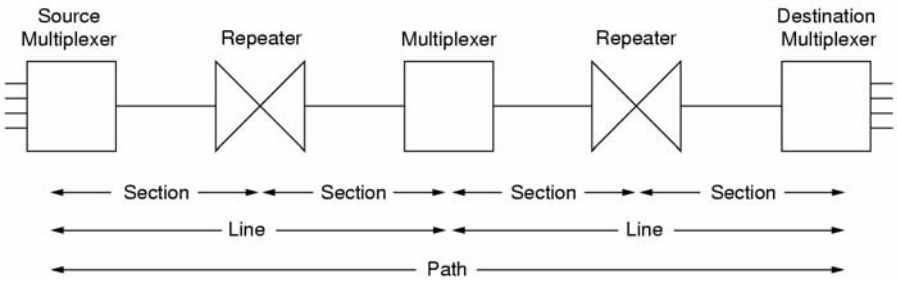
TDM carrier: multiplexing



- T2 = T1 x 4 = 1.544 x 4 = 6.176 + bit ctrl
- T3 = T2 x 6 = 6.312 x 6 = 37.872
- analoghe definizioni per multiplexing E1 in diversi carrier Ex

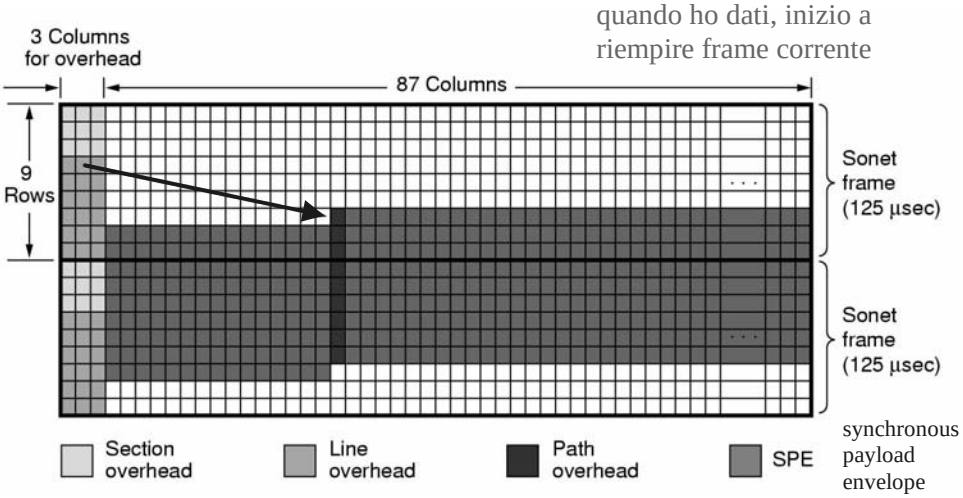
SONET/SDH

- standard per TDM su fibra ottica
- Synchronous Optical NETwork/Synchronous Digital Hierarchy
- Obiettivi:
  - interoperabilità tra carrier e tra sistemi PCM
  - uso canali alta banda
- dati inviati a intervalli fissati; ctrl di master clock



Framing in SONET

quando ho dati, inizio a riempire frame corrente

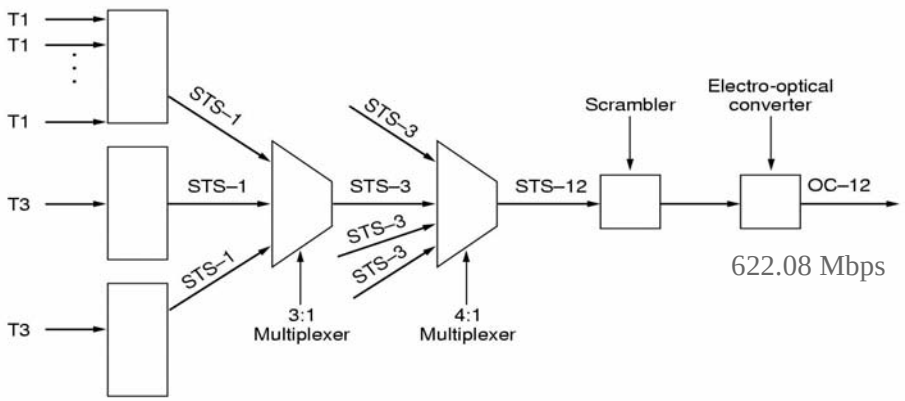


(87+3)x9=810 byte; (810x8)x8000=51.84 Mbps canale base OC−1

((87x9)x8)x8000 = 50.112 Mbps dati; 1.728 Mbps ctrl (synchro)

- in frame overhead ptr a inizio dati

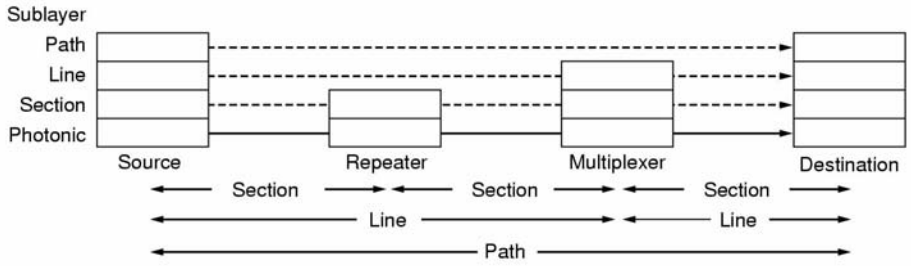
Framing in SONET



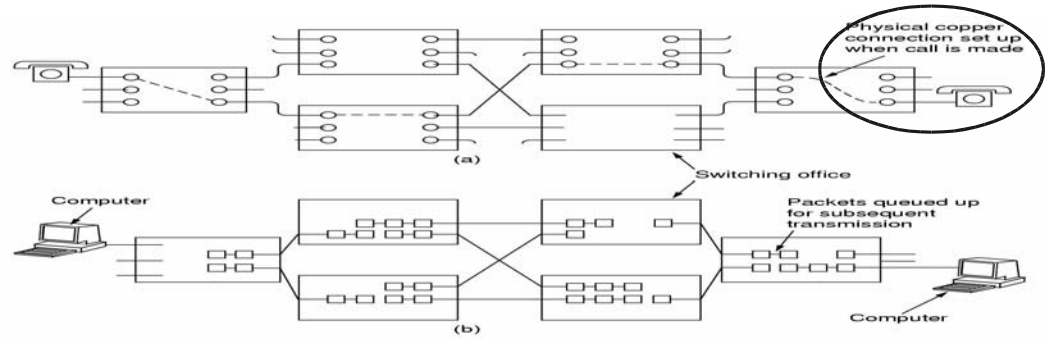
- frame size in funzione bit rate canale
  - OC−3: 155.52 Mbps/8000 = 2430B per frame
  - facilita multiplexing e synchro
- in frame scritto 1 byte per ogni tributario a turno
- NB: rate differenti per USA, Giappone, Europa

Architettura SONET

- photonic: caratteristiche fibra e radiazione usata
- section: invio/ricezione frame su 1 hop; eventualmente rigenerazione
- line: (de)multiplexing
- path: problematiche end−to−end, interfacce con livello data−link



Tecniche di commutazione

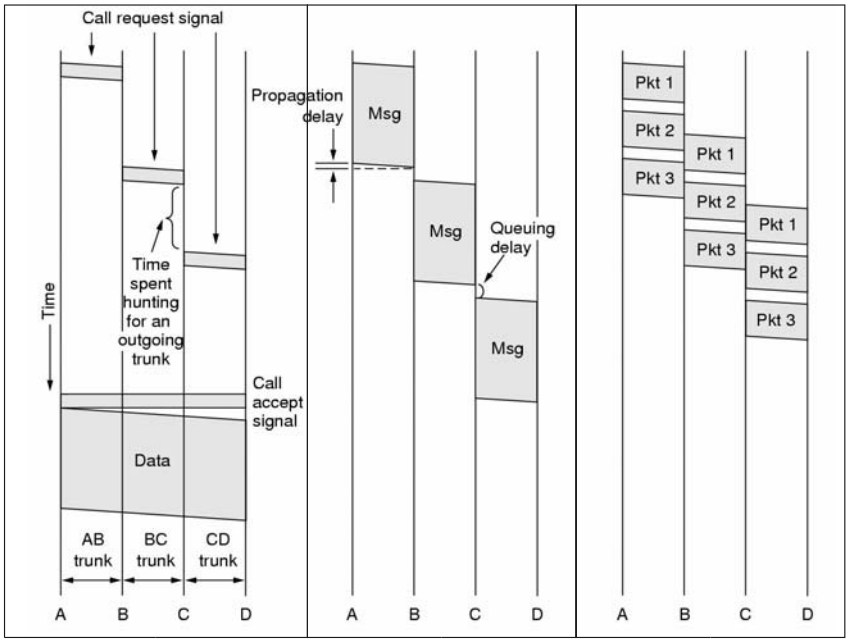


- reti p2p: più interfacce rete per un router
- path = determinata sequenza interfacce
- commutazione di circuito: telefonia
  - path fisico costruito a set−up fino a fine chiamata
  - tutti i dati seguono il medesimo path riservato

Msg switching, pkt switching

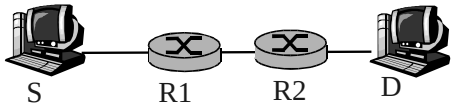
- circuit switching inefficiente wrt. uso bwth
- msg switching, *packet* switching
  - store−and−forward
    - s−a−f delay = pkt size / link bwth
  - ordinamento non garantito
  - statistical multiplexing
  - migliore condivisione bwth wrt. circuit switching ma rischio *congestione*
  - maggiore o/h ctrl per header pkt
  - migliore tolleranza ai fallimenti

Confronto tecniche switching



Confronto tecniche switching

- msg=7.5 Mb; link bwth=1.5 Mbps

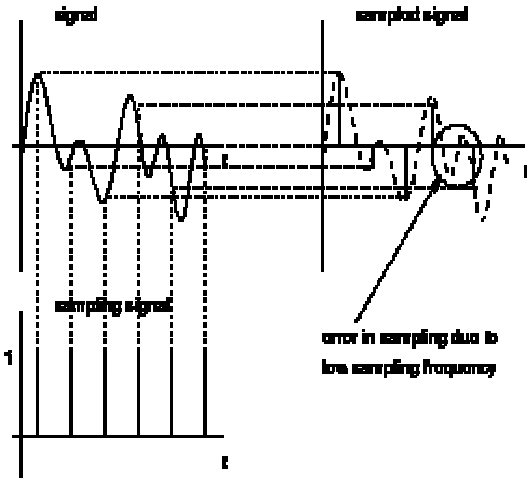


  - transm. del. su 1 link: 7.5 Mb / 1.5 Mbps = 5 sec.
  - ricezione a destinazione dopo 15 sec.
- msg frammentato in 5000 pkt da 1.5 Kb
  - transm. del. su 1 link: 1.5 Kb / 1.5 Mbps = 1 msec.
  - a t_0 trasmette 1º pkt, che raggiunge R1 dopo 1 msec; a t=1 msec trasmette 2º pkt. Situazione a t=2 msec.?
  - ultimo pkt trasmesso a 4.99 sec.
    - raggiunge dest. dopo 4.99+(0.001x3)=5.002 sec.

Rappresentazione dell'informazione

- segnale analogico: variazione continua
- segnale digitale: variazione discreta
  - alta fedelta', indipendenza dal tempo, indipendenza dalla sorgente
- trasmissione dati: trasformazione da digitale a analogico

- campionamento e quantizzazione

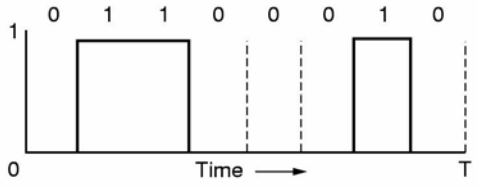


Banda segnale

- bandwidth = range di frequenze occupate dal segnale (Hz)
- bit rate = velocita' di trasmissione dell'informazione (bps)
- Th. Nyquist−Shannon
  - *un segnale analogico di banda B puo' essere completamente ricreato dalla forma digitalizzata se e' stato campionato con un rate s >= 2B*
- es: voce (bwth = 4 KHz) quantizzata con valori di 8 bit
  - s >= 2B = 2 x 4KHz = 8000 campioni
  - bit rate deve essere 8 bit x 8000 = 64000 bps

Codifica delle informazioni

- variazione di voltaggio per rappresentare i bit
  - dispositivi non generano segnale preciso
  - resistenza mezzo attenua/distorce segnale
- baud rate = # cambiamenti del segnale che lo hw puo' generare per secondo
  - limitazione sul bit rate
    - se 2 livelli di segnale: bit rate = baud rate
    - se k livelli di segnale: bit rate = 2^k baud rate



Limitazioni sul bit rate

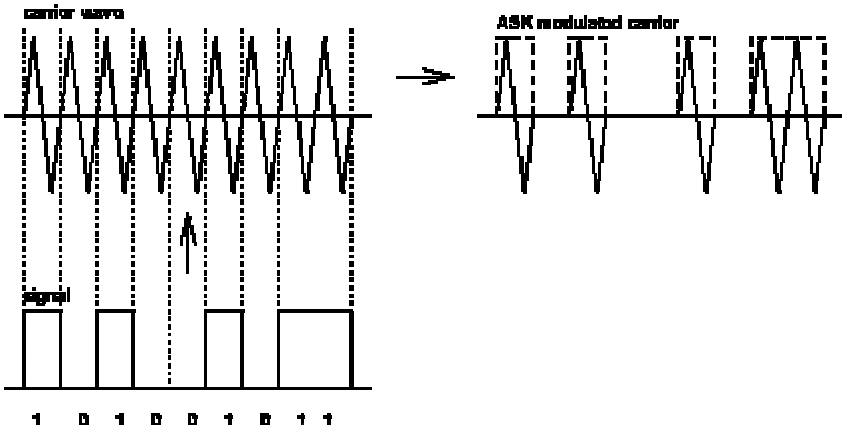
- Legge di Nyquist
  - data banda B del segnale avente V livelli discreti
  - max bit rate = 2 B $\log_2$ V bit/sec
  - es: canale di 3 KHz che trasmette segnale binario ha max bit rate 2 x 3000 x $\log_2$ 2 = 6000 bps
- valore teorico max. In realta': noise disturba...
- Legge di Shannon
  - SNR: signal−to−noise ratio = 10 $\log_{10}$ (S/N) dB
  - max bit rate = B $\log_2$ (1 + S/N)
  - es: canale di 3 KHz con SNR di 30 dB ha S/N=1000 e max bit rate 3000 x $\log_2$ (1 + 1000) <=30000

Comunicazione su lunga distanza

- NB: banda 0 porta informazione nulla!
- come si fa a raggiungere bit rate piu' alti?
  - attenuazione distrugge segnale...
- scoperta: i segnali "oscillanti" riescono a propagarsi piu' lontano
  - uso di onde sinusoidali = carrier
  - il carrier e' continuamente trasmesso sul carrier
  - il trasmettitore modifica il carrier per inviare dati: modulazione
  - il ricevente conosce carrier, e modalita' di modifica del trasmettitore: ricostruisce dati

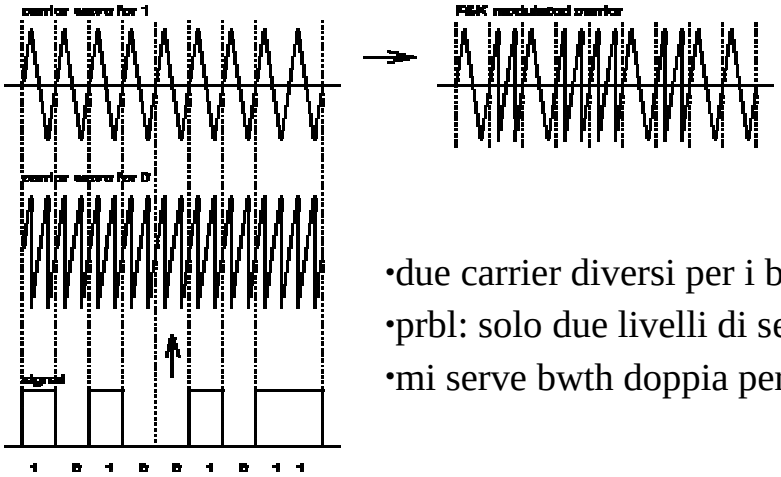
Modulazione di ampiezza (AM)

- carrier costante; src varia l'ampiezza a seconda del bit che deve trasmettere
- PRBL: solo due livelli segnale



Modulazione di frequenza (FM)

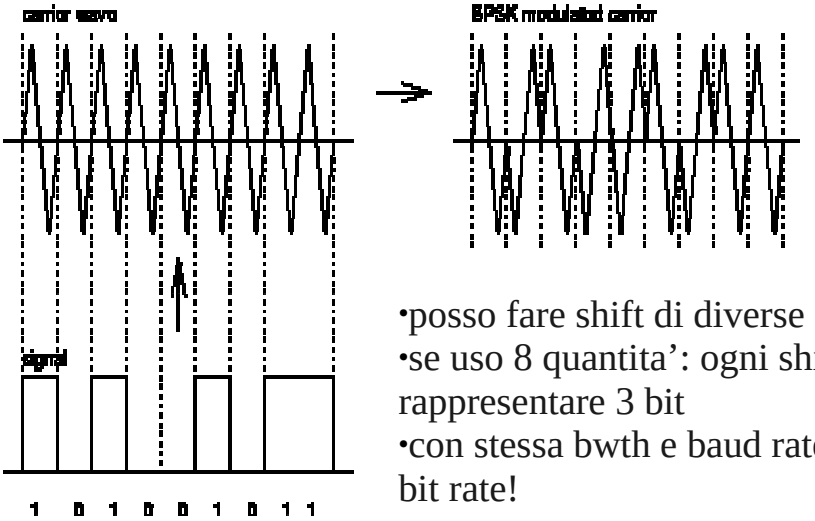
- *Due o piu' segnali che usano frequenze di carrier differenti, possono essere trasmessi sullo stesso mezzo senza interferenze*



•due carrier diversi per i bit 0 e 1
•prbl: solo due livelli di segnale
•mi serve bwth doppia per stesso rate

Modulazione di fase

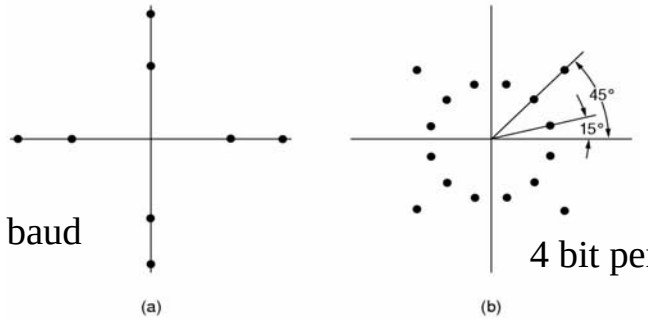
- faccio shift dell'onda sinusoidale, di quantita' differente a seconda dell'informazione che invio



•posso fare shift di diverse quantita'
•se uso 8 quantita': ogni shift puo' rappresentare 3 bit
•con stessa bwth e baud rate, aumento bit rate!

Constellation pattern

- modem usano combinaz. AM/modulaz. fase
- diagrammi rappresentativi di combinazioni valide di ampiezza e fase
- due modem per comunicare devono fare riferimento alla stessa costellazione

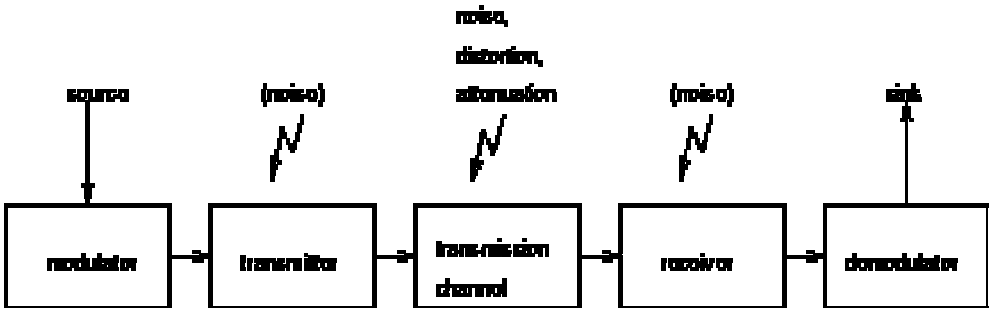


3 bit per baud

4 bit per baud

Trasmissione dell'informazione

- con baud rate R, T bit codificati da ogni shift: max bit rate = 2 R T
- full−duplex: mod/demod in ogni stazione
  - carrier diversi o synchro per evitare trasmissione contemporanea (half−duplex)



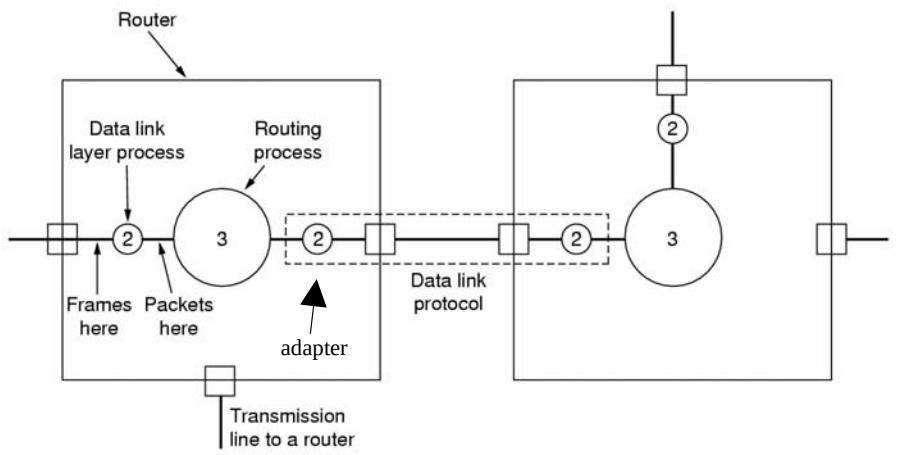
Funzionalità di Data Link L.

- framing dei pkt e addressing
- rilevazione e correzione di errori
- accesso dei frame al canale
- consegna affidabile per link con alto tasso di errore
- controllo di flusso
- trasmissione half−duplex o full−duplex
- NB: un frame può essere gestito da diversi protocolli DLL, che offrono servizi differenti

Servizi di DLL

- unacknowledged connectionless
  - per traffico real−time o su canali con basso tasso errore
- acknowledged connectionless
  - per canali inaffidabili (eg. wireless)
  - migliora throughput wrt. ritrasmissione msg
  - può generare frame duplicati
- acknowledged connection−oriented
  - frame ricevuti in ordine 1! volta
  - uso #seq + timer per last ack

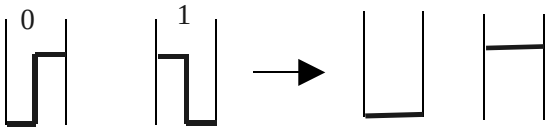
Architettura di DLL



- adapter = Network Interface Card
  - composto da bus interface + line interface
- processo DLL gestisce tutte le linee usando strutture dati distinte

Framing

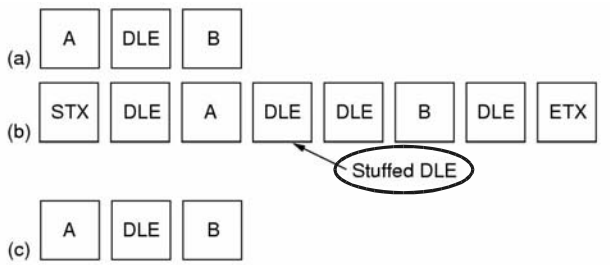
- PRBL: Phy.L. consegna sequenze bit. DLL deve riconoscere limiti frame
- time gap tra frame: inaffidabile
- character count nello header del frame
  - errori possono alterare valore
  - difficile risincronizzazione
  - usato in combinazione con altri metodi
- violazione codifica Phy.L.



possono essere delimiter

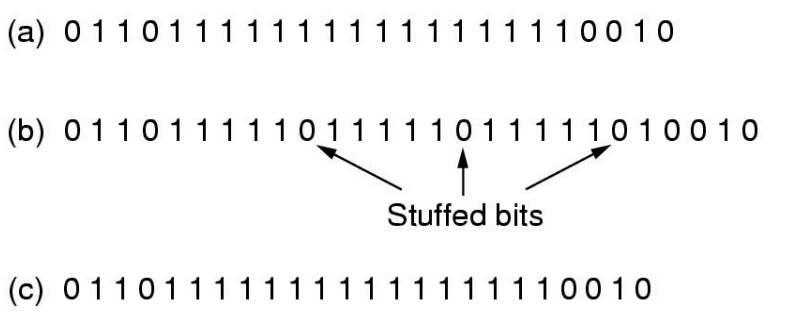
Character stuffing

- più facile risincronizzazione
- DLE = Data Link Escape
- STX = Start of Text ; ETX = End of Text
- DLE STX = start of frame
- DLE ETX = end of frame



Bit stuffing

- meno legato a codifica ASCII
- flag = 01111110 a start/end frame
- 011111… nei dati → 0111110…



Gestione errori

- mezzo fisico soggetto a guasti
  - frame persi → ack + ritrasmissioni
  - frame danneggiati (rumore) /* es. wireless */
  - ES: trasmesso 1 0 0 1 1 0 1 1 1 0 0 1
    ricevuto 1 1 0 0 1 0 1 1 0 1 1 1
    errore singolo burst
- servizio acknowledged connection oriented: DLL deve rilevare / correggere errori
  - rilevazione: si sono verificati errori?
  - correzione: quali errori si sono verificati?
  - ridondanza informazioni

Distanza di Hamming

- frame di m bit → 2^m parole possibili
  - ES: m=3 , 2^m=8
    000 001 010 011 100 101 110 111
    000 —error→ 010 entrambe parole legali

2. r bit di ridondanza: n = m+r , k = $2^m < 2^n$ parole legali
   - $2^n - k$ parole illegali: se le ricevo so che esiste errore
   - ES: m=3 , k=8 , r=1
     0000 0011 0101 0110 1001 1010 1100 1111
     0000 → 0100 errore! ; 0000 → 0101 ???

- Dist. Hamming tra 2 parole = #bit differenti (XOR)
  - ES: 0 0 1 0 0 0 1 1 0 D_H=4
    1 0 0 1 0 0 1 0 0 4 errori trasformano una nell'altra

Distanza di Hamming

- distanza di Hamming di un codice $\mathcal{C}=\{w_1, \ldots, w_n\}$
  - $D_H(\mathcal{C})$ = min D_H $\{w_i, w_j\}$ per ogni coppia di parole
- se per $\mathcal{C}$, $D_H(\mathcal{C})$ >=2d+1 allora posso
  - rilevare gli errori che cambiano fino a 2d bit in un frame
  - correggere gli errori che cambiano fino a d bit in un frame
- ES: $\mathcal{C}$ ={00000 , 00111 , 11100}
  - $D_H(\mathcal{C})$ = min{3, 3, 4} =3 → d=1 (2d =2)
  - rilevo 2 errori: 11000 deriva da 00000 o da 11100
  - correggo 1 errore: 10000 puo' solo derivare da 00000
  - non rilevo 3 errori: 00111 puo' essere 00000 errato
- stima error rate della rete

Codifica

- trade off tra overhead e capacità rilevaz./correz.
- bit di parita': 1 bit per frame t.c. rende pari gli '1'
  - $D_H(\mathcal{C})$ = 2: rilevo errore singolo, non correggo
  - burst: probabilita' di rilevare errore = ½
- lower bound #bit ridondanti per correzione errori singoli
  - 2^m messaggi legali, r bit ridondanza, n = m + r
  - per ogni msg esistono n msg illegali a distanza 1
    - per correggere singolo errore: n+1 bit pattern riservati per ogni msg
    - ES: per 000 riservo 001 010 100
  - $(n+1)\, 2^m <= 2^n \rightarrow (m + r + 1)\, 2^m <= 2^{m+r} \rightarrow m + r + 1 <= 2^r$
    - ES: m = 1000 → $1001+r <= 2^r \rightarrow r >= 10$

Codifica di Hamming

- soddisfa lower bound precedente: rileva/corregge errori singoli, con bit di parita'
  - bit i controllato da insieme P_i di bit parita'
  - scomposizione di i in somma di potenze di 2
  - ES: m=5 $\rightarrow$ 6+r <= 2^r $\rightarrow$ r >= 4 e n=9
  - bit di parita' in posizione 1, 2, 4, 8; schema frame:

posizione dati			1		2	3	4		5
	p	p	d	p	d	d	d	p	d
scomposizione			1+2		1+4	2+4	1+2+4		1+8

NB: ogni bit dati ha scomposizione unica

  - codifica 0 0 1 1 0 = 1 0 0 0 0 1 1 0 0
  - se bit modificato, sono sbagliati tutti i suoi bit di parita'

Hamming: correzione burst

- correzione burst di lunghezza k
  - scrivo k messaggi in colonne
  - mando bit leggendo per righe
  - k bit errati in sequenza cambiano max 1 bit per colonna
    - max 1 bit errato per frame: so correggere!
- ES: n=5 , k=3
  - devo mandare 00110 10010 11100

0	1	1
0	0	1
1	0	1
1	1	0
0	0	0

spedisco 0 1 1 0 0 1 1 0 1 1 1 0 0 0 0

ricevente aspetta n gruppi di k bit, che scrive per righe e legge per colonne

Es. codice Hamming

- m=10 (r=4) ; codifica di 1001100111
  - struttura frame p p 1 p 0 0 1 p 1 0 0 1 1 1
- 3=1+2 ; 5=1+4 ; 6=2+4 ; 7=1+2+4 ; 9=1+8 ; 10=2+8 ; 11=1+2+8 ; 12=4+8 ; 13=1+4+8 ; 14=2+4+8
- bit 1 controlla bit 3, 5, 7, 9, 11 e 13
  - bit 1 controlla 1 0 1 1 0 1 percio' vale 0
- bit 2 controlla 3, 6, 7, 10, 11, 14 = 1 0 1 0 0 1 → 1
- bit 4 controlla 5, 6, 7, 12, 13, 14 = 0 0 1 1 1 1 → 0
- bit 8 controlla 9, 10, 11, 12, 13, 14 = 1 0 0 1 1 1 → 0
- codifica frame e' 0 1 1 0 0 0 1 0 1 0 0 1 1 1
- ricevo 0 1 1 0 1 0 1 0 1 0 0 1 1 1 → K=1+4
- ricevo 0 0 1 0 0 0 1 0 1 0 0 1 1 1 → K=2

Cyclic Redundancy Check

- msg come rappresentazione polinomio
  - es: 0 0 1 0 1 1 corrisponde a $x^3 + x^1 + x^0$
- aritmetica in modulo 2: somme e sottrazioni senza riporto
- codifica usando polinomio generatore G(x) t.c.
  - ha grado inferiore a lunghezza frame [r]
  - ha primo e ultimo bit '1'
- aggiungo r bit '0' in coda al msg → $M(x)\, x^r$
- divido stringa ottenuta per G(x) e sottraggo resto
  - trasmetto $T(x) = M(x)\, x^r - (M(x)\, x^r \bmod G(x))$
- implementato in hardware

Esempio CRC

- M(x) = 1 1 0 1 0 1 1 0 1 1 ; G(x) = 10011
- x^r M(X) = 1101011011 0000 // G(x) ha grado 4

```
1 1 0 1 0 1 1 0 1 1 0 0 0 0 : 1 0 0 1 1 = 1 1 0 0 0 0 1 0 1 0
0 1 0 0 1 1
  0 0 0 0 0 1
    0 0 0 0 1 0
      0 0 0 1 0 1
        0 0 1 0 1 1
          0 1 0 1 1 0
            0 0 1 0 1 0
              0 1 0 1 0 0
                0 0 1 1 1 0  RESTO
```

trasmetto T(x)=11010110111110
arriva 10010010110111
Errore 01000100001001

- se frame ricevuto non e' divisibile esattamente per G(x), si e' verificato un errore
- errori rilevabili dipendono da G(x)

Proprieta' G(x)

- singolo errore: E(x) = x^i rilevati
  - G(x) ha >= 2 termini, percio non divide x^i
- due errori isolati: E(x)=$x^i + x^j = x^i (x^{j-i} + 1)$
  - rilevati se G(x) non divide x^k +1 per k <= m−1
- #dispari errori: *non esiste polinomio con #dispari di termini che abbia (x+1) come fattore nell'aritmetica mod 2*
  - DIM: sia E(x) = (x+1) Q(x) . Allora E(1)=1 sostituendo 1 nei termini in #dispari. Ma E(1)=(1+1) Q(x) =0
- burst di lunghezza <=r: E(x)=$x^i (x^{k-1} + ... + 1)$; k <= r
  - rilevato: G(x) ha termine x^0, percio' non divide x^i; $(x^{k-1} + ... + 1)$ ha grado <r, percio' non e' divisibile per G(x)
- burst di lunghezza >r: la prob. di rilevare errore e' $1/(2^{r-1})$

