

RETI INFORMATICHE

Donazioni sempre ben accette ❤️
<https://www.paypal.me/mgiannini01>

Introduzione

• INTERNET

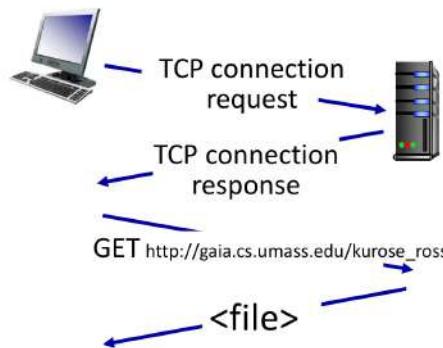
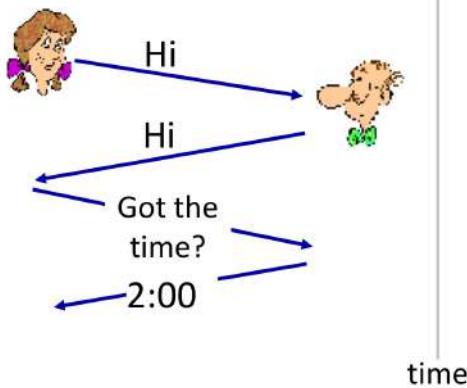
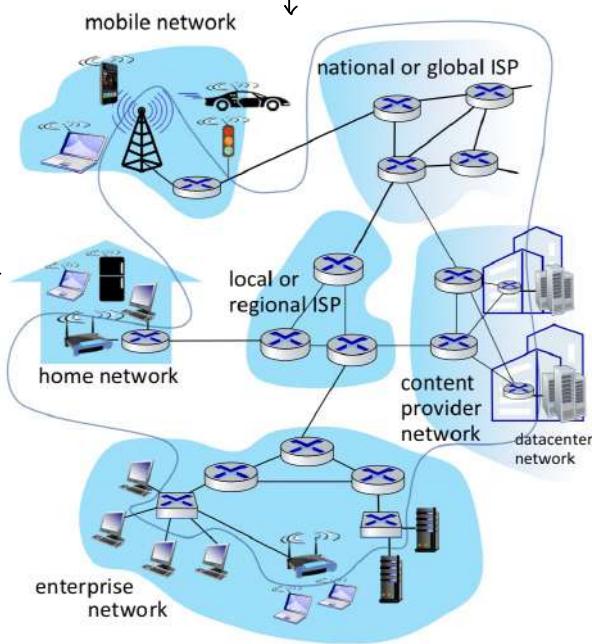
Sistema che interconnecte un certo numero di computer (host) che ospitano applicazioni. Sono collegati tramite reti di accesso e sono presenti router che fungono da interno dicendo: non generano traffico e ricevono pacchetti generati da host per trasporto a recuperarli dall'host destinatario che lo indirizza al prossimo destinatario. I computer e i router sono collegati da link di comunicazione ove vicinanza l'informazione. Questa viene codificata in un segnale e transita sul link (fibra ottica, rete, wireless). Ciò che ci interessa è il bitrate (bit trasmessi al secondo). Tutti questi dispositivi formano delle reti.

I dispositivi collegati ad internet sono di numerose categorie → IoT: Internet of things = collegano ad internet vari oggetti

Internet = rete di reti. Per la comunicazione sono necessari dei protocolli per porre in comune ciascuno: insieme di regole rispettate da tutti

↓ Come è un protocollo?

Si ha uno scambio di messaggi che però hanno un certo formato e che devono essere mandati in un certo ordine: vengono definiti dal protocollo insieme anche alle azioni fatte dagli interlocutori



→ Come definiscono protocollo?
Sono via via RFC (Request for comments) poi con gli standard e commentato da IETF (Internet Engineering Task Force) che decide infine se approvare o no

↓ Visione più comune (dall'alto)

Sistema di comunicazione universale per fare varie cose (streaming video, comunicazione, mail, giochi, e-commerce, social, IoT). Possiede anche di sviluppare applicazioni distribuite fornendo un'interfaccia.

PERIFERIA di INTERNET

Host, server △ Client e server indicano i processi anche se si mani per indicare la macchina su cui quei processi girano

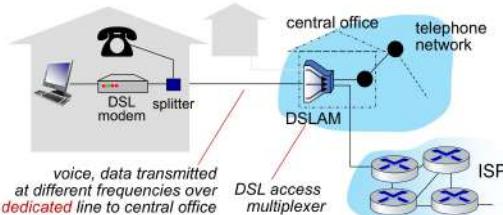
Collegati tramite reti di accesso che connettono router ad internet collegati ad host tramite link di comunicazione

NETWORK CORE: Reti fornite da ISP (Internet Service Provider), possono essere locali, regionali, nazionali, internazionali.

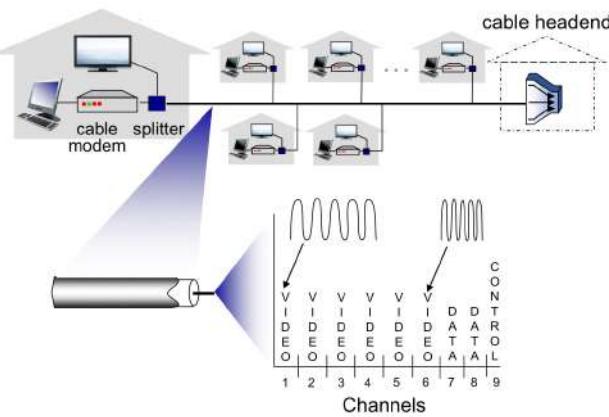
Vari tipi di reti di accesso: Residenziale (interna ad abitazione), Istituzionale (ad esempio rete di UniPi), Pubbliche (GSM/3G, WiFi). In tutti i casi possono esistere vari tipi di collegamento. WiFi, Ethernet, rete cellulare. Sono interconnessi poco di più che attraverso il link e se il link è dedicato o condiviso. Nel secondo caso la velocità viene spartita da più utenti.

↓ Tipi collegamento

ADSL: Asimmetria fra download e upload. Si fa questo perché le applicazioni client-server in cui si fa molto più download che upload. Tuttavia le linee possono essere simmetriche (SDSL). Si usa un modem e filtri per non avere interferenze. Soffre di distorsioni per trasmitti suono.

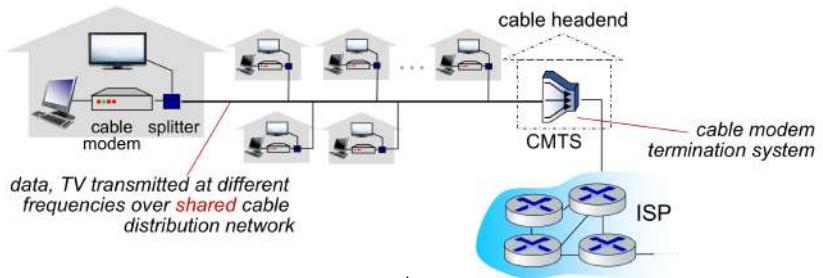


CABLE-BASED ACCESS:



→ Banda divisa in vari canali, alcuni per video, altri per internet

FIBRA OTTICA:



↓ VARI TIPI DI COLLEGAMENTO

R **ADSL**



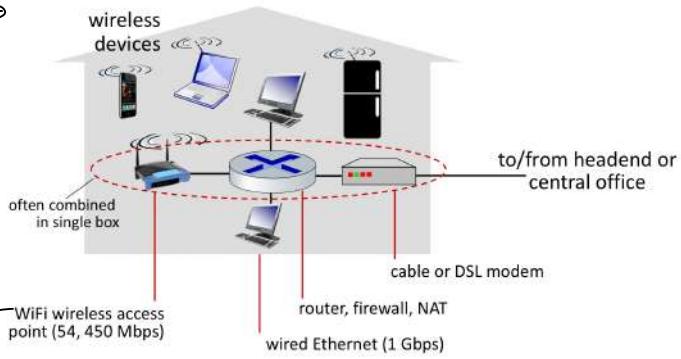
F **FTTH**



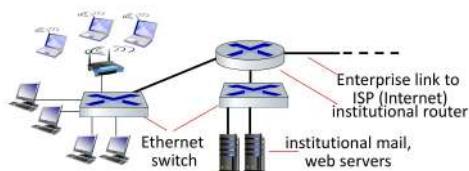
FR **FTTC**



ESEMPIO: reti di casa →



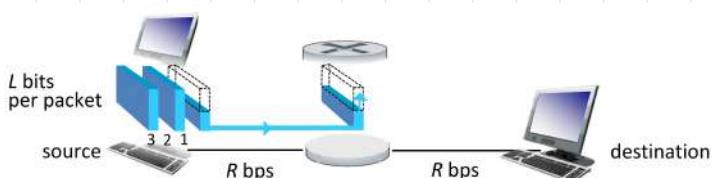
Varie tipologie: standard 802.11 tipo b/g/n a seconda del bitrate
reti intituzionali →



• PACCHETTI

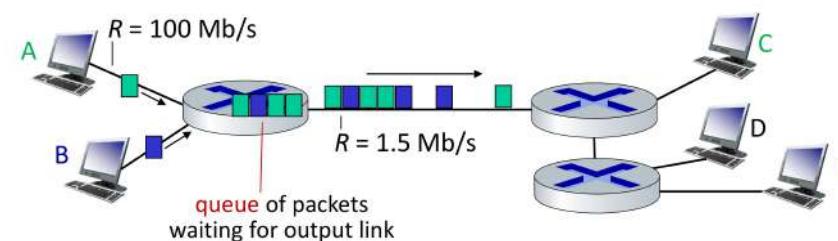
Un host connesso ad una rete di accesso invia e riceve pacchetti come risultato dei messaggi originati dalle applicazioni. L'host invia pacchetti sul link verso il router con un certo transmission rate e con un certo tempo. Un pacchetto è costituito da 2 hit che devono essere codificati in segnali (che richiede tempo). Se il hitrate è R , per trasmettere un pacchetto di 2 hit ci vuole un tempo $2/R$ (transmission delay). Una volta codificati i hit si propagano sul link → 2 tipi: guidati, ove il segnale si propaga sul link (radio) e non guidati (segnali elettromagnetici). I doppini telefonici sono un esempio di link guidato. Questi sfruttano campi magnetici per la comunicazione ed evitano interferenze per attenuare i disturbi. Altro esempio sono i casi in fibra ottica dove il segnale è di tipo luminoso. Le fibre ottiche permettono di ottenere hitrate più elevate ed hanno anche un hit error rate più basso. Il segnale non è inoltre soggetto a disturbi elettromagnetici. Permette quindi di coprire distanze più elevate. Non si usa sempre in quanto costa di più. Vi sono poi i collegamenti wireless guidati i collegamenti WiFi, quelli a microonde, le reti cellulari.

• PRESTAZIONI



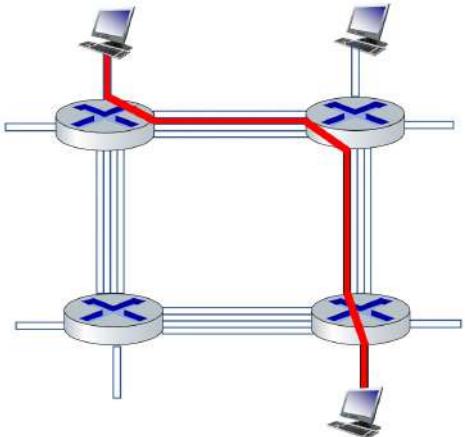
- se l'hitrate è trascorsa il link. Se vi sono più host che generano pacchetti ad una velocità > uscita si crea una coda ed i pacchetti vengono memorizzati fino a fine spazio → Si può scaricare un pacchetto già memorizzato o uno nuovo. La coda genera ritardo e può generare anche perdita di pacchetti. Questo approccio si chiama packet-switching: il link di uscita viene usato per mandare pacchetti di varie fonti → permette di scaricare costi ma genera ritardi

I router usano politica store and forward: riceve il pacchetto sul link, lo memorizza, guarda la destinazione e lo inoltra su una linea di uscita. Il delay adesso sarà $2L/R$. Il tempo di memo-



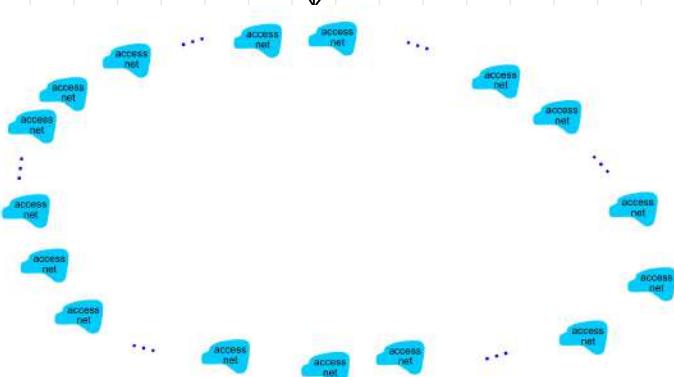
o scaricare anche perdita di pacchetti. Questo approccio si chiama packet-switching: il link di uscita viene usato per mandare pacchetti di varie fonti → permette di scaricare costi ma genera ritardi

↓ Alternativa

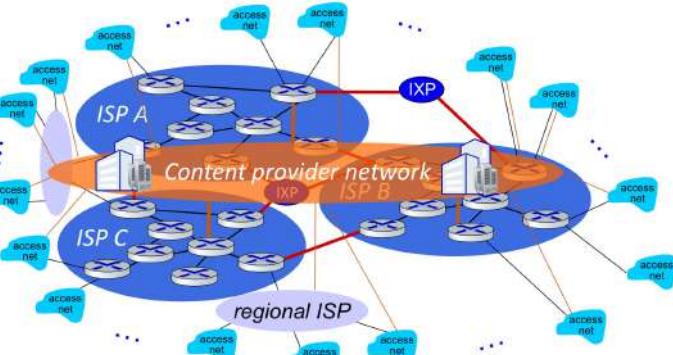


Circuit switching: Circuito dedicato tra i nodi. I denti d'incisione che non creano congestioni. Si ha quindi un link dedicato. Non conviene se ci genera poco traffico mentre conviene nel caso opposto. Il traffico dei computer è però tipicamente intermittente (bursty), quindi il packet switching è più conveniente.

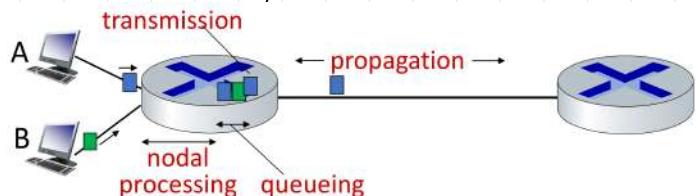
Se abbiamo molti utenti e poco traffico conviene multiplexare. Se si usa circuit switching tutti gli utenti devono pagare lo stesso anche se ne godono di una linea dedicata. Si dimostra che, se ci hanno 35 utenti, la probabilità che vi manca più di 10 utenti attivi (e quindi congestione) è di 0.0004 e quindi mi conviene packet switching in quanto si ottengono le stesse prestazioni con prezzo minore.



Ogni rete di accesso dove è nere collegata alle altre. Non è possibile connettere un router a tutti gli altri. Si crea quindi un ISP planetario che connette tutte le reti di accesso. Questo crea però una situazione di monopolio ed è possibile censurare le informazioni a piacimento. Vi sono quindi più ISP a più livelli.



Riprendiamo il concetto di coda → un pacchetto deve attendere il tempo di servizio + il ritardo di accodamento. Il primo è fino menta il secondo varia a secondi del numero di pacchetti che sono in coda. Se la coda cresce troppo chiama perdita.



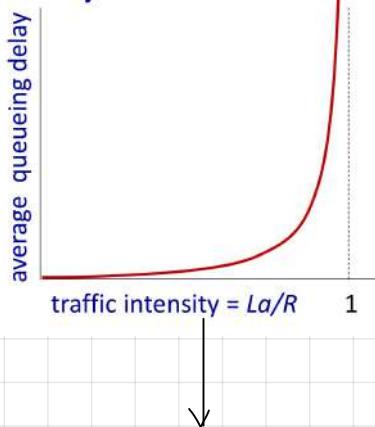
$$d_{\text{nodal}} = d_{\text{proc}} + d_{\text{queue}} + d_{\text{trans}} + d_{\text{prop}}$$

→ Il router fa store and forward: deve fare una copia in memoria, verificare la correttezza e vedere l'indirizzo di destinazione. Si parla di delay (ms) e in genere è di decine di millesimi di secondo. Una volta che il pacchetto è stato elaborato questo viene accodato e

si ha il tempo di accodamento (queuing). Prima o poi si arriva in cima alla coda e si ripetuta quindi in tempo di trasmissione L/R. Il pacchetto viene trasmesso da impulsi con un certo tempo di propagazione.

Questo dipende dal tipo di segnale e dal effetto. distanza (d/s) propagation. Se il traffico aggregato (totale) eccede la capacità del link si ha ritardo di accodamento e di conseguenza il ritardo complessivo cresce.

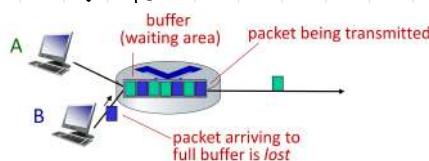
sited)



α = average arrival rate: tempo medio generazione medio
 R = bandwidth: bit secondi al secondo (bps)
 L = lunghezza pacchetto

$L_a = \frac{\alpha}{R}$ = bit al secondo in ingresso al router.
 $L_a \rightarrow$ Se $\alpha < R$ è come non ci sono code. Se $\alpha > R$ inizialmente non accade nulla ma poi il rapporto cresce in maniera vertiginosa. Comunque sia, quando rapporto $\rightarrow 1$, ritardo $\rightarrow \infty$ se buffer illimitato. Non è quindi necessario superare 1 (overload).