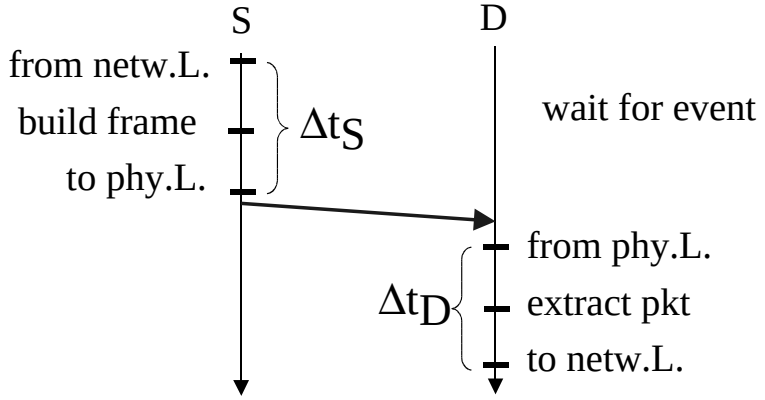
Uso CRC

- esistono polinomi standard
- CRC−12: $x^{12} + x^{11} + x^3 + x^2 + x^1 + 1$ per char di 6 bit
- CRC−16: $x^{16} + x^{15} + x^2 + 1$ per char di 8 bit
- CRC−CCITT: $X^{16} + x^{12} + x^5 + 1$ per char di 8 bit
- CRC−16 e CRC−CCITT catturano: errori singoli, errori doppi, #dispari errori, burst di lunghezza <= 16
  - 99.997% errori in burst di lunghezza 17
  - 99.998% errori in burst di lunghezza >= 18

Protocolli di DLL

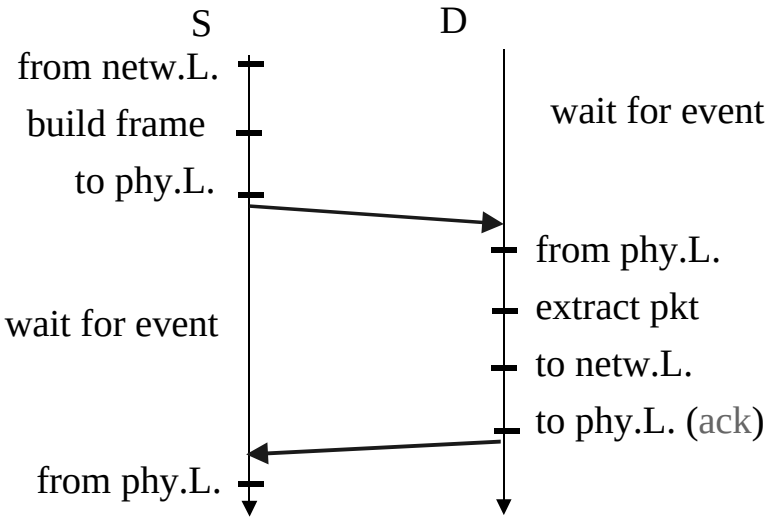
- S desidera serv. affidabile conn.oriented a D
- assunzioni:
  - dati da netw.L. sempre disponibili
  - non considerato contenuto pkt né processing per error control
  - frame portano #seq in {0,MAX_SEQ}
  - pkt size fissa di MAX_PKT byte
  - frame header = <kind, #seq, ack, info>
  - canale fisico può perdere frame
- timer per attesa acknowledgement

Protocollo 1



- OK se physical link esente da errori
- PRBL: se D è lento a processare ($\Delta t_S < \Delta t_D$) e DLL di D *non* ha buffer infinito?
- attese fissate: inaffidabile e inefficiente

Proto 2: Stop−and−Wait

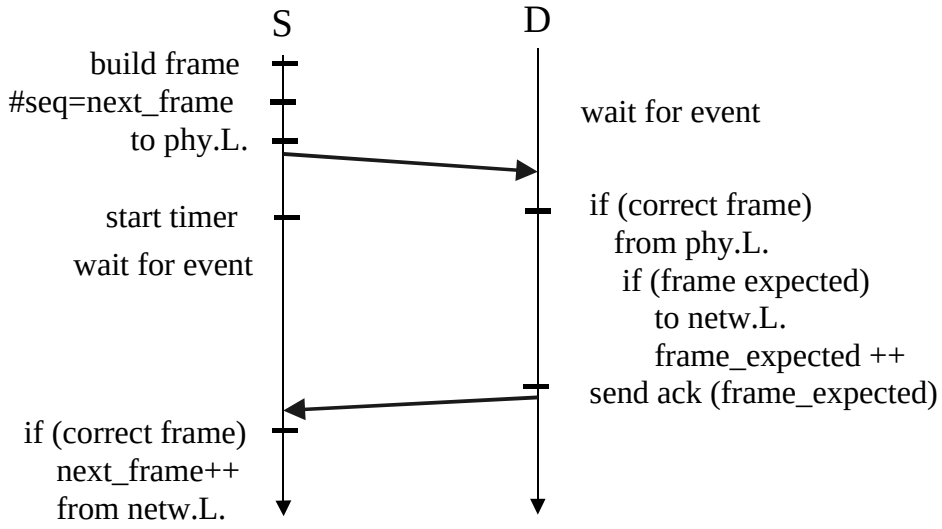


- stretta alternanza (basta half−duplex)
- PRBL: è efficiente? Cosa accade se frame perso?

Proto 3: link inaffidabile

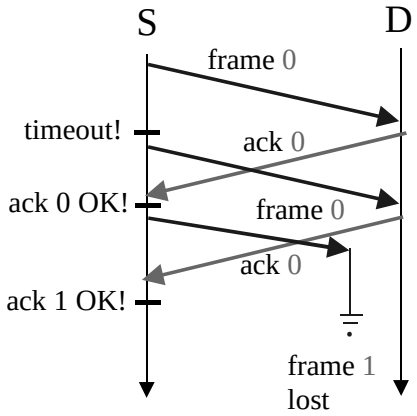
se uso solo timer in S?

- se perdita ack consegnati pkt duplicati



PAR

prestazioni e correttezza dipendenti da lunghezza timeout
(circa round−trip time)

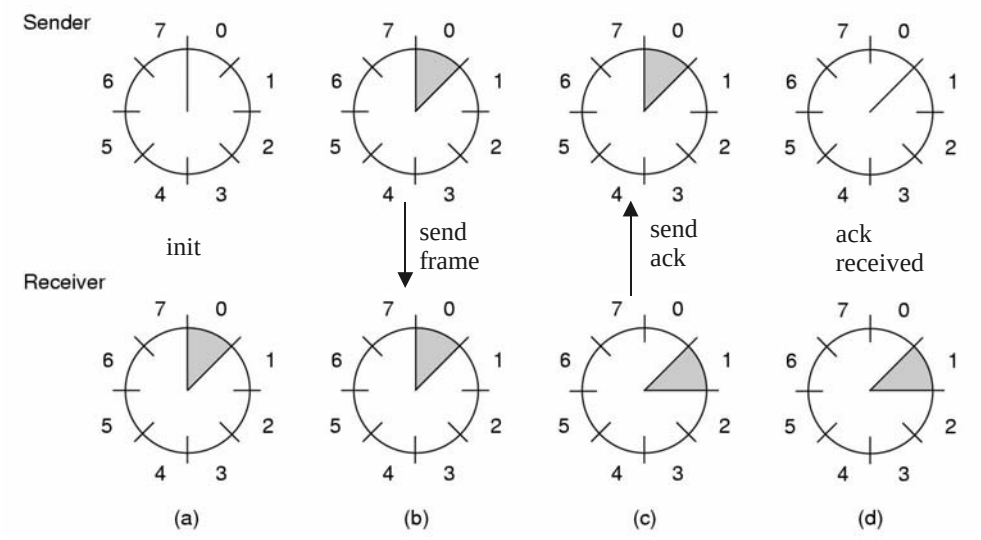


- ➢ ack non caricano #seq frame a cui si riferiscono
- ➢ se timer breve: D vedrà gap nella sequenza

Discussione protocolli

- ❑ se desiderata comunic. bidirezionale → full−duplex
  - ❖ piggybacking per efficiente uso bwth
    - probl dimensione timer attesa invio ack via piggybacking
- ❑ PAR sensibile a dimensionamento timer
- ❑ per ogni rtt inviato al max 1 frame
  - ❖ se rtt >> processing delay, link sottoutilizzato
- ❑ parallelismo tra frame → ***sliding window***
- ❑ sender window per frame inviati in attesa ack
  - ❖ a ack per frame con min #seq → shift avanti finestra
- ❑ receiver window per frame attesi e frame ricevuti fuori ordine
  - ❖ shift a frame con min #seq; frame fuori finestra dropped

Es: sliding window 1−bit



in generale: come si dimensionano finestre?

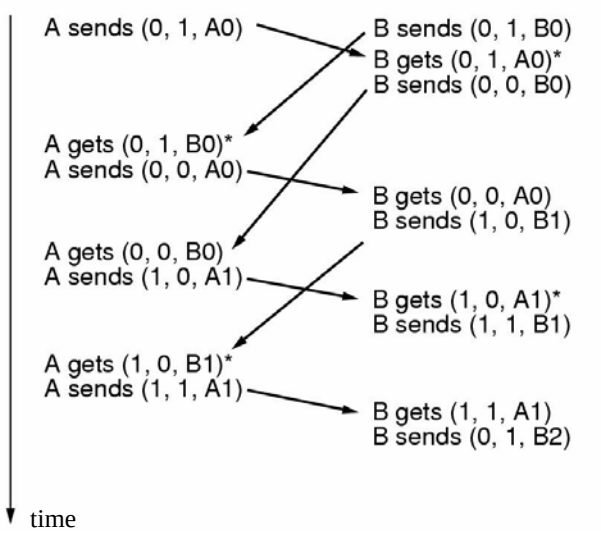
Proto 4: 1 bit bidirezionale

```
#define MAX_SEQ 1

next_frame=frame_expected=0;
from_netw_L(&buffer);
s.info=buffer;
s.seq=next_frame;
s.ack=frame_expected;
to_phy_L(&s);
start_timer(s.seq);
while (true) {
  wait for event;
  // correct fr., damaged fr., t.o.
  if (correct frame arrival) {
    from_phy_L(&r);
    if (r.seq==frame_expected) {
      to_netw_L(r.info);
      frame_expected++;
    } // end if
    if (r.ack==next_frame) {
      from_netw_L(&buffer);
      next_frame++;
    } // end if
  } // end if
  s.info=buffer;
  s.seq=next_frame;
  s.ack=1−frame_expected;
  to_phy_L(&s);
  start_timer(s.seq);
} // end while
```

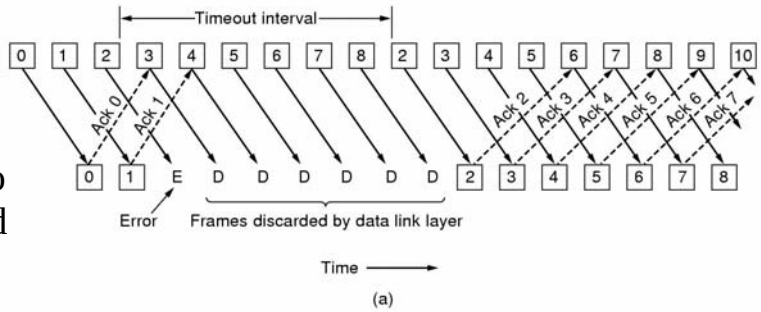
Discussione proto 4

- ancora stop−and−wait
- piggyback: ack contiene #seq ultimo frame ricevuto correttamente
  - non più sensibile a timeout troppo breve, ma può essere inefficiente
- prbl synchro se A e B inziano a inviare contemporaneamente

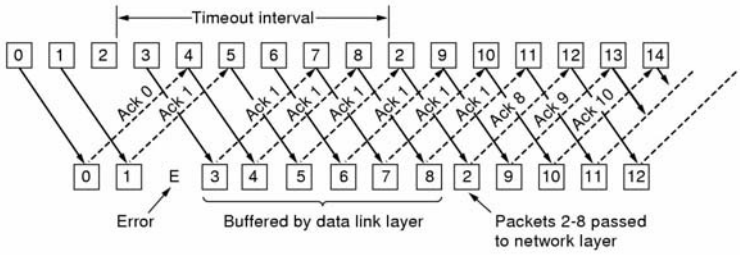


Ottimizzazione

consentiamo >1 outbound frame alla volta



(a)



(b)

Pipelining

- rtt=500 msec; frame size=1 Kbit; bwth=50 Kbps
- transmission delay:
- propagation delay: 1Kbit/50Kbps=20 msec
- arrivo a rtt/2=250 msec
- ack ricevuto 250+20=270 msec
- sender aspetta per >=270+250=520 msec
- utilizzo rete 500 msec
- utilizzo 100% per 20msec/520msec= 3.85%

520/20=26 frame outbound

Utilizzo rete

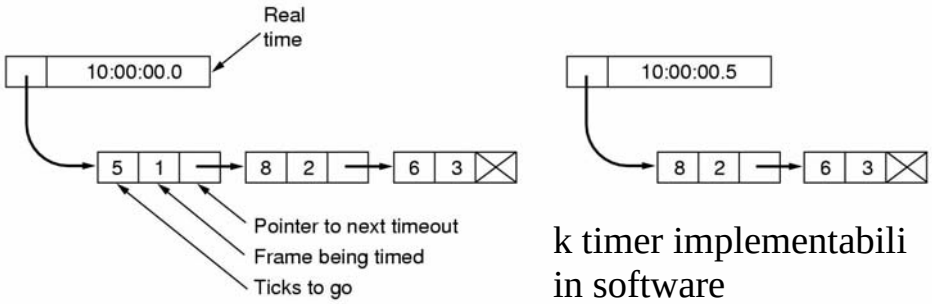
- l=frame size; R=rtt; b=bwth
- transmission delay Tx = l/b
- propagation d. Tp = R/2= lungh.cavo/vel.propag.
- stop−and−wait: linea occupata per l/b e idle per R
  - utilizzo U=(l/b) / (l/b + R) = l/(l+bR) = Tx/(Tx+2Tp)
  - se l<bR, utilizzo<50%
- sliding window con finestra k
  - linea occupata per k Tx = kl/b in intervallo Tx+R=Tx+2Tp
  - U=kTx/(Tx+2Tp)=k/(1+2Tp/Tx)=kl/(l+bR)
  - U → 100% quando k Tx → Tx+2Tp da cui si ottiene dimensione ottimale finestra di trasmissione

Gestione errori

- pipelining ottimizza utilizzazione bwth
- se gap nella sequenza o frame errati?
  - destinatario riceve altri *k−1* frame prima che sender se ne accorga
- go−back−n : ricevente scarta tutti i frame successivi a quello mancante
  - basso uso buffer a destinatario (k=1)
  - alto spreco bwth se rtt grande (perché?)
- selective−repeat : ack per frame corretti
  - sender ritrasmette solo frame mancanti
  - finestra destinatario>1, efficiente uso bwth
- trade off buffer space vs. disponibilità banda
- in entrambi sender deve usare k timer

Dettagli implementativi

- #seq può assumere MAX_SEQ+1 valori
- MAX_SEQ t.c. k <= MAX_SEQ
  - se k=MAX_SEQ+1 e arriva ack per #seq=MAX_SEQ+1 non so distinguere se si riferisce a finestra attuale o precedente



k timer implementabili in software

Simulatore

- in linguaggio C, disponibile sul sito del corso (*Materiale didattico*)
- *README* contiene info compilazione e uso
- *protocol.h* contiene template procedure
- *exercizes* contiene proposte esercizi per studio/confronto protocolli
- provare a cambiare valore costanti, operazioni eseguite

… buon divertimento! …

Proto 5: go−back−n (1)

- formato frame: <dati, #seq, ack>
  - ack cumulativo: *ack=k* conferma arrivo frame in finestra sender con #seq <= *k*
- inizializzazione:
  - ack atteso = #seq frame da mandare = 0
  - #seq frame atteso = #frame in window = 0
- interfaccia sender/netw.L.
  - a init e finchè spazio in src window: netw.L. abilitato
  - se src window piena: netw.L. disabilitato
  - se netw.L. abilitato e ha pkt: da pkt produco e invio nuovo frame; aggiorno #seq frame da mandare
- finestra src gestita come array circolare

Proto 5: go−back−n (2)

- *procedura* invio_frame:
  - riempio campi nuovo frame e passo a phy.L. /*ack da #seq frame atteso*/
  - set timer che rileva perdita frame o ack
- schema funzionamento: ciclo infinito
  - attesa evento – gestione– (dis)abilitaz. netw.L.
  - eventi possibili:
    - nuovo pkt da netw.L. /* già discusso */
    - arrivo frame danneggiato
    - arrivo frame corretto
    - t.o. per frame in finestra src
- arrivo frame danneggiato: frame dropped

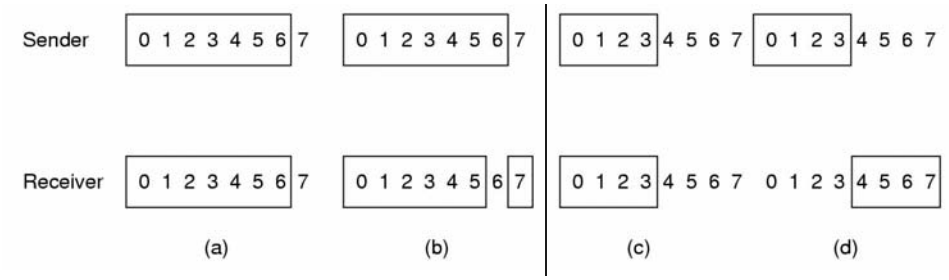
Proto 5: go−back−n (3)

- scadenza t.o. per frame in finestra src:
  - re−invio_frame per frame da quello con #seq = ack atteso e re−set dei relativi timeout
- arrivo frame corretto: prelevo da phy.L.
  - se #seq t.c. è frame atteso: passo a netw.L /*altrimenti drop*/
    - aggiorno #seq frame atteso
  - ctrl se frame ha piggybacked ack in finestra /*ack atteso <= frame ack < #seq frame da inviare*/
    - libero tutti i posti nella fin. relativi a frame acknowledged e ne azzero i timer
    - aggiorno #frame in window e ack atteso

Politica selective−repeat

- destinatario accetta frame nella finestra anche fuori ordine (ma consegna ordinata)
  - cumulative ack
- timer per decidere invio ack in assenza di traffico di ritorno evita blocco (< rtt!)
- negative ack per velocizzare recupero
- timer sender: se piccolo, nak inutili ma spreco bwth per ritrasmissioni extra
- finestra ricezione fissata = k_recv

SR: dimensione finestra k_recv



- b) receiver manda ack e aggiorna finestra
  - se ack persi, sender rimanda 0
  - receiver tiene 0 come nuovo e rimanda ack 6
  - sender avanza finestra e continua…
- se k_recv<=(MAX_SEQ+1)/2 no problem

Proto 6: selective−repeat (1)

- formato frame: <tipo, dati, #seq, ack, nak>
  - ack cumulativo, nak individuale (max 1 a frame)
- inizializzazione come prima; inoltre:
  - oldest_frame (per t.o.), array arrived[] per frame ricevuti correttamente ma fuori ordine
- gestione finestre: 2 array circolari
  - limiti fin. src: [ack atteso, #seq frame da mandare −1]
  - limiti fin. rcv: [#seq frame atteso, too_far=(MAX_SEQ+1)/2
- interfaccia sender/netw.L. come prima

Proto 6: selective−repeat (2)

- *procedura* invio_frame:
  - riempio campi e passo a phy.L. /*ack da #seq frame atteso, è sempre piggybacked*/
  - se frame dati: set timer attesa ack
  - azzero timer per decisione invio ack separato
- schema funzionamento: ciclo ¥ come prima
  - eventi possibili:
    - nuovo pkt da netw.L. /* già discusso */
    - arrivo frame danneggiato
    - arrivo frame corretto
    - t.o. per frame in finestra src
    - t.o per invio separato ack

Proto 6: selective−repeat (3)

- arrivo frame danneggiato: invio_frame di tipo nak con #seq frame atteso
- t.o. frame in finestra src: invio_frame di tipo dati contenente oldest_frame
- arrivo frame corretto: prelevo da phy.L.
  - se dati ed è frame in fin.rcv *non duplicato*:
    - inserisco in fin.rcv e ricordo ∃nza in arrived[]
    - passo a netw.L. maggior # possibile frame in sequenza
    - aggiorno limiti fin.rcv e arrived[] in accordo
    - ricordo che non devo mandare nak e set timer per invio separato ack

Proto 6: selective−repeat (4)

arrivo frame corretto <cont.>

- se dati ma ≠ frame atteso e no nak da inviare
  - invio_frame di tipo nak con #seq frame atteso
- se nak per un frame in fin.src: invio_frame di tipo dati con frame richiesto
- se frame ha piggybacked ack per frame in fin.src:
  - libero tutti i posti in fin.src relativi a frame acknowledged e ne azzero i timer
  - aggiorno limite inferiore fin.src e #frame contenuti, e ack atteso

- t.o. invio separato ack: invio_frame di tipo ack con #seq frame atteso

Serial Line IP

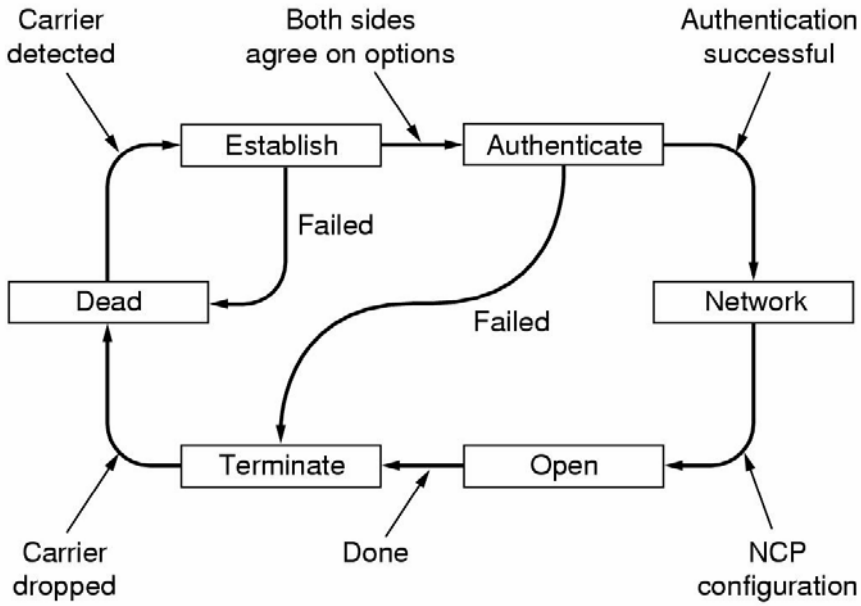
- in Internet: p2p per connessione LAN (leased line) o PC a router (ISP) remoto
- SLIP: connessione via modem (RFC 1055)
  - uso character stuffing
  - compatibile solo con IP
    - indirizzi IP assegnati staticamente e noti a priori
  - compressione TCP/IP header
  - non supporta error detection/correction né autenticazione

Point−to−point protocol

- supporta error detection, multipli proto netw.L., autenticazione, ind. IP dinamici, header compression
  - usa character stuffing
  - opzionale: servizio affidabile
- Link Control Proto per set−up/negoziazione
  - e.g. max payload size, proto autenticazione
- Netw. Ctrl Proto dipendente da proto netw.L.
  - e.g. assegnazione indirizzo IP

Bytes	1	1	1	1 or 2	Variable	2 or 4	1
	Flag 01111110	Address 11111111	Control 00000011	Protocol	Payload	Checksum	Flag 01111110

PPP: automa a stati



MAC sublayer

- regola l'accesso a canale broadcast
- politiche statiche (FDM, TDM) inefficienti
  - 1/N bwth sprecato quando un utente è idle
  - non più di N utenti ammessi comunque
  - aumento del ritardo di N volte:

    C=bwth canale, T=tempo attesa servizio

    λ frame/sec arrival rate, $1/\mu$ bit/frame

    $T = 1/(\mu C - \lambda)$ dalla teoria delle code.

    Se banda disponibile è C/N, il tempo di attesa è

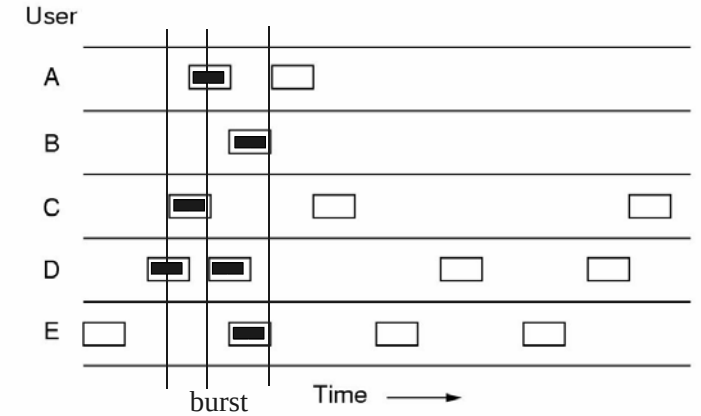
    $T' = 1/(\mu C/n - \lambda/N) = N/(\mu C - \lambda) = NT$

Politiche *dinamiche*

- protocolli random access
  - una stazione trasmette quando ha frame da mandare, usando tutta la bwth canale
  - se due trasmissioni contemporanee: collisione
  - protocollo specifica come risolvere collisioni
- assunzioni sul sistema
  - al canale accedono N stazioni indipendenti che generano 1 frame alla volta
  - stazioni comunicano solo attraverso canale condiviso
  - due frame i cui tempi di trasmissione si sovrappongono sono totalmente distrutti
  - tutte le stazioni rilevano le collisioni
  - tempo può essere continuo o slotted (synchro)
  - stazioni possono eseguire carrier sense

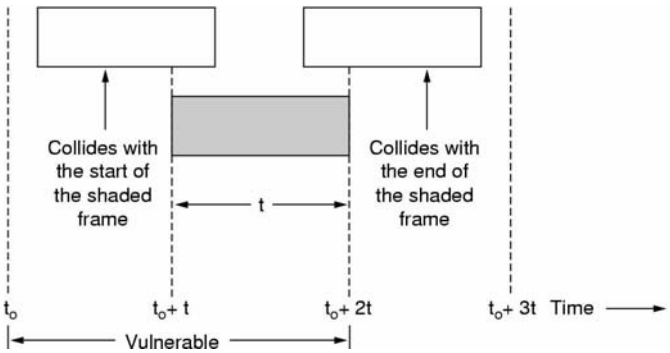
Aloha (1970)

- ogni sender trasmette quando ha frame
  - se collisione, attesa per tempo random (perchè?) prima di ritrasmissione
- frame time = transm. delay per 1 frame



Analisi Aloha

- G = probabilità tentativo trasmissione 1 frame in 1 frame time (Poisson) <= 1
- successo se 1! frame trasmesso in un intervallo (2 x frame time) = 2 Tf



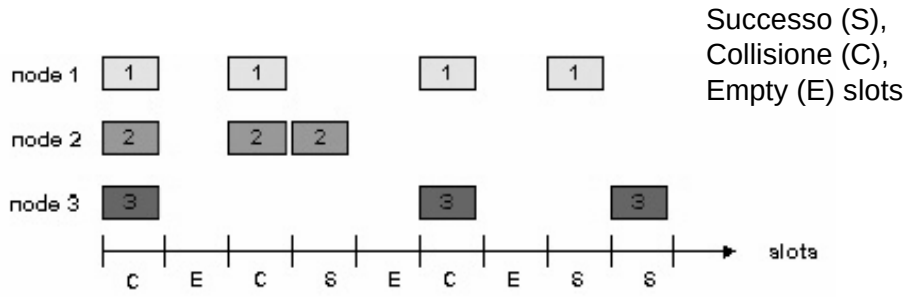
Analisi Aloha (cont.)

- prob trasmissione 1 frame in Tf: $G\,e^{-G}$
- prob. nessun frame trasmesso in Tf: e^{-G}
- prob. successo: $G\,e^{-G}\,e^{-G} = G\,e^{-2G}$
- max: uguaglio a 0 la derivata

  $D=e^{-2G}(1-2G) \Rightarrow G = ¥$ (min) o $G= ½$ (max)
- per $G= ½$, $Ge^{-2G} = ½\,e^{-1} = 1/2e$ ~ 18.4%
- NB: $e^{-1/2}$ ~ 60.6% tempo inutilizzato!

Slotted Aloha (1972)

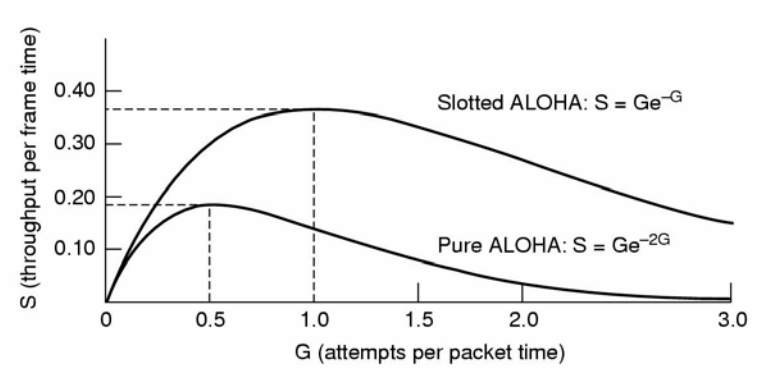
- tempo diviso in slot di durata = pkt trans. time
- nodo può trasmettere solo a inizio slot
  - collisione solo tra frame inviati nello stesso slot
  - se collisione: ritrasmetti pkt in slot successivi con prob p fino a successo
  - master clock sincronizza i nodi



Successo (S), Collisione (C), Empty (E) slots

Analisi Slotted Aloha

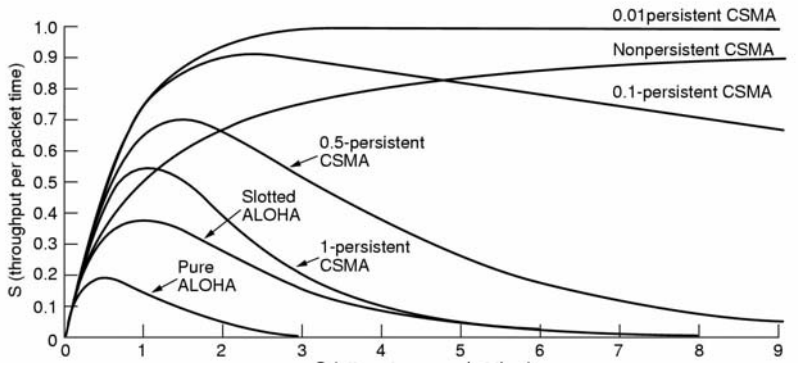
- simile a precedente ma contesa su Tf
- max efficienza = 1/e ~ 36.8% per G=1
- prob slot vuoto = 36.8%



Carrier−Sense−Multiple−Access

- approccio distribuito
- stazione fa test canale; appena libero, invia frame (1−persistente)
- se collisione, attesa tempo random e retry
  - collisioni da propagation delay e contemporaneità stazioni
- non−persistente: se occupato, ricontrollo dopo tempo random
- p−persistente (*slotted*): probabilità p di trasmettere in slot idle

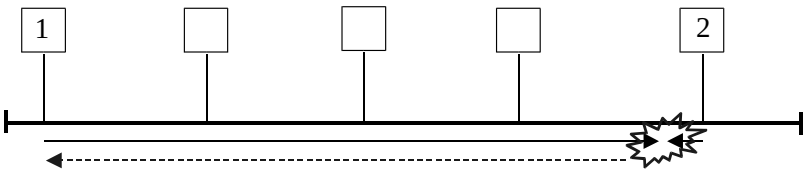
Confronto politiche CSMA



- 1−persistente: maggiori prob collisione per partenza simultanea stazioni
- non−persistente: migliore utilizzo canale, ma maggiori ritardi

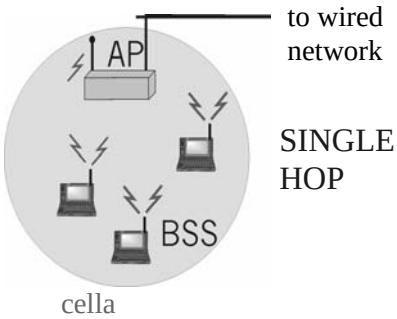
CSMA−CD

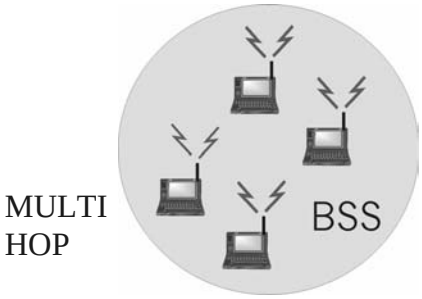
- stazione ascolta canale mentre trasmette
  - se collisione, trasmissione interrotta (backoff)
  - codifica t.c. consente rilevazione collisioni
- tempo rilevamento dipende da propagation delay τ
  - entro 2τ stazione è certa che tutte le altre hanno rilevato sua trasmissione



W−LAN

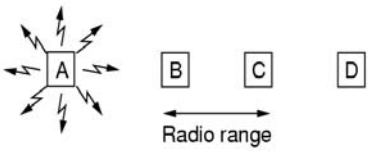
- wireless link: radio , infrarosso (bcast)
- architettura single−hop vs. multi−hop
- elevato tasso fallimento, handoff
- collision detection difficilmente attuabile



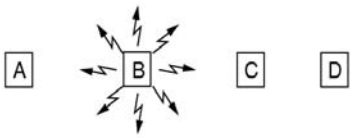


W−LAN: CSMA

- hidden station: C non sente A e trasmette a B
- exposed station: C sente B e non trasmette a D
- CSMA rileva attività intorno alla sorgente ma non intorno al ricevente

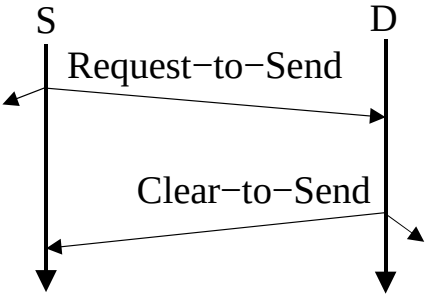


hidden



exposed

CSMA with Collision Avoidance



<RTS, data frame size, t_{start}>

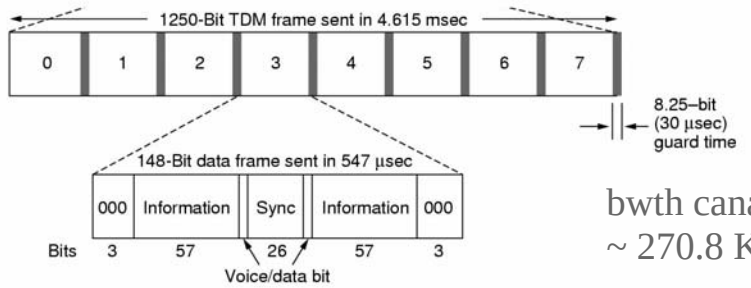
<CTS, data frame size>

802.11

- chi sente non trasmette per T_{frame} (+ T_{CTS})
- risolve hidden/exposed
- se collisione (RTS contemporanei) usa backoff
  (sotto quale ipotesi è vantaggioso?)
- MACAW: ack e ritrasmissioni a DLL

Global System for Mobile Comm

- bande a 900 MHz e 1800 MHz; circuit switched
- FDM: 124 bande frequenza (x2: upstream e downstream) da 200 KHz ognuna
- TDM: ogni banda supporta 8 connessioni



bwth canale
~ 270.8 Kbps

- handoff ~ 300 msec, tariffe a tempo

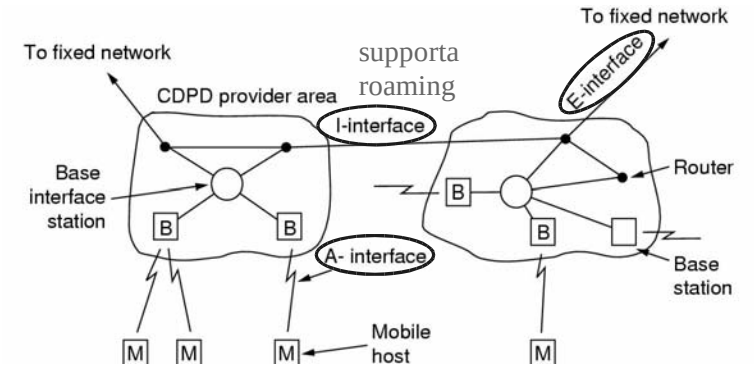
GSM

multiframe con canali controllo

- broadcast ctrl da base station per decisione handoff da parte dei MH
- dedicated ctrl per scambio info locazione e registrazione tra BS e MH
- common ctrl, suddiviso in canali logici
  - paging per annuncio chiamate arrivate
  - random access (Aloha) per richiesta slot in dedicated ctrl
  - access grant per assegnazione slot

Cellular Digital Pkt Data

- pkt switched, TACS compatibile, compressione e cifratura dati, error correction
- 274 bit dati → 378 bit codice + 42 bit flag
- up/down link TDM, bwth=19.2 Kbps (netta ~ 9600 bps)
- Digital Sense Multiple Access per uplink ≈ CSMA
  - BS informa di eventuali collisioni



NB: canali dati prelazionabili per voce

Code Division Multiple Access

- ogni sender usa codice differente
- bit codificato in chip sequence (o complem.)
- codici usabili simultaneamente → ortogonali
  - prodotto scalare
  - $S \bullet T = S \bullet \overline{T} = \overline{S} \bullet T = \overline{S} \bullet \overline{T} = 0 \,;\, S \bullet S = 1 \,;\, S \bullet \overline{S} = -1$
- ogni sender usa intera bwth canale
- chip sequence ortogonali si sommano quando inviati contemporaneamente

CDMA

A: 0 0 0 1 1 0 1 1	A: (−1 −1 −1 +1 +1 −1 +1 +1)
B: 0 0 1 0 1 1 1 0	B: (−1 −1 +1 −1 +1 +1 +1 −1)
C: 0 1 0 1 1 1 0 0	C: (−1 +1 −1 +1 +1 +1 −1 −1)
D: 0 1 0 0 0 0 1 0	D: (−1 +1 −1 −1 −1 −1 +1 −1)
(a)	(b)

Six examples:

− − 1 −	$\mathbf{C}$	$S_1 = (-1\ +1\ -1\ +1\ +1\ +1\ -1\ -1)$
− 1 1 −	$\mathbf{B} + \underline{\mathbf{C}}$	$S_2 = (-2\ \ 0\ \ 0\ \ 0\ +2\ +2\ \ 0\ -2)$
1 0 − −	$\mathbf{A} + \underline{\overline{\mathbf{B}}}$	$S_3 = (\ 0\ \ 0\ -2\ +2\ \ 0\ -2\ \ 0\ +2)$
1 0 1 −	$\mathbf{A} + \overline{\mathbf{B}} + \mathbf{C}$	$S_4 = (-1\ +1\ -3\ +3\ -1\ -1\ -1\ +1)$
1 1 1 1	$\mathbf{A} + \mathbf{B} + \mathbf{C} + \mathbf{D}$	$S_5 = (-4\ \ 0\ -2\ \ 0\ +2\ \ 0\ +2\ -2)$
1 1 0 1	$\mathbf{A} + \mathbf{B} + \overline{\mathbf{C}} + \mathbf{D}$	$S_6 = (-2\ -2\ \ 0\ -2\ \ 0\ -2\ +4\ \ 0)$

(c)

$S_1 \bullet C = (1 +1 +1 +1 +1 +1 +1 +1)/8 = 1$
$S_2 \bullet C = (2 +0 +0 +0 +2 +2 +0 +2)/8 = 1$
$S_3 \bullet C = (0 +0 +2 +2 +0 -2 +0 -2)/8 = 0$
$S_4 \bullet C = (1 +1 +3 +3 +1 -1 +1 -1)/8 = 1$
$S_5 \bullet C = (4 +0 +2 +0 +2 +0 -2 +2)/8 = 1$
$S_6 \bullet C = (2 -2 +0 -2 +0 -2 -4 +0)/8 = -1$

(d)

- decodifica: prodotto scalare tra segnale ricevuto e chip sequence sender di interesse
- prbl: synchro tra source e destination

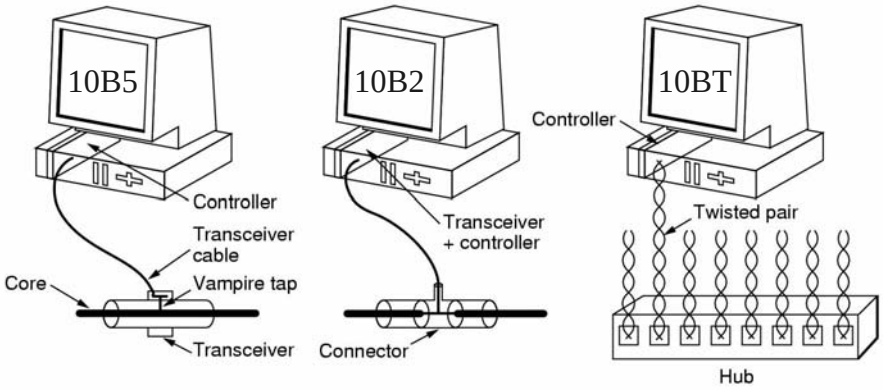
$$S \bullet C = (A + \overline{B} + C) \bullet C = 1$$

IEEE 802.3 (Ethernet)

- 1−persistente CSMA−CD
- time domain reflectometry per rilevare guasti: 10BaseT

tipo	cavo	max. segm.	nodi/segm.	pros
10Base5	coax (thick)	500 m.	100	per backbone
10Base2	coax (thin)	200 m.	30	+ economico
10BaseT	doppino	100 m.	1024	facile manutenzione
10BaseF	fibre	2 Km.	1024	su lunga distanza

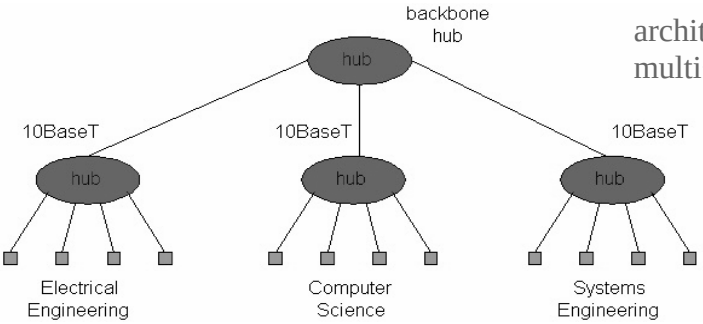
Connessioni tra LAN



- transceiver: collision detection, carrier sense
- transceiver cable: 5 doppini per data in/out, ctrl in/out, alimentazione
- controller: framing, calcolo e verifica checksum, gestione buffer

Hub

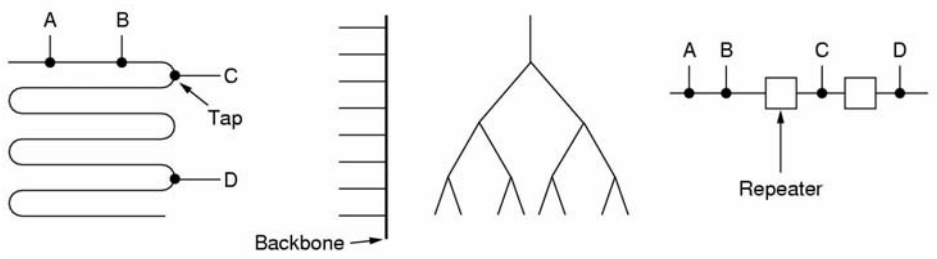
- device di phy.L., fa broadcast input su ogni oif
- tutti i LAN segment in stesso dominio di collisione
  - riduzione del thput aggregato
- backbone dà resistenza ai fallimenti
- usabile solo con cablature omogenee



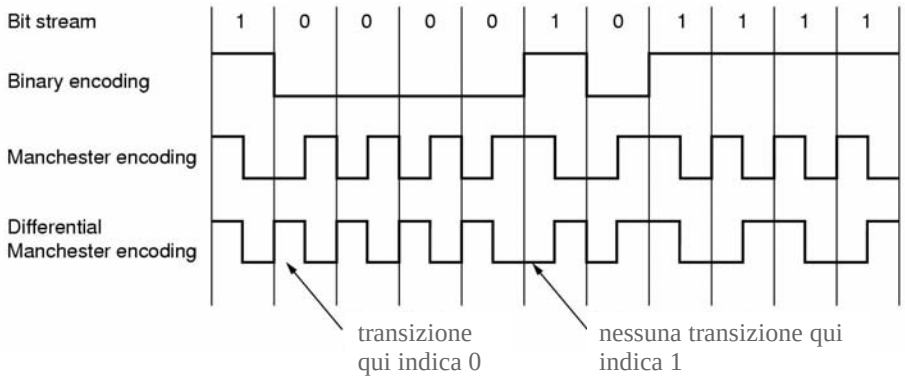
architettura multi−tier

Schemi cablatura

- repeater: riceve, amplifica e ritrasmette
  - specifica: max 2500 m. e 4 ripetitori tra 2 transceiver
- adapter manda segnale direttamente nel canale broadcast (± 0.85 V)

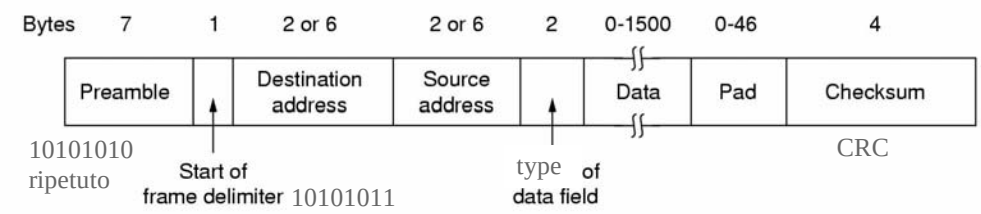


Manchester encoding



- 0 volt indica sender inattivo
- transizione a metà bit: facile synchro
- schema differenziale: miglior resistenza al rumore

802.3 MAC proto



- preamble: synchro clock recv con sender
- indirizzi: highest bit 0 per unicast, 1 per multicast; tutti 1 per broadcast
  - 46º bit distingue indirizzi locali o globali
- data min 46B, altrimenti padding
  - min size per rilevamento/gestione collisioni

Binary exponential backoff

- $2\tau \cong 51.2$ μsec (a 10 Mbps → 512b=64B) = slot time
- se rilevata collisione, stazione manda 48 bit rumore
- src attende random (0,1) prima di retry
- ad ogni nuova collisione, attesa random in $[0, 2^k-1]$ (max k=10)
- dopo 16 tentativi riportato fallimento
  - adattamento dinamico a # sorgenti
- assenza collisioni *non* garantisce ricezione corretta
  - aggiunta ack

Analisi 802.3

- k stazioni sempre pronte a trasmettere
  - p probabilità trasmissione
- prob. successo Ps = K p $(1-p)^{k-1}$
  - max funz 1/e = 37% per k → ¥ e p= 1/k
- prob contesa per esattamente j intervalli Pc[j] = Ps $(1-Ps)^{j-1}$
- #medio slot contesa E[Pc]:
- durata slot = 2τ → E[Pc]=2τ/Ps
  - al meglio E[Pc] = 2τ/e = 139 μsec
- efficienza=Tx/(Tx+ 2τ/Ps) = Tx/(Tx+ 2τe)

Analisi 802.3

- efficienza si può anche scrivere come

$$U = \frac{l/b}{l/b + 2\tau\ e}$$

- se L=lunghezza cavo, c=velocità propagaz.

$$U = \frac{l/b}{l/b + 2\tau\ e} = \frac{1}{1 + 2eLb/cl}$$

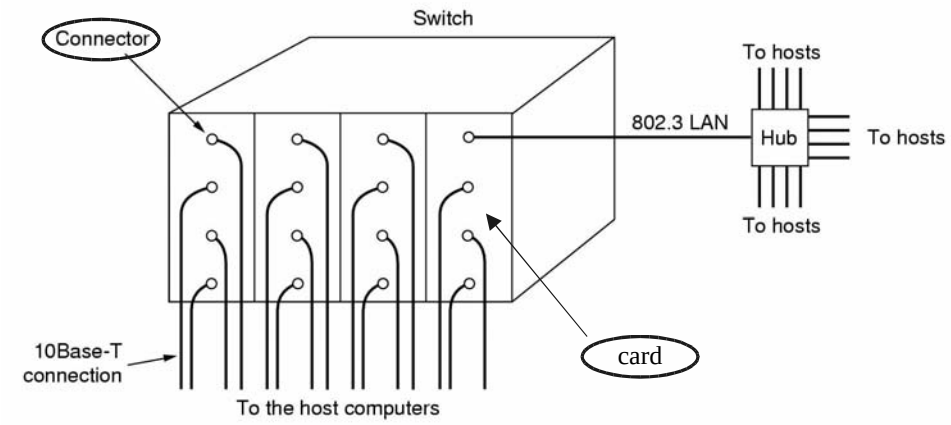
  - aumentando Lb diminuisce utilizzo…
  - diminuisce anche all'aumentare del #stazioni che cercano di spedire e al diminuire della taglia del frame

Bridge

- device di DLL: esamina indirizzo livello 2 e inoltra su oif più adatta
- ogni LAN segment è un distinto dominio di collisione
  - se buffer a ifc: operazioni full−duplex su tutte le porte
  - aiuta a supportare affidabilità e sicurezza consentendo isolamento LAN
- usabile con cablature Ethernet o LAN eterogenee
  - esegue traduzioni formati
- switch: bridge con alto #interfacce e alte prestazioni

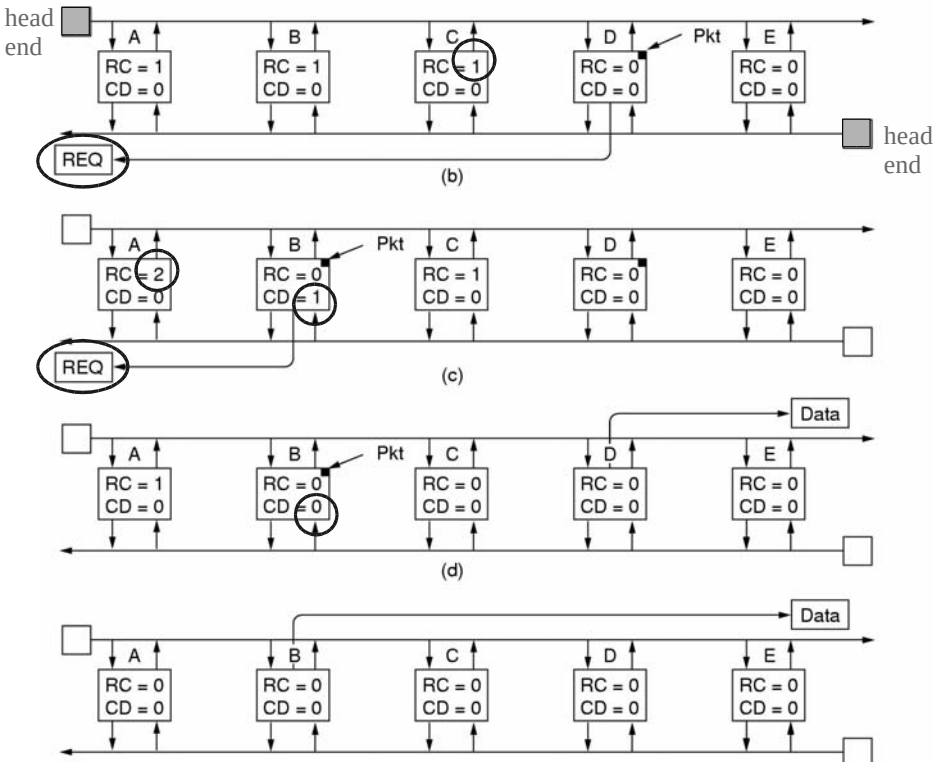
Switched LAN

- crescita LAN → saturazione
  - o aumento bwth, o nuova politica

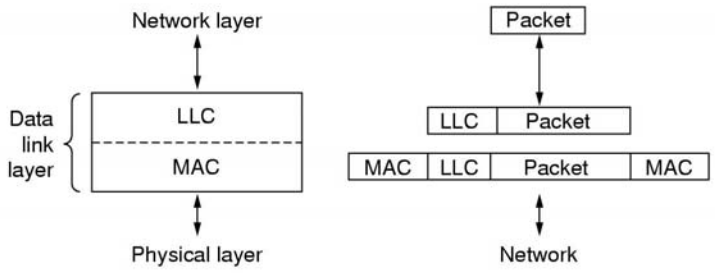


IEEE 802.6 (DQDB)

- head−end genera continuamente celle da 53B (44B payload, Busy/Request bits)
- ogni src deve sapere se dest a destra o sin
- trasmissione in ordine FIFO
  - ❖ src manda Request su reverse bus wrt dest
  - ❖ ogni nodo ricorda quanti nodi a valle devono mandare prima di lui
  - ❖ quando è suo turno, manda cella Busy con dati



Logical Link Control (802.2)



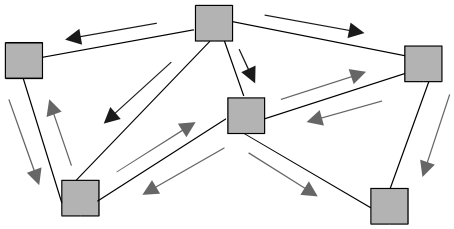
- nasconde differenze tra vari 802.x
  - interfaccia uniforme presentata a netw.L.
- fornisce error ctrl, flow ctrl
- header LLC contiene #seq, ack per servizi
  - unreliable datagram
  - acknowledged datagram
  - reliable connection−oriented

Bridge

- interconnessione tra LAN eterogenee
- riformatta frame, adegua header, ricalcola checksum
  - se frame troppo grande per LAN output: non c'è modo di gestirlo (drop)
- buffering frame se le LAN interconnesse operano a bit rate differenti
  - prbl se bottleneck wrt timer livelli superiori
- se una ma non tutte le LAN supportano priorità? Se una ma non tutte le LAN supportano ack?

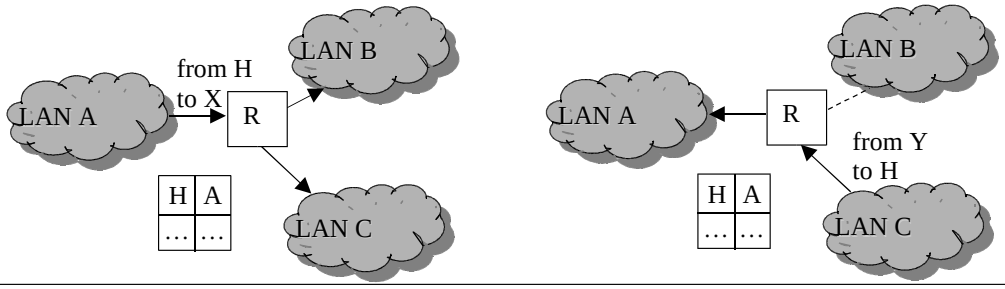
Transparent bridge

- obiettivo: plug&play tra LAN eterogenee
- hash table per inoltro frame
  - frame destinato a LAN da cui arriva è dropped
  - altrimenti: instradamento sulla LAN opportuna se nota oif
  - altrimenti flooding



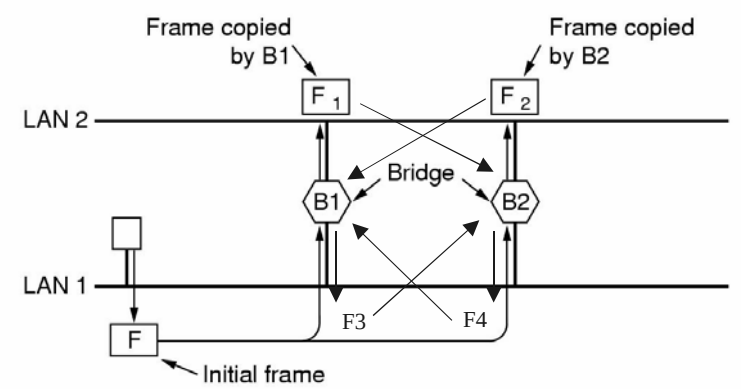
Backward learning

- usato da transparent bridge per imparare path: rete si auto−configura
- bridge vede tutti i frame generati in LAN connesse → impara dove sono src
- entry hash table: soft−state (minuti) rinfrescato ad ogni frame (in VLSI)



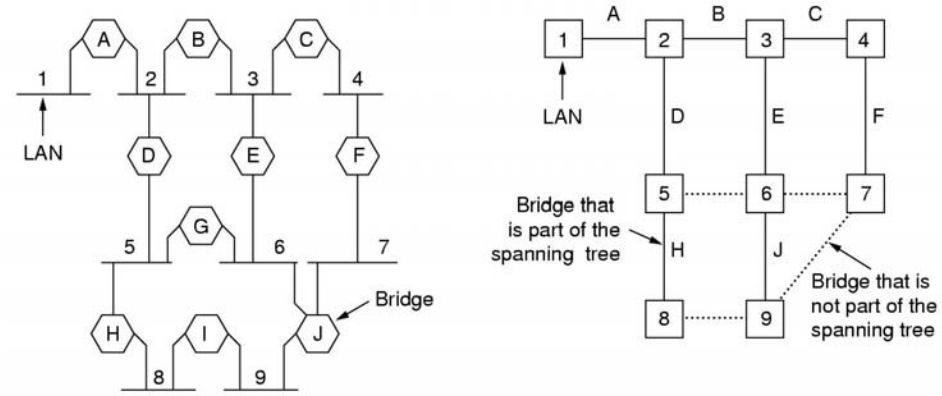
Affidabilità bridge

- più bridge operano in parallelo
  - minori probabilità disconnessione rete
- pericolo loop → costruzione spanning tree



Spanning tree

- sottografo aciclico: 1! path per ogni coppia di nodi
- scelta root (e.g. bridge con minore ID)
  - algo per path migliori da root ad ogni bridge
- (ri)costruito ad ogni cambio topologia

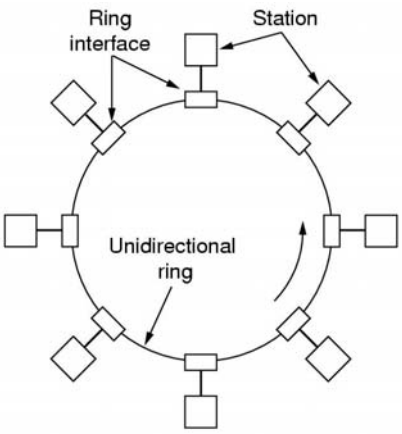


Analisi transparent bridge

- source routing: in header frame per LAN differente, src indica path completo
  - ogni host deve conoscere topologia rete!

caratteristica	transparent	source rtng
servizio	connectionless	conn.oriented
trasparenza a host	totale	*non* trasparente
configurazione	automatica	manuale
routing	subottimo	ottimo
localizzazione	backward learning	discovery frames
fallimenti	gestiti da bridge	gestiti da host
complessità	nei bridge	negli host

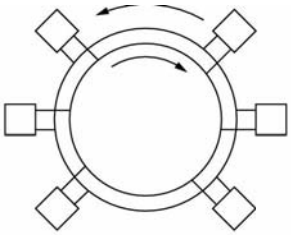
Token ring (802.5)



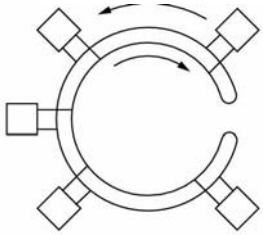
- token circola continuamente
- src lo rimuove da canale a invio frame
- token rigenerato a fine trasmissione
- round−robin per possibilità invio
- efficienza rete prossima a 100%

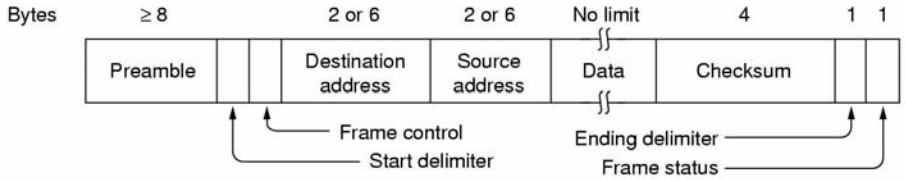
Fiber Distributed Data Intfc

- token ring a 100 Mbps; distanze <= 200 Km; #stazioı <= 1000; high−speed LAN
- stazioni di classe A o B; codifica “4 out of 5”
- circolazione contemporanea di >1 frame con preambolo per synchro clock



fibre multi−mode
src LED





FDDI MAC

- token holding timer: upper bound su durata trasmissione (anti−unfairness)
- token rotation timer: se scade il token è perso
- valid transmission timer: errori transienti

- FDDI usata principalmente come backbone tra LAN

Fast Ethernet (802.3u)

- basato su 10BaseT

tipo	cavo	max. segment	note
100Base−T4	doppino	100 m.	categoria 3
100Base−TX	doppino	100 m.	full−duplex
100Base−F	fibra	2 Km	full−duplex

- 100BaseT: 4 doppini a 25 MHz con codifica 8B6T t.c. mando 4 bit a ogni ciclo → 100 Mbps
- 100BaseTX: 2 doppini a 125 MHz con codifica 4B5B t.c. mando 4 bit ogni 5 cicli
- 100BaseF: 2 fibre multi−modali

Connessione Fast Ethernet

- shared hub:
  - le LAN formano unico dominio collisione
  - usuali meccanismi 802.3
- switched hub ≈ switch
  - tutte le stazioni possono contemporaneamente trasmettere
  - frame bufferizzati nello hub
  - usati per 100BaseF
- supportano anche 10 Mbps LAN

Reti satellitari

- satellite: collezione di transponder che ricevono segnale, amplificano e inviano bcast
  - transp. usano diverse freq per up e down link
  - antenne puntate su area per dwell time
- geosincroni (h~36000Km), 50 Mbps per ogni transponder
  - end−to−end delay ~ 250−300 msec.
  - uso VSAT (Very Small Aperture Terminal)+hub
- orbita bassa: costellazioni (h~750Km)
  - handoff tra satelliti
- utilizzi: broadcasting, mobile, appl. militari

WAN satellitari

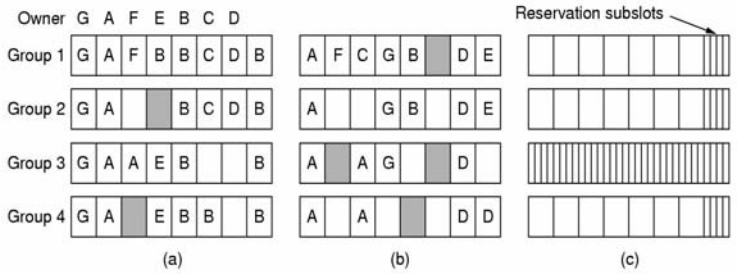
- carrier sense impraticabile: uplink non ascoltabile
- usati Aloha, FDM, TDM, CDMA
- Polling: satellite lento a coordinare →
  - stazioni collegate a rete di scarsa bwth
  - token ring: ogni stazione conosce il suo turno
- FDM:
  - ctrl potenza stazioni per separazione canali
  - SPADE: allocaz.dinamica frequenze
    - TDM signaling channel per prenotazione canale, OK su downlink

WAN satellitari

- Aloha:
  - ground reference station bcast segnale orario, periodicamente, per slotted
  - 2 uplink per aumentare utilizzazione a 73% con buffer ¥ ()
- CDMA: meno efficiente di TDM
- TDM:
  - stazioni devono essere synchro
  - slot assegnati staticamente per poche stazioni, altrimenti dinamicamente
  - politiche statiche (e.g. ACTS centralizzato) richiedono conoscenza sistema per garantire alto utilizzo

WAN satellitari: TDM

- [Binder'75]: #slot >= #stazioni
  - slot di stazioni idle assegnati Aloha, fino a collisione con owner
- [Crowther'73]: slotted Aloha; chi ha successo continua a usare stesso slot
- [Roberts'73]: subslot di ctrl per prenotazione



Servizi di Network Layer

- interfaccia con trasporto t.c. esistenza sottoreti multiple eterogenee e' nascosta
  - indirizzi netw.L. determinati con meccanismo uniforme
- servizi: connectionless o conn.oriented? affidabili o meno?
  - best effort IP
- frammentazione segmenti in pkt
- determinazione cammini e routing
  - datagram o circuito virtuale?
- indipendenza servizio offerto e struttura rete

Datagram vs. virtual circuit

- datagram:
  - ogni pkt instradato indipendentemente
    - possono arrivare in disordine
  - tabelle instradamento: 1 entry per ogni destinazione
    - pkt header: indirizzo destinatario
- virtual circuit
  - tutti i pkt di un circuito seguono stesso path
  - set−up path a priori (signaling protocol)
  - tabella instradamento: 1 entry per ogni circuito
    - pkt header: ID circuito

Datagram vs. virtual circuit

caratteristica	datagram	circuito virtuale
set−up sessione	non necessaria	richiesta
indirizzamento	pkt ha indirizzo dest	pkt ha ID circuito
info stato	subnet non ne ha	subnet ha 1 entry per ogni circuito
routing	ogni pkt e' indipendente	tutti i pkt in sessione usano stesso path
fallimenti router	max. 1 pkt perso	falliscono tutti i VC che attraversano router
controllo congestione	difficile	facile se allocazione buffer ai VC

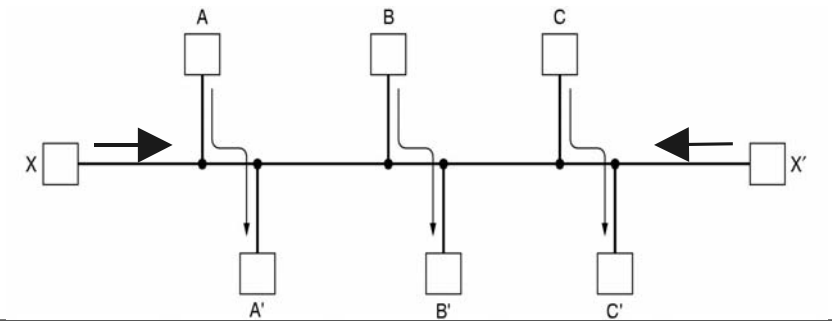
Datagram vs. virtual circuit

- diverse possibili combinazioni di servizi connectionless e connection−oriented con i tipi di sottorete sottostanti

Upper layer	Type of subnet: Datagram	Type of subnet: Virtual circuit
Connectionless	UDP over IP	UDP over IP over ATM
Connection-oriented	TCP over IP	ATM AAL1 over ATM

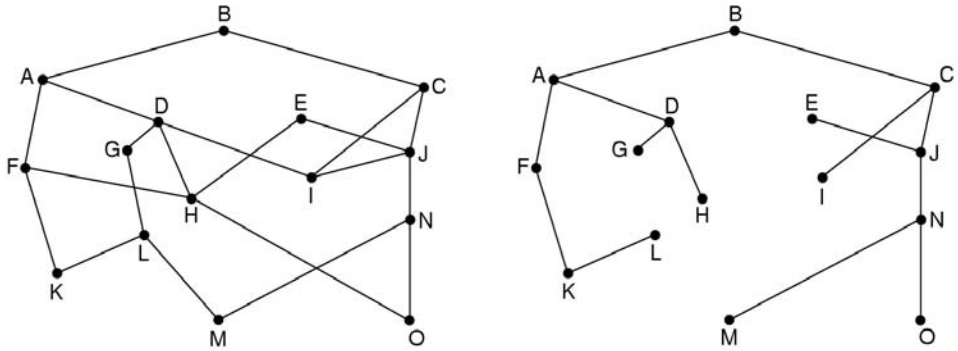
Algoritmi di routing

- correttezza: path fornito deve esistere, no loop
- semplicita' computazione
- robustezza a fronte di cambiamenti topologia
- stabilita' nel fornire path; fairness vs. ottimalita' path
- algo non adattivi (statici) vs. adattivi



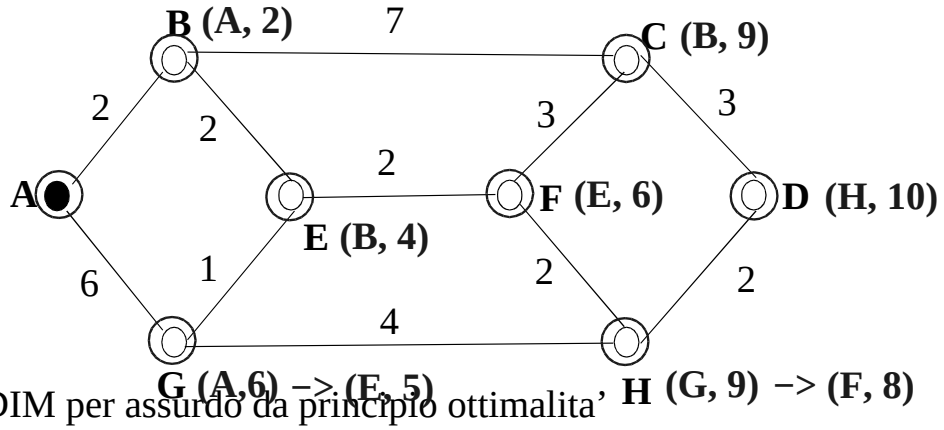
Principio di ottimalita'

- *se il router J e' sul cammino ottimale tra S e D, allora il cammino ottimale tra J e D si sovrappone a quello da S a D*
- sink tree = merge dei cammini ottimali
  - PRBL: scoprire sink tree per ogni destinazione



Shortest path routing (Dijkstra)

- produce path ottimale qualunque sia la metrica



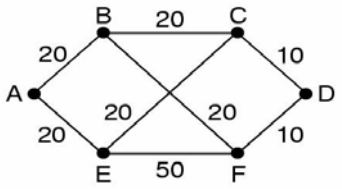
- DIM per assurdo da principio ottimalita'
  - AXYZE ottimale invece di ABE
- PRBL: conoscenza intera topo, costa $O(n^2)$

Flooding

- gia' visto a proposito di transparent bridge
  - prbl terminazione: time−to−live
  - storia pkt gia' inoltrati (pulizia?)
- selective flooding
  - uso linee (oif) che +/− vanno dalla parte giusta
- frequentemente adottato in wireless
- costa in bwth, ma robusto e sicuramente usa path ottimo...