Come misuriamo ritardi e perdite? Usiamo traceroute: tempo impiegato per raggiungere destinazione → dipende dal traffico e quindi informazione imprecisa. Questo funziona mandando 3 pacchetti da host a router. Ottengono come risultato una lista di router da host di partenza ad host destino fatto.



consumo di energia throughput

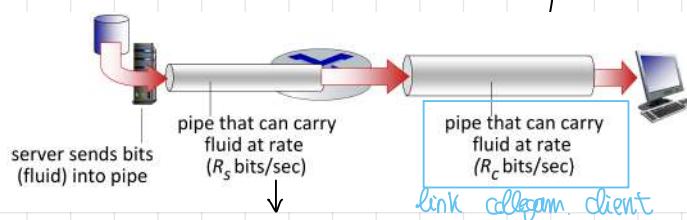
Numero di bit che il sistema riceve a destinatario (bit/s). Questo è \neq del link in quanto tutto il link misurato

→ 2 politiche: posso opporre

Se mi perde un pacchetto questo aumenta il ritardo, il traffico ed il consumo

scorrere ultimo arrivato più vecchio

pacchetto questo va rimandato anche i router intermedi



Client riceve un R_s, R_c bit/s: il link minimo costituisce il collo di bottiglia del link. Questo vale anche per collegamenti più complessi.

• SICUREZZA

È necessario avere delle misure di sicurezza. La prima misura di sicurezza è sicurezza dell'avvenire. Si hanno in rete diversi malware quali virus, worm, spyware. Abbiano poi attacchi Dos: si identifica un target a cui si mandano miliardissime richieste di servizio da parte di host infettati. Per rispondere a queste richieste il server non risponde più alle richieste ordinarie ed il server crolla. L'attacco più semplice è però quello di "cullare" le informazioni da un messaggio broadcast. Questo attacco è pericoloso in quanto possono essere acceduti dati sensibili quali numeri di carta di credito o password. Un altro tipo di attacco è l'IP spoofing nel quale il pacchetto viene firmato con un nome diverso da quello reale.

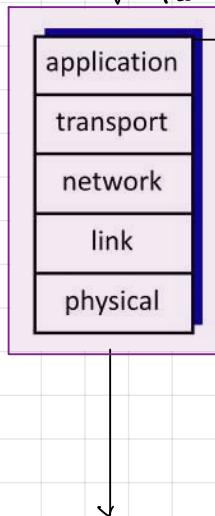
• STRATI PROTOCOLLARI

Come strutturiamo il software di rete per far comunicare app su host diversi? Si utilizza una struttura
 ↓ Esempio: Oi vuole comunicare con Os

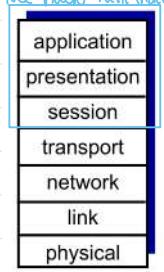


→ Internet ha un modello stratificato: più facilmente gestibile e più veloce grazie a suddivisione lavoro.

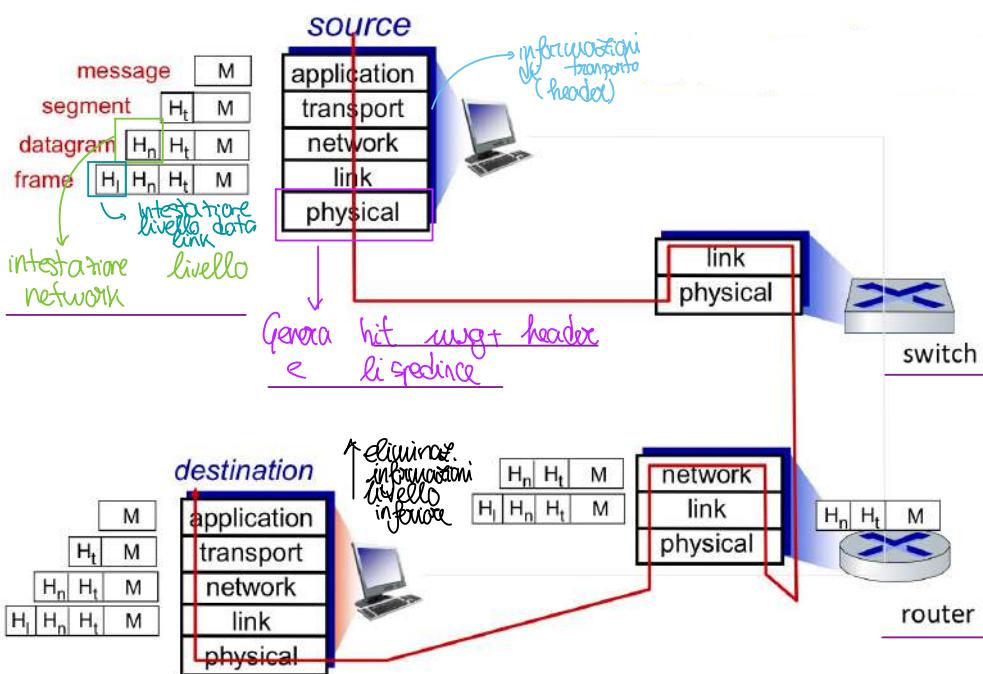
↓ Modello di internet



→ A livello applicazione ci sono i processi applicativi che generano i messaggi. Questi vengono moltiplicati dal livello di trasporto e vengono trasmessi sul link attraverso regoli fissi: **△ Ora momento** in cui si passa all'esterno. Quando questi arrivano a destinazione (dal basso verso l'alto), **Modello Originale** viene fatto nel nuovo tutto insieme:



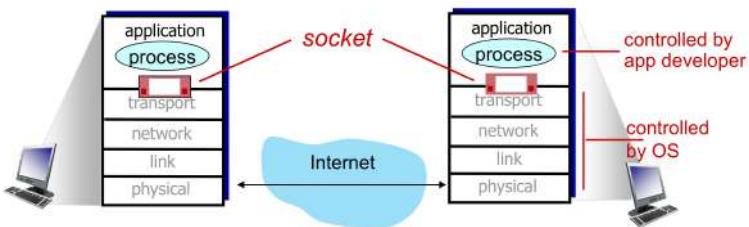
← Si passa a isolosi



Applicazioni di rete

• PRINCIPI DI BASE

Le applicazioni girano sugli host (periferia). Non è necessario per questo utilizzare tutto lo stack sui dispositivi rete. Sono costituite da diversi 2 processi (client-server) e sono comuni di scambiarsi direttamente messaggi. Ci sono 2 modelli per strutturare app. di rete: client-server e peer-to-peer. Nel primo ci sono processi di 2 categorie: il server offre il servizio ed i client ne sfruttano. Questo può essere web, e-mail, etc. C'è una netta divisione dei ruoli ed il server per questo deve essere always-on, dove avere un indirizzo IP permanente (noto) e dove avere ricco di risorse (capacità computazionali, banda, memorizzazione). Il client e la nostra macchina e si connette al server per richiedere il servizio. Non è necessario che sia sempre connesso e può avere un IP casuale dinamico. Non deve avere ricco di risorse e non comunicare mai tra di loro. Il secondo modello è quello peer-to-peer. Vi sono entità tutte dello stesso livello e possono fare sia da client che da server a seconda che richiedano/offrano un servizio. Possono avere sia client che server nello stesso momento. Non devono avere sempre attivi e possono avere IP dinamico. Non è necessario avere ricchi di risorse. Questo è perché in quanto la capacità è distribuita su molti peer. In generale abbiamo sempre processi (client-server o peer) che devono comunicare le memorie condivise (stiva) e receive messaggi da dispositivo del mittente operativo (va bene anche per computer diversi) → Ogni processo può fare send/receive su un socket da e un'astrazione del SO. Poiché il



fatta un'operazione in rete vengono coinvolti tutti i livelli sotto stanti. Poiché la "cassetta di posta" funzioni è necessario che sia associata ad un indirizzo. Nel nostro caso è l'indirizzo a 32 bit IP che identifica l'host.

All'interno dell'host vi sono però diversi processi e quindi occorre dare l'identificativo del processo e per questo si utilizza il numero di porta.

Vi sono numerose applicazioni che comunicano usando protocolli di comunicazione. I protocolli sono diversi per ogni applicazione. Questo definisce la tipologia di messaggi scambiati (request/response), le sintassi del messaggio (campi di certa lunghezza), la semantica del messaggio (mi accedo che nei campi ci sarà la cosa giusta) e regole che curano che le app facciano ciò che devono fare in risposta a messaggi. Alcuni protocolli sono open (pubblici) definiti in RFC e scambiabili liberamente. Altri sono invece di pubblico dominio ma non definiti da entità di standardizzazione. Infine vi sono protocolli proprietari che quelli d'azienda mette a disposizione l'applicazione ma non come funziona. Le applicazioni possono anche avere dati regolanti quali integrità di dati (che devono tutti essere interi, fondamentale per file eseguibili ed esecutibili mentre per altre app quali Skype questo non è necessario in quanto si tollera perdita), regolanti nel tempo (il file transfer è inversivo mentre i giochi online sono time sensitive così come il VoIP) e regolanti di throughput (sicurezza - encryption, detta

integrity 1. Le richieste sono leggibili ma possono non avere rispettato.

Internet offre 2 tipi di servizio: **TCP** → Servizio uffidabile: i bit arrivano tutti, nello stesso ordine e correttamente. C'è un servizio di controllo di overflow e congestione. Non si garantisce nulla nel ritardo, né throughput minimo e nulla sicurezza.

UDP → Non uffidabile: unaggio può non arrivare o può arrivare corretto o può non avere mantenuto l'ordine. Non ci sono garanzie né per gli altri appelli. Questo è però utile per dare ripetizioni in quanto riduce il ritardo derivante dai controlli.

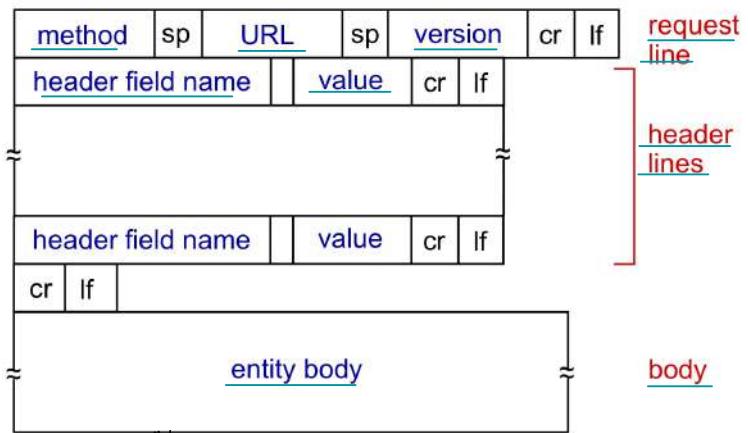
application	application layer protocol	transport protocol
file transfer/download	FTP [RFC 959]	TCP
e-mail	SMTP [RFC 5321]	TCP
Web documents	HTTP 1.1 [RFC 7320]	TCP
Internet telephony	SIP [RFC 3261], RTP [RFC 3550], or proprietary	TCP or UDP
streaming audio/video	HTTP [RFC 7320], DASH	TCP
interactive games	WOW, FPS (proprietary)	UDP or TCP

Per la sicurezza né TCP né UDP offrono niente quindi tutto è in chiaro. Sono però state introdotte delle soluzioni quali i TLS (transport layer security) che è uno strato ulteriore tra application e transport. Questo offre encryption, integrity ed end-to-end.

Web ed HTTP

Oggi diversi mezzi fatti su server diversi. Una pagina web è costituita da un file HTML e da oggetti embedded che compongono la pagina complessiva. Tutto può essere indirizzato tramite una URL preceduta dal protocollo. La URL è composta da host name e path name. Questo è un nome univoco che corrisponde all'indirizzo IP. Per facilitare la garanzia di unicità definiamo i domini. Per la comunicazione client-server si usa HTTP che usa il protocollo TCP in quanto si fa uno file-transfer multiplo. Questo anche perché TCP è connection oriented: al primo tempo per apertura e chiusura della connessione. **⚠ Non confondere app. da protocollo mato dell'app.** HTTP è il protocollo che sostiene le applicazioni web. Ogni volta che si rispetta la stessa richiesta vengono fatte sempre le stesse operazioni → il server è "stale". Non si ricorda le richieste che sono state fatte precedentemente. Questo è fatto per semplicità: memorizzare le informazioni introduce complessità. Vi sono 2 tipi di HTTP: il non-persistent prevede l'apertura di una connessione, l'invio di un oggetto ed infine la chiusura della connessione. È possibile invece, per il persistent HTTP, tenere aperta per più tempo la connessione per inviare più oggetti. La connessione viene chiusa dopo un po' di attivita' in ragione di una specie di attacco DDOS. Questa va chiamata quindi dopo un certo timeout. Il più efficiente è il persistent HTTP.

↓ Struttura di una richiesta
formato ASCII
GET /index.html HTTP/1.1\r\n linea di richiesta: path da seguire, protocollo, fine riga
Host: www-net.cs.umass.edu\r\n nome host
User-Agent: Firefox/3.6.10\r\n tipo client
Accept: text/html,application/xhtml+xml+xml\r\n file accettati
Accept-Language: en-us,en\r\n lingua
Accept-Encoding: gzip,deflate\r\n
Connection: keep-alive\r\n
\r\n connessione persistente



sviluppo di risposte

```

prodotto accettato
HTTP/1.1 200 OK\r\n
Date: Sun, 26 Sep 2010 20:09:20 GMT\r\n
Server: Apache/2.0.52 (CentOS)\r\n
Last-Modified: Tue, 30 Oct 2007 17:00:02
GMT\r\n
Content-Length: 2652\r\n (byte)
Keep-Alive: timeout=10, max=100\r\n many richieste
Connection: Keep-Alive\r\n (or persistent)
Content-Type: text/html\r\n
\r\n
data data data data ...

```

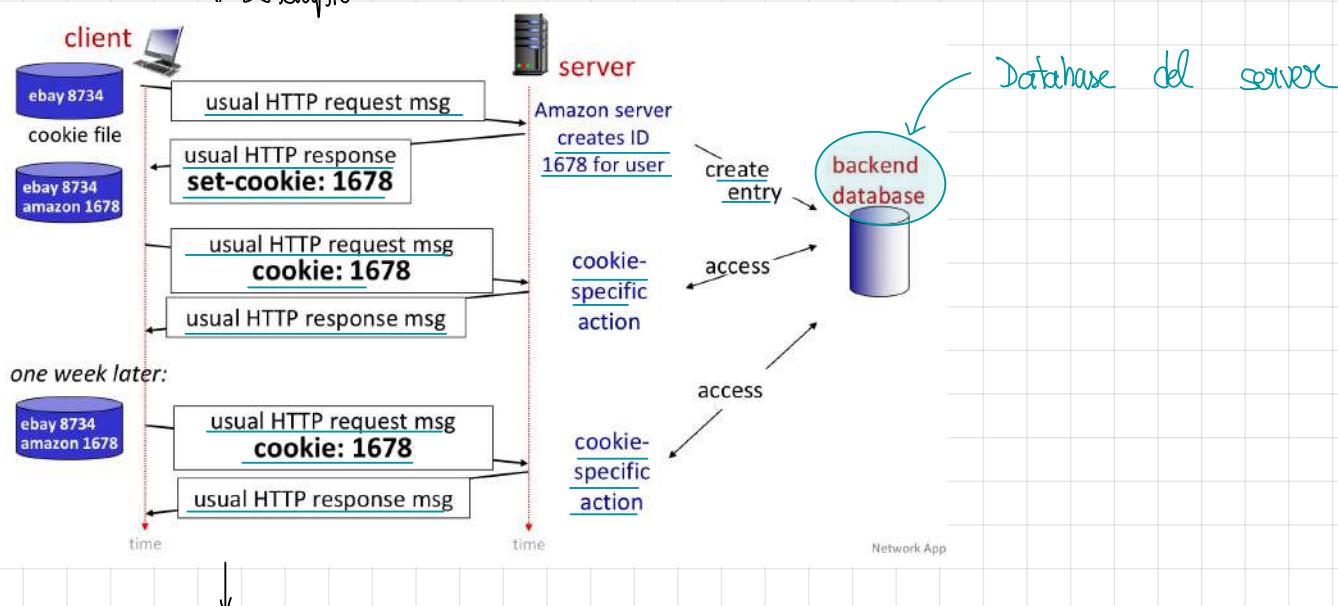
↓ codici

200 = OK, 301 = oggetto spostato, 400 = Bad request: non capisco niente,
404 = not found, 505 = HTTP Version not supported

↓

In alcuni casi statoless non è il massimo. Per introdurre memoria i cookie. Il meccanismo prevede di includere una linea di intestazione nel sviluppo di richieste e nel sviluppo di risposte ed un file dei cookie sulla macchina client ed un database nel server

↓ Esempio



Usi successivi: autenticazione (password), carrello, suggerimenti mirati, sessioni utente
↓ (web ed e-mail)

Problemi per la privacy: le informazioni sensibili oppure le informazioni vengono rilevate grazie all'analisi. Che cosa ne fa il server di questi dati? L'uso può essere etico, non etico, accettabile, non accettabile. Questo accade anche a causa della possibilità di vendere i dati a terzi ed anche al fatto dei dati possibili.

web cache

Client e server possono essere molto lontani ed il server potrebbe avere congestionato e quindi il response delay può essere molto alto. Per ridurre il delay si usa il meccanismo della cache in modo da avvicinare l'informazione. Si mettono nel mezzo dei proxy server che sono delle macchine dove gira il proxy

ed è client-server che intercetta le richieste del browser e la dà in cache in modo trasparente. Se il proxy ha già l'informazione gliela invia ed perché il proxy è situato sulla rete locale del client. Si comporta da client nel caso di redirezione verso il server. In questo modo si ottiene un tempo di risposta più veloce. Questo perché tipicamente più persone di un'organizzazione accedono alle stesse informazioni ed anche i singoli utenti accedono più volte alle stesse informazioni. Oltre alla velocità, un beneficio è quello di ridurre l'accesso al link che va dalla rete dell'organizzazione a quello del provider ed anche il traffico sulla rete internet in generale.