Flow−based routing

- se matrice di traffico *costante* $\{F_{ij}\}$ e capacita' link $\{C_{lk}\}$ sono disponibili a *priori*
- calcolo del ritardo medio pkt usando la teoria delle code $T = 1 / (\mu C - \lambda)$



Source \ Destination	A	B	C	D	E	F
A		9 AB	4 ABC	1 ABFD	7 AE	4 AEF
B	9 BA		8 BC	3 BFD	2 BFE	4 BF
C	4 CBA	8 CB		3 CD	3 CE	2 CEF
D	1 DFBA	3 DFB	3 DC		3 DCE	4 DF
E	7 EA	2 EFB	3 EC	3 ECD		5 EF
F	4 FEA	4 FB	2 FEC	4 FD	5 FE	

λ	14	11	12	13	8	10	6	8
link	AB	AE	BC	EF	EC	BF	CD	FD

Flow−based routing

- mean pkt size=$1/\mu$=800 bit; μ = 0.00125
- $\Sigma \lambda_i$ =82 ; peso = λ_i /82

i	**line**	**λ_i**	**C_i**	**μC_i**	**T_i**	**Weight**
1	AB	14	20	25	91	0.171
2	BC	12	20	25	77	0.146
3	CD	6	10	12.5	154	0.073
4	AE	11	20	25	71	0.134
5	EF	13	50	62.5	20	0.159
6	FD	8	10	12.5	222	0.098
7	BF	10	20	25	67	0.122
8	EC	8	20	25	59	0.098

Flow−based routing

1. costruzione matrice $\{C_{lk}\}$
2. scelta algoritmo routing tentativo
3. costruzione matrice traffico con quell'algo
4. calcolo mean delay sistema
5. ripetere punti 2−4 per diversi algoritmi e scegliere il migliore
6. cablare nei router i path forniti dall'algo scelto

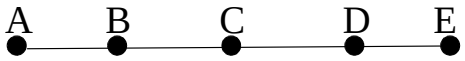
- oppure: algoritmo ottimizzazione
- *perche' sono algoritmi piu' che protocolli?*

Distance vector

- algo Bellman−Ford / Ford−Fulkerson
- RIP = Routing Information Protocol in Arpanet
- SPR e Flow−based forniscono intero path src−dest (source−routing, virtual circuit)
- DV in ogni router, per ogni destinazione, fornisce *miglior* oif e *distanza* verso dest da quella oif
- ogni router prende decisioni instradamento usando solo info locali
- minore o/h memoria, path subottimi, computazione distribuita

Distance vector

- inizio: ogni router conosce solo distanza dai vicini
- periodico scambio routing table



t_0: ogni nodo sa che e' a 1 hop dai suoi vicini

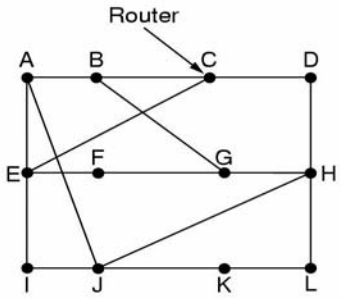
t_0: C) 1 hop B, D
B) 1 hop A, C
D) 1 hop C, E

t_1:
→ B lo dice a C
→ D lo dice a C

A	2	B
B	1	B
C	0	C
D	1	D
E	2	D

a t_1 C sa: 0 hop a C
1 hop a B e D
1+1=2 hop ad A via B
1+1=2 hop a E via D

Distance vector



(a)

To	A	I	H	K	New estimated delay from J	Line
A	0	24	20	21	8	A
B	12	36	31	28	20	A
C	25	18	19	36	28	I
D	40	27	8	24	20	H
E	14	7	30	22	17	I
F	23	20	19	40	30	I
G	18	31	6	31	18	H
H	17	20	0	19	12	H
I	21	0	14	22	10	I
J	9	11	7	10	0	–
K	24	22	22	0	6	K
L	29	33	9	9	15	K
	JA delay is 8	JI delay is 10	JH delay is 12	JK delay is 6	New routing table for J	

Vectors received from J's four neighbors

(b)

Count−to−infinity

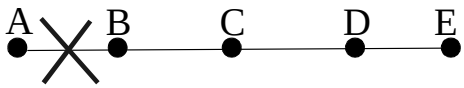
- DV impara rapidamente nuovi path migliori ma reagisce lentamente a problemi
  - un router a distanza K da un nuovo nodo N impara path a N in K scambi

A	B	C	D	E	
	∞	∞	∞	∞	Initially
	1	∞	∞	∞	After 1 exchange
	1	2	∞	∞	After 2 exchanges
	1	2	3	∞	After 3 exchanges
	1	2	3	4	After 4 exchanges

(a)

A	B	C	D	E	
	1	2	3	4	Initially
	3	2	3	4	After 1 exchange
	3	4	3	4	After 2 exchanges
	5	4	5	4	After 3 exchanges
	5	6	5	6	After 4 exchanges
	7	6	7	6	After 5 exchanges
	7	8	7	8	After 6 exchanges
		⋮			
	∞	∞	∞	∞	

Split Horizon



t_0) C:

A	B	C	D	E
B	B	C	D	D
2	1	0	1	2

C−>B

A	B	C	D	E
¥	1	0	1	2

C−>D

A	B	C	D	E
2	1	0	1	¥

D−>C

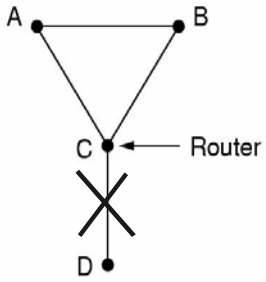
A	B	C	D	E
¥	¥	1	0	1

al passo successivo C ha dist. ¥ da A, ecc...

quando AB cade, B sa di avere dist. ¥ da A e propaga

Split Horizon

- anche noto come poisoned reverse
- fallisce!



A−>C

A	B	C	D
0	1	1	¥

e similmente per B

C−>A
C−>B

A	B	C	D
1	1	0	¥

B−>A

A	B	C	D
1	0	1	2

A−>B

A	B	C	D
0	1	1	2

A conosce

A	B	C	D
0	1	1	3
A	B	C	B

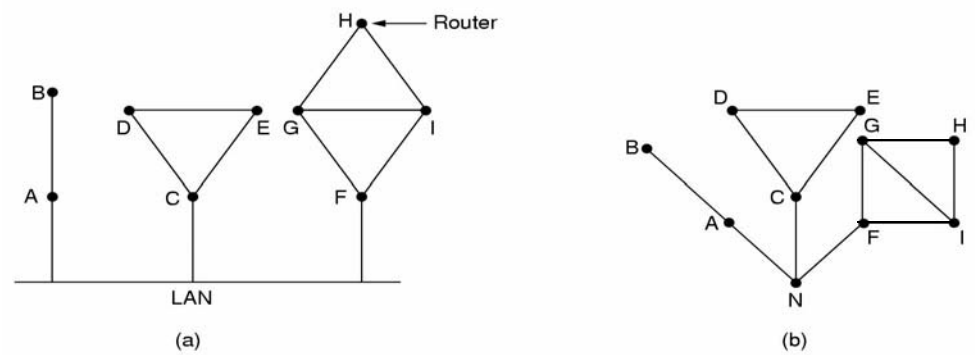
count to ¥!

Link state

- DV ha problema di count−to−infinity
  - puo' considerare una metrica alla volta
  - in funz. metrica, puo' convergere lentamente (es. bwth)
  - puo' trovare path subottimi
- LS trova path ottimi
  - ogni nodo scopre info sui propri vicini
  - ogni nodo dice a tutti queste info
  - ogni nodo impara intera topologia
  - ogni nodo calcola SPR a ogni altro (Dijkstra)

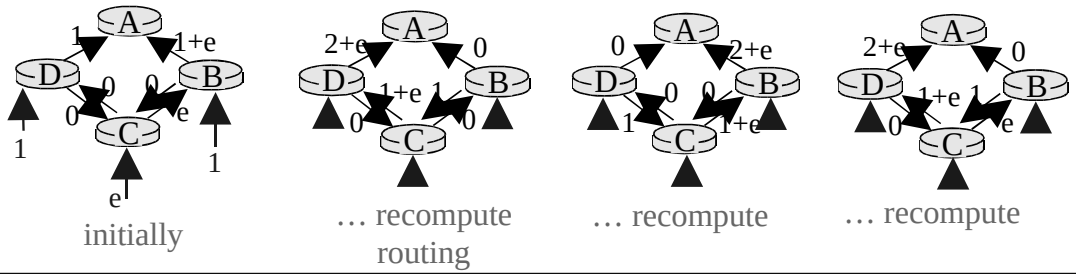
Link state: step 1

- a boot: Hello su ogni linea point−to−point
  - chi riceve, risponde con proprio indirizzo
  - indirizzi globalmente unici, LAN come nodo
  - percio' ogni nodo scopre propri vicini



Link state: step 1

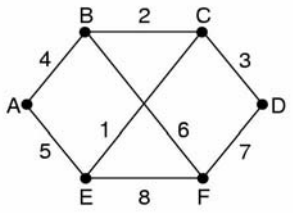
- caratteristiche linee stimate sperimentalmente
  - Echo pkt per ping: delay (round−trip−time)
  - hop, banda...
- stima carico ai nodi
  - considerato puo' portare a path migliori
  - considerato puo' portare a oscillazioni



initially … recompute routing … recompute … recompute

Link state: step 2

- pkt = < src, #seq, age, {neigh_i, delay_i} >
- scambiati periodicamente o quando cambia qualcosa
- scambiati affidabilmente e usati per computare SPR
  (perche' serve affidabilita'?)
- ogni nodo, quando genera/riceve pkt per D, inoltra verso next−hop per D nella sua routing table



(a)

Link State Packets

A	
Seq.	
Age	
B	4
E	5

B	
Seq.	
Age	
A	4
C	2
F	6

C	
Seq.	
Age	
B	2
D	3
E	1

D	
Seq.	
Age	
C	3
F	7

E	
Seq.	
Age	
A	5
C	1
F	8

F	
Seq.	
Age	
B	6
D	7
E	8

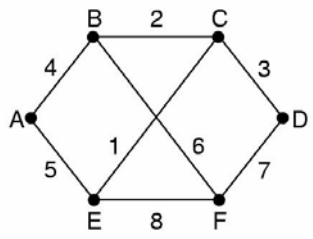
(b)

Link state: step 2

- PRBL: vista topologia temporaneamente inconsistente: loop !
- link state pkt flooded
  - ogni nodo ricorda pkt ricevuti e scarta duplicati e versioni (#seq) obsolete
- #seq di 32 bit evita wrap−around
- age (uint 16 bit) ricordata e decrementata ogni sec
  - per crash router o bit error su #seq
  - quando arriva a 0: reset stato router
  - anche come TTL per i pkt

Link state: step 2

- pkt sono acknowledged
- flag: dove pkt va mandato, dove devo inoltrare ack



Source	Seq.	Age	Send flags A	Send flags C	Send flags F	ACK flags A	ACK flags C	ACK flags F	Data
A	21	60	0	1	1	1	0	0	
F	21	60	1	1	0	0	0	1	
E	21	59	0	1	0	1	0	1	
C	20	60	1	0	1	0	1	0	
D	21	59	1	0	0	0	1	1	

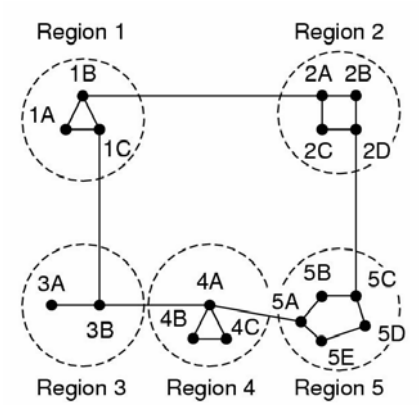
Distance vector vs. Link state

caratteristica	DV	LS
o/h msg	O(n)	$O(n^2)$
o/h memoria	O(n)	$O(n^2)$
o/h computaz.	O(k)	O(kn)
convergenza	count−to−¥	oscilla
path internet	subottimo RIP (intraAS) BGP (interAS)	loop ottimo contemporanei OSPF (intraAS)

n = #router; k = #vicini per ogni router

Instradamento gerarchico

- o/h DV e LS proporzionale a #nodi
- raggruppamenti nodi in regioni. Ogni nodo
  - ricorda esatte info routing per altri nodi in stessa regione
  - ricorda info approssimative per nodi in altre regioni



Dest.	Line	Hops
1A	–	–
1B	1B	1
1C	1C	1
2	1B	2
3	1C	2
4	1C	3
5	1C	4

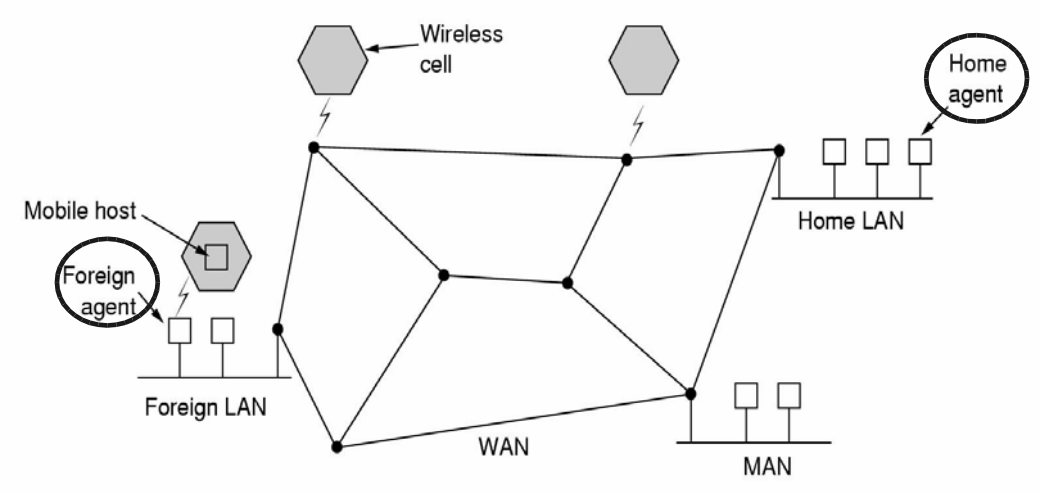
tabella per nodo 1A ha 7 entry invece di 17

Instradamento gerarchico

- indirizzo nodo deve essere gerarchico
  - non so arrivare esattamente a nodo ma so dirigermi verso la zona in cui si trova
- piu' livelli: regioni raggruppate in cluster, raggruppati in zone, raggruppate in gruppi...
- routing eseguito ai diversi livelli gerarchici
- pros & cons:
  - < spazio memorizzazione routing table
  - < computazione, o/h pkt ctrl
  - cammini peggiori ? (indiretti) ES: 1A − 5C
- prestazioni ottimali: (ln N) livelli con (e ln N) entry per router: degrado path accettabile

Instradamento per MH

- difficile per continui cambiamenti topologia

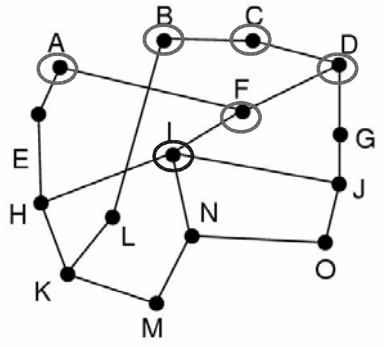


Instradamento per MH

1. foreign agent annunciano periodicamente propria esistenza
2. MH in nuova cella cerca foreign agent e si registra presso di lui
3. foreign agent notifica nuova posizione MH a Home agent, e riceve eventuali info stato (handoff)
4. msg per MH mandati a Home agent, che inoltra a corrente foreign agent (tunneling)
5. src possono essere informate di nuova locazione

Broadcast / Multicast

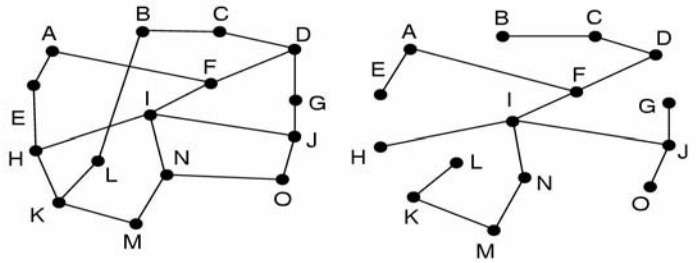
- applicazioni: TV via cavo, news, videoconferenza, lavoro cooperativo
- PROBLEMI:
  - conoscere destinazioni (tutte per bcast)
    - bitmap vs. indirizzo gruppo
  - inviare dati a piu' stazioni in modo efficiente
  - unicast non va bene!
  - flooding?



I src; A B C D F dests

Broadcast routing

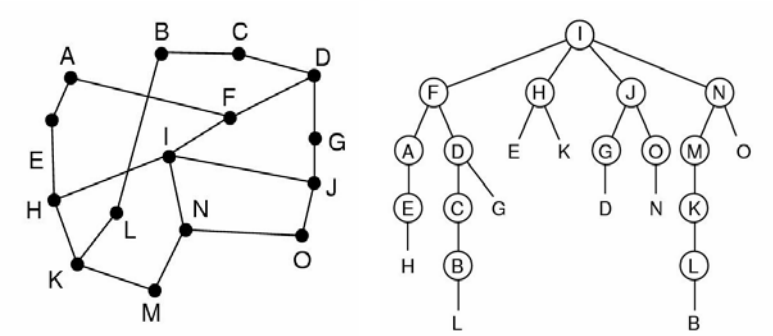
- spanning tree: grafo aciclico che copre tutti i nodi di una rete
- radice = sender; foglie = destinatari (e possibilmente nodi interni)
- usa path ottimi
  - PRBL: come calcolarlo? LS, ma DV non va bene



4 hop,14 pkt per raggiungere tutti i nodi

Reverse Path Forwarding

- se pkt arriva da ifc migliore verso la src, allora inoltro su tutte le ifc, altrimenti scarto pkt
- efficiente e facile da implementare



5 hop, 24 pkt per raggiungere tutti i nodi

Multicast routing

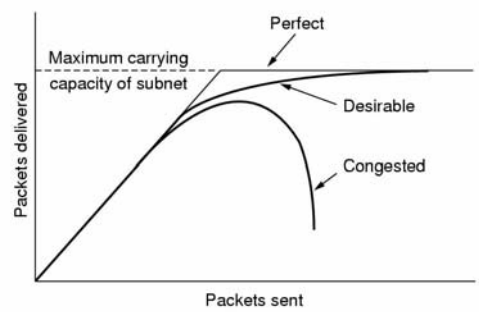
1. faccio broadcast e destinazioni filtrano
   - inefficiente: gruppi sparsi vs. densi
2. unicast inefficiente per gruppi densi

- PRBL: gestione membership gruppo, dinamicamente variabile nel tempo
- strutture ad albero: source−based vs. shared

➢ *approfondimenti nel corso di Protocolli di Gruppo*

Controllo di congestione

- PRBL: capacità link limitata
  - buffer nei router si riempiono
    - crescita delay end−to−end
  - a buffer pieni, pkt dropped
    - ritrasmissioni aumentano carico
    - Nagle: se router hanno memoria ¥ situazione peggiora



Controllo di congestione

- prbl di sottorete: garantire che essa possa gestire traffico offerto
- diverso da flow ctrl: problema *end−to−end*
  - garantire che ricevente possa consumare dati alla velocità a cui sender invia
- congestion ctrl per Quality−of−Service
  - applicazioni richiedono garanzie particolari (bwth...)
  - traffic policing per accounting
  - separazione tra flussi

Principi generali

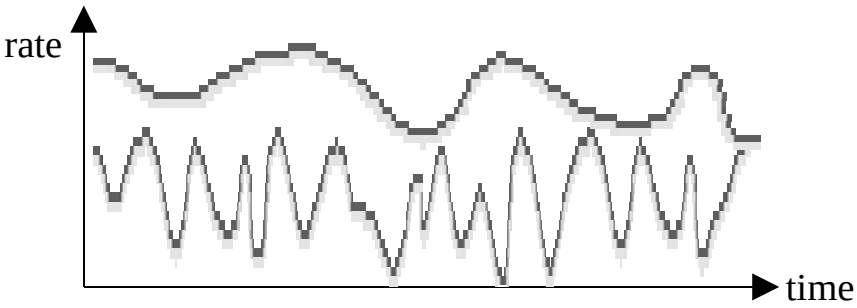
- prevenire problema (open loop)
  - admission control
  - shaping a ingresso rete
  - basso utilizzo rete (?)
- risolvere problema (closed loop)
  - monitoraggio sistema
  - intervento quando si verifica congestione
  - maggiore utilizzo ma possibile aggiustamento in ritardo

Prevenzione congestione

Livello	**Politiche**
trasporto	1. ritrasmissione e acknowledgement 2. out−of−order caching 3. controllo flusso 4. dimensionamento time−out
network	1. datagram vs. circuiti virtuali 2. politiche di accodamento, servizio e drop pkt 3. algoritmi di instradamento 4. gestione TTL pkt
data link	1. ritrasmissione e acknowledgement 2. out−of−order caching 3. controllo flusso

Traffic shaping

- traffico irregolare (bursty) più difficile da controllare
- Variable (e.g. Mpeg) vs. Constant Bit Rate



- customer e carrier si accordano su profilo traffico ammissibile

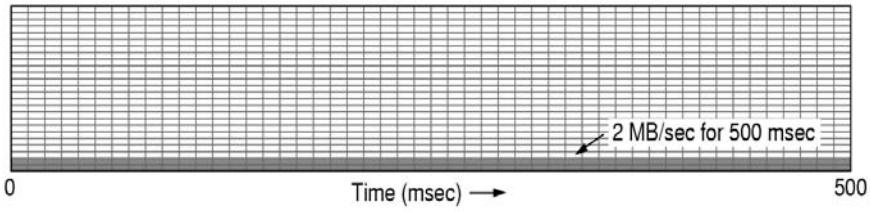
Specifica flussi

- caratteristica input per imformare sistema riguardo a carico generato
- insieme a servizio richiesto: risorse necessarie per gestione traffico
  - admission control

caratteristiche input	tipo servizio
max pkt size (B)	loss interval (microsec) [LI]
rate medio (B/s)	loss [LS], loss rate [LS/LI]
buffer (B)	loss per burst (pkt)
max rate (B/s)	min. delay
	max jitter

Leaky bucket

- 2 processi:
  - producer genera ***r*** token/sec senza credito
  - consumer: quando arriva pkt
    - **if** token disponibile **then** invia pkt e decrementa #token disponibili
    - **else if** esiste spazio in buffer **then** accoda pkt
    - **else** pkt dropped



Leaky bucket

- in generale: token puo' equivalere a pkt (se pkt size costante) o a # byte/bit
  - NB: cio' non significa che frammento pkt!
- es: src produce a 200 Mbps per 40 msec; bwth disponibile a src = 2 MBps = 16 Mbps
  - leaky bucket t.c. r_l = 16 Mbps ; C_l = 8Mb
  - traffico in ingresso al LB:

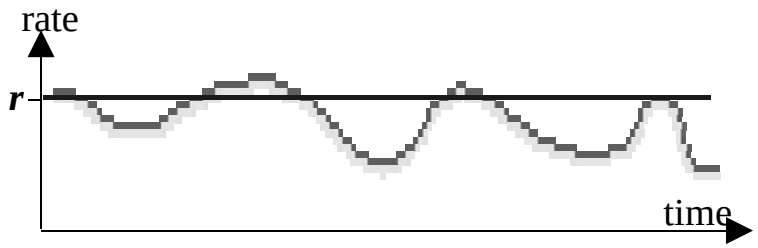
    200 Mbps x 0.040 sec = 8 Mb <= C_l

    nessun dato e' dropped
  - tempo di uscita dal leaky bucket

    T = 8 Mb / 16 Mb = 0.5 sec

Token bucket

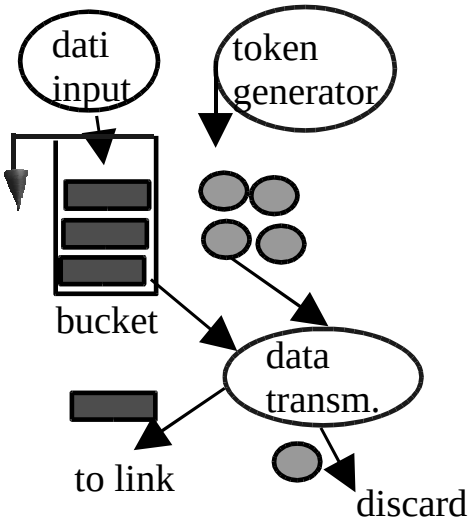
- buffer dati *illimitato* (ideale)
- *bucket contiene token e* mantiene credito
  - *senza credito:* ***r*** tasso generazione token e' ***maximum*** output rate
  - *con credito:* ***r*** tasso generazione token e' ***mean*** output rate



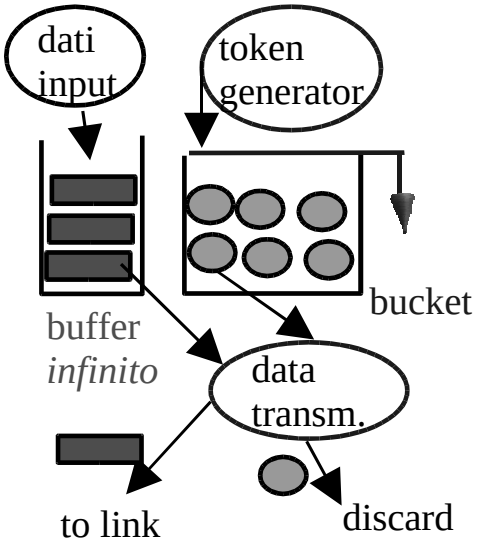
NB: dati sono sempre inviati alla piena bwth del link

leaky vs. token bucket

LEAKY BUCKET



TOKEN BUCKET



token bucket

- bucket di capacita' ***C*** : drop token in eccesso
  - credito consente burst limitato a breve termine, rispettando rate medio su lungo termine
- es: bucket pieno a t_0; C_t = 8 Mb; r_t = 16 Mbps; M_t = 100 Mbps
  - src produce a 200 Mbps per 20 msec

    200 Mbps x 0.02 sec = 4 Mb traffico da smaltire

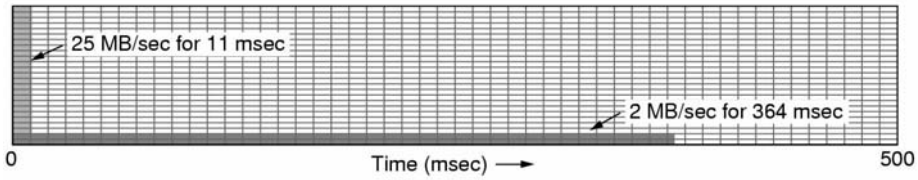
    4 Mb <= C_t percio' invio a peak rate

    durata burst = 4 Mb / 100 Mbps = 40 msec
  - ... credito residuo ?! posso smaltire piu' di 4 Mb

    posso smaltire 4 Mb + (***r*** x 40 msec.)

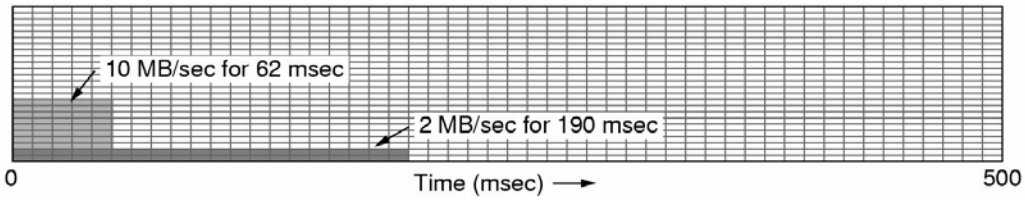
token bucket

- es: src produce a 200 Mbps per 40 msec.; C = 2 Mb ; r = 16 Mbps , M = 200 Mbps; S = durata *massima* burst
- dati inviati in S sono MS = C + (r x S) → S = C/(M − r)
  - es: S = 2 Mb / (200 Mbps − 16 Mbps) = 11 msec
  - MS = 200 Mbps x 0.011 = 2.2 Mb
  - traffico src = 200 Mbps x 0.040 sec = 8 Mb
  - rimangono 8 Mb − 2.2 Mb = 5.8 Mb che invio al rate medio r, smaltendoli in tempo T=5.8/16= 0.36 sec



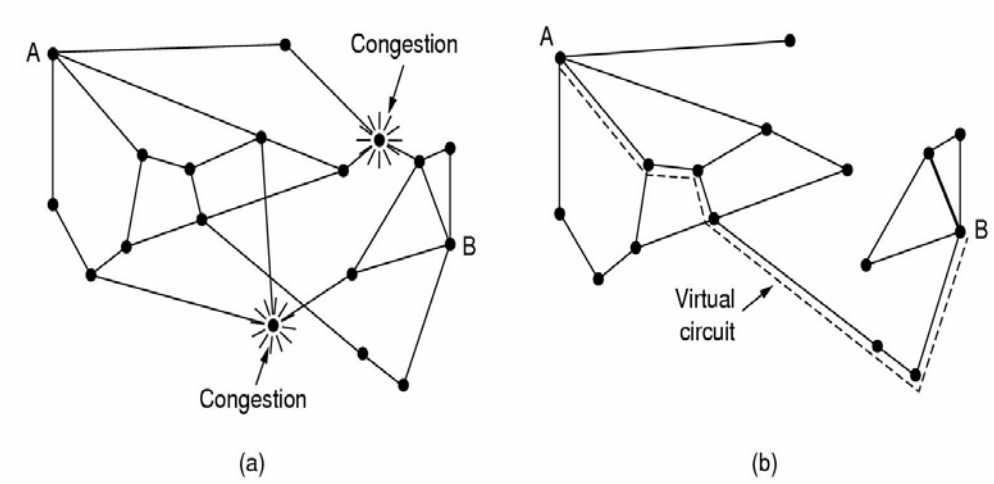
token + leaky bucket

- ha senso se $r_T < r_L < M_T$
  - durata burst e' $S = C_T / (r_L - r_T)$
- **es:** r_L = 80 Mbps; r_T = 16 Mbps; M_T = 200 Mbps; C_T = 4 Mb; dati da sorgente di 8 Mb
  - S = 4/(80−16)=0.0625 sec.
    - in cui smaltisco S x r_L = 80 x 0.0625 = 5 Mb
  - restano 8−5=3 Mb che smaltisco a rate r_T impiegando un



Reti a circuito virtuale

- admission control

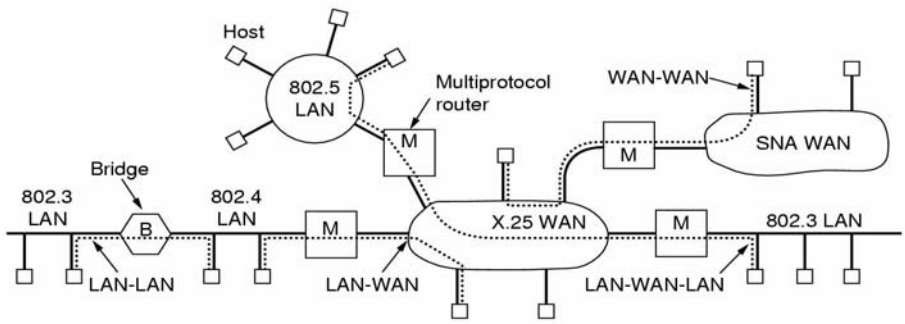


Soluzione congestione

- monitoring: lunghezza buffer, #pkt dropped, e2e delay, #timeout e ritrasmissioni
  - notifica a sorgenti: fairness
- stabilità e efficacia:
  - se reagisco troppo presto, sistema oscilla
  - se reagisco troppo tardi...
- soluzioni:
  - aumentare risorse: traffic/network engineering
  - ridurre il carico

Internetworking

- esistenza reti differenti (TCP/IP, DECnet, Novell, ATM...)
  - ogni rete usa differente tecnologia, percio' differenti protocolli
- connessioni LAN/LAN , LAN/WAN , WAN/WAN, LAN/WAN/LAN

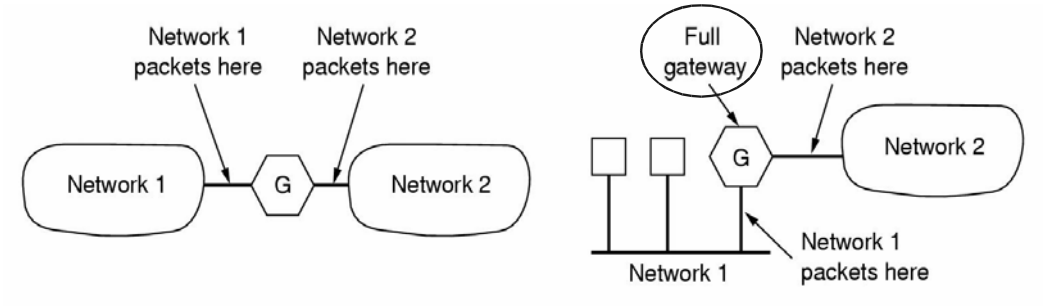


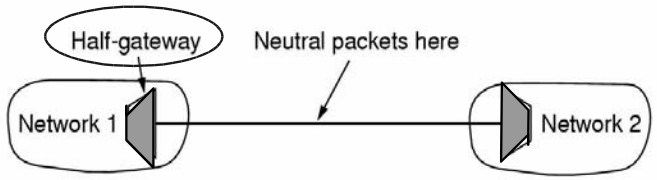
Apparati di rete

- **repeater** (L1): amplificano/rigenerano/replicano segnale [hub]
- **bridge** (L2): store−and−forward; puo' eseguire *traduzione* frame fra reti
- **router** (L3) multiprotocollo: supportano interoperabilita' tra diversi protocolli routing eseguiti in reti differenti
- **Transport/Application gateway** (L4−7): supportano connessione tra reti eterogenee a livelli alti architettura

Gateway

1 host esegue traduzioni tra proto differenti





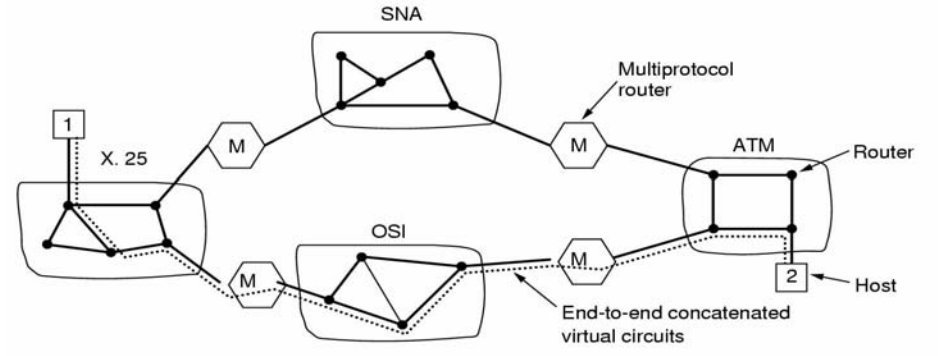
2 host gestiti separatamente, che comunicano via protocollo di interconnessione

Problemi interconnessione

- conversioni alle interfacce

caratteristica	possibilita’
servizio offerto	conn−oriented vs. connectionless
protocolli	IP, AppleTalk, DECnet ...
indirizzamento	flat o gerarchico
multicast/QoS	supportato
pkt size	upper bound variabile
gestione errori	affidabile, consegna ordinata
ctrl flusso/congestione	supportato o meno
sicurezza	cifratura
parametri	timeout, specifiche flussi

Connection−oriented



- costruzione circuiti virtuali concatenati da un router al successivo
  - garantisce arrivo in ordine
- tabelle conversione ID circuiti
- meglio per reti (~) omogenee

Connectionless

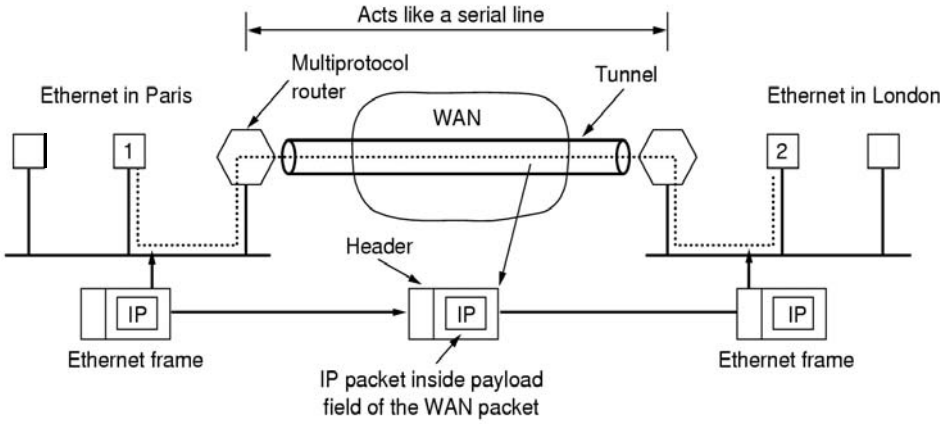
- servizio best−effort, datagram
- pkt diversi possono seguire path differenti
- PROBLEMI: addressing, frammentazione, conversione fra formati pkt
- anche per insiemi di reti eterogenee

Instradamento

- *Autonomous System*: rete (generalmente sotto il controllo di un unico amministratore) connessa ad altri AS attraverso multiprotocol router
- *interior gateway protocol (IGP)*: in AS
- *exterior gateway protocol (EGP)*: tra AS

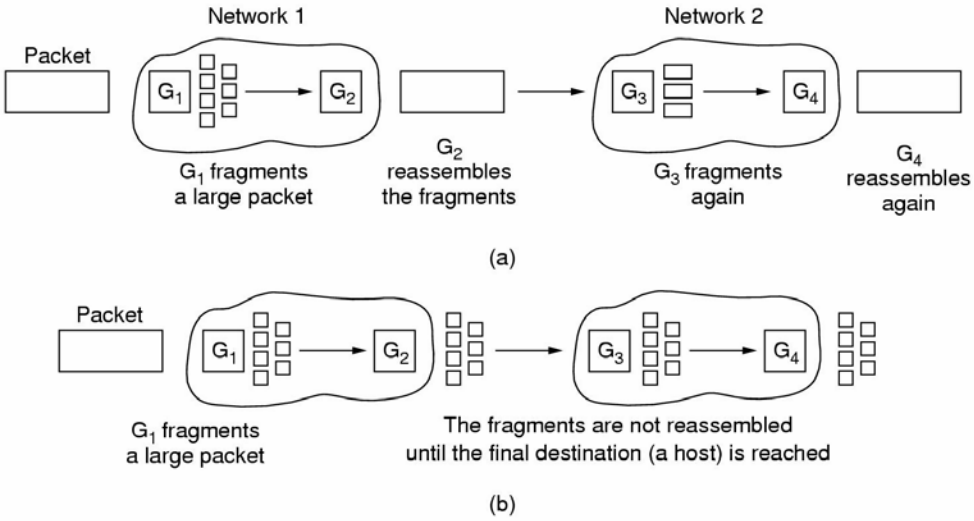
Tunneling

- src/dest fra loro omogenei
- incapsulamento pkt per trasmissione tra multiprotocol router



Frammentazione

- reti diverse hanno diverso upper bound su dimensione pkt
  - frammentazione pkt grandi puo' essere un problema...

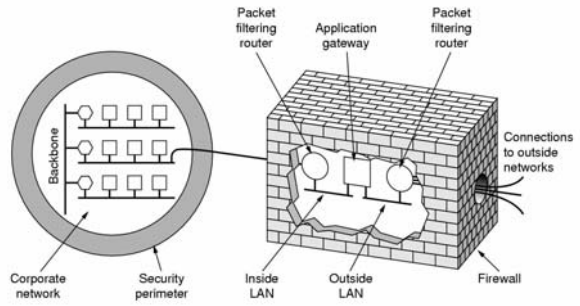


Frammentazione

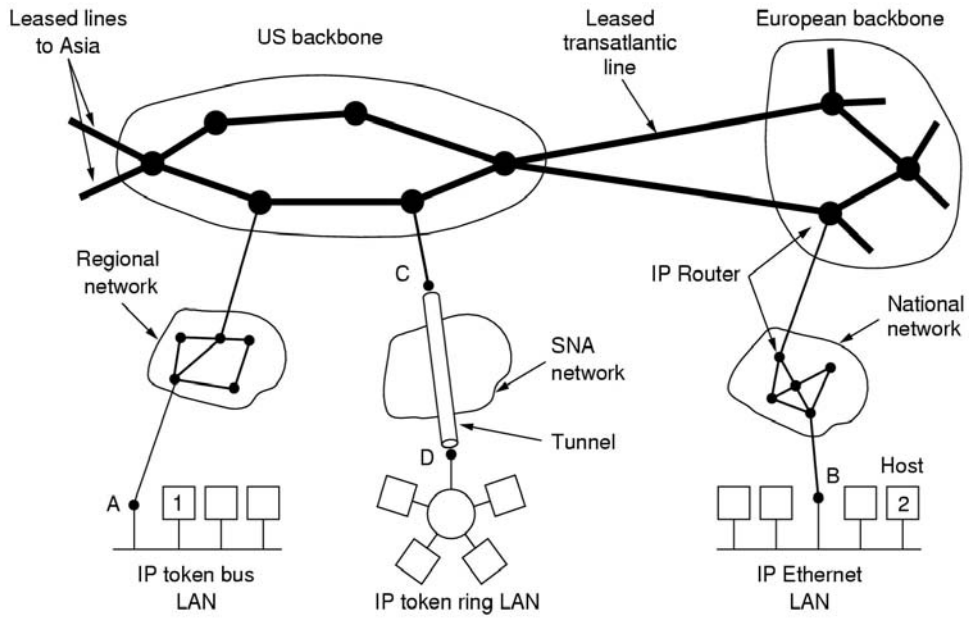
- transparent f.: basse prestazioni per vincoli su path e ripetuta (de)frammentazione
- *non* trasparent: host devono (saper) riassemblare, o/h per header
- numerazione frammenti ad albero
  - prbl se perdo frammento e internetworking connectionless
- accordo su max PDU (MTU) a priori
  - numerazione: < #pkt, #frammento >

Firewall

- domini amministrativi con diverse politiche di sicurezza
- 1! punto di accesso alla rete
- pkt filter ispeziona ogni pkt in transito e decide cosa farne
- application gateway ispeziona ad alto livello

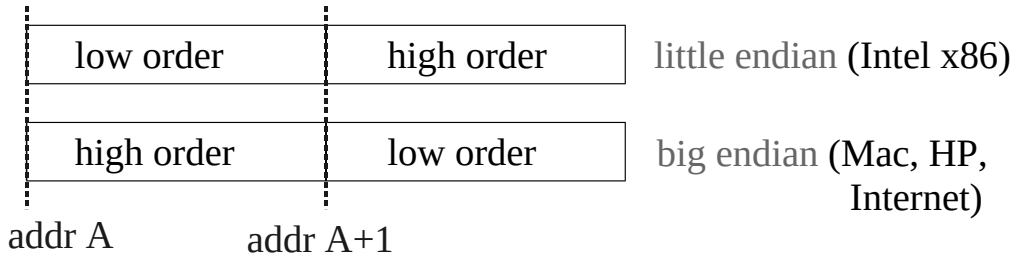


Internet

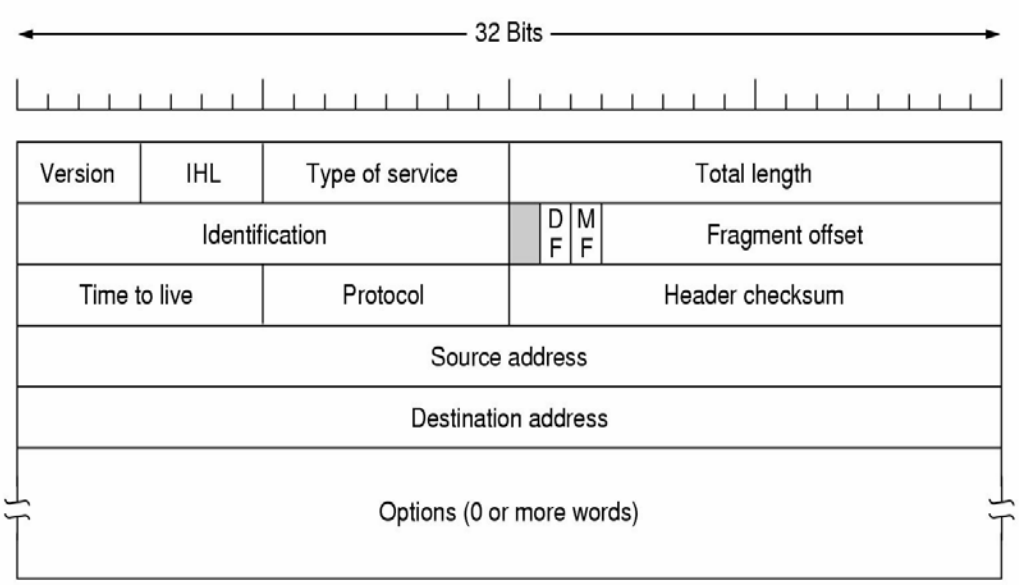


Internet Protocol

- fornisce serv. trasm. datagram su reti interconnesse
- servizio best effort
- datagram fino a 64 KB
  - instradamento attraverso struttura gerarchica
- byte order: non tutti gli host memorizzano i byte all'interno di una parola nello stesso ordine
  - rete: big endian (byte piu' significativo a sinistra)



IP header



IP header

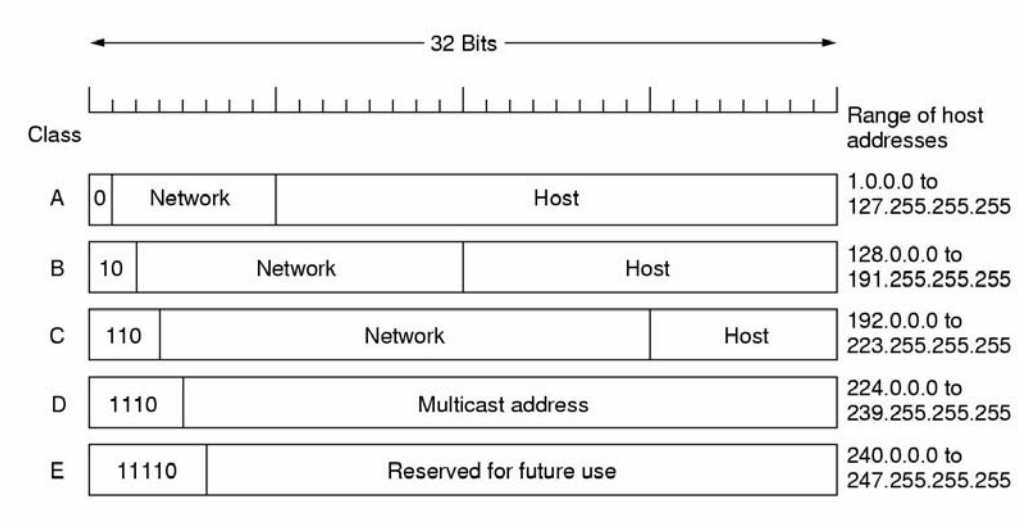
- opzioni; 1B codice + 1B lunghezza + dati

opzione	descrizione
security	specifica quanto e' riservato il datagram
strict source routng	path completo da seguire
loos sorce routng	router da attraversare lungo il path
record route	ogni router aggiunge proprio IP addr
timestamp	ogni router aggiunge proprio IP addr e timestamp

Indirizzamento IP

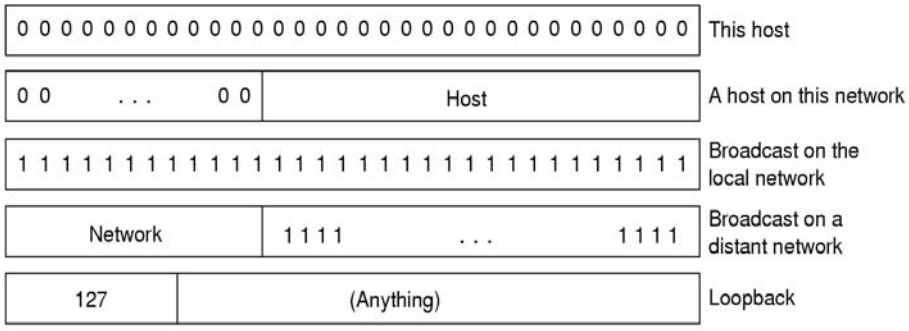
- host ha associato nome simbolico (es. tic.silab.dsi.unimi.it) e indirizzo IP
  - in dotted notation: 159.149.134.9
  - in formato rete: unsigned long di 32 bit
- indirizzo IP contiene network ID + host ID
- 5 classi di indirizzi
  - classe A: 127 reti (7 bit) da $2^{24}-1$ host
  - classe B: $2^{14}-1$ reti da $2^{16}-1$ host
  - classe C: $2^{21}-1$ reti da 255 host
  - classe D per multicast ($2^{28}-1$ gruppi)
  - classe E per usi futuri

Indirizzamento IP



Indirizzamento IP

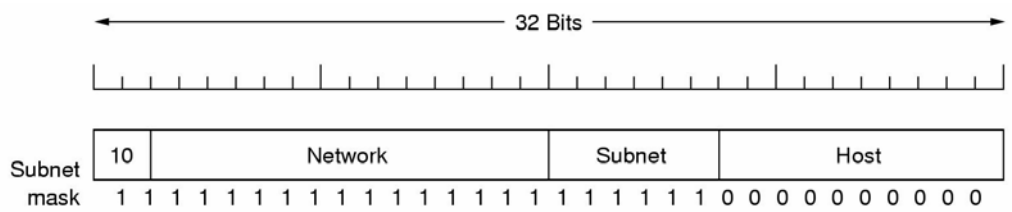
- alcune combinazioni di bit hanno significati particolari



- loopback: non arriva alla rete
  - /etc/hosts contiene sempre indirizzo loopback e indirizzo IP locale

Subnetting

- problema: esaurimento indirizzi IP per sovradimensionamento reti
  - aggiunta reti se sottodimensiono, richiede aggiornamento di tutti i router
- suddivisione logica di reti grandi
  - es: rete classe B (14 bit netw ID, 16 bit host ID)



Subnetting

- subnet mask: tutti bit 1 in corrispondenza di network e subnet addr; tutti bit 0 per host addr
- entry routing table:
  - < network addr, 0 > per host remoti
  - < 0 , host addr > per host locali
  - < 0 , subnet addr , 0 > per host locali ma in altre subnet
  - < 0, 0, host addr > per host in subnet locale
- router calcola AND tra indirizzo IP destinazione in header pkt e subnet mask
  - isola indirizzo rete e confronta con entry sopra

Internet Control Message Proto

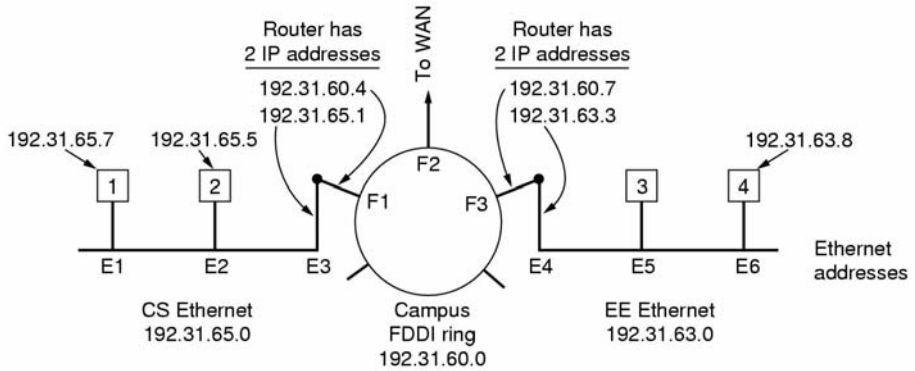
- pkt controllo scambiati tra router per monitoraggio rete e notifica eventi

tipo pkt	**uso**
dest unreachable	pkt non consegnabile
time exceeded	TTL arrivato a 0
parameter prbl	header non valido
source quench	ctrl congestione
redirect	segnala errore rtng
echo req/reply	test attivita' router
timestamp req/rep	simile a echo

ping

Interfaccia Netw.L. − DLL

- pkt arrivati su LAN destinataria inoltrati da MAC
  - DLL non conosce indirizzi IP…
- traduzione: Address Resolution Protocol



ARP

- scoperta matching indirizzi IP / DLL sulla rete locale
- bcast query: “chi ha indirizzo x.y.z.w ?”
  - bcast reply: “io: x.y.z.w = <DLL address>”
- IP src passa a DLL src i pkt e il DLL address dest, da cui costruire frame
  - IP dest verifica correttezza IP address destinazione indicato in pkt header

ARP: ottimizzazioni

- bcast query: altri host sulla rete fanno caching
  - lo stesso per bcast reply
- host a boot invia query con proprio IP addr e DLL addr
  - nessuno risponde, ma tutti ricordano in chache
  - anche rilevazione errori configurazione
- cache ripulita dopo pochi minuti
- prbl: routing tra LAN differenti
  - src cerca DLL addr di dest che sta altrove
    - nessun host su LAN locale risponde

Proxy ARP

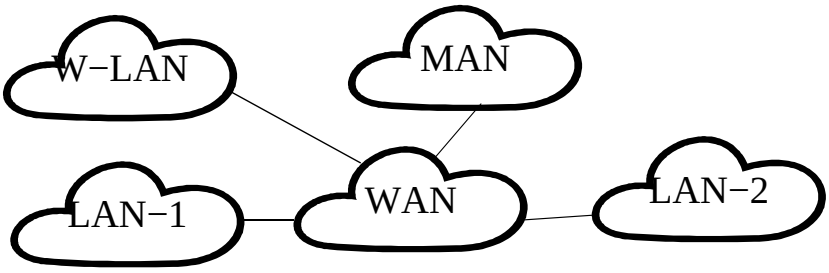
- proxy: entita' che esegue operazioni per conto di altra (e.g. web proxy)
- src nota che IP addr e' in altra LAN
  - manda pkt a default DLL addr: router(s)
  - routers instradano secondo proprie tabelle
- router configurato per rispondere a richieste relative a sottorete a cui porta, con il proprio MAC addr
  - lo stesso per inoltro pkt da router a router

Reverse ARP , BOOTP

- host diskless configurati da remoto
  - scaricano da server i bin del S.O. e la configurazione base
  - *Scalabilita*': server NON mantiene dettagli config di tutti gli host (riuso immagine)
  - host a boot conosce indirizzo server e proprio MAC addr (da NIC)
- host bcast query con proprio MAC addr
  - RARP server risponde con IP addr dell'host
- **BOOTP**: se server (e.g. RARP) non disponibile su ogni rete
  - usa UDP: accede a server su reti remote
  - maggiori info config reperibili

Routing

- determinazione path "migliore" tra src e dest
  - diverse metriche: #hop, ritardo, bwth disponibile...
- servono info riguardo a topologia rete: tutta ?
- Autonomous System: collezione di reti gestite da unico admin, che usano stesso algo routing
  - differenti hw/sw devono interoperare



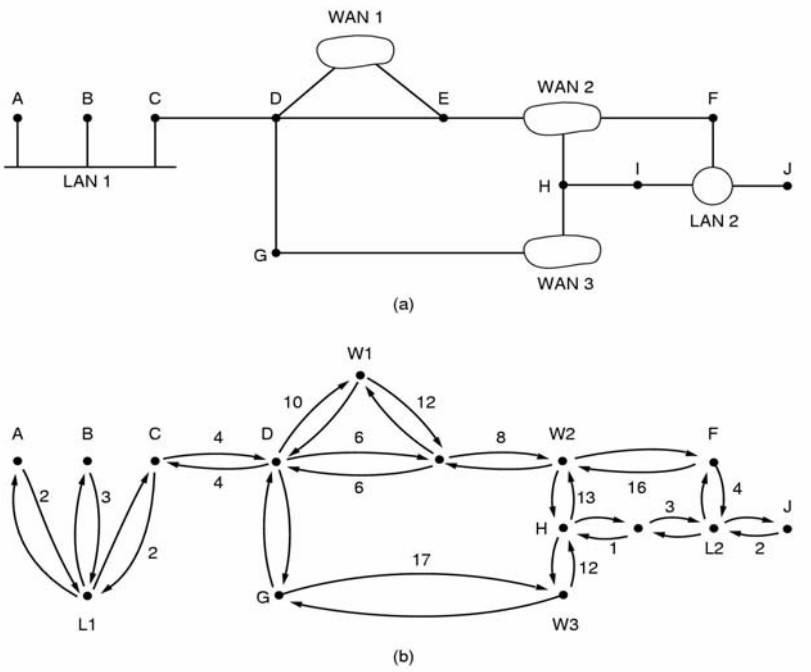
Intra− vs. Inter−dominio

- intra−dominio; piu' facile raccogliere info
  - # limitato apparati, uniso sysadmin
  - piu' info piu' accurate: migliori path
  - interior gw proto: era RIP (Bellman−Ford)
    - problema count−to−infinity
    - ora: link−state (OSPF)
- inter−dominio:
  - alto #device, difficile/costoso mantenere info aggiornate
  - exteriori gw proto: distance−vector (BGP)
- prbl: interfacciamento a confini domini

Open Shortest Path First

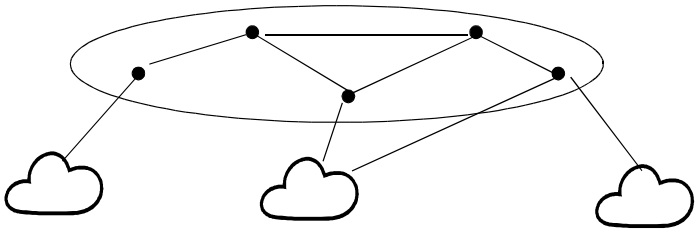
- standard non proprietario (RFC 1247)
- supporta diverse metriche (1 tabella per ognuna)
- instradamento dinamico : adattabilita'
- supporta Quality−of−Service (QoS)
- fa load balancing tra cammini (multipath)
- tre tipi di connessioni
  - linee point−to−point tra 2 router
  - LAN: multiaccess + broadcast
  - WAN: multiaccess + p2p

OSPF



OSPF

- AS rappresentato come grafo pesato
- se AS grande: partizionato in aree non sovrapposte t.c.
  - router interni appartengono a 1! area
  - area border router connettono >=2 aree
  - esiste area backbone t.c. tutte le aree sono connesse ad essa (~ centro stella)
  - AS boundary router comunicano con router in altri AS



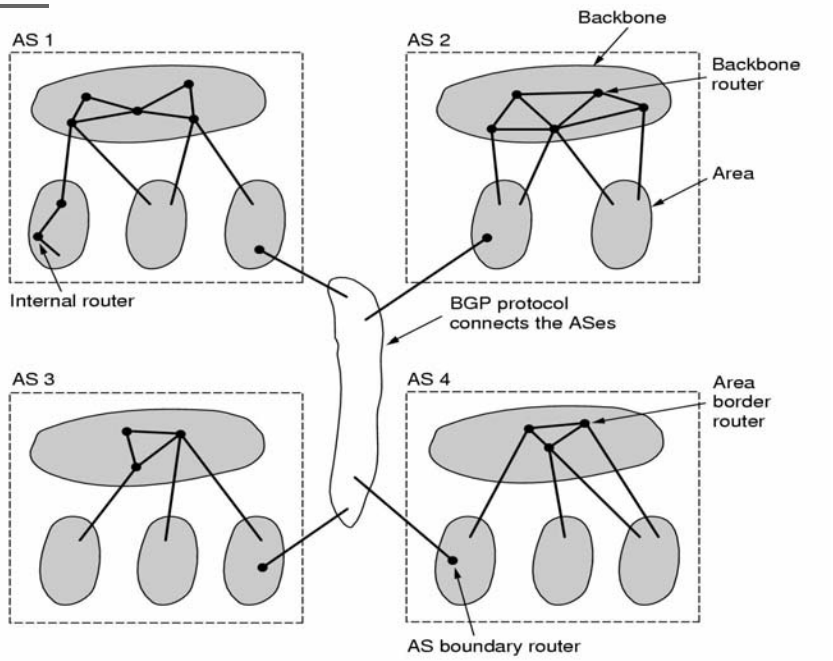
OSPF

- se AS piccolo: path migliore tra ogni coppia di router (su grafo pesato per particolare metrica)
  - NB: ottimizzare "troppe" metriche alla volta e' NP−hard
- se AS gerarchico
  - link state all'interno di ogni area
  - area border router eseguono link state per ogni area che connettono
- routing tra aree
  - DA area src A backbone
  - attraversamento backbone
  - DA backbone A area dest

OSPF: protocollo

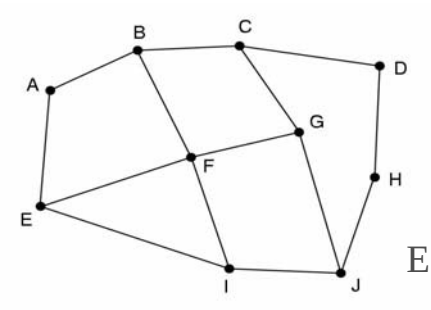
- gruppo ALL_ROUTERS: tutti i rtr connessi alla LAN
- a boot/periodico: HELLO msg per scoperta vicini
- DESIGNATED rtr su LAN per soppressione duplicati
  - adiacente a tutti gli altri attaccati alla LAN
- scambio periodico LINK STATE UPDATE tra adiacenti
  - con #seq, acknowledged per affidabilita' e up−to−date
  - usato per ricalcolo routing table
- DATABASE DESCRIPTION msg: ultimo #seq posseduto
- LINK STATE RQ: per aggiornarsi (e.g. a boot)

OSPF



Border Gateway Proto

- prbl sicurezza: path S−>D deve passare da / evitare determinati nodi
  - configurazione manuale
- basato su distance vector (con problemi relativi)



stub netw: 1 link a grafo
multiconnected netw: non consente attraversamento
transit netw: utilizzabili

ES: crash G verso D

- evita count−to−infinity ricordando path completi

Classless InterDomain Routing

- PRBL: gli indirizzi in IPv4 sono in esaurimento
- amministratori evitano indirizzi di classe C (fino a 256 host) a favore della classe B (fino a 65000 host)
- molte delle esistenti reti di classe B hanno meno di 256 host...
- meccanismo per utilizzo piu' efficiente degli indirizzi e pre controllare dimensioni routing table

CIDR

- aumentare livelli in indirizzamento gerarchico? peggio!
- IDEA; raggruppare classi C in insiemi da usare come spazio contiguo di indirizzi
  - ES: 32 reti C per 32 x 256 = 8192 host per gruppo
- 194.0.0.0 − 195.255.255.255 Europa
- 198.0.0.0 − 199.255.255.255 Nord America
- 200.0.0.0 − 201.255.255.255 Centro−Sud America
- 202.0.0.0 − 203.255.255.255 Asia, Australia

CIDR

- reti di classe C: ultimo ottetto per host ID
- 194.0.0.0 − 194.255.255.255 − 195.0.0.0 − 195.255.255.255

  256 x 256 = 65536 + 65536 = 131072
- sulle 4 regioni 131072 x 4 = 524288 classi C
- per ogni regione 131072 x 256 = 33.554.432 indirizzi di host
- avanzano 204.0.0 − 223.255.255 ovvero 256 x 256 x (223 − 203) = 1.310.720 reti di classe C

CIDR e routing

- se mi capita pkt con indirizzo 194._ o 195._ so che devo inoltrarlo verso l'Europa
  - analogamente per gli altri raggruppamenti
  - 1 entry in routing table per 33M indirizzi
- indirizzamento con maschere nelle routing table
- ES: entry per rete 194.24.8.0 − 194.24.11._

  indirizzo base 11000010.00011000.00001000.00000000

  mask calcolata come 256 − #reti assegnate
  - mask deve far risultare indirizzo di base. nascondendo bit aggiuntivi delle altre reti del gruppo

maschere CIDR

- Cambridge: 194.24.0._ a 194.24.7._ ovvero da 11000010.00011000.00000000._ a 11000010.00011000.00001011._
  - sono 256 x 8 = 2048 indirizzi su 8 reti −> 256−8=248
  - mask: 255.255.248.0 = [1].[1].11111000.[0]
- Oxford: 194.24.16._ a 194.24.31._ ovvero 16 reti per (256 x 16=) 4096 indirizzi, da 11000010.00011000.00010000._ a 11000010.00011000.00011111._
  - mask: 255.255.240.0 = [1].[1].11110000.[0]
- Edinburgo: 194.24.8._ a 194.24.11._ ovvero 4 reti per (256 x 4=) 1024 indirizzi, da 11000010.00011000.00001000._ a 11000010.00011000.00001011._
  - mask: 255.255.252.0 = [1].[1].11111100.[0]

CIDR e routing

- ES: Edinburgo 194.24.8._ a 194.24.11._ (mask 11111111.11111111.11111100.[0])
  - _.8 11000010.00011000.000010 00
  - _.9 11000010.00011000.000010 01
  - _.10 11000010.00011000.000010 10
  - _.11 11000010.00011000.000010 11
- in tutti e 4 i casi lo AND con mask produce 11000010.00011000.00001000 che e' l'indirizzo ricordato nella routing table
- la cardinalita' dei gruppi e' una potenza di 2

CIDR e routing

- calcolo (pkt dest address AND mask)
- confronto risultato con indirizzi base
- indirizzo base che corrisponde e' usato per decidere routing
- NB: stesso principio adattato a tutti gli indirizzi. Percio' si puo' allocare spazio indirizzamento indipendentemente da classi.