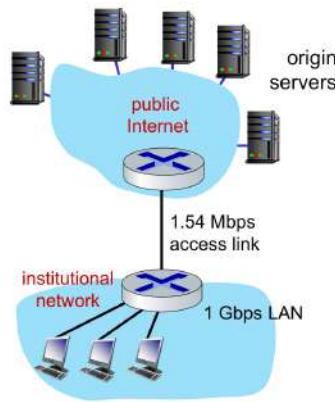
↓ Esempio

Scenario:

- access link rate: 1.54 Mbps
- RTT from institutional router to server: 2 sec
- Web object size: 100K bits
- Average request rate from browsers to origin servers: 15/sec
 - average data rate to browsers: 1.50 Mbps

Performance:

- LAN utilization: .0015
- access link utilization = **.97**
- end-end delay = Internet delay + access link delay + LAN delay
 $= 2 \text{ sec} + \text{minutes} + \text{usecs}$



1: Aumento velocità link:

Scenario:

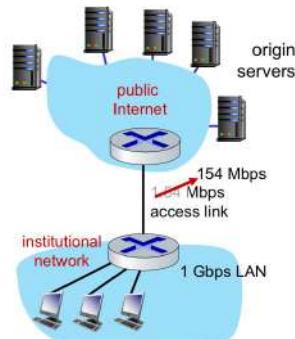
- access link rate: **154 Mbps**
- RTT from institutional router to server: 2 sec
- Web object size: 100K bits
- Avg request rate from browsers to origin servers: 15/sec
 - avg data rate to browsers: 1.50 Mbps

Performance:

- LAN utilization: .0015
- access link utilization = **.97** → .0097
- end-end delay = Internet delay + access link delay + LAN delay
 $= 2 \text{ sec} + \text{msecs} + \text{usecs}$

Cost: faster access link (expensive!) → msecs

↓ Risoluzione



2: Installazione web cache

Scenario:

- access link rate: 1.54 Mbps
- RTT from institutional router to server: 2 sec
- Web object size: 100K bits
- Avg request rate from browsers to origin servers: 15/sec
 - avg data rate to browsers: 1.50 Mbps

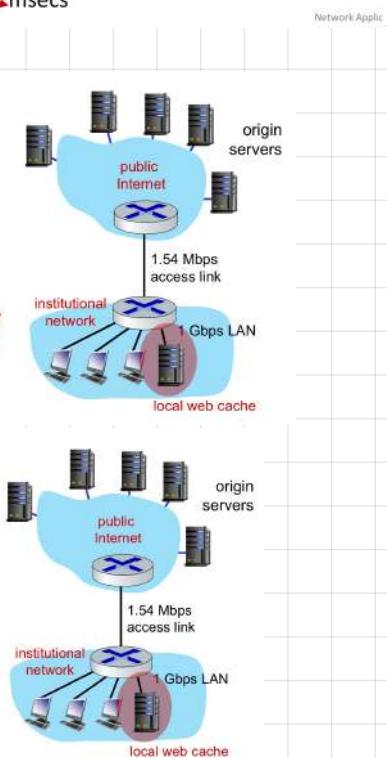
Performance:

- LAN utilization: ?
- access link utilization = ? How to compute link utilization, delay?
- average end-end delay = ?

Cost: web cache (cheap!)

Calculating access link utilization, end-end delay with cache:

- suppose cache hit rate is 0.4: 40% requests satisfied at cache, 60% requests satisfied at origin
- access link: 60% of requests use access link
- data rate to browsers over access link
 $= 0.6 * 1.50 \text{ Mbps} = .9 \text{ Mbps}$
- utilization = $0.9 / 1.54 = .58$
- average end-end delay
 $= 0.6 * (\text{delay from origin servers}) + 0.4 * (\text{delay when satisfied at cache})$
 $= 0.6 (2.01) + 0.4 (\sim \text{msecs}) = \sim 1.2 \text{ secs}$

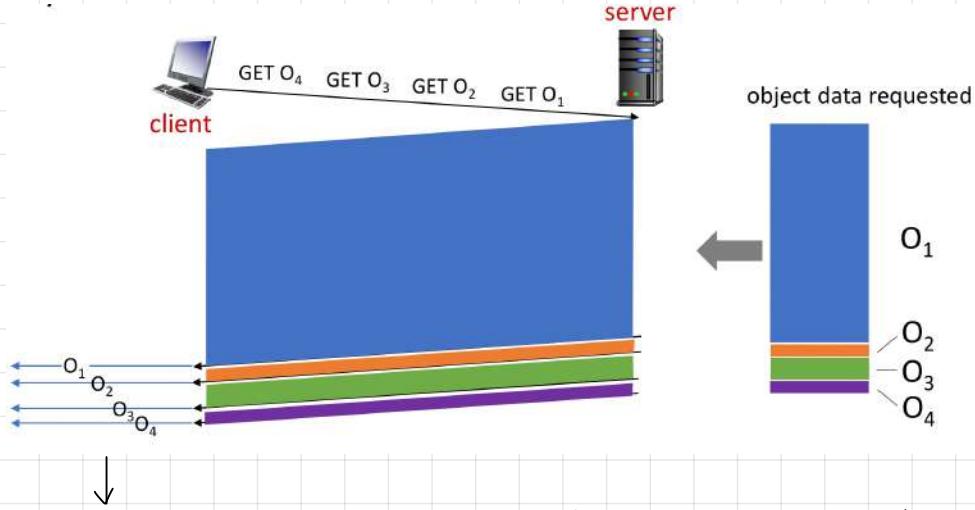


lower average end-end delay than with 154 Mbps link (and cheaper too!)

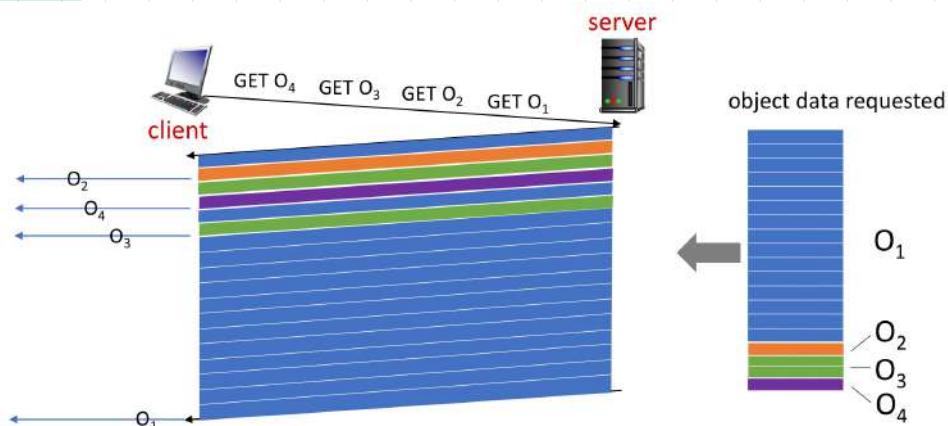
Siamo sicuri che la copia inviata dal proxy sia buona? le copie presenti nel proxy potrebbero non essere aggiornate → proxy richiede aggiornamento: non può richiedere troppe frequentemente → viene fatto solo se viene fatto la richiesta e la copia non è aggiornata: conditional GET → guarda la data di ultima modifica dell'oggetto che ora sta comunicata dal server

HTTP/2

Se un browser vuole ricevere molti dati può verificarsi il fenomeno dell'idle line blocking: il primo della linea tiene bloccato il server per molto tempo → connessioni diverse in parallelo per richiedere oggetti diversi. Inoltre, il protocollo TCP è fair quindi tutte le connessioni hanno lo stesso throughput e questo prevede l'allocatione di eccezionali. Si ha dead of line blocking a causa dell'utilizzo di TCFB (first come first scheduling).



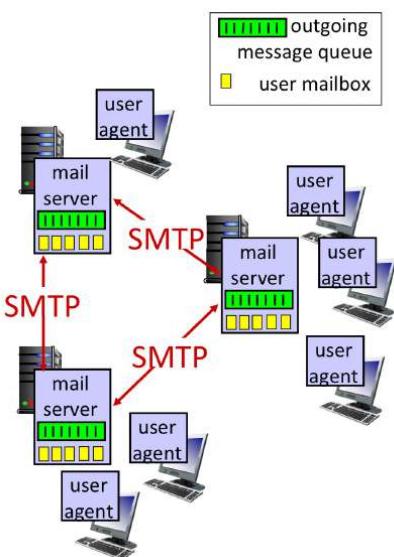
Si passa a schedulazione round robin (un po' per uno) dividendo il file in frame



Con HTTP/2 si può anche richiedere a punti specificando l'ordine in cui si vogliono ricevere gli oggetti. Il client può anche non richiedere gli oggetti e mandare automaticamente al server (quelli embedded). In HTTP/2 non sono presenti meccanismi di sincronizzazione
↓ transazione in corso
HTTP/3: utilizza protocollo QUIC anziché il TLS

E-MAIL

3 componenti fondamentali: user-agent (programma di posta elettronica) e protocollo SMTP (Simple Mail Transfer Protocol)



Il server funziona sia per i messaggi di ingresso che per quelli di esodo ed ha una coda per gestirli. In particolare quelli in entrata vengono accodati nella mailbox dell'utente. I server comunicano con il protocollo SMTP. Il mail server fa il client quando invia un messaggio ad un server e fa il server quando riceve. Il portale Server indica la macchina in quanto ci sono 2 processi diversi per inviare e ricevere. SMTP usa TCP come protocollo di trasferimento in quanto vogliamo integrità imitabilmente mandando la porta 25. I processi client e server comunicano con il protocollo TCP con C che apre la connessione con S. Vi sono 3 fasi: handshaking, trasferimento dell'info e closure. L'interazione è di tipo command/response: si invia codificato in ASCII da

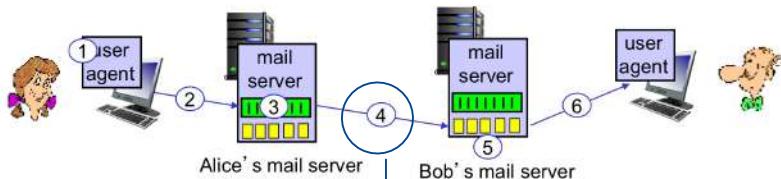
U invia comandi e S risponde 7 hit.

↓ Alice vuole mandare un messaggio a Bob

- 1) Alice uses UA to compose e-mail message "to" bob@someschool.edu
- 2) Alice's UA sends message to her mail server; message placed in message queue
- 3) client side of SMTP opens TCP connection with Bob's mail server

- 4) SMTP client sends Alice's message over the TCP connection

- 5) Bob's mail server places the message in Bob's mailbox
- 6) Bob invokes his user agent to read message