Servizi di Transp. L.

- indipendenza da eterogeneita' rete sottostante (e2e)
- connection−oriented vs. connectionless
  - conn.oriented affronta problemi dovuti a natura rete, fallimenti…
    - es: in subnet non esiste ack: msg di Transp.L.
- insieme uniforme di primitive indipendenti da tecnologia sottostante (e.g. Wi−Fi)
- servizi a valore aggiunto (e.g. QoS)
  - uniformi e garantiti indipendentemente da rete
  - scelti da appl/utenti (SLA): negoziazione

Quality−of−Service

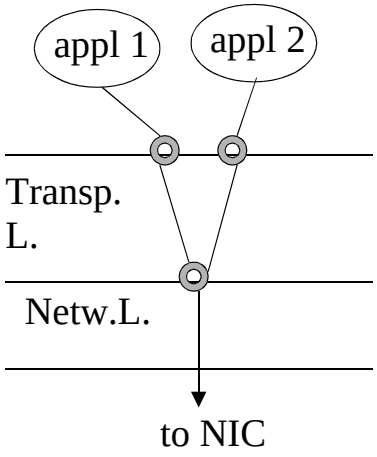
- connection establishment delay: VoIP
- connection establishment failure probability: telefonia
- throughput: file transfer, mmedia
- transit delay: dati real−time (lettura di sensori…)
- residual error ratio: eg. < x% per audio/video
- protezione: cifratura, firma digitale
- priority tra flussi/sessioni diverse
- resilience: resistenza connessione a fallimenti

Funzionalita' Transp.L.

- simile a DLL: ctrl errori, ordinamento, flow ctrl…
- indirizzamento destinazione
- monitoraggio comportamento netw.L.
  - es. per dimensionamento timer ritrasmissione
  - deve tener conto di possibile congestione
- addressing: connessione tra 2 processi, invece che tra 2 host come a netw. L.

Indirizzamento a Transp. L.

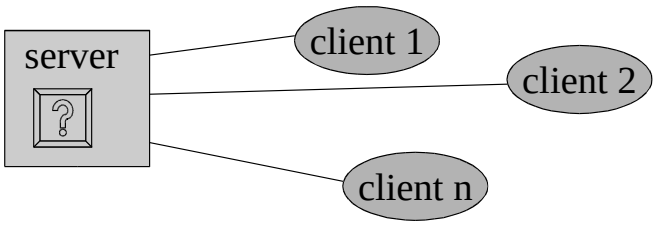
1. distinzione tra diversi processi src/dest di dati su stesso host
2. IP addr solo non basta
3. PID non va bene: remoto?
4. **port**: ID unico *legato* a processo
   - in TCP/IP: unsigned short (16 bit)
   - well−known per servizi standard



Addressing in Unix

- nslookup: per trovare IP address
  - /etc/hosts per indirizzi locale e piu' usati
  - altrimenti interroga name server
- gethostbyname(), gethostbyaddr()
  - per trovare IP address in programma C
- getservbyname() per trovare port server standard
  - /etc/services in accordo a standard (RFC1700)

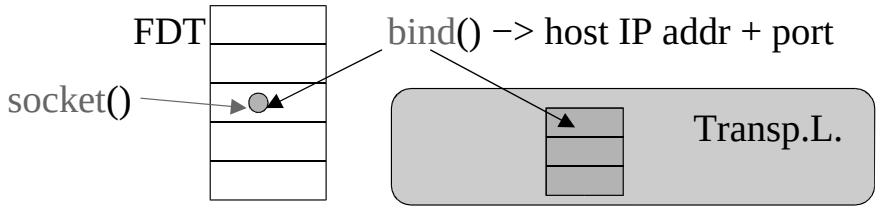
Approccio client−server



- server: fornitore di servizi (e.g. DB, web) o gestore risorse condivise (e.g. printer)
- riceve request da client, esegue operazioni richieste e (eventualmente) invia reply
  - resilience, sicurezza, potenza…
- process server (per servizi non standard o poco usati, inetd)

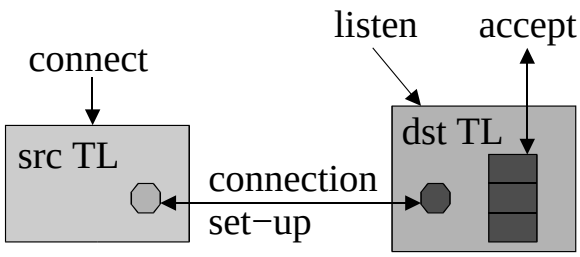
Primitive Transp. L.

- *dove operano? tra quali livelli?*
- differenti a seconda del servizio richiesto
  - diversa visibilita' sistema fornita all'utente
  - diversa complessita' proto di livello superiore
- inizializzazione locale /* libreria C */
  - socket(): creazione end−point
  - bind() a indirizzo noto all'esterno



Primitive Transp.L.

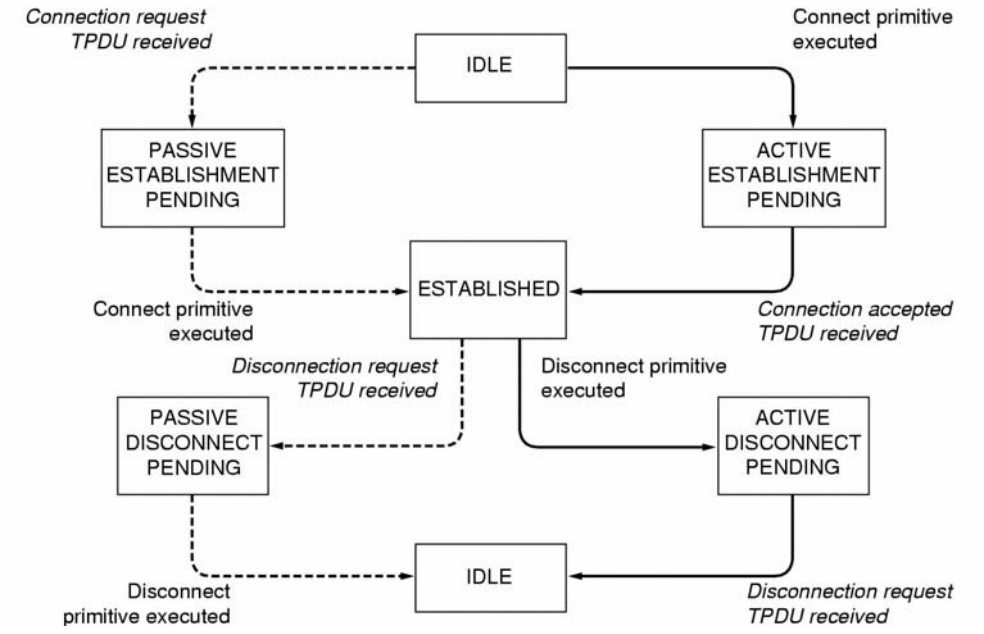
- associazione <proto, IPsrc, port−src, IPdst, port−dst>
- connection set−up [conn.oriented]
  - listen
  - connect
  - accept



Primitive Transp.L.

- sendto(), receivefrom() per scambio TPDU (transp. proto data unit)
- chiedono fornitura servizio a entita' trasp.
- proto appl. per stabilire: di chi e' turno di parola, formato dati contenuti in TPDU, tipo msg di livello applicativo…
- close() per chiusura connessione
  - asimmetrica
  - simmetrica: negoziazione

Primitive Transp.L.

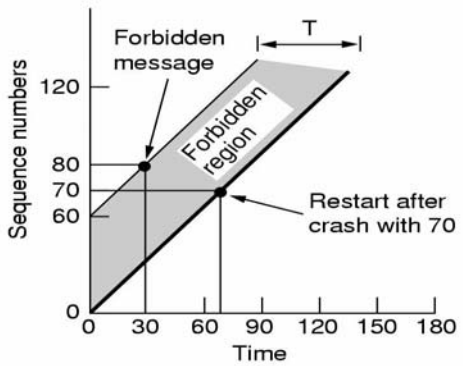


Delayed duplicate

- rete fa buffering: dati possono riemergere dopo tempo impredicibile
- prbl: legare dati a connessione per rilevare obsoleti
- one−time transp. address: come gestire well known server?
- connection ID: stato mantenuto quanto a lungo?
  - problema scelta ID univoco
- temporizzazione: msg restano in giro per tempo limitato
  - src genera msg con uguale ID dopo che tutte le info sul precedente (e.g. ack) sono scomparse dalla rete
  - ctrl path, hop count, timestamp in msg

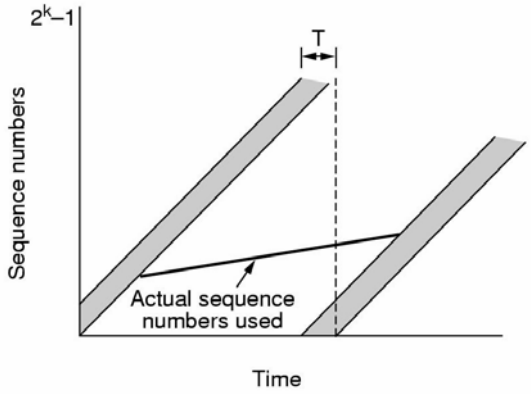
Metodo di Tomlinson

- src sceglie #seq iniziale come k bit meno significativi del proprio clock (k t.c. no problemi wrap around)
- ipotesi: T upper bound su tempo vita msg
  - clock synchro funzionanti anche in caso di crash
- crash host:
  - sender reste idle per T (inefficiente)
  - sender attivo ma non genera #seq in forbidden region



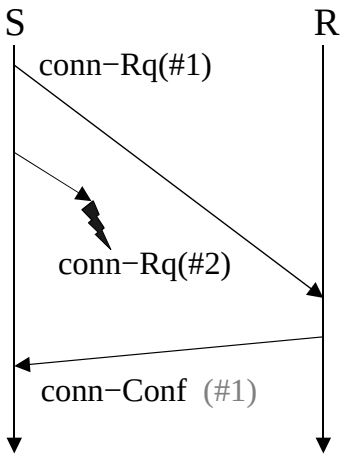
Metodo di Tomlinson

- max data rate = 1 TPDU per clock tick
- ingresso in forbidden region se data rate basso
  - resynch #seq
- serve protocollo agreement #seq tra sender e receiver
- serve global clock



Connection set−up

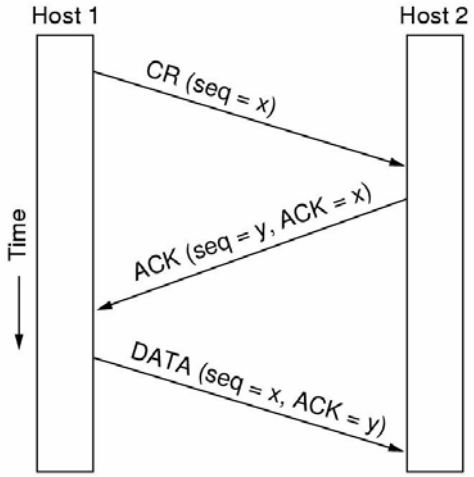
- sender e receiver usano uguale #seq per traffico bidirezionale
  - sender invia connect−Rq(#seq), confirmed



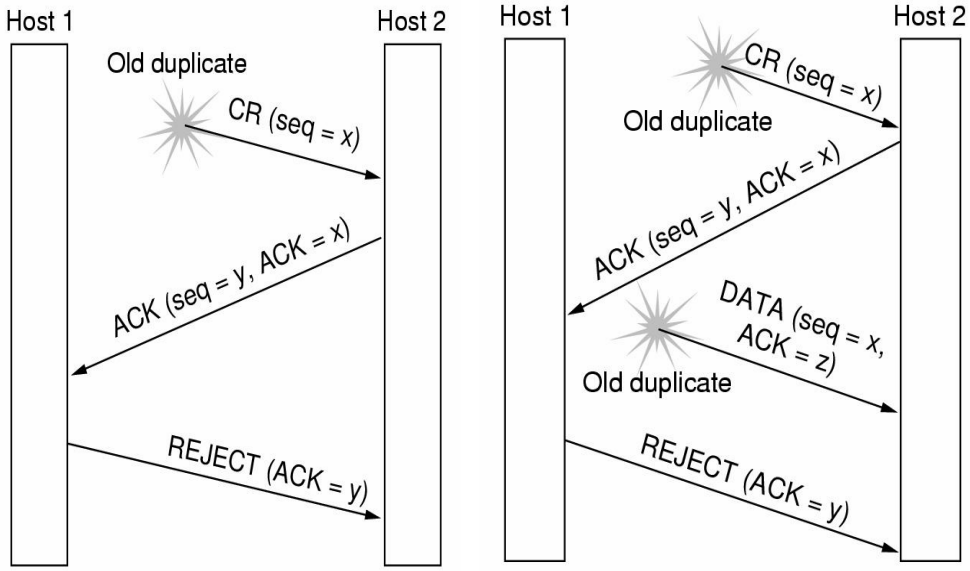
- se 1° invio in ritardo e 2° perso, dest chiude connessione con 1ª Req, ma sender interpreta erroneamente e le due parti non sono in accordo

Three−way−handshake

- sender e recv usano #seq diversi per flussi traffico nelle 2 direzioni
- ogni processo fa riferimento a proprio clock
- funziona sempre correttamente anche in presenza di delayed duplicate di ctrl TPDU



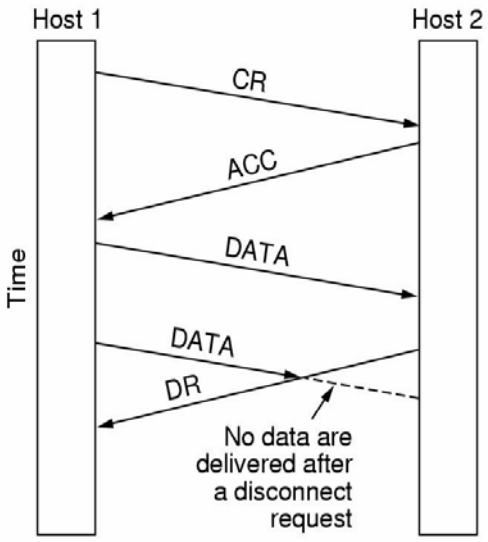
Three−way−handshake



duplicate conn−Rq duplicate conn−Rq e ack

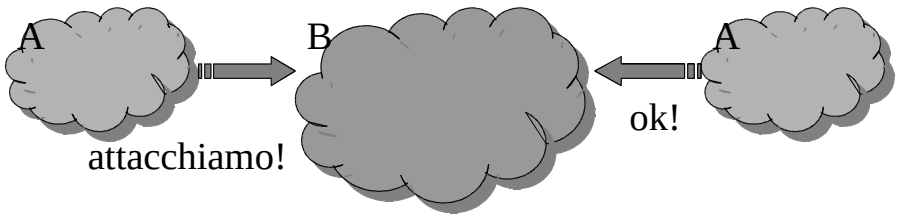
Rilascio connessione

- cancella tutte le info sulla connessione
- rilascio asimmetrico puo' provocare perdita di eventuali dati ancora in transito al momento della chiusura
- agreement sulla disconnessione?



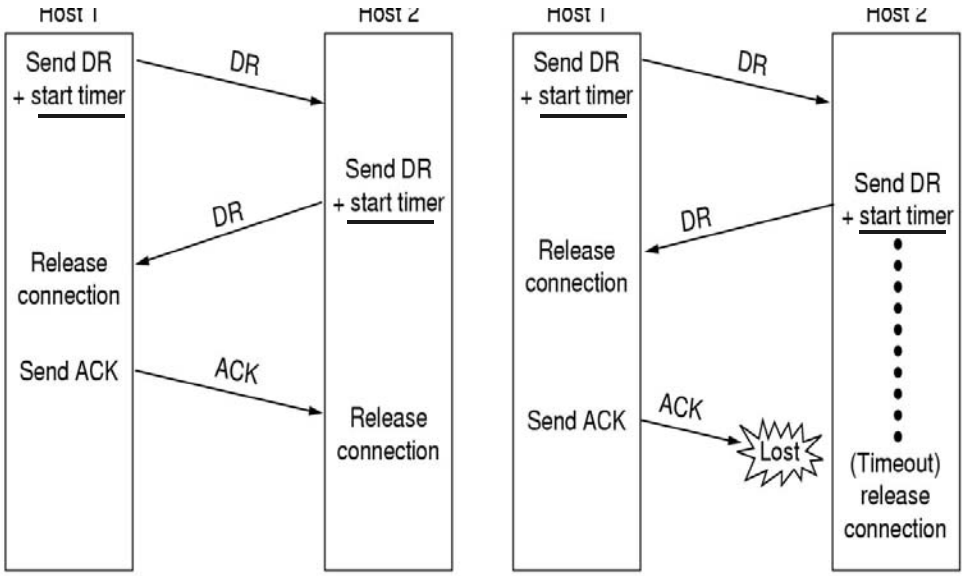
Problema dei due eserciti

- rilascio simmetrico: chiusura connessione deve essere fatta in entrambe le direzioni
- potrebbe non andare mai a buon fine…



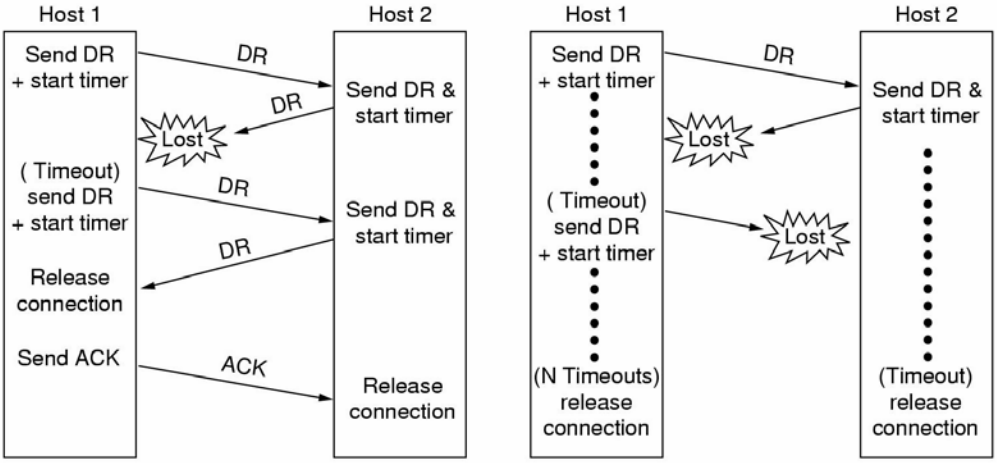
- improbabile per infinito #msg scambiati
  - in pratica: 3−way handshake + timer

Rilascio connessione



normale perdita ack

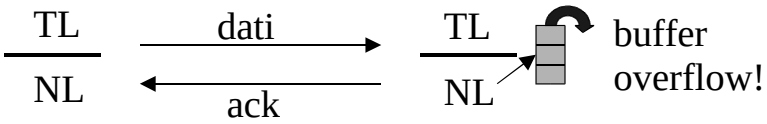
Rilascio connessione



- possibilita' conness. parzialmente aperta se perdo DR iniziale
- rilascio se conness. inattiva per T (keepalive TPDU)

Controllo di flusso

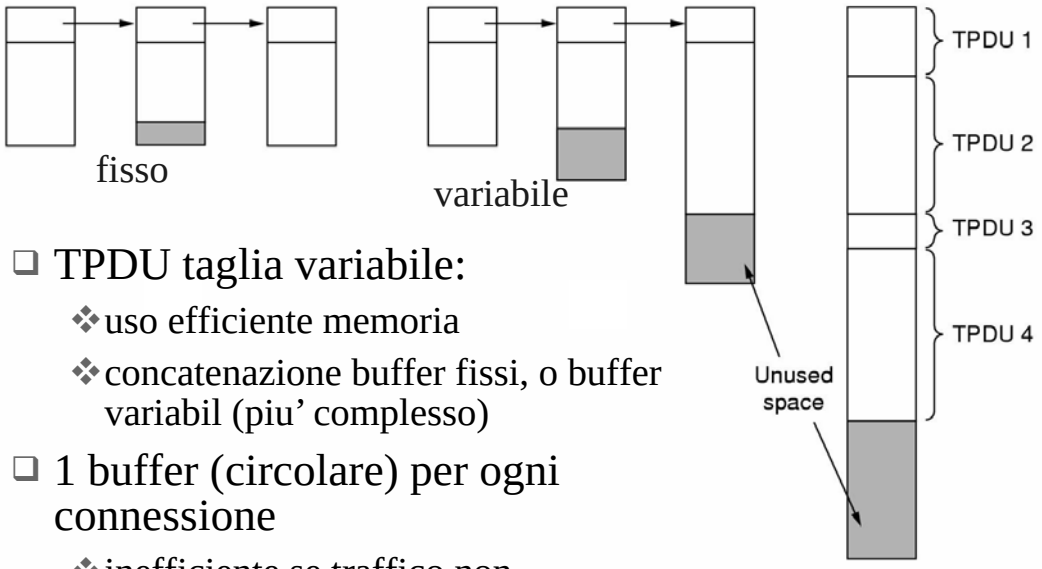
- in DLL entita' controlla poche linee
  - in Transp.L. possono esistere molte connessioni aperte
- Transp.L. puo' richiedere buffer grandi
- a receiver, spazio memoria puo' essere condiviso tra varie connessioni
  - allocazione dinamica spazio
- a sender: rete che fornisce servizio acknowledged non da' alcun aiuto…



Dimensionamento buffer

- DLL go−back−n
  - variable size a lato sender, =1 lato receiver
- DLL selective−repeat
  - lato sender = lato receiver
    - *perche' ne' piu' piccolo ne' piu' grande?*
- TRANSP.L. : dipende da taglia TPDU
  - puo' essere differente per diverse connessioni
  - se taglia fissa: buffer come concatenazione di elementi di uguale dimensione

Dimensione buffer



- TPDU taglia variabile:
  - uso efficiente memoria
  - concatenazione buffer fissi, o buffer variabil (piu' complesso)
- 1 buffer (circolare) per ogni connessione
  - inefficiente se traffico non omogeneo tra connessioni

Gestione buffer

- basso carico offerto: recvr puo' non bufferizzare (*perche'?*)
  - deve farlo per alto carico
- allocazione dinamica memoria a receiver
  - se non disponibile, sender deve essere stopped
  - ri−negoziazione dinamica window size
- problemi:
  - per efficienza: sender deve essere in grado di aumentare quando possibile
  - feedback da receiver (in ack)

Gestione buffer

#seq da 4 bit

	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 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

Ctrl congestione e buffering

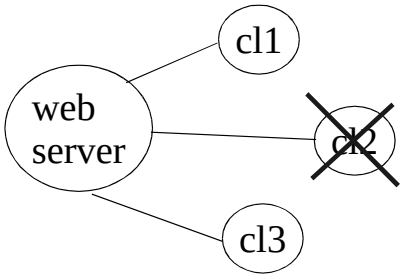
- finestra sender deve limitare traffico iniettato in rete
- stima condizioni della rete
  - ritardo round−trip time r
  - max throughput dalla rete (TPDU/sec) t
  - finestra sender $t \cdot r$: linea piena

Multiplexing

- upward multiplexing: aggregazione di flussi di dati generati da diversi processi su un host, sulla stessa NIC
- accounting delle risorse di rete per un tempo d'uso invece che per quantita' d'uso
- downward multiplexing per load balancing
  - offerta > throughput aggregato sfruttando parallelamente diverse connessioni (di netw. L.)
- ES: #seq di n=8 bit; rtt=540 msec (satellite); pkt size = 128B
  - con go−back−n thput=(2^8-1) x 128 x 8=485 Kbps

Resistenza ai fallimenti

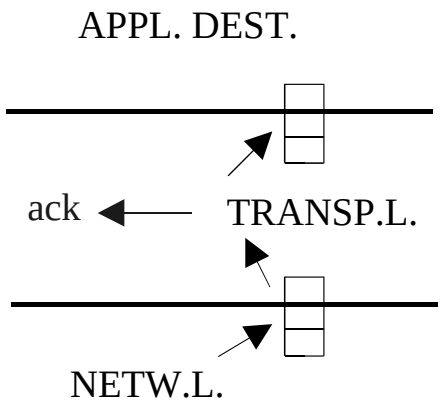
- transp.L. puo' dover essere in grado di superare fallimenti di rete e di host
  - rete: puo' provvedere (in parte) netw.L.
  - host: funzionamento di processi su un host deve essere indipendente da quanto accade su altri host
- ES:



- crash cliente 2
- server deve continuare a operare per altri client
- piu' difficile il contrario...

Resistenza ai fallimenti

- server crash e fa reboot: chiede (bcast) stato ai client
  - S0: client non ha dati unacknowledged
  - S1: client ha inviato dato e non ha ack
- client ritrasmette o meno?
  - S1 e ritrasmette: server potrebbe *consegnare* duplicato
  - S0 e non ritrasmette: dato acknowledged ma non *consegnato*: perso!



Resistenza ai fallimenti

- azioni recvr: A: invio ack; W: consegna msg; C: crash
  - end−to−end argument: recovery in alto dove e' nota semantica

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

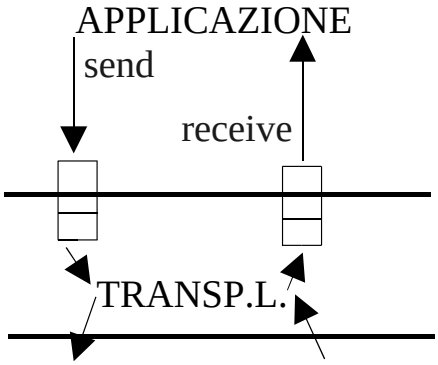
Internet Transport Protocols

- livello rete (IP) solo connectionless
- User Datagram Proto: connectionless
  - aggiunge a IP alcune info controllo, ctrl errore...
  - scambio msg
- Transmission Control Proto: connection−oriented
  - servizio ordinato e affidabile
  - controllo di flusso e di congestione
  - scambio byte stream

Indirizzamento TCP/UDP

- accesso a servizio (TSAP): socket
  - punti terminali del canale di comunicazione
  - insieme di primitive che permettono di accedere ai servizi di livello trasporto (e rete ~)
- indirizzi: IP address + port
  - per UDP, IP address puo' essere anche multicast o broadcast, per TCP no
- port 0−255: well−known per servizi standard (RFC 1700)
- port 256−1023 per servizi in test
- port 1024−5000 per uso sistema
- port > 5000 per protocolli utente

Interfacce tra livelli



- send: si blocca se non esiste spazio in buffer interfaccia
- receive: si blocca se non esistono dati in buffer
- send: dati possono essere bufferizzati
  - prbl in caso di fallimenti, > efficienza w.r.t. o/h header
  - push flag: dati inviati senza attesa in buffer
  - out−of−band (urgent) data: trasmissione di tutto quello che c'e' per la connessione, e consegna con trap a dest

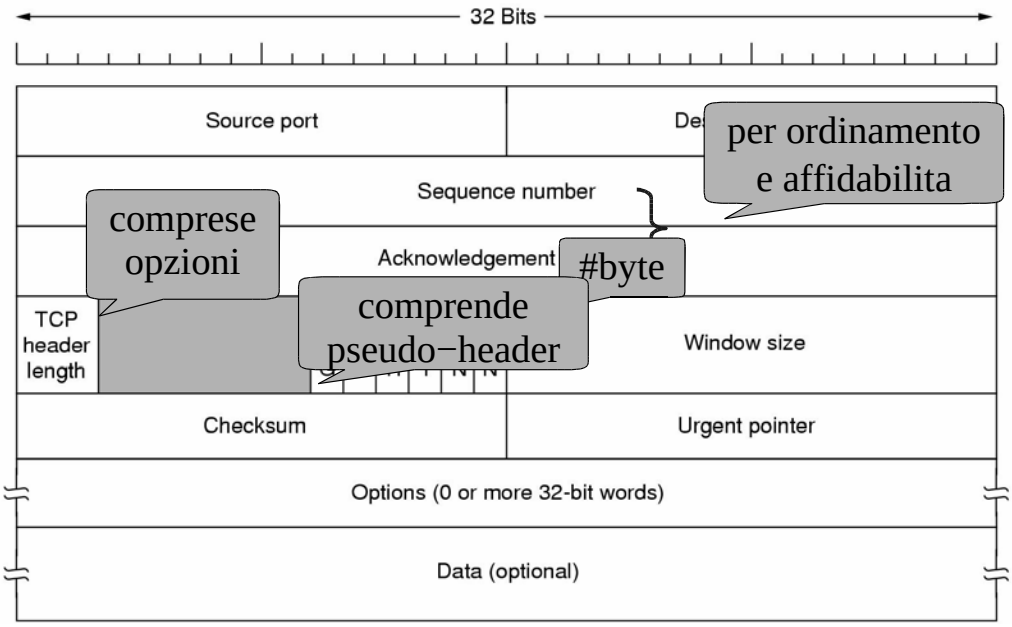
TCP service model

- byte stream spezzato in IP datagram e inoltrato a IP insieme a TCP header
- a dest: riassemblaggio datagram e consegna byte stream
- NON preservato confine msg
  - 2 blocchi da 512B possono essere consegnati come tali, come 1 blocco da 1024B, come 1024 blocchi da 1B...
  - dipende da protocollo di livello superiore
  - non posso trasmettere altri tipi di dati (int, float...) *Perche'*?