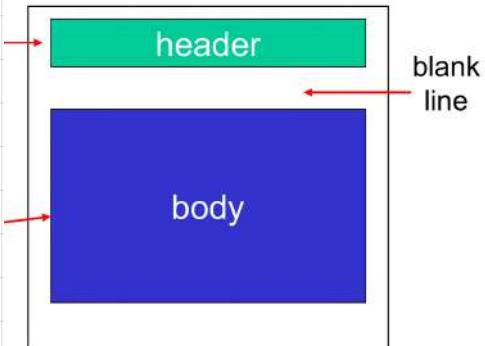


```

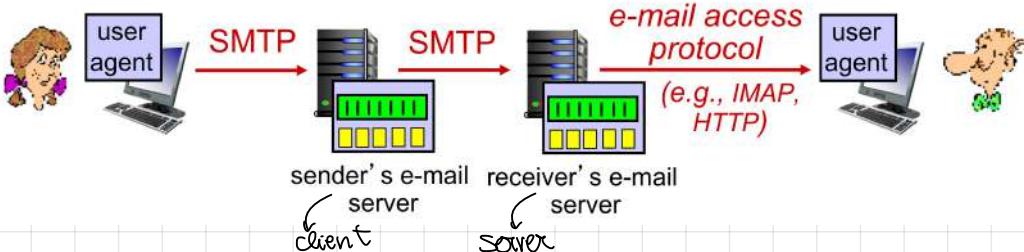
S: 220 hamburger.edu
C: HELO crepes.fr
S: 250 Hello crepes.fr, pleased to meet you
C: MAIL FROM: <alice@crepes.fr>
S: 250 alice@crepes.fr... Sender ok
C: RCPT TO: <bob@hamburger.edu>
S: 250 bob@hamburger.edu ... Recipient ok
C: DATA
S: 354 Enter mail, end with "." on a line by itself
C: Do you like ketchup?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

Confronto con HTTP: Nel caso di HTTP C si collega ad S per scaricare i dati mentre se ci si parla con S per inviare un messaggio. Nel primo caso si ha un comunicamento pull mentre nel secondo push. Entrambi hanno messaggi e comandi ASCII e SMTP usa commessioni persistenti (invia messaggi con la stessa connessione). In HTTP ogni oggetto è encapsulato in un messaggio di risposta mentre per SMTP quando C invia messaggi ad S ollega anche tutti gli attachments (tutti i messaggi). In HTTP si ha una riga con solo ":" come terminazione mentre in HTTP la linea vuota.

| Formato messaggio



header: intestazioni del tipo (To:, From:, Subject:)
Keyword: valore
Specificati da utente ma Δ non è un utente
Body: msg in ascii + file che include testo + attachments.

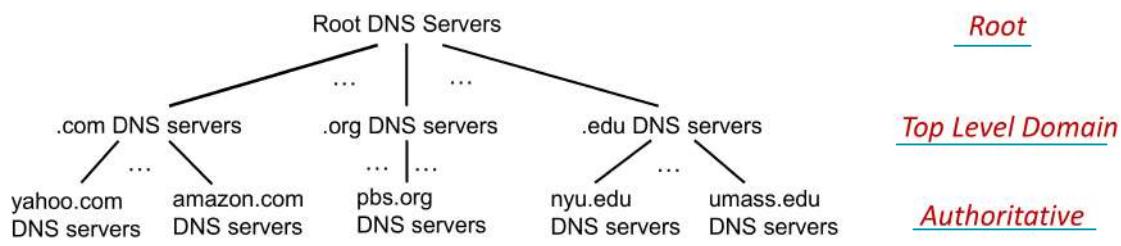


Il mail never potrebbe anche girare sui computer di Alice e Bob ma questo approccio è decaduto con l'avvento del possesso di più dispositivi a persona e quindi lo user agent si collega al mail server tramite SMTP che poi provvede ad inviare il messaggio. Questo vale anche dal lato di Bob. Non c'è nessun vantaggio ad avere il mail never im locale in quanto i messaggi arrivano ed ora gliel'arrivo e quindi il pc dovrebbe stare sempre acceso. Si usano quindi i mail server messi a disposizione dai provider a cui i client si collegano tramite il protocollo IMAP per scaricare la posta e per gestire la posta sul server. Si può accedere alla posta anche tramite protocollo HTTP con l'interfaccia web

• DOMAIN NAME SYSTEM

Utilizziamo UDP in quanto manda un messaggio alla volta e quindi è efficiente. Ricordiamo che gli host sono identificati dagli indirizzi IP (numericci). Questo non è utilizzabile quotidianamente. Per questo sono stati introdotti gli indirizzi simbolici (www.google.it) che devono avere una voce su rete globale (analoga - avete d CF delle persone). Per semplificare dividiamo il dominio globale in top-level domain (.com, .it, .eu) e per ciascuno di essi ne definisce un'autorità di registrazione a cui è necessario registrarsi per registrare un dominio e questa fornisce o nega l'autorizzazione. Il dominio costa un canone. Una volta registrato un dominio è sufficiente garantire l'unicità nei sottodomini e nei livelli inferiori. Per loro gestore è interna all'organizzazione. Si ottiene quindi una gerarchia di nomi organizzata ad albero.

Questo non è sufficiente però per avere i nomi simbolici



Si utilizza un traduttore (da syn a binary): DNS → Non va bene un DNS server centralizzato per motivi politici e perché costituisce un single point of failure. Inoltre dovrebbe gestire una quantità di traffico immensissima perché non scala — Decentralizziamo: il DNS viene implementato come un BB distribuito nei vari server in diverse parti del mondo: il traffico si distribuisce e l'affidabilità aumenta. È necessario che le varie parti comunicino fra loro: viene

implementato come un'applicazione client-server: C'è richiesta traduzione, S accade al DS → complexity at the edge: niente nel retro.

↓

Servizio principale: hostname to IP translation

Servizi associati: associazione nome reale server ad alias per riconoscimento più facile. Il DNS permette di fare la traduzione da clien a name canonic, il quale può avere tradotto un IP (host e ind. email). Vi sono poi alcuni server web che sono replicati per evitare single-point of failure e distribuire il carico. Per distribuire il carico il DNS manda gli IP dei server corrispondenti al dominio richiedendo la lista di una posizione + client: il web browser accede al primo IP. Il database viene distribuito mantenendo la struttura gerarchica (vedi figura sopra). Ad ogni livello si possono avere più server DNS per ogni sottodominio.

Client wants IP address for www.amazon.com; 1st approximation:

- client queries root server to find .com DNS server
- client queries .com DNS server to get amazon.com DNS server
- client queries amazon.com DNS server to get IP address for www.amazon.com

Si hanno molti server per il root: ~200 neli USA, 13 neli dc che lo implementano. È gestita dalla ICANN che controlla la root e poi ci sono le autorità che gestiscono le parti sovrastanti (.it = Registro.it) → Authoritative: in grado di fornire traduzione → Alcuno un sottodominio autoritativo

local DNS server: C'è un server che local DNS server che può già sapere la traduzione o può chiederla alla cerca di DNS server → mantiene una cache locali che contiene le associazioni hostname - IP. Questo deve avere vicino a C' è quindi sulla stessa rete di accesso dell'utente: per le reti residenziali è alla rete del provider. Nella configurazione di rete se ne specificano 2: il primario ed il secondario a cui ci si rivolge se il primario non va

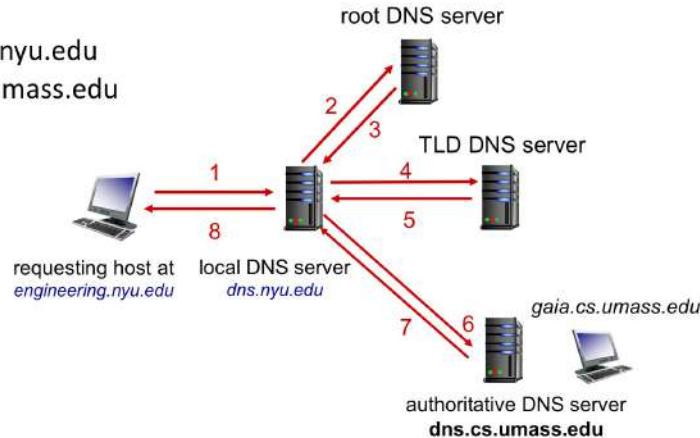
↓ Esempio

Example: host at engineering.nyu.edu wants IP address for gaia.cs.umass.edu

Iterated query:

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

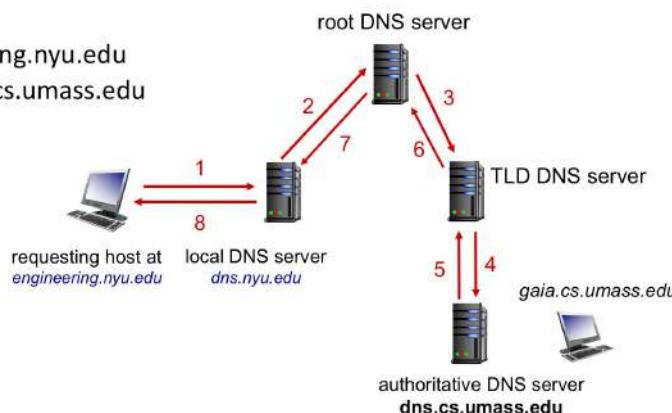
↓ Passaggio ad approccio ricorsivo



Example: host at engineering.nyu.edu wants IP address for gaia.cs.umass.edu

Recursive query:

- puts burden of name resolution on contacted name server
- heavy load at upper levels of hierarchy?



È sempre bene avere le cache: se le informazioni sono in cache non è necessario chiederle in giro occupando rete e consumando energia. Le informazioni vengono eliminate ogni tanto perché obsolette

Per lo scambio delle informazioni in una IP: le richieste sono non essere recapitate però come vantaggio si ha una velocità nettamente superiore fondamentale per questo scopo.

↓ formato geniale record

RR format: (name, value, type, TTL)

→ TTL di vita

→ Vari RR: name e value hanno + significati

- ↓
→ A: supponiamo traduzione hostname - IP → name = hostname, value = IP
- CNAME: traduzione da alias a nome canonico host → name = alias, value = nome canonico
- MX: name = alias posta elettronica, value = mailserver (nome simbolico)
- NS (Name system): name = dominio (unipi.it), value = nome simbolico host autoritativo per quel dominio ⚠ Entrambi nomi simbolici

← 2 bytes → ← 2 bytes →

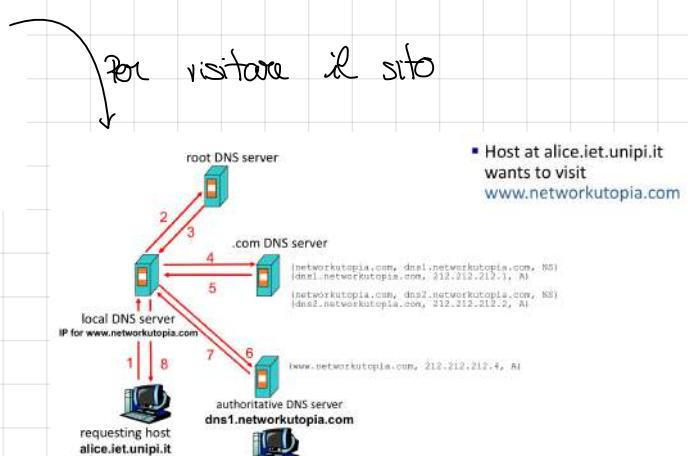
identification	flags
# questions	# answer RRs
# authority RRs	# additional RRs
questions (variable # of questions)	
answers (variable # of RRs)	
authority (variable # of RRs)	
additional info (variable # of RRs)	

↓ insolito record, esempio

Example: new startup "Network Utopia"

società di registrazione: necessaria per registrare

- register name networkutopia.com at **DNS registrar** (e.g., Network Solutions)
 - provide names, IP addresses of authoritative name server (primary and secondary)
 - registrar inserts NS, A RRs into .com TLD server:
(networkutopia.com, dns1.networkutopia.com, NS)
(dns1.networkutopia.com, 212.212.212.1, A)
- create authoritative server locally with IP address 212.212.212.1
 - type A record for www.networkutopia.com
 - type MX record for networkutopia.com



- **APPLICATIONI P2P** (utilizzate per condividere contenuti, comunicazioni telefoniche, IM, streaming) In genere hanno una componente CS ed una P2P. Questo perché prima di scaricare contenuti è necessario dire con chi comunicare: indice → Interlocutore deve avere collegato e si deve sapere il suo IP. Gli indici sono scelti casualmente con un DB Chiave-voto ove la chiave identifica il contento (nove Skype della persona) e il campo voto è l'identificatore. L'indice deve essere organizzato in modo efficiente e

Dove essere aggiornato. È possibile fare operazioni di insert dove si comunicano nome e IP ed il gestore provvede all'inservimento allo stesso tempo per l'eliminazione. L'idea più semplice è quella di farlo centralizzato in cui gli utenti si collegano al gestore che rientra loro gli IP per le loro richieste. Un'applicazione quindi abbiamo una rete centralizzata ed una peer-to-peer. Si ha quindi un'architettura ibrida.

↓ Esempio: Napster

Napster

- P2P system for music sharing
- Centralized index
 - napster.com

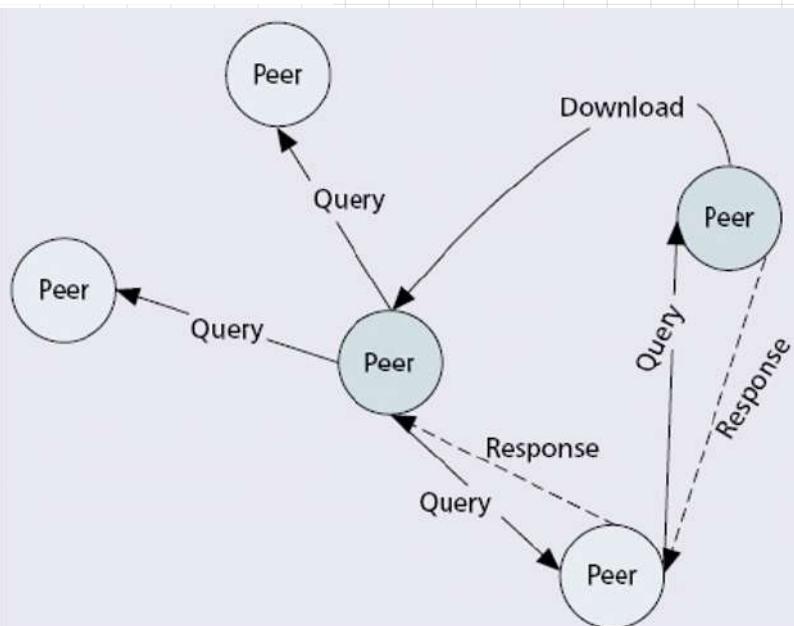
Drawbacks

- Single point of failure
- Performance bottleneck
- Copyright Infringement

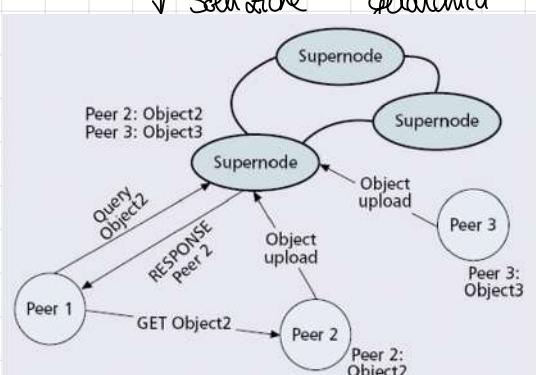


Si passa ad un approccio completamente decentralizzato: indice distribuito su tutti i peer → se metto a disposizione 10 MP3 realizzo un indice con 10 titoli con di fianco il mio IP. Diventa però più difficile trovare i file. Ma come faccio a sapere dove trovare la canzone? Nel nostro caso ogni peer è connesso con un certo numero di peer, i cosiddetti vicini (non geograficamente). Il nostro peer per cercare un file fa una query che influisce sui vicini. Se questi non hanno la ricerca, o poi, se è il contenuto, subito che la ricerca è molto più cercata del resto del circuito. Il problema di overhead in quanto il circuito si riduce e non c'è più il controllo della query. Per imparando solti possibili soluzioni non avere ricercate.

↓ Soluzione peer-to-peer



la query viene propagata ai rispettivi vicini. Poi, se il peer richiedente capisce che i vicini non controllano il contenuto, invia il peer richiedente il contenuto del vicino vicino. Si ha così controllo esatto. Si ha anche che i peer perdono il controllo del contenuto del vicino vicino. Per imparando solo possibili soluzioni non avere ricercate.

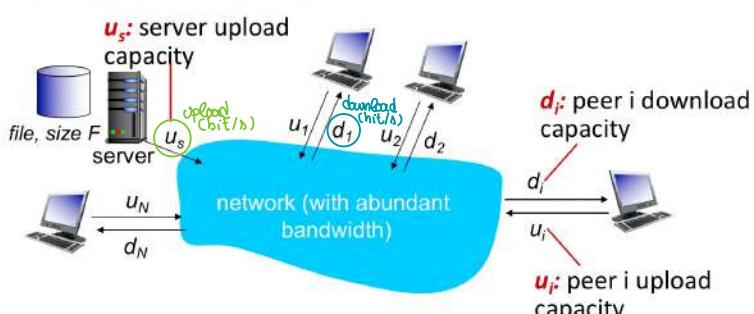


Supernodi: peer cerca gli altri ma realizzano indice. Questo rimane distribuito (su tutti i supernodi) ma non totalmente (<# supernodi < # nodi). Per condurre un contenuto al peer deve registriarsi con un superpeer per cercare in contenuto il peer si rivolge al superpeer che restituisce l'indirizzo del peer dell'oggetto. Se non lo ha chiede agli altri supernodi con il metodo del query flooding.

vogliamo un'applicazione

Q: how much time to distribute file (size F) from one server to N peers?

- peer upload/download capacity is limited resource

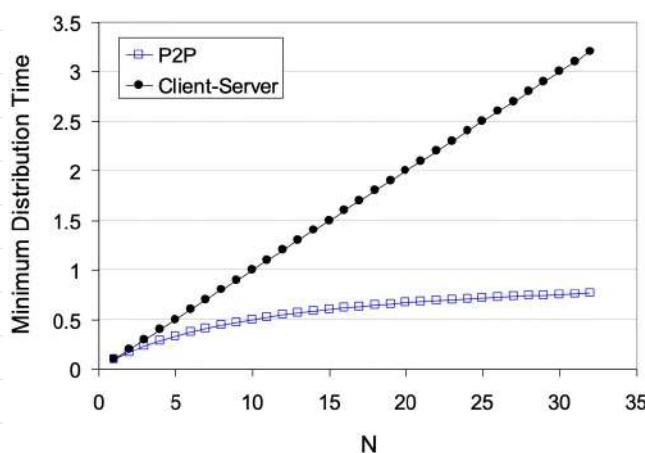


Minimizzazione tempo di distribuzione:
+ da nulla a disposizione
copia ad ultimo client che la scarica.

Chiamiamo DCS il ritardo nel caso C-S: X copie del file distribuite dal server per costituire una limitazione inferiore per DCS. Per il download consideriamo il C con download per tel client sarà F/d_{min} . \rightarrow DCS $\geq \max\{Xt/u_s, F/d_{min}\}$
Nel caso P2P il "server" deve fare immediatamente una copia in un tempo T/u_s . Il download rimane incerto considerando il costo-tempo de d_i in tempo F/d_{min} . Il modello P2P è un modello di ritardo distribuito e quindi possono vedere come un utente che ha a disposizione una velocità di upload pari a $\leq u_i$. Questo deve distribuire NF hit per un tempo dato da $\frac{X}{u_i}$. Allora quindi $t_{min} = \frac{X}{u_i}$

$$D_{P2P} \geq \max\{F/u_s, T/u_s, NF/(u_i + \epsilon u_i)\}$$

\downarrow per $X \rightarrow \infty$



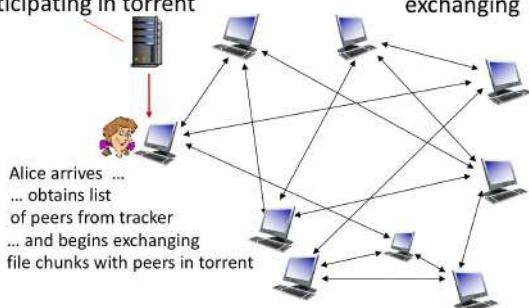
$$\begin{aligned} \text{con } u_1 = u_2 = \dots = u_n = u \\ F/u = 1h \\ u = 10u \\ d_{min}, u \end{aligned}$$

\downarrow Utilizzo

Vogliamo scaricare utilizzando più peer contemporaneamente

tracker: tracks peers participating in torrent

torrent: group of peers exchanging chunks of a file



per facenti parte del torrent. Questa lista è mantenuta più o meno aggiornata da messaggi keep-alive. Questo non è un problema a causa della redundanza. I file sono divisi in chunks da 256 kB. Una volta ricevuta la lista di peer Alice prova ad aprire più connessioni TCP possibili con i peer.

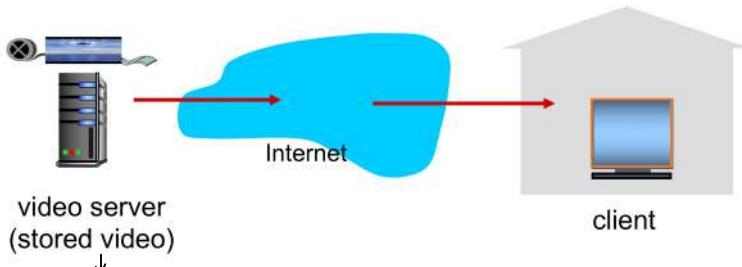
BitTorrent: invece di peer che condividono un file, Torrent, che possono avere già una copia completa, non averla od averla parziale. Se neamo avere una copia completa il torrent si esaurisce. Il torrent è dinamico con continue entrate ed uscite dei peer. I tracker tengono traccia dei peer e vengono contattati con l'ip presente nel file torrent. Questi forniscono una lista di ip di

I peer con cui ha stabilito una connessione vengono detti vicini e può comunicare con loro. Alice vuole ricevere tutti i chunks e non è interessata al loro ordine. All'inizio Alice non ha chunks e richiede ai vicini la lista dei chunks in loro possesso. Per la ricezione, Alice pone dei chunk meno frequenti. Quando Alice ha scaricato dei chunk può fare da server selezionando i peer dando la priorità a quelli più veloci. La graduatoria viene stilata ogni 10 secondi. Se però faccio così mi fissa i vicini. Ogni 30 secondi viene scelto un nuovo vicino a cui inviare chunk. La speranza è quello di trovare un vicino più veloce dei precedenti. Questo metodo prioritari i peer veloci facendoli comunicare con i peer veloci. I free rider che vogliono solo scaricare sono molto pericolosi.

STREAMING

Lo streaming video è la categoria che consente la più ampia quantità di banda in rete. Un video è una sequenza di immagini visualizzate con un certo rate (FPS), ciascuna di esse riconoscibile in array di pixel a cui sono associate informazioni. Tutto è tradotto in hit trannei con un certo bitrate che viene ricavato da chi guarda il video. Il bitrate generalmente è molto elevato e dipende dalla qualità del video. Molti dei pixel di ciascuna immagine sono uguali (correlazione spaziale) ed anche tra un'immagine e la successiva si hanno pixel comuni (correlazione temporale). Questo permette di risparmiare hit. Vi sono vari tipi di codifica: CBR = bitrate costante, VBR = bitrate variabile → vengono usate le correlazioni per ridurre il bitrate. Vi sono diverse tecniche di codifica che permettono di avere hit differenti e quindi qualità differenti.

Streaming di contenuti registrati



Problema: internet è una rete a pacchetti best-effort in cui i pacchetti vengono trannei con un certo ritardo (jitter). Questo influenza anche la trasmissione video, anche se trannei in modo diverso. C'è poi il tranneo in modo uniforme → buffering buffer: rimuovere il ritardo immagazzinando un po' di video.

Recentemente è stata introdotta DASH (Dynamic Adaptive Streaming over HTTP). Lo streaming è adattivo e la codifica è adattata alle condizioni di rete. Si codifica i chunk del video con bitrate (= qualità) diverse e fornisce il manifest in cui il chunk è riconosciuto l'URL di codifica. C'è richiede un chunk con una determinata codifica e periodicamente misura la banda C-S. Se il throughput è basso ed il buffer si sovra, C'è che si cambia codifica con bitrate più basso.

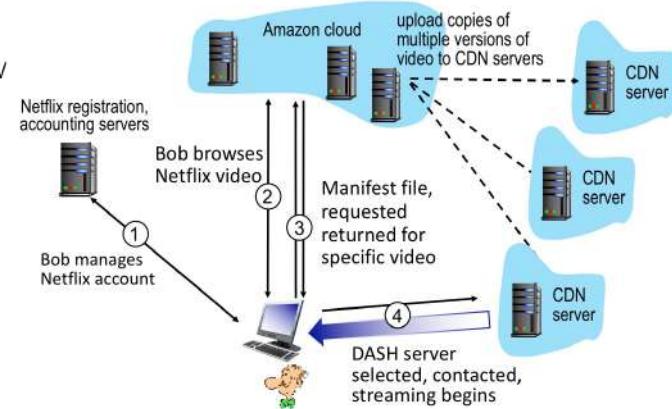
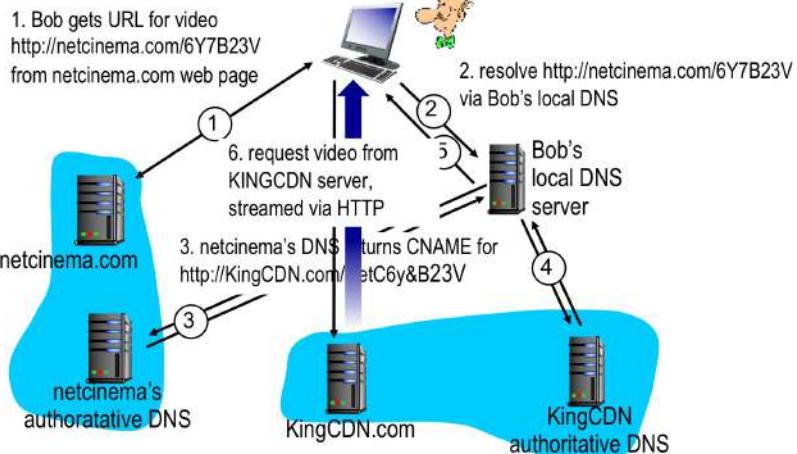
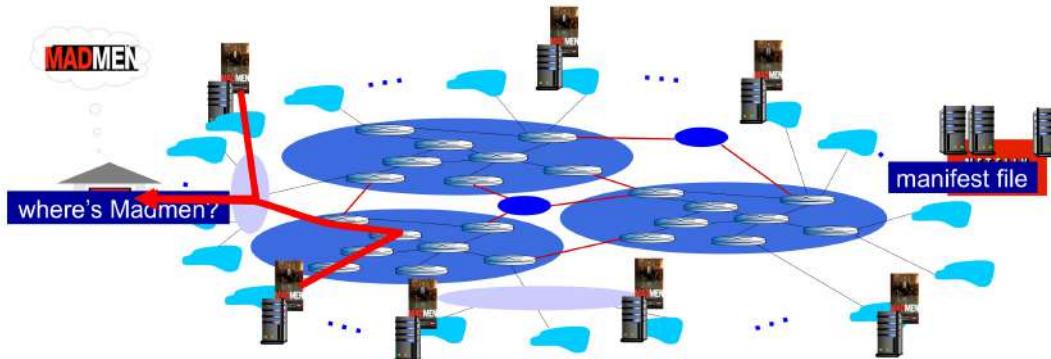
↓ Organizzazione del network (garantisca qualità)

Potiamo pensare di fare un grande distributore che serve tutti che però presenta le criticità già viste ed in questo caso si vede che influenza il ritardo e la qualità (in modo negativo, > perdite). Inoltre la soluzione non scala: tante copie dello stesso file a utenti diversi → alto traffico su internet → si fanno CDNs (Content distribution networks) che sono di 2 tipi. Il primo è enter deep in cui i server sono nella rete di accesso di C, il 2° è

C richiede a S video il flusso video. Il flusso viene inviato a C che lo manda verso lo utente. Utente crea un buffer per non doverne bloccarsi nella visione.

bring home in cui si hanno cluster più grandi in modo vicino vicino alla rete di caccia di c.

↓ Esempio



• STORIA → vedi le slide

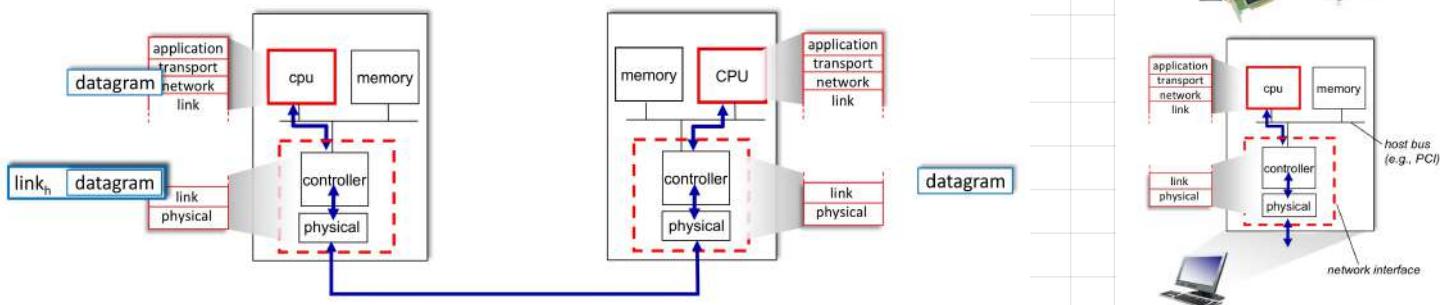
Reti a connessione diretta



→ link di comunicazione fra host: ci sono codi fissi le informazioni in hit e successivamente variano in valori di tensione. Gli impulsi si propagano sul link e raggiungono la destinazione che si occupa della decodifica → Trasmettitore e ricevitore che estranne le spettri
↓

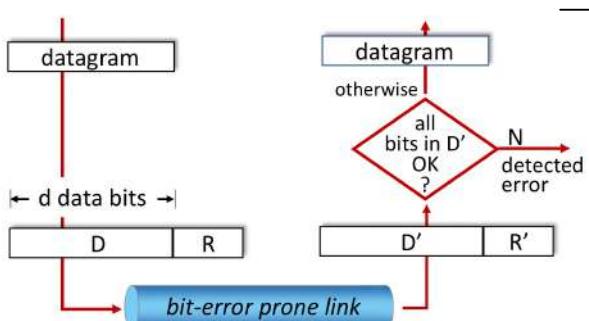
livello fisico: codifica e trasmissione non è ideale → Attenuazione e distorsione: il ricevitore riceve il segnale distorto e prende delle decisioni sul segnale ricevuto → hit error rate: nr. hit sbagliati ricevuti
↓ ulteriore livello

data link layer: il hit error rate aumenta con la lunghezza della sequenza → Viene diviso in frame (framing). Questi devono avere simboli iniziali per ricostituire il file corrispondente e vogliamo sapere se i frame ricevuti sono corretti. L'operazione si chiama error detection e viene fatta sia in invio che in ricezione vengono fatte in fase di ricezione. Se vi sono errori può essere fatta un'operazione di error correction oppure il frame può essere riconosciuto. Questo è inutile se il link è molto rumoroso ma rende invece con alta probabilità gli errori se il link è stabile. Il link layer implementa anche funzionalità di flow control e funzionalità half duplex o full duplex (T/R / T/R) → livello data link implementato in parte in hw nella scheda di rete ed in parte in sw sopra.



ERROR DETECTION

Ricevitore deve capire se frame ricevuto è integro



→ Vengono aggiunti al hit di redundanza prodotti con un algoritmo. Dopo che il pacchetto viene inviato sul link (non affidabile). Il ricevitore riceve il pacchetto nella forma D'R' (potrebbero essere + da DR inviati). Il ricevitore esegue lo stesso algoritmo per generare la redundanza di T su D'. Se R' generato = R' ricevuto allora i dati sono corretti. Altrimenti potrebbero esserci errori nella transmisione di D o di R ma il ricevitore non può discriminare fra questi casi e quindi esegue il frame come esatto. La probabilità di errore dipende dall'algoritmo e dalla "spesa" (hit di redundanza)

Algoritmi

Se hit error rate è bassa posso usare algoritmo poco complesso

codice a parità: Si devono trasformare d hit. Si aggiunge 1 hit detto hit di parità → parità pari: hit aggiunto fa diventare numero i pari

Ricevitore ↓ → parità dispari: hit aggiunto fa diventare numero i dispari

Se accordati su parità pari R conta hit=1. Se pari tx corretta altrimenti esatto. Il costo è molto basso ma si ha esatto

quando 2 bit sono invertiti \rightarrow Si ha quando bit erano rotti e non hanno (BUS, incidento a wire)

metodo di checksum:

utilizzato dai protocolli di trasporto. si hanno d bit divisi in blocchi da m bit di cui si fa somma bit a bit ottenendo sempre risultati su m bit. Questo risultato viene inviato in aggiunta ai dati. Il ricevitore confronta il checksum ricevuto con quello calcolato e se sono uguali assume che i dati sono corretti.

Il costo è \rightarrow una la probabilità di errore è c del bit di parità \rightarrow solo protocolli di trasporto