Protocollo TCP

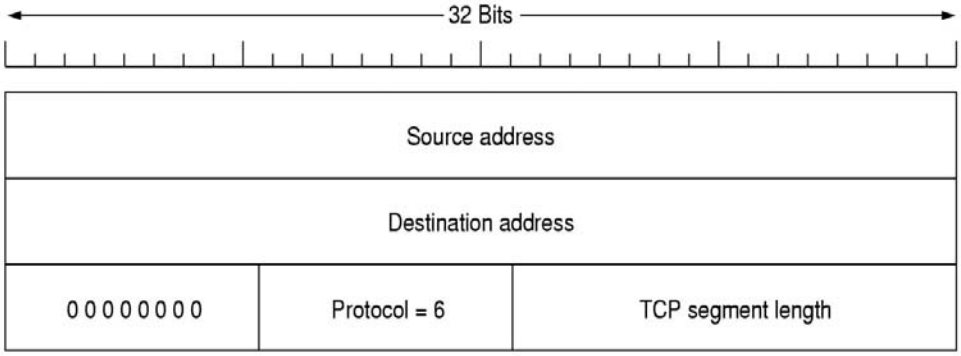
- ogni byte puo' avere #seq
  - #seq di 32 bit per ordine e affidabilita'
- segmento: 20B header + dati
  - un segmento deve stare in (64 KB) IP datagram
  - se definita MTU: segment size <= MTU
- sliding window mechanism
  - timer (~ e2e rtt...) + ack (next #seq atteso)
  - frammentazione segment crea problemi per ack
- TCP e' famiglia (ampia!) di protocolli

Struttura segmenti TCP



Pseudo−header TCP

- pseudo perche' contiene informazioni di livello inferiore, che non dovrebbero essere viste a livello Transp.



Opzioni TCP

- indicazione max paylod accettabile (TPDU)
  - segmento grande ammortizza info ctrl header, ma non tutti possono essere in grado di accettarlo
  - tutte devono poter accettare 536B
- possibilita' di ampliare la finestra (da negoziare!)
- scelta tra selective−repeat e go−back−n
- possibilita' di usare negative ack

TCP: creazione connessione

- meccanismo 3−way handshake visto in precedenza
  - server in attesa su accept()
  - client esegue connect()
  - sfruttati alcuni flag presenti nello header TCP:

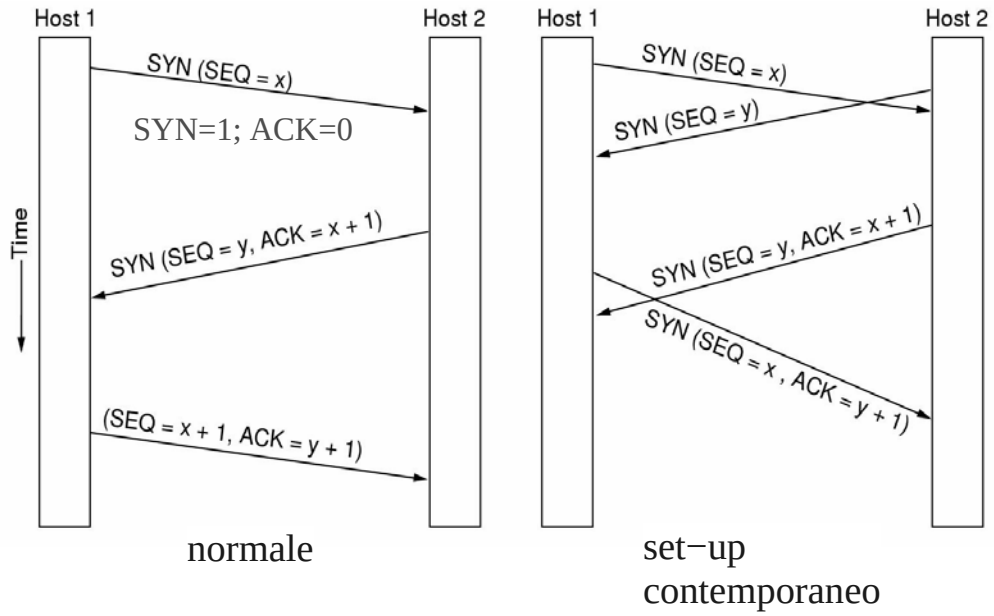
SYN =1 : il segmento (SYN segment) e' usato per creare nuova connessione

ACK =1 : indica che il valore di ack contenuto nel segmento e' valido

RST =1 : indica connessione rifiutata (es.: perche' e' sbagliato l'indirizzo [port] del server)

FIN =1 : chiusura connessione (simmetrica)

TCP: creazione connessione



normale

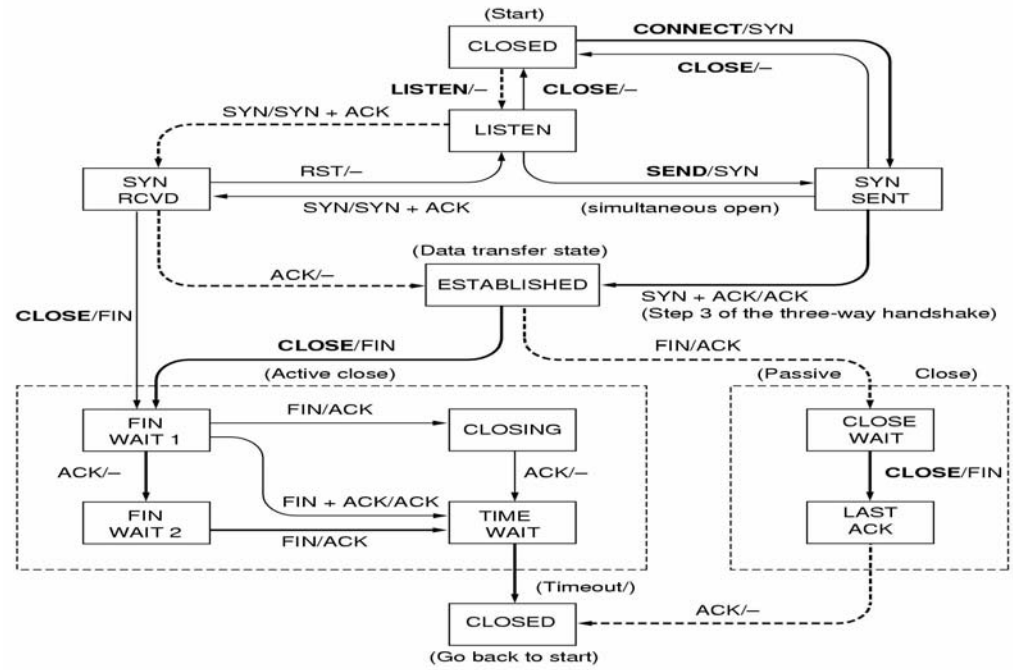
set−up contemporaneo

TCP: creazione connessione

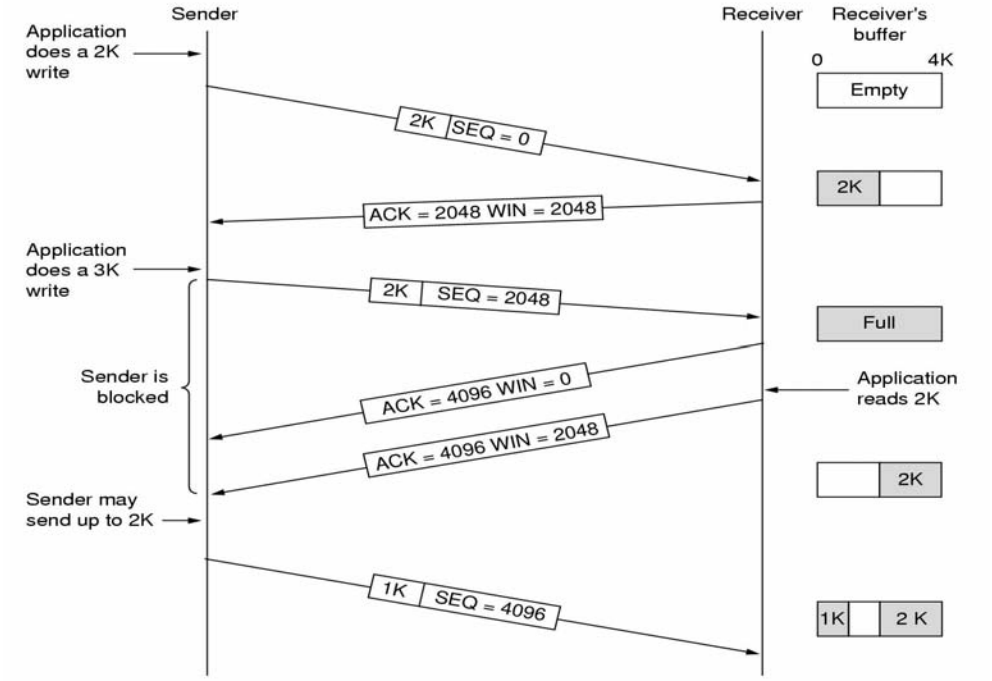
- #seq iniziale derivato da clock (tick=4 microsec.)
- forbidden region, riutilizzo port:
  - no riapertura connessione con uguale associazione per max segment lifetime (120 sec.)
  - problema dei due eserciti
    - timer chiusura connessione (default 240 sec.)
- automa a stati

CLOSED	FIN WAIT 1
LISTEN	FIN WAIT 2
SYN RCVD	TIMED WAIT
SYN SENT	CLOSING
ESTABLISHED	CLOSE WAIT
	LAST ACK

TCP: creazione connessione



TCP: trasmissione dati



TCP: gestione finestra trasmissione

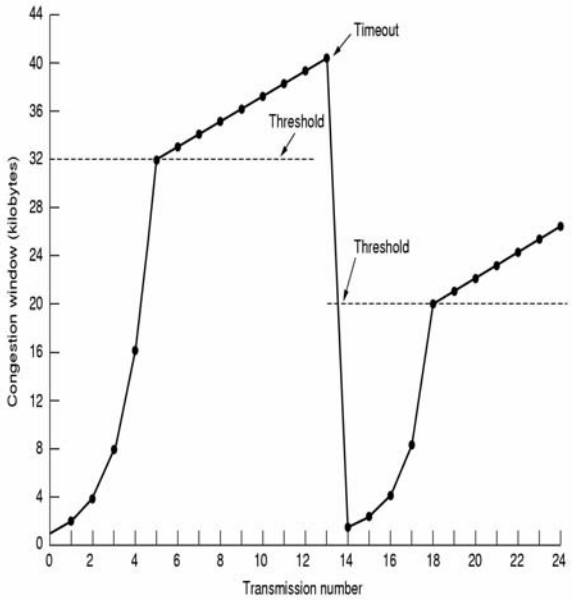
- dati urgenti possono essere inviati sempre
  - NB: di default, esiste limitazione 1B urgent (oob) data
- deadlock avoidance: sender periodicamente invia 1B per provocare ack da recvr con window size
- ottimizzazione uso banda: algo Nagle
  - es: telnet con tempi risposta 500 msec (echo)
  - non va bene per movimento mouse...
- sindrome silly window: approccio Clark
  - spazio disponibile >= min {buffer/2 , max TPDU}
  - read blocca fino ad arrivo *congrua* quantita' di dati

TCP: controllo congestione

- flow ctrl gestito negoziando finestra receiver
- congestione: recvr vede arrivare poco traffico e non ridimensiona la finestra trasmissione
- errori trasmissione (su rete wired!) infrequenti
  - HYP: timeout si verifica per congestione sottorete
- soluzione: manipolazione dyn finestra trasmissione
  - /** NB: "falsi" timeout per errata stima rtt... **/
  - se "stringo" finestra trasmissione, inietto meno carico in rete e allevio congestione
  - ... ma se stringo troppo posso essere inefficiente

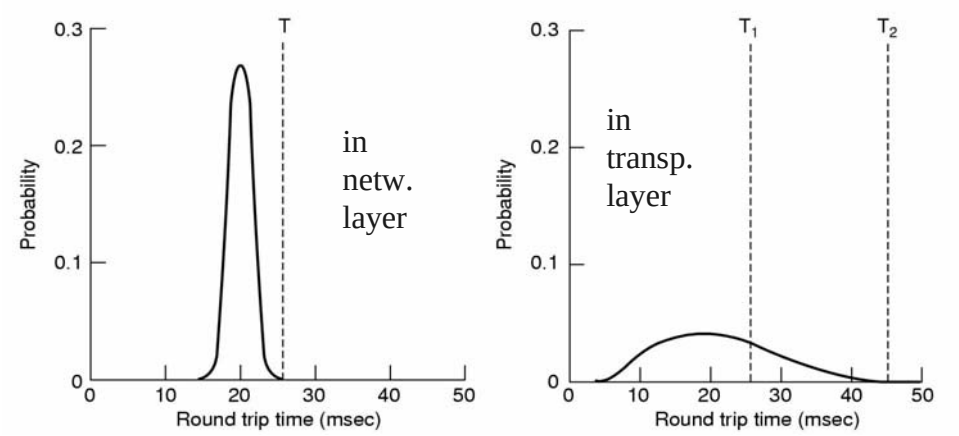
TCP: congestion window

- sender tiene 2 finestre: trasm. e congestione (CW)
  - puo' trasmettere il min tra le due ampiezze
- CW inizializzata a max TPDU (prudente...)
- ogni volta che invio con successo: raddoppio CW
- se deve ritrasmettere: riduco finestra
  - **slow start algo**



TCP: gestione timer ritrasmiss.

- timer ottimale = round−trip time *(perche' ne' + ne' − ?)*
- se fallimento: devo allargare timer
  - a differenza di netw.L., entita' aumento e' difficilmente stimabile



TCP: stima RTT

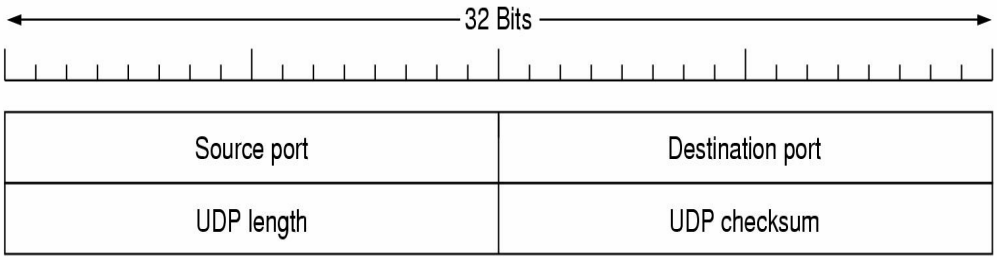
- algo Jacobson per stima dinamica
- stima corrente aggiornata con stima nuova M:
  - RTT = α x RTT + (1− α) x M
  - α e' peso "*storia*"; in generale α = 7/8
  - mi serve per adattare se va bene
- se va male, posso *raddoppiare* timeout
- oppure [Jacobson] vario di fattore proporzionale alla deviazione standard D (figura precedente)
  - D = α D + (1−α) | RTT − M |
  - timeout = RTT + 4 D
  - parametri ottimizzati matematicamente

TCP: gestione timer

- prbl: se ritrasmetto, stime successive possono essere errate
- algo Karn:
  - raddoppio timer ad ogni failure
  - aggiorno stima quando 1° invio corretto
- persistence timer per prevenzione deadlock
  - recvr impone a sender finestra 0
  - successivo update finestra e' perso
  - entrambi i processi sono in attesa uno dell'altro
  - sender dopo un po' manda probe per avere news
- keepalive timer; timer per *timed wait*

UDP

- incapsula IP datagram e aggiunge checksum
  - checksum calcolato su pseudo−header
- serve per evitare costo set−up connessione e accesso a routine di sistema (socket raw)
- conserva limiti di messaggio



Application−layer protocols

API: application programming interface

- defines interface between application and transport layers
- socket: Internet API
  - two processes communicate by sending data into socket, reading data out of socket
  - socket: punto terminale di un canale di comunicazione di alto livello
  - differente da pipe:
    - per comunicazione tra procs su host diversi
    - per comunicazione full−duplex

Socket programming

socket: astrazione di un canale di comunicazione tra processi distribuiti in rete

Socket API

- introduced in BSD4.1 UNIX, 1981
- explicitly created, used, released by apps
- client/server paradigm
- two types of transport service via socket API:
  - unreliable datagram
  - reliable, byte stream−oriented

socket

a *host−local, application−created/owned, OS−controlled* interface (a ‘‘door”) into which application process can both send and receive messages to/from another (remote or local) application process

Application design

- entita' appl.L. hanno proprio protocollo
- in funzione semantica appl., va scelto appropriato servizio di transp.L.
  - in funzione servizio, diverso schema interazione tra client e server
    - TCP tipicamente per scambio grosse quantita' dati, lunghe interazioni, affidabilita' necessaria
    - UDP se o/h set−up connessione sproporzionato w.r.t. scambio dati; puo' rendere piu' complesso livello applicativo
  - in accordo, opportune primitive per interfaccia
- server iterativo vs. concorrente
  - client piu' semplice da progettare

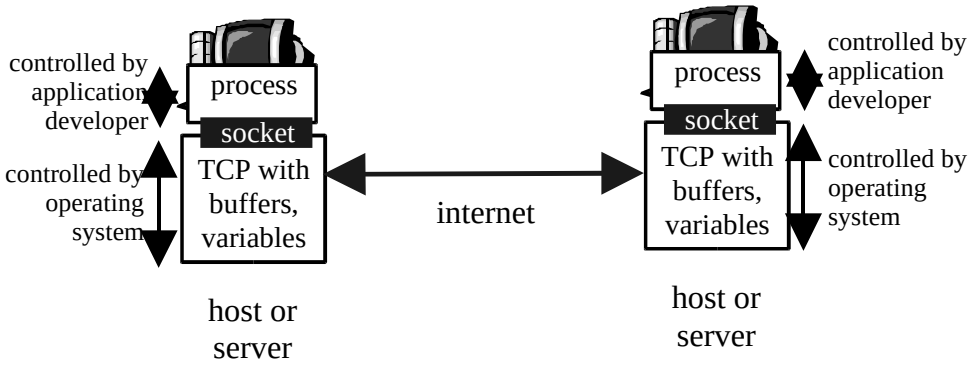
Socket programming in Java

- java.net: package contenente
  - InetAddress: class per manipolazione indirizzi
  - Socket: class per uso servizi TCP a lato client
  - ServerSocket: class per uso servizi TCP a lato server
  - DatagramSocket: class per uso servizi UDP
    - indifferenziata per client e server. *Perche'*?
  - DatagramPacket: class per manipolazione UDP segment
- java.io: package contenente classi per la gestione di I/O stream (necessario per uso TCP)
- descrizione dettagliata classi disponibile a http://java.sun.com/products/jdk/1.2/docs/api/java/net/<*nomeclasse*>.html

Socket−programming using TCP

Socket: a door between application process and end−to−end−transport protocol (UDP or TCP)

TCP service: reliable transfer of **bytes** from one process to another



Socket programming *with TCP*

Client must contact server
- server process must first be running
- server must have created socket (door) that welcomes client's contact

Client contacts server by:
- creating client−local TCP socket
- specifying IP address, port number of server process

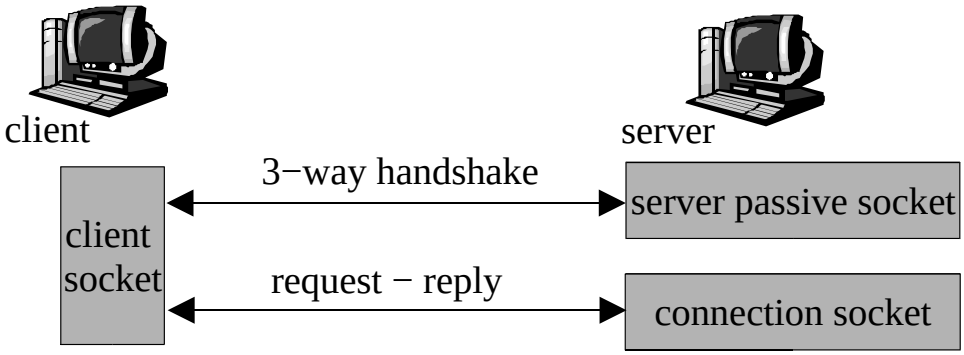
- When client creates socket: client TCP establishes connection to server TCP
- When contacted by client, server TCP creates new socket for server process to communicate with client
  - allows server to talk with multiple clients

application viewpoint

TCP provides reliable, in−order transfer of bytes (''pipe") between client and server

Socket programming with TCP

- passive socket del server legata a well−known port
  - tutti i client conoscono
- specifico client usa poi socket (e port) personale
  - gestione piu' client sfruttando thread



Java class InetAddress

- rappresenta indirizzo host in formato rete (ulong)
- non ha costruttori
- metodi:
  - equals(Object obj)
  - getAddress() torna IP address locale (ulong)
  - getByName(String host): IP addr da nome simbolico
  - getHostAddress(): IP addr in dotted notation
  - getHostName() nome simbolico locale
  - ....

Java class Socket

- costruttori:
  - Socket() crea socket non connessa
  - Socket(InetAddress addr, int port) crea socket e la connette allo specifico server; si puo' specificare anche addr locale
  - Socket(String host, int port) come sopra, ma specifica server con nome simbolico
- metodi:
  - getInputStream(), getOutputStream()
  - getInetAddress(), getLocalAddress(), getLocalPort(), getPort()
  - close(): rilascio connessione
  - metodi per get/set buffer size, timeout, algo Nagle

Java class ServerSocket

- costruttori:
  - ServerSocket(int port)
- metodi:
  - accept() attende connessione
  - close(), getInetAddress(), getLocalPort() come class Socket

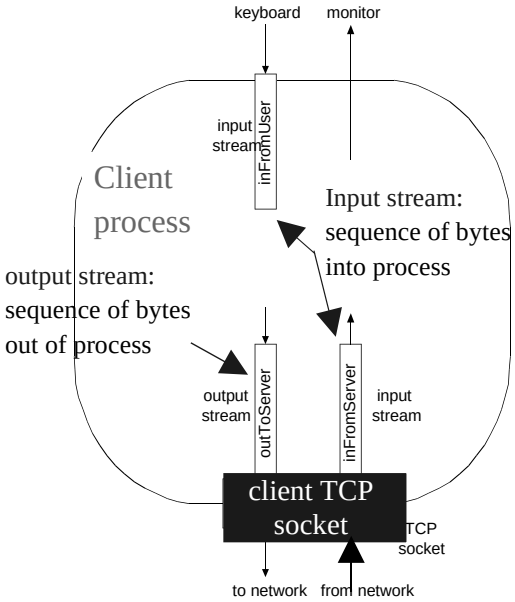
SocketExceptions

- class per la gestione degli errori che si possono verificare nell'uso delle socket
- alcune eccezioni della classe sono:
  - BindException: porta locale in uso o non assegnabile
  - ConnectException: connessione rifiutata dal server
  - NoRouteToHostException: host remoto iraggiungibile (connect timed out)

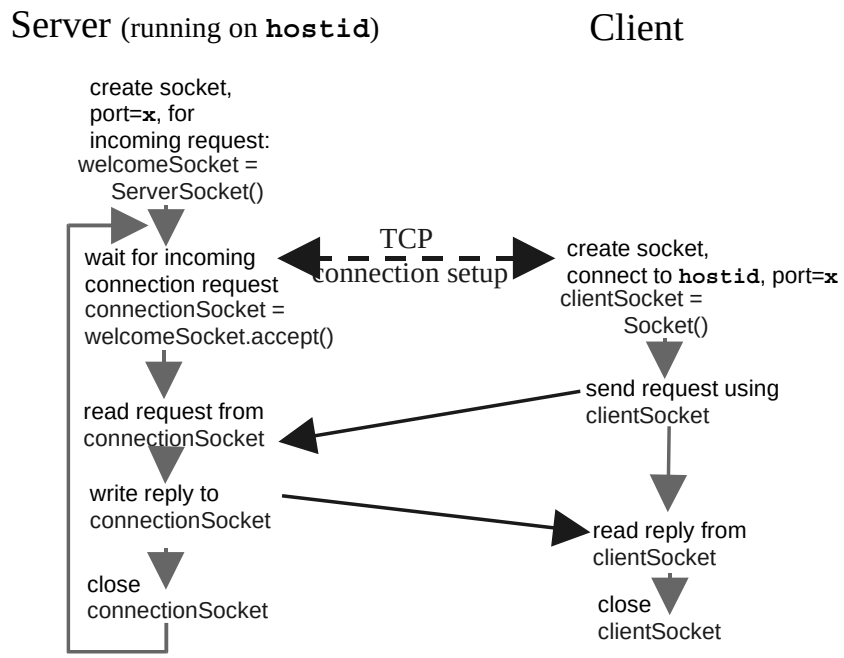
Socket programming with TCP

Example client−server app:

- client reads line from standard input (**`inFromUser`** stream) , sends to server via socket (**`outToServer`** stream)
- server reads line from socket
- server converts line to uppercase, sends back to client
- client reads, prints modified line from socket (**`inFromServer`** stream)



Client/server socket interaction: TCP



Example: Java client (TCP)

```
import java.io.*;
import java.net.*;
class TCPClient {

    public static void main(String argv[]) throws Exception
    {
        String sentence;
        String modifiedSentence;

        BufferedReader inFromUser =
          new BufferedReader(new InputStreamReader(System.in));

        Socket clientSocket = new Socket("hostname", 6789);

        DataOutputStream outToServer =
          new DataOutputStream(clientSocket.getOutputStream());
```

Create input stream

Create client socket, connect to server

Create output stream attached to socket

Example: Java client (TCP), cont.

Create input stream attached to socket

```
BufferedReader inFromServer =
  new BufferedReader(new
  InputStreamReader(clientSocket.getInputStream()));

sentence = inFromUser.readLine();
```

Send line to server

```
outToServer.writeBytes(sentence + '\n');
```

Read line from server

```
modifiedSentence = inFromServer.readLine();

System.out.println("FROM SERVER: " + modifiedSentence);

clientSocket.close();

    }
}
```


Example: Java server (TCP)

server iterativo

```
import java.io.*;
import java.net.*;

class TCPServer {

  public static void main(String argv[]) throws Exception
   {
    String clientSentence;
    String capitalizedSentence;
```

Create welcoming socket at port 6789

```
    ServerSocket welcomeSocket = new ServerSocket(6789);

    while(true) {
```

Wait, on welcoming socket for contact by client

```
      Socket connectionSocket = welcomeSocket.accept();
```

Create input stream, attached to socket

```
      BufferedReader inFromClient =
        new BufferedReader(new
        InputStreamReader(connectionSocket.getInputStream()));
```

Example: Java server (TCP), cont

Create output stream, attached to socket

Read in line from socket

Write out line to socket

```
DataOutputStream  outToClient =
  new DataOutputStream(connectionSocket.getOutputStream());

clientSentence = inFromClient.readLine();

capitalizedSentence = clientSentence.toUpperCase() + '\n';

outToClient.writeBytes(capitalizedSentence);
    }
  }
}
```

End of while loop, loop back and wait for another client connection

Server concorrente

- multithreading: un thread del server per ogni client collegato
- struttura del server:

```
ServerSocket welcomeSocket = new ServerSocket(6789);
while (true) {
    Socket Incoming = Server.accept();
    thread Handler = new ThreadHandler();
    Handler.start();
}
```

Socket programming *with UDP*

UDP: no ‘‘connection” between client and server

- no handshaking
- sender explicitly attaches IP address and port of destination for each message
- server must extract IP address, port of sender from received datagram

UDP: transmitted data may be received out of order, or lost

application viewpoint

UDP provides unreliable transfer of groups of bytes (‘‘datagrams”) between client and server

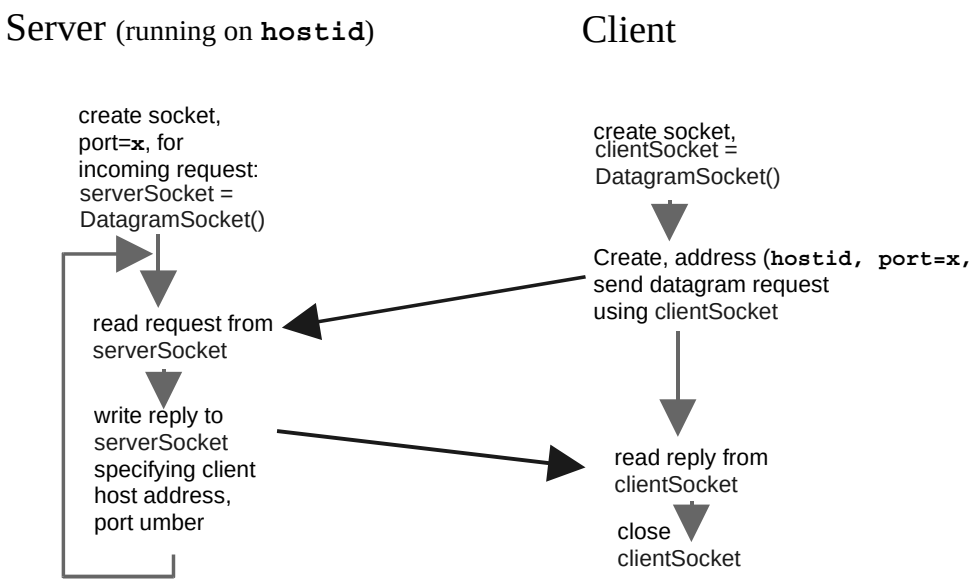
Java class DatagramPacket

- 2 costruttori:
  - DatagramPacket(byte[] buf, int length) per ricevere
  - DatagramPacket(byte[] buf, int length, InetAddress address, int port) per inviare
- metodi per get/set campi msg:
  - getAddress() ; setAddress(InetAddress iaddr) /* remoto */
  - getData() ; setData(byte[] buf)
  - getLength() ; setLength(int length)
  - getPort() ; setPort(int iport) /* remoto */

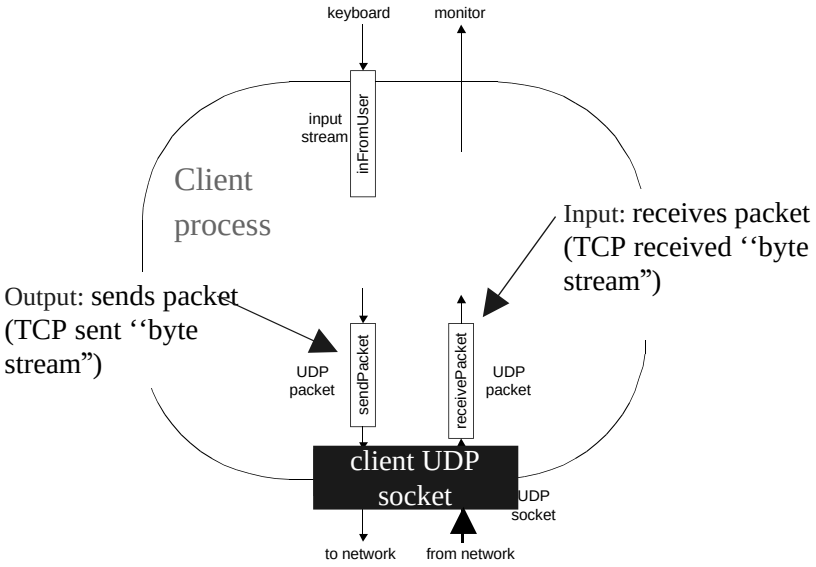
Java class DatagramSocket

- costruttori:
  - DatagramSocket() /* port scelta da S.O. */
  - DatagramSocket(int port)
- metodi:
  - send(DatagramPacket p) ; receive(DatagramPacket p)
  - getInetAddress(); getLocalAddress(); getLocalPort(); getPort() come class Socket
  - close()
  - metodi per get/set buffer, timeout

Client/server socket interaction: UDP



Example: Java client (UDP)



Example: Java client (UDP)

```
import java.io.*;
import java.net.*;

class UDPClient {
    public static void main(String args[]) throws Exception
    {
```

Create input stream

```
      BufferedReader inFromUser =
        new BufferedReader(new InputStreamReader(System.in));
```

Create client socket

```
      DatagramSocket clientSocket = new DatagramSocket();
```

Translate hostname to IP address using DNS

```
      InetAddress IPAddress = InetAddress.getByName("hostname");

      byte[] sendData = new byte[1024];
      byte[] receiveData = new byte[1024];

      String sentence = inFromUser.readLine();
      sendData = sentence.getBytes();
```

Example: Java client (UDP), cont.

Create datagram with data−to−send, length, IP addr, port

Send datagram to server

Waiting to read datagram from server

```
DatagramPacket sendPacket =
  new DatagramPacket(sendData, sendData.length, IPAddress, 9876);

clientSocket.send(sendPacket);

DatagramPacket receivePacket =
   new DatagramPacket(receiveData, receiveData.length);

clientSocket.receive(receivePacket);

String modifiedSentence =
    new String(receivePacket.getData());

System.out.println("FROM SERVER:" + modifiedSentence);
clientSocket.close();
}
}
```


Example: Java server (UDP)

Create datagram socket at port 9876

condivisa da tutti i client

Create space for received datagram

Receive datagram

```
import java.io.*;
import java.net.*;

class UDPServer {
  public static void main(String args[]) throws Exception
    {

      DatagramSocket serverSocket = new DatagramSocket(9876);

      byte[] receiveData = new byte[1024];
      byte[] sendData  = new byte[1024];

      while(true)
        {

          DatagramPacket receivePacket =
             new DatagramPacket(receiveData, receiveData.length);
          serverSocket.receive(receivePacket);
```

Example: Java server (UDP), cont

```
String sentence = new String(receivePacket.getData());

InetAddress IPAddress = receivePacket.getAddress();

int port = receivePacket.getPort();

        String capitalizedSentence = sentence.toUpperCase();

sendData = capitalizedSentence.getBytes();

DatagramPacket sendPacket =
   new DatagramPacket(sendData, sendData.length, IPAddress,
                      port);

serverSocket.send(sendPacket);
  }
 }
}
```

Get IP addr port #, of sender

Create datagram to send to client

Write out datagram to socket

End of while loop, loop back and wait for another datagram

Socket programming: references

Java−tutorials:

- ‘‘All About Sockets” (Sun tutorial), http://www.javaworld.com/javaworld/jw−12−1996/jw−12−sockets.html
- ‘‘Socket Programming in Java: a tutorial,” http://www.javaworld.com/javaworld/jw−12−1996/jw−12−sockets.html

Link sul sito del corso

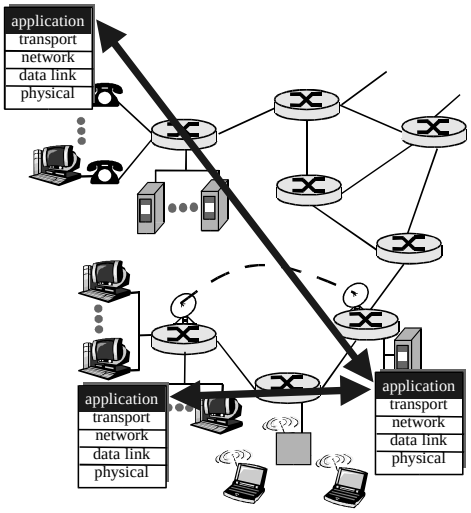
Applications and application−layer protocols

Application: communicating, distributed processes

- running in network hosts in "user space"
- exchange messages to implement application
- e.g., email, ftp, Web

Application−layer protocols

- one "piece" of an application
- define messages exchanged by apps and actions taken
- use communication services provided by lower layer protocols (TCP, UDP) via socket



Struttura applicazioni

- user agent + protocollo client/server
- user agent: software process, interfacing with user "above" and network "below".
  - implements application's protocol (***non*** application−layer protocol…)
  - Web: browser
  - E−mail: mail reader
  - streaming audio/video: media player

DNS: Domain Name System

Internet hosts, routers:

- IP address (32 bit) – usato dai protocolli di rete
- nome simbolico, es. gaia.cs.umass.edu

- nome simb. case insensitive
  - max 63 char per ogni componente
  - max 255 char per nome intero
  - rigida struttura gerarchica per localizzaz. geografica

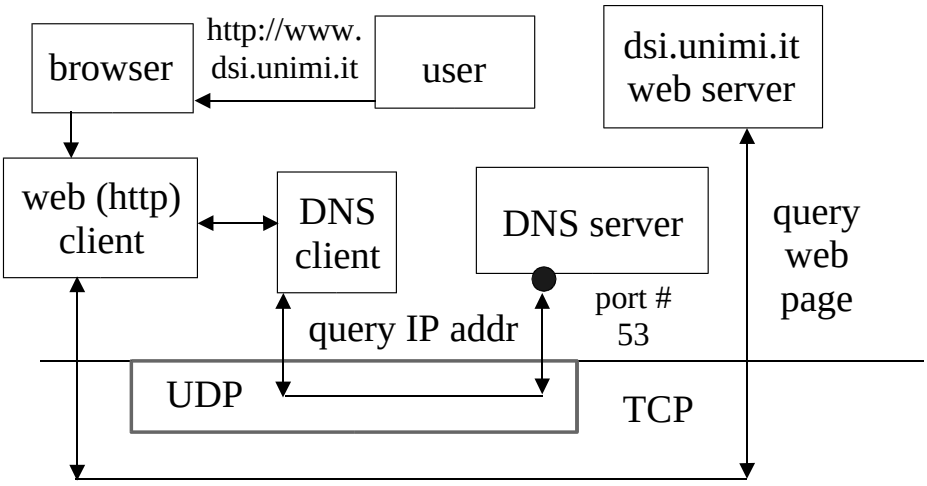
Domain Name System (RFC 1034, 1035):

- *distributed database* implemented in hierarchy of many *name servers*
- *application−layer protocol* : host, routers, name servers need to *resolve* names (address/name translation)
  - note: core Internet function, implemented as application−layer protocol
  - complexity at network's "edge"

DNS: servizi forniti

- e' applicazione client−server (nslookup)
- servizi:
  - traduzione nome simbolico in IP address
  - host aliasing: nome canonico
    - mercurio.srv.dsi.unimi.it − www.dsi.unimi.it
  - mail server aliasing
  - distribuzione carico per server replicati
    - rotazione lista a ogni interrogazione
    - Akamai, Content Distribution Networks

DNS: schema funzionamento



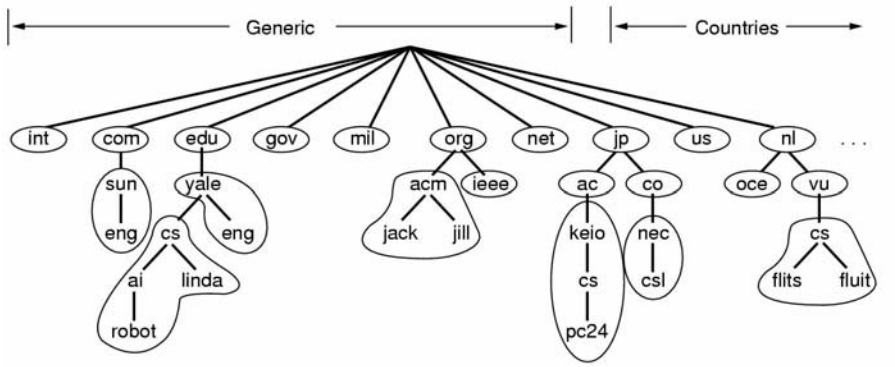
… + traduzione IP addr in MAC addr …

DNS: implementaz. centralizzata

- agli albori di Arpanet… (hosts.txt)
- single point of failure
- bottleneck: alto volume traffico, alto ritardo
  - distant centralized database: alto delay e2e
- difficolta' gestione
  - DB enorme per ogni mapping, alias…
  - frequenti aggiornamenti
  - sicurezza (authentication, authorization)

doesn't *scale!*

DNS: spazio dei nomi



local name server: primo punto di ricerca (default)

- cablato in file config sys (+ server secondario backup)

authoritative name server: responsabile dominio

- for a host: stores that host's IP address, name
- can perform name/address translation for that host's name

root name server: interrogato se local non sa rispondere a query

DNS: spazio dei nomi

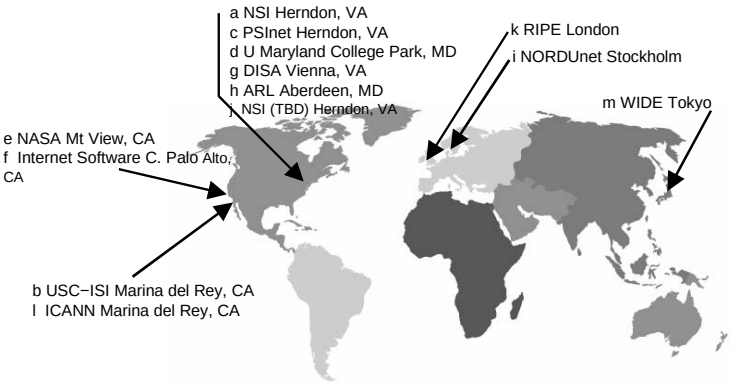
- quando un nome simbolico non e' specificato per intero, sistema tenta completamento aggiungendo il nome del dominio locale
  - es: tic.silab −> tic.silab.dsi.unimi.it
- albero globale diviso in zone
  - zona: insieme di nodi con unica radice
  - gestione distribuita tra le zone
  - ogni server conosce almeno i mapping della propria zona

DNS: gestione spazio dei nomi

- ogni host deve essere registrato presso almeno 2 authoritative server (ans)
- ans ha sempre indirizzo corretto degli host registrati
- ans controlla allocazione sub−domain
  - ES: unimi.it controlla dsi.unimi.it …
  - no duplicazione, traccia degli ans dei subdomain
  - NON applicato iterativamente!
- non legato a confini amministrativi
- diverse centinaia di domini top−level

DNS: Root name servers

- contacted by local name server that can not resolve name
- root name server:
  - contacts authoritative name server if name mapping not known
  - gets mapping and returns mapping to local name server



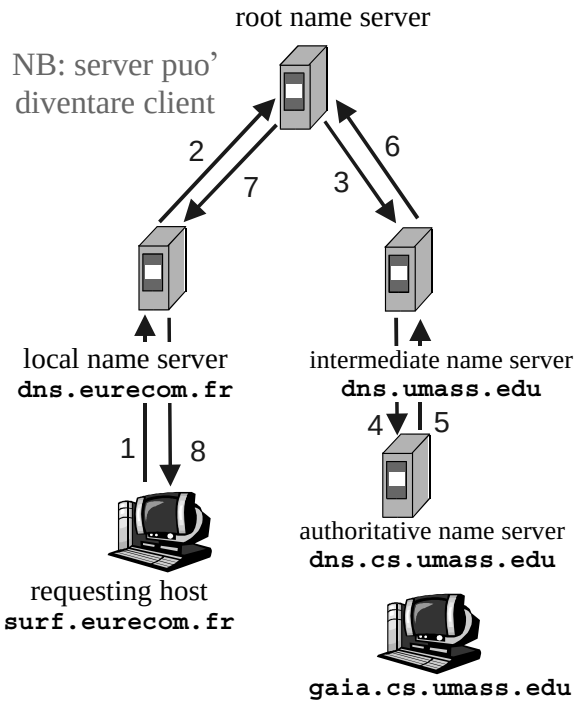
13 root name servers worldwide

DNS example

host **`surf.eurecom.fr`** wants IP address of **`gaia.cs.umass.edu`**

1. contacts its local server, **`dns.eurecom.fr`**
2. **`dns.eurecom.fr`** contacts root name server, if necessary
3. root name server contacts ans, **`dns.umass.edu`**

if root does not know ans: it may know *intermediate name server:* who to contact to find ans



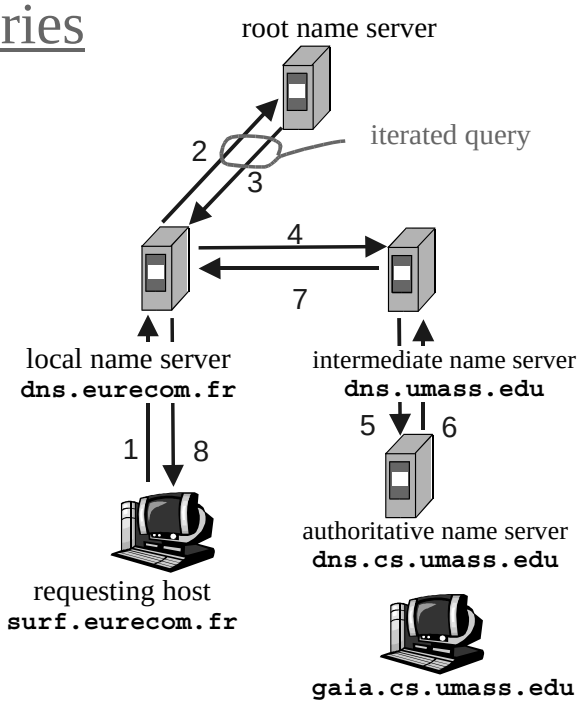
DNS: iterated queries

recursive query:

- puts burden of name resolution on contacted name server
- heavy load?

iterated query:

- contacted server replies with name of server to contact
- "I don't know this name, but ask this server"



DNS: gestione query

- query iterativa riduce carico a root server; usata se root non e' capace di ricorsione
- tempi di risoluzione anche lunghi
- once server learns mapping, it *caches* mapping
  - risposta non−authoritative
  - cache entries timeout after some time
- fault−tolerance: query a tempo. Se scade timeout, client inviera' query successiva a un altro server
- correntemente aggiornamento manuale
  - update/notify mechanisms under design by IETF (RFC 2136)
  - http://www.ietf.org/html.charters/dnsind−charter.html

DNS records

DNS: distributed db storing resource records (RR)

RR format: **`(domain-name, ttl, type, class, value)`**

- Type=A
  - **`name`** is hostname
  - **`value`** is IP address
- Type=NS
  - **`name`** is domain (e.g. foo.com)
  - **`value`** is IP address of authoritative name server for this domain
- Type=CNAME

  IN
  - **`name`** is alias name for some "canonical" (the real) name

    `www.ibm.com` is really `servereast.backup2.ibm.com`
  - **`value`** is canonical name
- Type=MX
  - **`value`** is name of mailserver associated with **`name`**

DNS: struttura database

- ttl <= 86400 sec. (1 giorno)
- nessun server possiede tutti i mapping
- indirizzo root server caricato a boot da file config sys

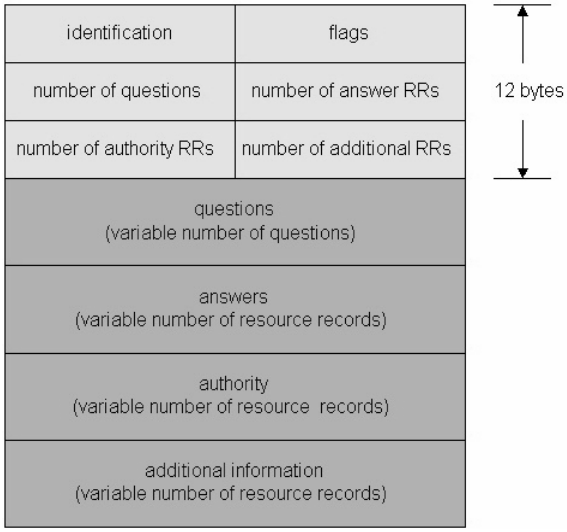
```
cs.vu.nl      86400  IN     TXT    "Vrije Universiteit"
cs.vu.nl      86400  IN     MX     1 zephyr.cs.vu.nl
PC1.cs.vu.nl  86400  IN     HINFO  Sun Unix
PC1.cs.vu.nl  86400  IN     A      130.37.16.112
PC1.cs.vu.nl  86400  IN     A      192.31.231.165
PC1.cs.vu.nl  86400  IN     MX     1 PC1.cs.vu.nl
PC1.cs.vu.nl  86400  IN     MX     2 zephyr.cs.vu.nl
www.cs.vu.nl  86400  IN     CNAME  zephyr.cs.vu.nl
...........
```


DNS protocol messages

DNS protocol : *query* and *reply* messages, both with same *message format*

msg header

- identification: 16 bit # for query, reply to query uses same #
- flags:
  - query or reply
  - recursion desired
  - recursion available
  - reply is authoritative



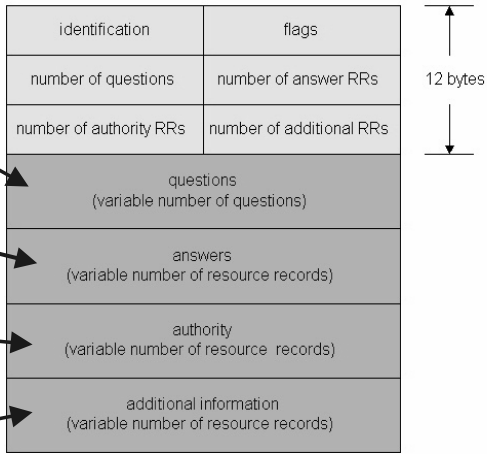
DNS protocol messages

Name, type fields for a query

RRs in reponse to query

records for authoritative servers

additional “helpful” info that may be used



Condivisione file

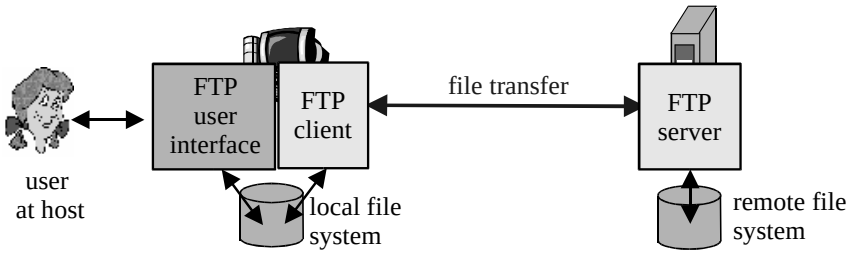
- approccio on−line (NFS)
  - modifiche immediatamente visibili a tutti
  - prbl: synchro tra utenti, bottleneck al server
- approccio off−line (FTP)
  - copia replicata su host utente
  - prbl: riconciliazione copie
  - trasferimento batch vs. interattivo
    - batch: noto meno latenza; interatt: risolvo subito errori
- FTP [RFC 959]: trasferimento interattivo
  - autenticazione utenti
  - gestione prbl derivanti da sistemi eterogenei (naming, rappresentazione interna dati)

FTP: modalita' d'uso

DNS + aliasing…

```
$ ftp ftp.cs.purdue.edu
Connected to lucan.cs.purdue.edu
220 lucan.cs.purdue.edu FTP server (Version wu-2.4.2-
  VR16(1) ready.
Name (ftp.cs.purdue.edu:users): anonymous
331 Guest login ok, send e-mail address as password
Password: ********
230 Guest login ok, access restrictions apply.
ftp> get pub/comer/tcpbook.tar bookfile
200 PORT command okay.    /* creazione port per client */
150 Opening ASCII mode data connection for tcpbook.tar
  (9895469 bytes).
226 Transfer complete.
9895469 bytes received in 22.76 seconds (4.3e+02
  Kbytes/s)
ftp> close
221 Goodbye.
ftp> quit
```

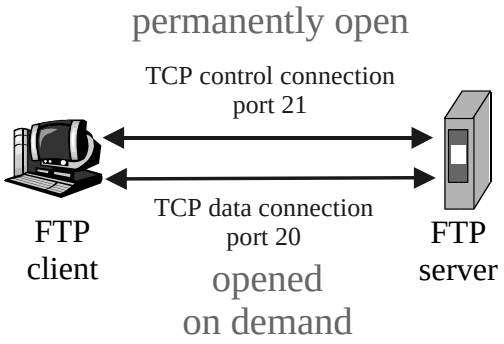
ftp: the file transfer protocol



- transfer file to/from remote host
- client/server model
  - *client:* side that initiates transfer (either to/from remote)
  - *server:* remote host
- ftp: RFC 959
- ftp server: port 21

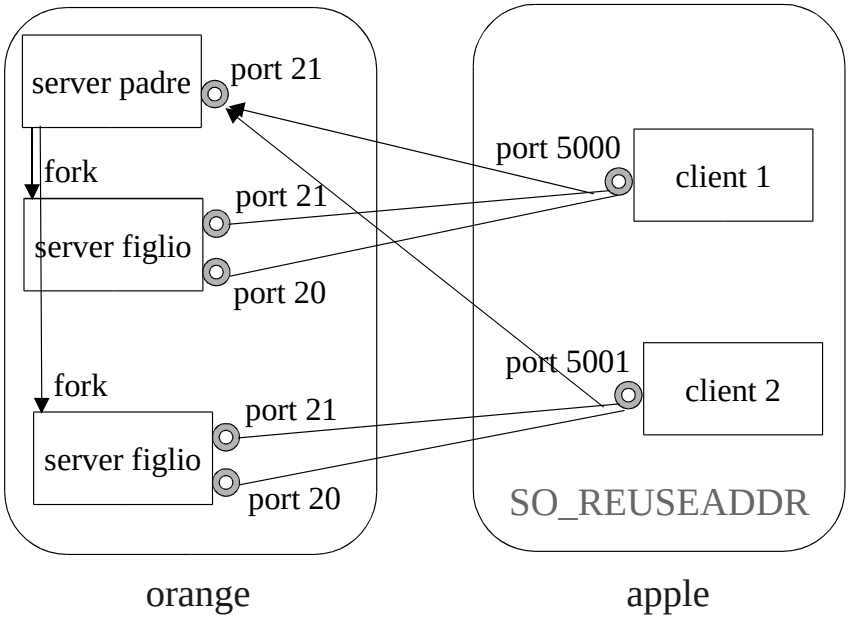
ftp: separate control, data connections

- ftp client contacts ftp server at port 21, specifying TCP as transport protocol
- two parallel TCP connections opened:
  - control: exchange commands, responses between client, server.
    "out of band control"
  - data: file data to/from server
- ftp server maintains "state": current directory, earlier authentication



HTTP stateless; trasferisce info controllo in−band; può usare connessioni TCP non−permanenti o permanenti

FTP: protocollo



ftp commands, responses

Sample commands:

- sent as ASCII text over control channel
- **`USER`** ***`username`***
- **`PASS`** ***`password`***
- **`LIST`** return list of file in current directory
- **`RETR filename`** retrieves (gets) file
- **`STOR filename`** stores (puts) file onto remote host

Sample return codes

- status code and phrase (as in http)
- **`331 Username OK, password required`**
- **`125 data connection already open; transfer starting`**
- **`425 Can't open data connection`**
- **`452 Error writing file`**

Electronic mail

- trasferimento asincrono di file
- differenze con FTP / HTTP
  - http: pull proto; e−mail: push proto
  - tutti e 3 usano TCP
  - mail deve essere sequenza di ascii char (7 bit): tutto deve essere convertito
  - http: ogni oggetto trasferito in msg separato
  - mail: tutti gli oggetti posti nello stesso msg
- componenti architettura: user agent, mail server, SMTP

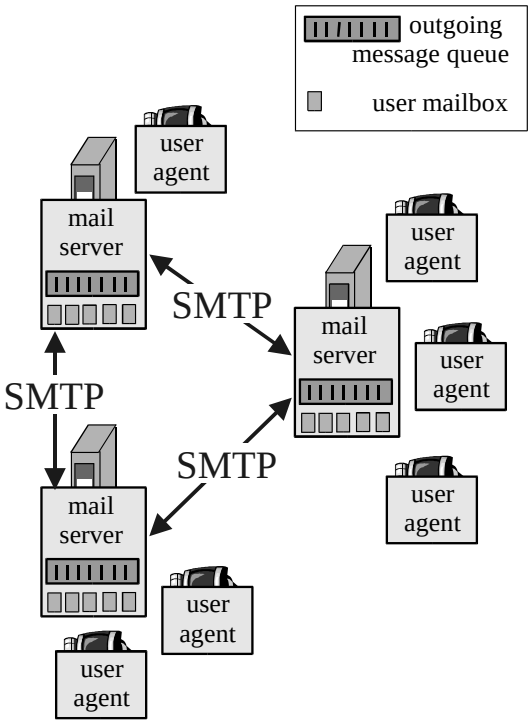
Electronic Mail

Three major components:

- user agents
- mail servers
- simple mail transfer protocol: smtp

User Agent

- a.k.a. “mail reader”
- composing, editing, reading mail messages
- e.g., Eudora, Outlook, elm, Netscape Messenger
- outgoing, incoming messages stored on server



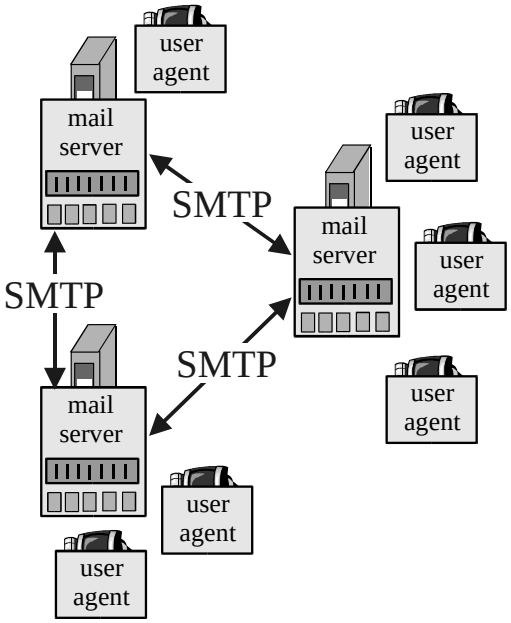
Email: funzionalita'

- composizione msg:
  - opportuno formato (header)
  - codifica eventuali attachment
- trasferimento: SMTP
  - reporting esito a sender (registered e−mail)
- display, disposition a ricevente
  - opportuna elaborazione attachment
- gestione mailing list, mail alias, sicurezza, Cc:, priorita' …

Electronic Mail: mail servers

Mail Servers

- mailbox contains incoming messages (yet to be read) for user
- message queue of outgoing (to be sent) mail messages
- smtp protocol between mail servers to send email messages
  - client: sending mail server
  - "server": receiving mail server



Electronic Mail: smtp [RFC 821, 822]

- client/server: both running for each mail server
- uses tcp to reliably transfer email msg from client to server, port 25
- direct transfer: sending server to receiving server
- three phases of transfer
  - handshaking (greeting): autenticazione, controllo esistenza destinatario
  - transfer of messages: tutti per il server dest. (connessione persistente)
  - closure
- command/response interaction
  - commands: ASCII text
  - response: status code and phrase

Sample smtp interaction

```
S: 220 hamburger.edu
C: HELO crepes.fr
S: 250  Hello crepes.fr, pleased to meet you
C: MAIL FROM: <alice@crepes.fr>
S: 250 alice@crepes.fr... Sender ok
C: RCPT TO: <bob@hamburger.edu>
S: 250 bob@hamburger.edu ... Recipient ok
C: DATA
S: 354 Enter mail, end with "." on a line by itself
C: Do you like ketchup?
C:   How about pickles?
C: .
S: 250 Message accepted for delivery
C: QUIT
S: 221 hamburger.edu closing connection
```

Try smtp interaction for yourself:

- **`telnet servername 25`**
- see 220 reply from server
- enter HELO, MAIL FROM, RCPT TO, DATA, QUIT commands

above lets you send email without using email client (reader)

smtp: caratteristiche msg

- smtp server uses `CRLF.CRLF` to determine end of message
- msg deve contenere header e body
- header per controllo: `<keyword: value>` /*1 riga testo */
  - alcuni header possono essere aggiunti da server

To:	Received:	Reply−To:
Cc:	Return−Path:	Message−Id:
Bcc:	Date:	In−Reply−To:
From:	Sender:	Subject:

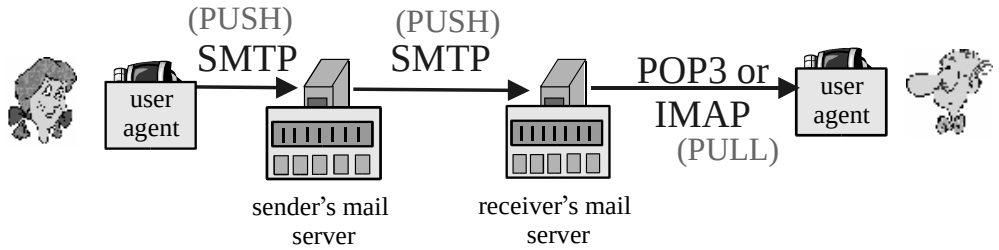
differenti da comandi SMTP!

- certain character strings not permitted in msg (e.g., `CRLF.CRLF`). Thus msg has to be encoded (usually into either base−64 or quoted printable)

User agent

- costruzione msg con opportune info header
- push via SMTP a mail server mittente
- invio: indirizzo destinatario@location
  - se destinazione e' mailing list, mail server espande
- ricezione:
  - mail server dest aggiunge propri header
    - e.g. Received: (>1 se .forward usato)
  - su richiesta: invio a user agent destinatario
- modalita' interazione, comandi in funzione del particolare agent

Mail access protocols



- SMTP: delivery/storage to receiver's server (sempre in esecuzione e connesso a Internet)
- Mail access protocol: retrieval from server
  - POP: Post Office Protocol [RFC 1939] /* port 110 */
    - download & delete ; download & keep
  - IMAP: Internet Mail Access Protocol [RFC 2060]
    - more features (more complex)
    - manipulation of stored msgs on server

POP3 protocol

authorization phase

- client commands:
  - **user:** declare username
  - **pass:** password
- server responses
  - **+OK**
  - **-ERR**

transaction phase, client:

- **list:** list message numbers
- **retr:** retrieve message by number
- **dele:** delete
- **quit: mailbox update**

```
S: +OK POP3 server ready
C: user alice
S: +OK
C: pass hungry
S: +OK user successfully logged on
```

```
C: list
S: 1 498
S: 2 912
S: .
C: retr 1
S: <message 1 contents>
S: .
C: dele 1
C: retr 2
S: <message 1 contents>
S: .
C: dele 2
C: quit
S: +OK POP3 server signing off
```


IMAP

- supporto a accesso utente da diversi host: NON scarica mail su host utente
  - POP3 solo se configurato “download & keep”
- consente mantenimento folder multipli per uno stesso utente
  - mantiene info stato tra sessioni diverse
    - POP3 non lo fa (stateless)
- permette download componente msg
  - per connessioni con poca bwth
- stato server: non−autenticato −> autenticato −> selected -> logout

Mail access protocols

- DMSP (Distributed Mail Access Proto, RFC 1056)
  - consente gestione mail in modo disconnesso e re−synch alla riconnessione
- HTTP−based e−mail
  - interazione tra user agent e mail server via http
  - interazione tra mail server via SMTP
  - > facilita' accesso ubiquo, ma puo' essere lento
- filtri per gestione e−mail, .forward, vacation daemon (effetti collaterali…)

Message format: multimedia extensions

- MIME: multi−purpose Internet mail extension, RFC 2045, 2046
- additional lines in msg header declare MIME content type
- regole codifica per dati non ascii (7 bit)

MIME version

method used to encode data

multimedia data type, subtype, parameter declaration

encoded data

```
From: alice@crepes.fr
To: bob@hamburger.edu
Subject: Picture of yummy crepe.
MIME-Version: 1.0
Content-Transfer-Encoding: base64
Content-Type: image/jpeg

base64 encoded data .....
.........................
......base64 encoded data
```

MIME types

`Content-Type: type/subtype; parameters`

Text

- subtypes: **`plain, html`**
  - charset: us−ascii, ISO−8859−1

Image

- subtypes: **`jpeg, gif`**

Audio

- subtypes: **`basic`** (8−bit mu−law encoded), **`32kadpcm`** (32 kbps coding)

Video

- subtype: **`mpeg`**

Application

- other data that must be processed by reader before "viewable"
- example subtypes: **`msword, octet-stream`**

MIME: tipi di dati

- multipart/mixed: per msg contenenti + oggetti
  - MIME header che descrive ogni oggetto
  - boundary characters tra oggetti diversi
  - per ogni oggetto azioni opportune (metamail)
- codifiche:
  - quoted printable: trasforma testo ascii arbitrario in testo senza caratteri indesiderabili (eg. CRLF.CRLF)
  - base64: gruppo 24 bit frammentato in 4 pezzi di 6 bit, ognuno codificato con ascii char
    - 'A'…'Z', 'a'…'z', '0'…'9', '+', '/'

Multipart Type

```
From: alice@crepes.fr
To: bob@hamburger.edu
Subject: Picture of yummy crepe.
MIME-Version: 1.0
Content-Type: multipart/mixed; boundary=98766789

--98766789
Content-Transfer-Encoding: quoted-printable
Content-Type: text/plain

Dear Bob,
Please find a picture of a crepe.
--98766789
Content-Transfer-Encoding: base64
Content-Type: image/jpeg

base64 encoded data .....
.........................
......base64 encoded data
--98766789--
```


Eterogeneita' sistemi, sicurezza

- e−mail gateway: traduce msg in transto tra sistemi che usano diversi formati e protocolli
  - mapping tra indirizzi con rappresentaz. diverse
  - manipolazione header non presenti/riconosciuti
- privacy: di default, tutto viaggia in chiaro
  - PGP (Pretty Good Privacy) fornisce segretezza, autenticazione, firma digitale, compressione
  - basato su schema a chiave pubblica
  - repository chiavi pubbliche: "*chi custodisce il custode?*"