D = bit di dati considerati come numero

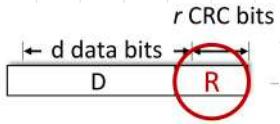
G = generatore di $r+1$ bit \rightarrow Accordo tra S ed R

\downarrow

Vogliamo generare bit di redundanza R : consideriamo la struttura D con la redundanza R . Questa può avere vista come $D \cdot 2^r$ (shift a sx) $\text{XOR } R$

\downarrow Generazione hit

Si devono scegliere x bit / $\langle D, R \rangle$ sia divisibile per G . Vogliamo quindi $D \cdot 2^r \text{ XOR } R = nG \rightarrow D \cdot 2^r = nG \text{ XOR } R$ e, dividendo per G vogliamo che il risultato sia 0: $R = \text{resto divisione } (D \cdot 2^r)$. L'operazione in hw è veloce e quindi non ha un costo computazionale alto. La verifica viene fatta in S ancora attraverso la divisibilità.



ERROR CORRECTION

Trasmettendo stringhe binarie, se sappiamo quali bit sono alterati è sufficiente invertirli \rightarrow Error correction codes ci dicono quali bit sono alterati. Vengono aggiunti hit ecc (error detection and correction) che costano di più di quelli redundanti. Questi vengono calcolati da S e R e dal calcolo di R si scopre correttamente ed eventuali bit alterati.

\downarrow

Parità bidimensionale:

		row parity		
		$d_{1,j}$	$d_{1,j+1}$	
		$d_{2,j}$	$d_{2,j+1}$	
column parity		\dots	\dots	\dots
$d_{i,1}$	\dots	$d_{i,j}$	$d_{i,j+1}$	
$d_{i+1,1}$	\dots	$d_{i+1,j}$	$d_{i+1,j+1}$	

\rightarrow Bit disposti a matrice. Per riga si calcola un bit di parità e poi si fa anche per colonna. Si calcola poi il bit di parità della gerca di redundanza che deve essere = al quello della colonna di redundanza. In totale si aggiungono $(j+1)r$ bit. Esempio, no errors: 101011 | 1
111100 | 0
011101 | 1
101010 | 0

detected and correctable single-bit error:	101011
101011	1
111100	0

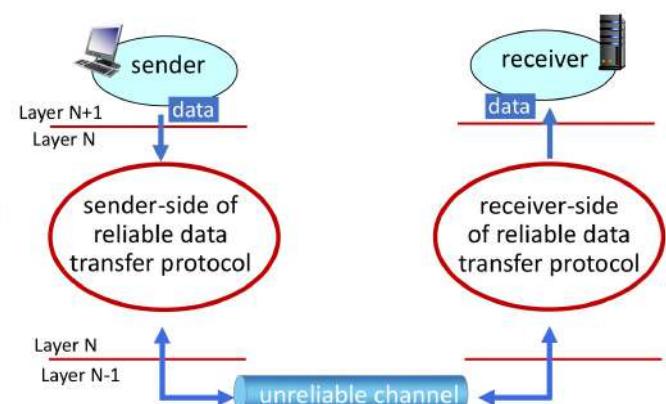
Meno se hit erano molti elevata per resto molto alto

• RELIABLE DATA TRANSFER

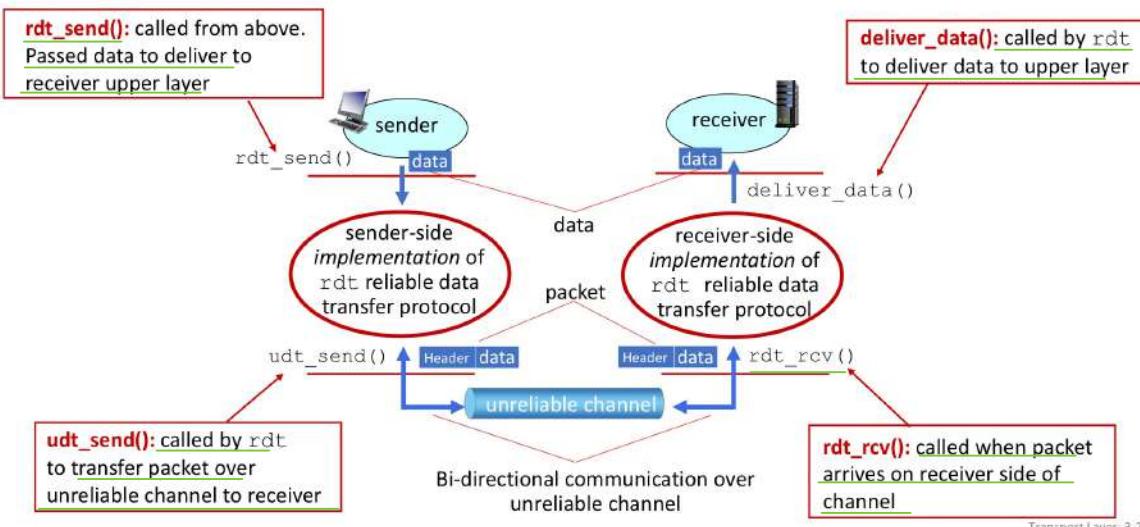
Cosa vorremo



cosa faccio



Per rendere il link affidabile occorre fare operazioni a seconda del livello di affidabilità. Si possono infatti avere errori su un singolo bit, pacchetti veri e propri ricevuti, pacchetti che sono ricodificati dal link sottostante. Se non accade niente avviene di ciò che accadrà ai dati sul link.



Supponiamo solo errori su singoli bit, trasformando i missaggi e cercare a stati finiti.

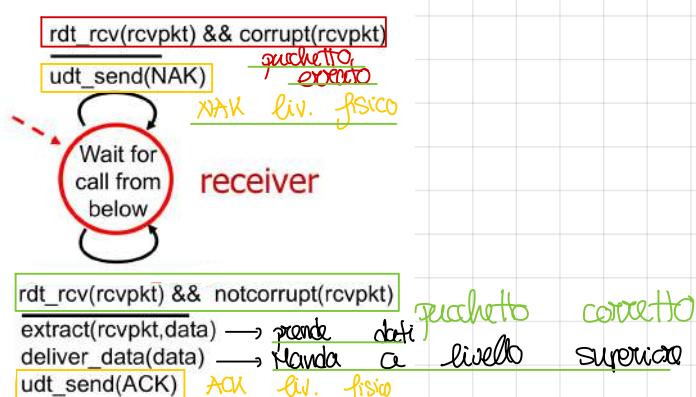
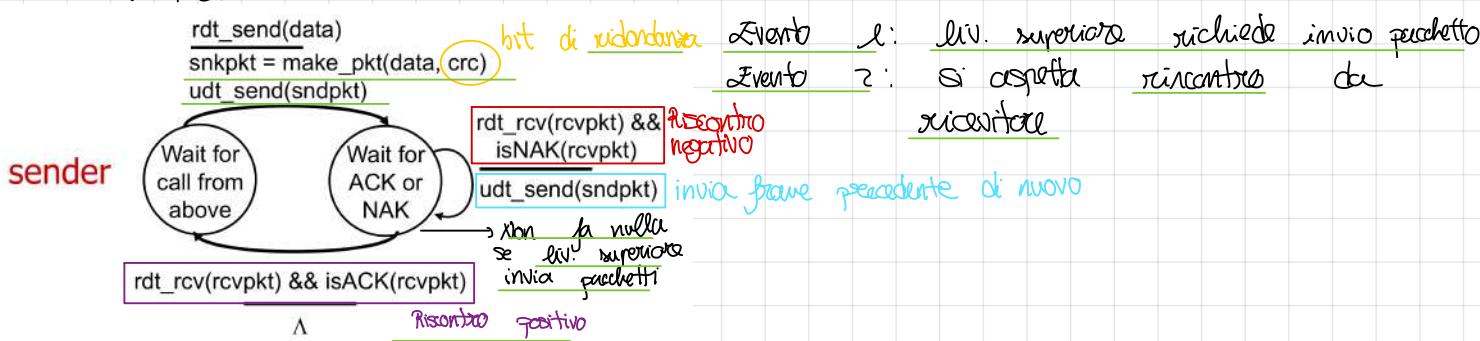
numero di stati finiti tra cui si hanno transizioni generati da eventi e che generano altri

rdt 1.0: Vogliamo un protocollo di trasmissione affidabile (no errori, no packet loss) → S invia dati sul link ed R li riceve



rdt 2.0: Possiamo capturare bit invertiti → Acknowledgements (ACKs): R informa S che ha capito, negative acknowledgements (NAKs): R dice ad S che non ha capito e quindi chiede retransmissione. → pacchetti sui cui è stato ricevuto NAK vengono ritrovati

⚠ Appoco stop and wait: S invia un pacchetto < aspetta ACK, NAK di R
↓ MSF



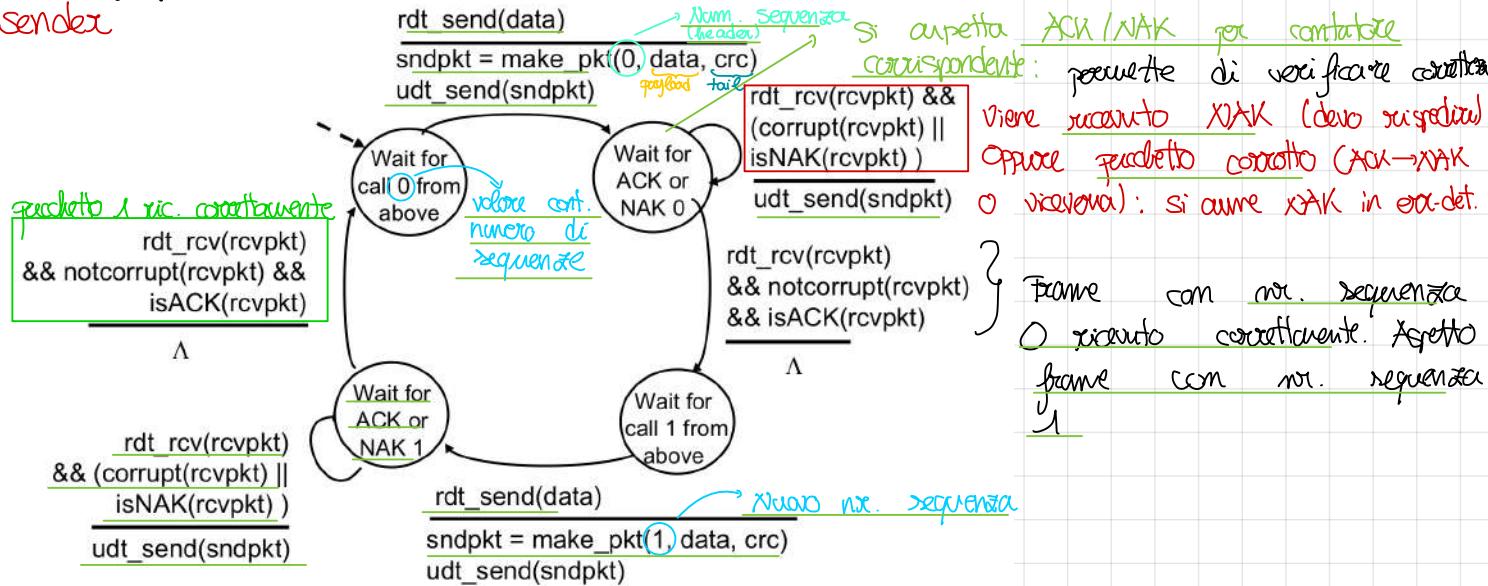
⚠ NON FUNZIONA: Abbiamo un errore che ACK e NAK siano ricevuti correttamente, il che non è vero → può accadere inversione hit. Non sono duplicati se S invia ACK ed R ricevere NAK

Ricevitore non ha memoria e non riceve duplicati: INACCETTABILE

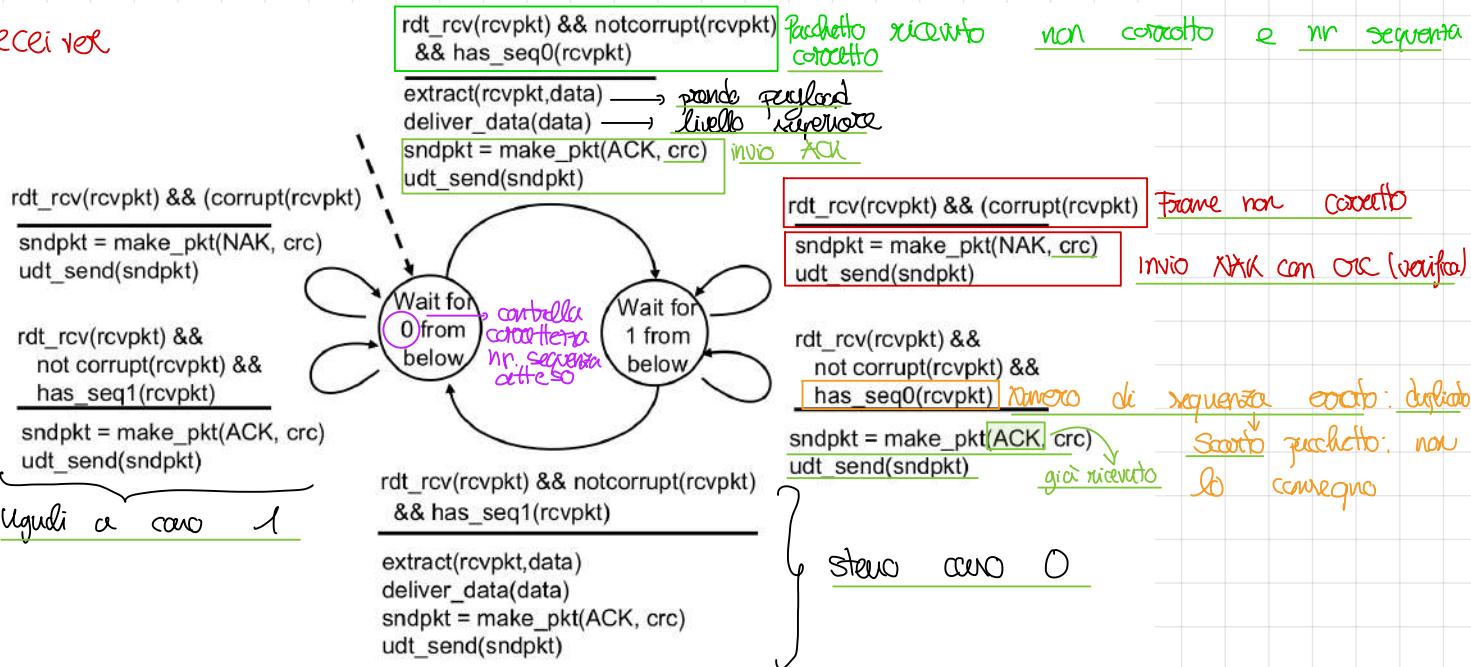
Ad ogni pacchetto viene assegnato un numero di sequenza: se lo ha già ricevuto elima via il pacchetto. Il campo dedicato a questo numero può essere lungo a piacere tenendo però presente l'overhead generato. È sufficiente avere un hit in quanto può trattare un solo pacchetto e i hit permette di rappresentare i valori 0 e 1.

rdt 2.1:

Sender



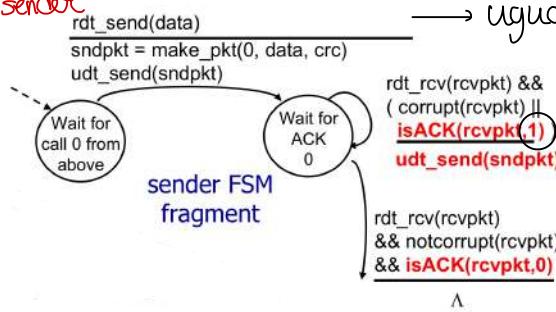
Receiver



Sender: Aggiungi \neq sequenza è 0,1¹, controllo su ACK/NAK → Raddoppio nr. stati
Receiver: Verifica duplicati

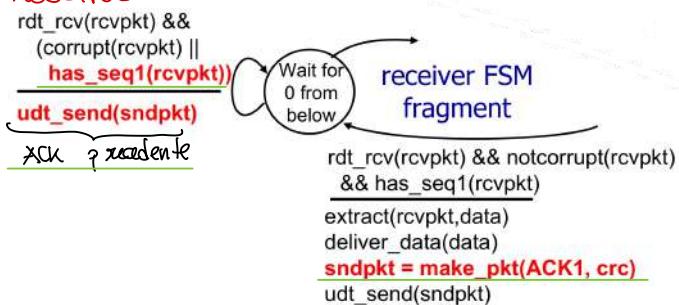
rdt 2.2: Si usa un protocollo che una nodo ACK (protocollo TCP). Se non si capisce si dice "ho capito la frase precedente" → ACK precedente

sender



relativo a pacchetto precedente
equivale a NAK

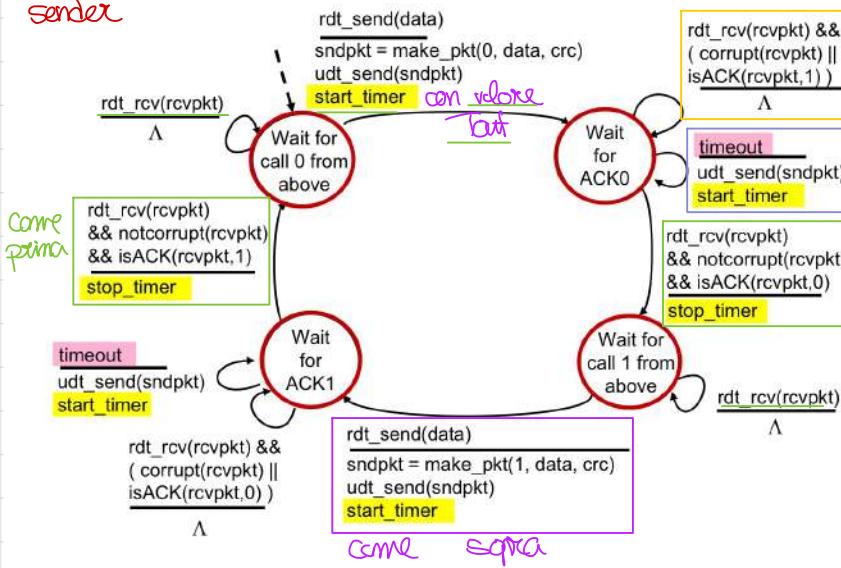
RECEIVER



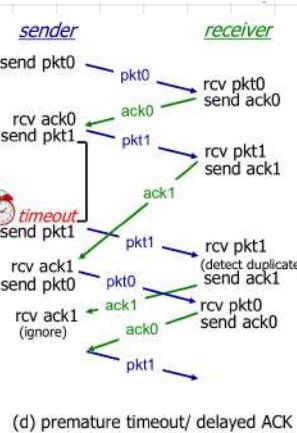
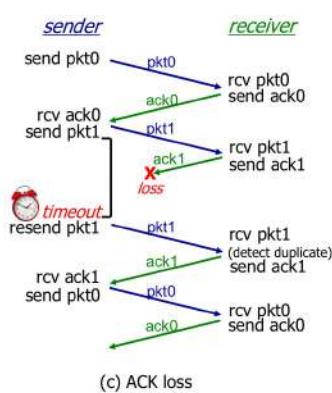
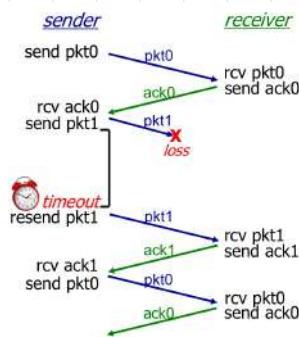
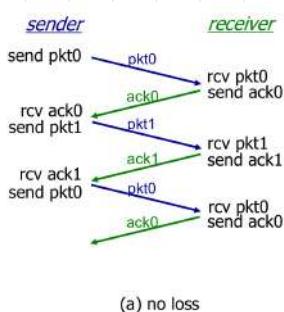
rdt 3.0:

Il pacchetto può non avere ricevuto da R (zero in connessioni punto-punto) → packet loss dovuto a router intermedio che lo scatta a causa della congestione. Come fa R a sapere che doveva ricevere un pacchetto perso? Si introduce timer impostato in base a $RTT = \frac{L}{R} + \frac{2d}{r} + \frac{\text{Loss}}{R} \cdot \text{bitrate}$ → Si sceglie $Tout > RTT$ in modo che non sia troppo grande o troppo piccolo per avere tolleranza.

sender



↓ Simulazione



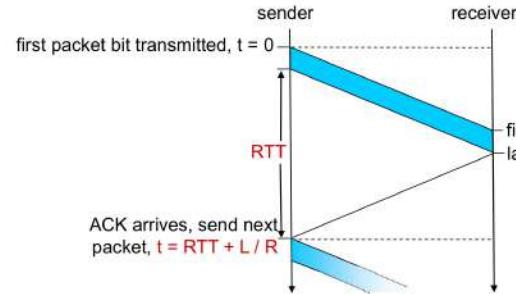
ACK corretto/precedente: Assetto timer per rimandare
Tout: copia non ricevuta.
Ritrasmetto e ricalco timer

Pacchetto integro, ACK corretto:
STOP TIMER

Prestazioni (stop-and-wait): $U_{\text{sender}} = \frac{\text{throughput}}{\text{capacità link}}$

↓ Esempio

Link 1Gbps, 15ms delay, 8000 bit packet



$$\rightarrow \frac{L}{R} = 8 \mu s$$

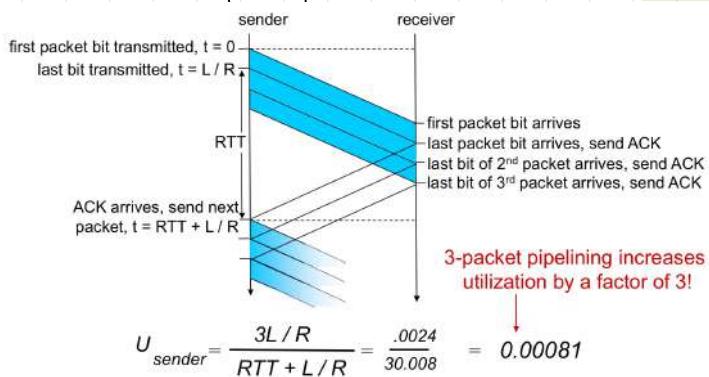
$$U_{\text{sender}} = \frac{C/R}{RTT + L/R} =$$

$$= 0.00027$$

⚠ troppo poco

Tempo di attesa troppo lungo: funziona bene solo se il ritardo di propagazione è piccolo (tranne che). In questo caso U_{sender} è molto elevata (rate locale, senza ritardi).

S mandano più pacchetti insieme (w) senza aspettare ACK → Approccio pipelining.

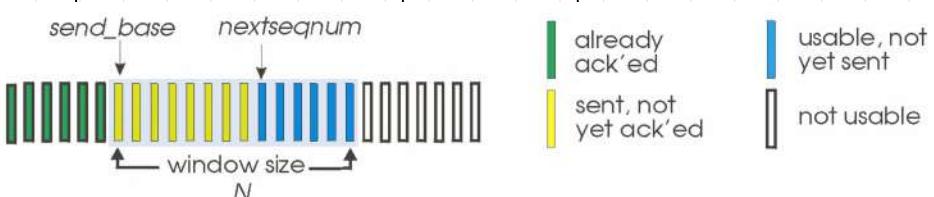


Scegliendo w più grande è possibile soffocare (o quasi il link). C'è però un prezzo da pagare. Operando a pipelining si devono tenere in memoria tutti gli n pacchetti inviati (anziché 1 nel caso stop-and-wait). Il ricevitore deve indicare sopra quando mandare ACK.

• PROTOCOLLI PIPELINING

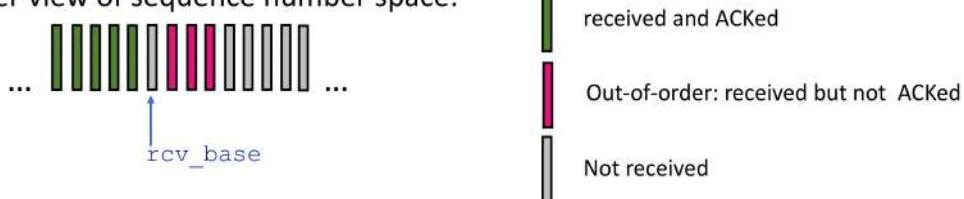
C'è back N: ACK cumulativi, "Ho ricevuto correttamente e in ordine fino a 3". Questo si applica anche ai successivi. Si deve mantenere un timer e cifrare tutti i pacchetti per cui non ha ricevuto ACK.

S può inviare al più N pacchetti consecutivamente senza aspettare ACK (Finestra).



(≤ N). Quando nr pacchetti inviati = N: req chiusura. nextseqnum rappresenta prima posizione inviabile. Il ricevitore fa error detection con numero ultimo pacchetto ricevuto in ordine. recv_base rappresenta n° pacchetto che R si aspetta di ricevere. Se R riceve un pacchetto fuori ordine più scaricare i successivi oppure può bufferizzare.

Receiver view of sequence number space:



Selective repeat.

ACK + pacchetto. S ha un timer + pacchetto inviato. Questi timer sono ottenuti lati su utilizzando un solo timer hw. Se S non riceve ACK per un pacchetto viene inviato di nuovo solo quello. R deve tenere tutti i pacchetti in memoria.

sender

data from above: → Events

- if next available seq # in window, send packet

timeout(n):

- resend packet n , restart timer

ACK(n) in [sendbase, sendbase+N]:

- mark packet n as received
- if n smallest unACKed packet, advance window base to next unACKed seq #

receiver

packet n in [rcvbase, rcvbase+N-1]

- send ACK(n)
- out-of-order: buffer
- in-order: deliver (also deliver buffered, in-order packets), advance window to next not-yet-received packet

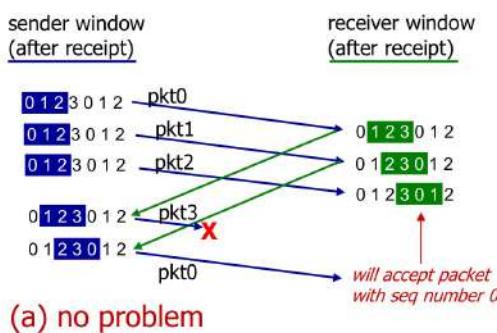
packet n in [rcvbase-N, rcvbase-1]

- ACK(n)

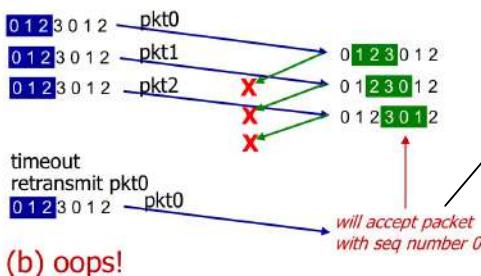
otherwise:

- ignore

↓ Problema zero R



→ 2 hit il numero di sequenza → finestra lunga 3. I pacchetti sono numerati da 0 a 3



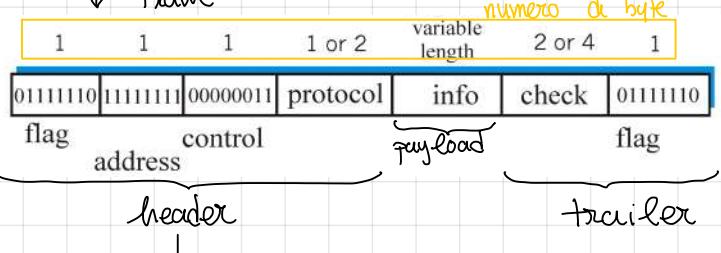
Errore: metto lo 0 dopo il 3 e quindi shuglio perché doveva cedere all'inizio della sequenza

Il spazio dei numeri di sequenza è troppo piccolo rispetto alla dimensione della finestra. Questa viene scelta preliminarmente. Il spazio dei numeri di sequenza deve avere la potenza di $2 >$ dimensione della finestra.

• PPP (Point-to-point protocol)

Implementa framing, ovvero i datagram vengono messi in frame gestiti indipendentemente. Se i datagram sono più grandi dei frame questi vengono spezzettati. Il protocollo insieme garantisce link transparency, ovvero la possibilità di mandare ogni tipo di link. Garantisce error detection e connectionless (ricorda e segnala errori). Inoltre, garantisce network layer address negotiation, fornendo quindi gli indirizzi IP agli host. → Servizio non relazionale: no error correction o ritrasmissione. Nei link punto-a-punto l'affidabilità è bassa (frame, FO) < Si suppone che seppa ai sicurezza. Un protocollo di trasporto (TCP) che garantisce affidabilità. Non permette immobile comunicazione punto-multipunto (importante per linee seriali) & flow control.

↓ Frame



CAMPI

check: Bit di redundanza in fondo poiché calcolati on the fly, dopo che pacchetto è stato trasmesso, rim bhw. → 16 o 32 bit per accettazione

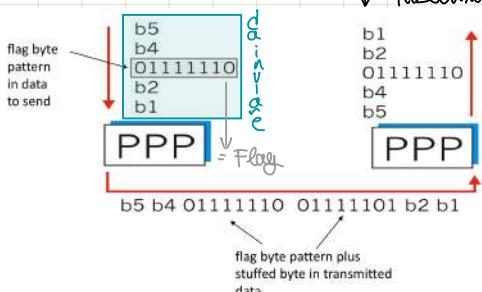
flag:

Iniziale → seconda inizio frame

All'utente possono essere negoziati:
div. campi info e check e invio
campi inutili

- Final** → indica fine frame
- address**: indirizzo ricevitore segnale per tutti. Può essere usato per point-to-multipoint.
- control**: possibili usi futuri
- protocol**: protocollo di livello superiore a cui devono essere forniti dati
- info**: dim. variabile [0, 1500] byte negoziabili

Problema se campo info = flag (scambiato con fine frame).



byte stuffing / unstuffing

T si accorge byte = flag e inserisce byte di escape prima di seq. = flag

- Sender (byte stuffing) byte di escape
 - 01111110 → 01111101 01111110
 - 01111101 → 01111101 01111101
- Receiver (byte unstuffing)
 - 01111101 01111110 → 01111110
 - 01111101 01111101 → 01111101

R butta via la sequenza di escape e considera ciò che è dopo come byte di dati e non come flag di fine.

MULTIPLE ACCESS PROTOCOLS

Per reti grandi si usa un link condiviso tra tutti i nodi di tipo broadcast → reti ad accesso multiplo: ethernet, reti wi-Fi, reti cellulari, TV via cavo, reti satellitari. Chiaramente la comunicazione va regolata per non avere conflitti

multiple access protocol

Evitare che 2 computer trasmettano contemporaneamente sullo stesso link (collissioni): algoritmo che determina il turno (accesso in mutua esclusione). Si ha un canale aggiuntivo per inviare la segnalazione (out of band) oppure i segnali sono inviati sullo stesso canale (in band). Il protocollo deve avere alcune caratteristiche. R = rate link condiviso, se c'è un solo nodo che trasmette vorremo che mane tutti gli R bit. Inoltre, se ci sono n nodi che vogliono trasmettere, vorremo di cui dovrebbe ricevere R/n link. Vogliamo che il protocollo sia decentralizzato. non c'è un nodo che coordina la trasmissione e non c'è una sincronizzazione stretta tra nodi (orologio comune). Infine, il protocollo deve essere semplice

3 categorie

Partizionamento del canale:

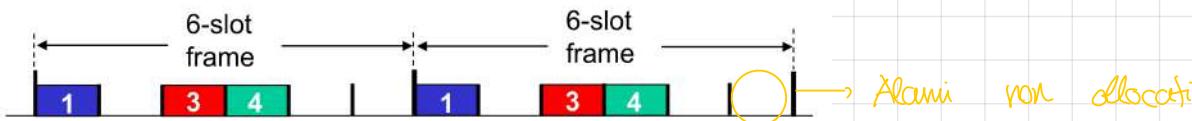
Si divide il canale in tanti pezzi (tempo, f, codice) ed a ciascun modo viene assegnato uno slot (di tempo, canale di frequenza): R si sintonizza su f che vuole andare

Accesso coniale:

non c'è un coordinamento stretto

Time Division

→ **TDMA**: tempo diviso in slot di durata prefissata toll da permettere trasmissione di pacchetto ed eventuali raggruppamenti in frame. Si ripetono nel tempo



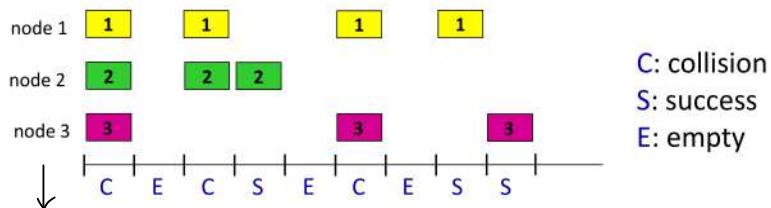
Se c'è un solo nodo questo non riusce ad uscire tutto il perché gli viene assegnato un solo slot. Il metodo è fair e obbliga a tutti i nodi gli stessi slot. I nodi, per trasmettere nello slot giusto, hanno bisogno di clock, e quindi di sincronizzazione il protocollo si può avviare semplice

FDM A: Spettro di frequenze diviso in bande e vi ha la trasmissione contemporanea e il deve dividere con la frequenza giusta

→ Un modo trasmette cercando la lunghezza di trasmettere: collisioni → c'è bisogno di individuarle e risolvere: protocollo CSMA → in arco durante trasmissione, STOP in caso di collisione e ripresa dopo un t casuale. Altri: CSMA/CD, CSMA/CA, Slotted Aloha, Aloha

↓
Slotted Aloha: Assumiamo che tutti i frame hanno la dimensione di uno slot in cui posso trasmettere pby e ACK. I nodi sono sincronizzati (clock) ed è possibile rilevare collisione

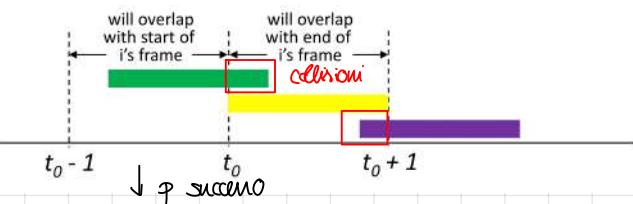
↓
Quando un nodo ha un pacchetto da trasmettere, ottiene presso slot di tempo e comincia a trasmettere. Se si è verificata una collisione (non ricevo ACK) il pacchetto viene ritrasmesso nello slot immediatamente successivo ma se tutti i nodi fanno così si avranno sempre collisioni. Si nega quindi una probabilità P relativa alla trasmissione nello slot immediatamente successivo. Può avere valori
↓ Esempio



Se c'è un solo nodo la banda viene saturata. Nel lungo periodo è fair ma non è completamente decentralizzato perché è necessario un orologio comune. È infine semplice
↓ Efficienza

n nodi, probabilità p. Ogni nodo ha sorgente pacchetto pronto. Il nodo i-esimo ha probabilità $p(1-p)^{n-1}$ di trasmettere (variabile geometrica) da probabilità che un nodo qualiasi della rete trasmitta è data dalla somma delle pi: $n p (1-p)^{n-1}$. Per ottenere la massima efficienza si trova $p^* / p = \max n p (1-p)^{n-1}$ facendo la derivata prima. Studiamo il $\lim_{n \rightarrow \infty} n p^* (1-p^*)^{n-1}$ ottieniamo $1/e = 0.37 = 37\%$

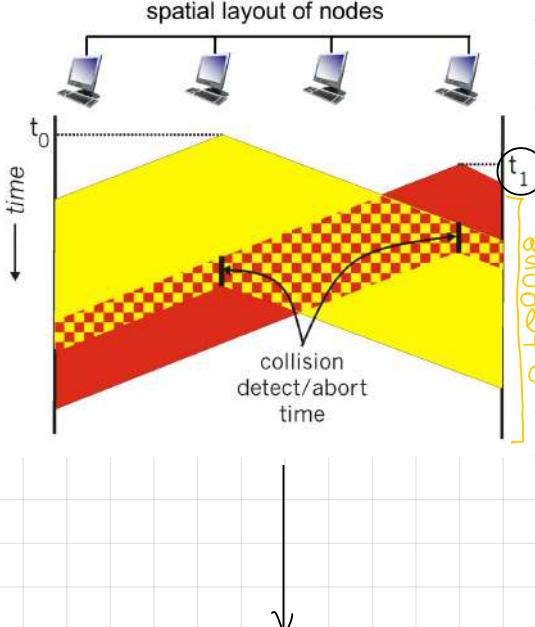
Aloha: Versione non slotted da trasmissione inizia subito e si sovrappone tutti i frame della stessa dimensione (stesso t di trasmissione). C'è collisione anche nel caso di collisione parallela



$$p(\text{nodo trasmette}) \cdot p(\text{nessun nodo in } [t_0-1, t_0]) \cdot p(\text{nessun nodo in } [t_0, t_0+1]) = p \cdot (1-p)^{n-1} \cdot (1-p)^{n-1} = p \cdot (1-p)^{2(n-1)} \xrightarrow{n \rightarrow \infty} \frac{1}{2e} = 18\% \longrightarrow \text{Completamente decentralizzato}$$

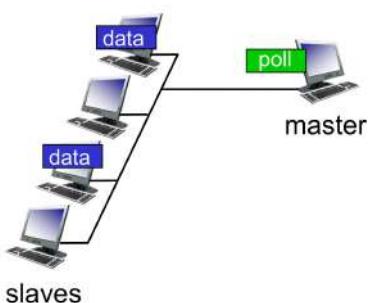
⚠ diuite: nodo non guarda se metto è occupato prima di tranneccere
 ↳ Mecanismo di sensing

CSMA: cariere sensing multiple access → Se channel = idle: tranneccere intero frame;
 se channel = busy: attende che si liber. Questo non risolve però le collisioni.
 perché tutti possono cominciare la tranneccere insieme → Mecanismo di collision detection (CSMA/CD): verifica durante comunicazione e abort. nel caso di collisione →

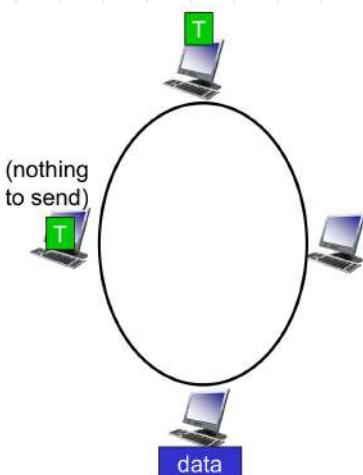


Messo libero: il segnale non è ancora accettato → protocollo eseguito correttamente.
 Il nodo T ascolta il segnale e misura la potenza ricevuta e la confronta con quella attesa. T si aspetta che $P \leq P_{attesa}$ (tralasciando rumore iniziale). Se $P > P_{attesa}$ c'è sicuramente una collisione. Quindi si raffigura tranneccendo un segnale con P_r per far capire all'altro che sta facendo collisione.

Piace a naturare tutto il link in rete di un nodo, è fair, è fully decentralized. I protocoli random access funzionano bene con un solo master (se ci sono tanti nodi: probabilità di collisione molto alta (→ sempre))



Se si guatta la master.

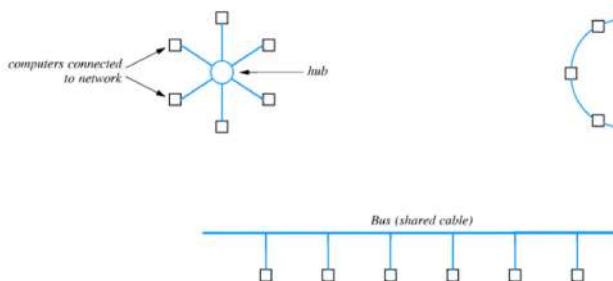


Polling: Nodi master che coordinano e slave che utilizzano. Il master invia un pacchetto di poll in cui chiede se slave ha spedito da tranneccere e fa questo ciclicamente con algoritmo di scheduling interno. Se c'è un solo nodo la rete è sfruttata al 100%. Nel caso di più nodi utilizzano invece 100% - pacchetti poll. Non è fair sempre ma decide il master ed è fair se questo interroga in modo Round Robin. Non è pienamente decentralizzato a causa del master e se non funziona fino a che non si elegge un altro.

Token passing: Perda solo chi ha il token: pacchetto specifico rilanciato alla fine dell'invio del pacchetto e girato a nodo successivo. Al netto del token si ottiene piena utilizzazione ed è fair se il resto che ha il token tranneccete in solo pacchetto. Non è completamente decentralizzato a causa del token stesso.

o LAN (Local Area Network)

Rete locale che connette computer con link condiviso di tipo broadcast. Tali host hanno bisogno un modo per comunicare con un solo host. Le LAN hanno una capacità di 100Mbps ed hanno un bit rate di 100~1000 Mbps.



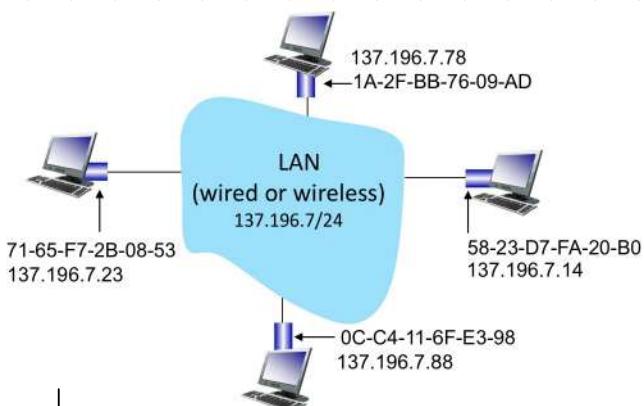
hub: Nodo centrale a cui sono connesi gli altri host.

bus: filo che connette tutti i nodi → nell'host di tipo broadcast. → Alle estremità si mette terminatore per diminuire eco (riflusso segnale).

ring: tutti host si scambiano token in modo circolare.

Indirizzamento

Il metodo è broadcast ma in generali la comunicazione deve essere fatta con un modo specifico. → Ogni nodo deve avere un nome univoco: unico a livello globale. Uniamo il MAC ed i 48 bit che identifica univocamente l'host sulla rete locale. Avendo che è univoco globalmente questo rimane univoco anche cambiando rete. Il MAC identifica Scheda di rete: 24 bit vendor, 24 bit scheda. L'IP viene invece assegnato a livello sw e lo identifica all'interno della rete locale.



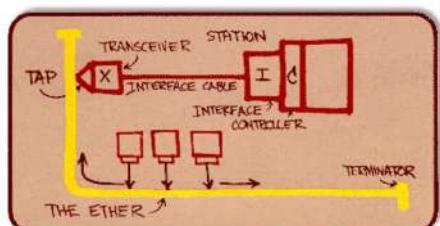
Il nodo che riceve indirizzo del pacchetto e se corrisponde con il suo lo può ce livello superiore. Se invece non corrisponde il pacchetto viene scartato.

Comunicazione unicast su metodo broadcast. Per mandare un messaggio a tutti viene utilizzato l'indirizzo di broadcast.

In modelli promiscua questa non scatta più

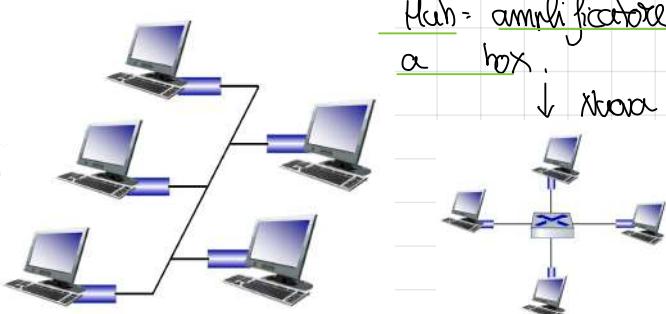
Ethernet

Rete più utilizzata e prima inventata. Si usa ancora perché è la più semplice e la meno costosa. Si raggiungono velocità tra 10 Mbps e 100 Gbps. È un cavo in cui controllate correttamente terminato. Gli host si collegano al cavo in rete (bus). Si ha un controllore che modifica i bit, e un interfaccia che lo collega al transceiver che frange i bit. Questo è collegato al cavo ethernet.



↓ Alternative

Hub - amplificatore
a box.



di potenza → rigenera il segnale: bus in

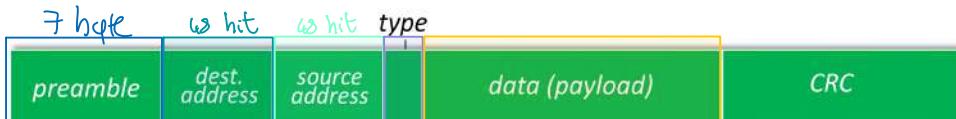
versione: switch

Riceve i frame e li inviada sulla porta corretta



Se due o più nodi trasmettono nelle prime versioni si verifica una collisione:
unico dominio di collisione → corretto da verificare switched

↓ Formato frame

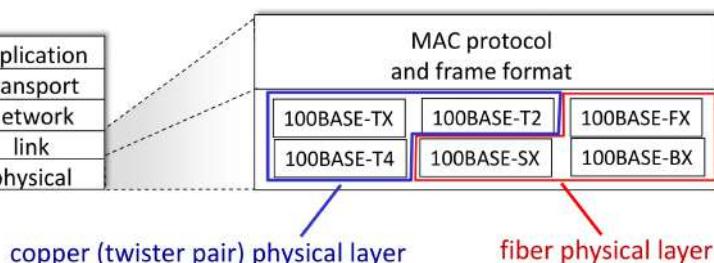


- 7 byte = 10101010 terminati da 10101011. Sono per sincronizzare il ricevitore poiché a livello fisico deve comprendere in base alla velocità del link e deve comprendere nel modo in modo da avere meno errore. Si sincronizza quindi il clock del ricevitore con quello del trasmettitore prima della trasmissione dei dati.
- Il frame inizia dopo il preambolo.
- Se dest address - destinatario frame viene esempiato
- Protocollo di livello superiore a cui farà payload.
- Dimensione minima = 60 byte e max = 1500 byte.

↓

Connessioni, quindi no handshake iniziale, immediata, infatti ci sono CRC per error detection che se dà esito positivo si accetta il payload mentre se dà esito negativo allora il frame viene scartato. Utilizza il protocollo CSMA/CD. La scheda di rete (NIC) riceve un pacchetto da livello superiore e crea frame. Anche poi il cardine è verifica che sia libero per un tempo pari a 96 bit (per 10Mbps si ha $\frac{as}{10 \cdot 10^6} = 9.6 \cdot 10^{-6} s = 9.6 \mu s$). Se non lo è allora aspetta durante la trasmissione la scheda aspetta il CSMA per verificare la presenza di collisioni. Se si rileva una collisione, per evitare si raffigura la collisione trasmettendo un altro. Per diminuire i tentativi successivi il nodo che ha colpito sceglie $K = \{0, 1\}$ e per trasmissione di 512 bit. Se $K=0$ inizia, altrimenti attende. Mentre tranne i colpi che gli altri nodi se ne accorgono, regole di 64 bit (jam) a potenza più faccio attendere un tempo casuale (backoff) la 2^a volta. Il viene moltiplicato per sicuri che gli altri nodi se ne accorgano, regole di 64 bit (jam) a potenza più faccio attendere un tempo casuale (backoff).

Supponiamo che ve ne sia molta. Il secondo intervallo di backoff viene moltiplicato per $\alpha = \{0, 1, 2, 3\}$ e poi si ripete lo stesso procedimento. Ogni collisione si raddoppia il range di $K \rightarrow$ Dopo la i -esima, $K = \{0, 1, 2, \dots, 2^{m-1}\}$ dove $m = \min(i, 10)$ per non allungare troppo i tentativi. Poi capite che vi mancano nuove collisioni e dopo 17 tentativi batte via il frame



Molti standard di livello fisico
100 indica velocità
BASE indica che segnale non viene modulato
ULTIMO TERMINALE dipende da metà di trasmissione utilizzata.

↓ limitazione sulla lunghezza del cavo
trasmettore B

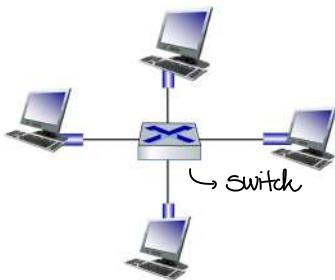
A frame da ricevere tempo T

Collisione: tranne quando inizia a ricevere. B si accorge inizialmente della collisione e A se ne accorge quando il segnale di B arriva ad A → tot 2T. A inoltre si accorge della collisione solo se sta ancora trasmettendo. Dove quindi essere $t_{trans} > 2T$. $2T = \Delta t$ ma se ci sono dei repeater questi generano un ritardo Δ . Avendo quindi $2T = \Delta t / v + \Delta$. Non posso fare quindi ethernet più lunghe di 1200 m e

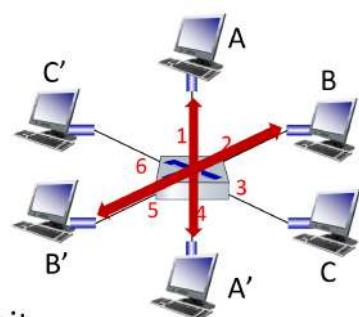
il frame di lunghezza minima deve essere di almeno
512 bit.

Packet Switched Networks

switched ethernet

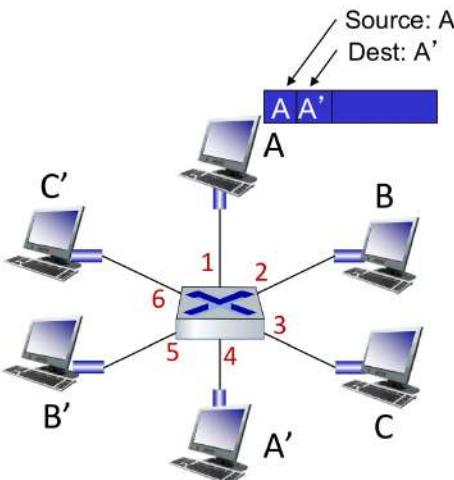


Punto su switch: opera al livello 2 (a diff. di hub che lavora al liv. 1) → riceve frame ed opera in modalità store and forward: memorizza il frame, vede la destinazione e lo indirizza alla porta destinataria. Si usa il protocollo CMP/CD per condivisione switch. La soluzione è trasparente all'utente. È un dispositivo plug-and-play in quanto basta collegarlo ed installarlo. → è self-learning: capisce da solo dove è il destinatario ed impara automaticamente.



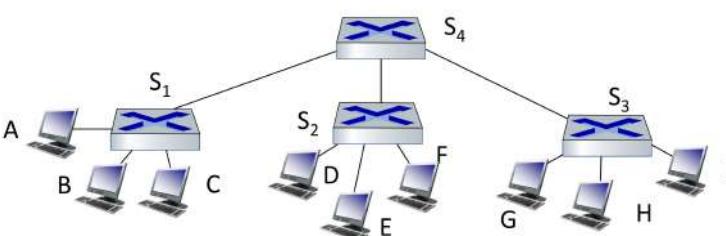
Il link è dedicato tra PC e switch: non ci sono collisioni → 1 trasmisso if porta dello switch senza avere collisioni: si aumenta il throughput. Come fa il nodo a sapere verso che porta di uscita indirizzare? viene letto l'indirizzo di destinazione e la switching table dello switch. Questa è scritta automaticamente dallo switch
↓ Come?

A vuole inviare frame ad A'. L'host non è consapevole della presenza dello switch. Questo ha una tavella iniziale vuota e quindi indica il portello a tutte le porte di uscita, trovando il destinatario su una porta. Si assume che gli host si comportino correttamente. Lo switch impara che A si trova sulla linea 1. Dopo che A' riceve il frame questo spesso risponde (ACK o frame bidirezionale). Lo switch adesso indirizza il frame ad A' trovando l'indirizzo nella tavola di switching. Se A' è collegato a A. Può però succedere che A' cambi in A'' in dove fare poi finta floating: va fatto periodicamente ogni TTL della tavola.
↓ Utilizzato in ethernet



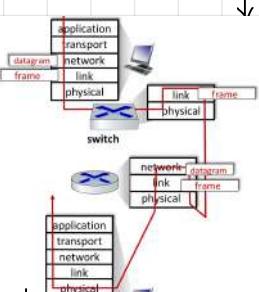
MAC addr	interface	TTL
A	1	60

MAC addr	interface	TTL
A	1	60
A'	4	60



Migliore throughput e si fa un numero di porte limitato → Soluzione gerarchica per collegare nn computer > nn porte
↓ corretto

A può mandare a qualciasi B grazie a self learning. Se una linea comune - hanno quindi indirizzi di più host → opera nel frame



Router: livello network → datagram
+

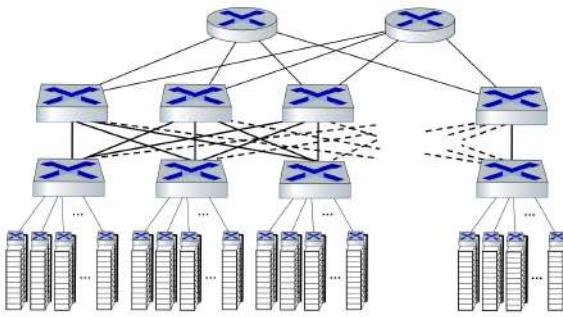
Visibilità switch: local, visibilità router: tutta internet.

Rete più grande: > numero hop quindi > tempo e dato che il servizio offerto da ethernet è unreliable aumenta anche la possibilità di packet loss.
 I web server e mail server vengono messi sullo switch per vicinanza, soprattutto per host utente.



In generale: Per evitare che sorgente e destinatario devono eseguire più percorsi per garantire affidabilità servizio.

Aziende offrono servizi → server posti in datacenter per gestione più efficiente: molte applicazioni molti client. Il network deve essere sempre disponibile e il caos deve avere distribuito e devono essere eliminati bottleneck → caratteristiche + ethernet e switch →

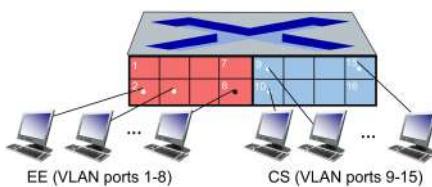


- Border routers**
 - connections outside datacenter
- Tier-1 switches**
 - connecting to ~16 T-2s below
- Tier-2 switches**
 - connecting to ~16 TORs below
- Top of Rack (TOR) switch**
 - one per rack
 - 40-100Gbps Ethernet to blades
- Server racks**
 - 20-40 server blades: hosts

Virtual LANs (VLANs)

Dominio di broadcast: Se rete trannelette ricevono tutti e gli host si comportano di conseguenza anche se i nodi potrebbero prendere tutti i dati. Versione switched: il broadcast avviene solo in caso di flooding ed il traffico rimane in grande limitato → alcuni frame devono essere mandati in broadcast: DMCP.

Si cerca di creare singolo dominio di broadcast: problemi rete grande → efficienza, privacy, security ed in più si organizza la rete per tipo di utenti. Può accadere che lo spazio in un gruppo si esaurisca e l'utente deve collegarsi in un gruppo che non è il suo virtuale.

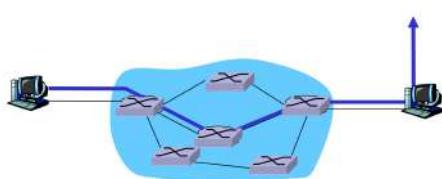


> efficienza per < traffico, > sicurezza per < host comuni

→ creata con switch configurando le porte: Assegnate a VLAN diverse: ciascuna è un unico dominio di broadcast staccato dalle altre VLAN.

Se nr utenti è otto allora a ciascuna VLAN possono essere assegnate porte di switch differenti. È possibile definire VLAN anche con indirizzi MAC. È possibile spostare utenti da una VLAN ad un'altra tramite i router. Anche gli switch possono essere usati come router.

• PACKET-SWITCHED WIDE-AREA NETWORKS (PANS)



Host diviso a maglie: mesh → switch gestiscono zonazione di porte. Quelli alla periferia fungono da punto di accesso.

Identificati da indirizzo univoco.

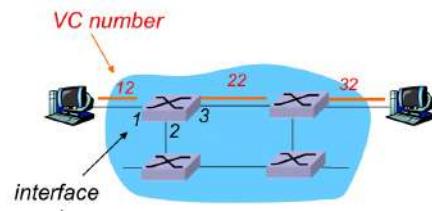
Può avere connection less, in cui ogni pacchetto viene inviato individualmente (datagram).

service) oppure ha fatto su connessione con un percorso stabilito preliminarmente da node sorgente a node destinatario. Avendo che il percorso è lo stesso i pacchetti circolano in ordine di inserimento nel percorso.

↓ Circuito virtuale

Come nei circuit switching: percorso su certo numero di switch da sorgente a destinatario → Call = setup percorso: nodo sorgente invia pacchetto richiesta in modo dettagliato e contiene le informazioni del destinatario seguendo un percorso. Gli switch in seguito neve il percorso indico verso il nodo sorgente. Quando raggiunge la sorgente la connessione è stabilita e si inizia trasferimento. Alla fine si fa il teardown in cui la connessione viene disattivata. Se circuito virtuale è characterizzato da un numero e questo viene attribuito dagli switch per l'istruzione. Il numero di VC è f & link: assegnato dagli switch che inoltzano indirettamente.

↓ Forwarding table



Forwarding table in A switch

Incoming interface	Incoming VC #	Outgoing interface	Outgoing VC #
1	12	3	22
2	63	1	18
3	7	2	17
1	97	3	87
...

Switches maintain connection state information!

↓ Alternativa: dettagli source

Pacchetti viaggiano per conto proprio, gli switch intermedi si preoccupano solo di connettere sorgente e destinazione. Il numero delle possibili destinazioni può diventare molto grande: no tabella di switching →

Destination Address Range	Link Interface
11001000 00010111 00010000 00000000 through 11001000 00010111 00010111 11111111	0
11001000 00010111 00011000 00000000 through 11001000 00010111 00011000 11111111	1
11001000 00010111 00011001 00000000 through 11001000 00010111 00011111 11111111	2
otherwise	3

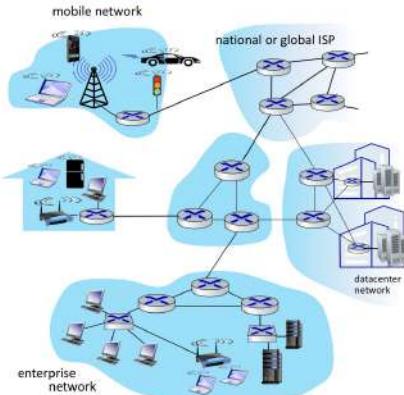
Link virtuale: Non mi interessa cosa c'è sotto ma solo percorso da sorgente a destinazione

→ Gli switch interventi ponono non creare circuito in avvenuta zona (banda)

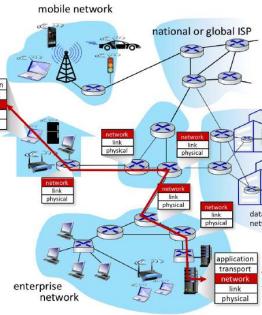
↓ Garanzie

Stabilità e qualità del servizio

Internetworking Data Plane



RETE di RETI: tecnologie diverse → devono poter comunicare tra di loro



Collegate da router e strato software sopra che rende conn trasparente

Livello networking: su tutti i router e host

S: prepara datagram e invia a livello network che passa a liv. inferiore e poi a router

forwarding

Datagram intradati verso rete di uscita : approccio connectionless con tabella di forwarding. Questa funziona come il navigatore del percorso effettuale l'uscita. Si fa quindi prima routing (percorso source e destination) e poi forwarding. Il routing dipende anche dalla tipologia della rete.

↓ 2 piani di lavoro router

piano dei dati: piano del controllo.

↓ 2 modalit

1. Tradizionale: router

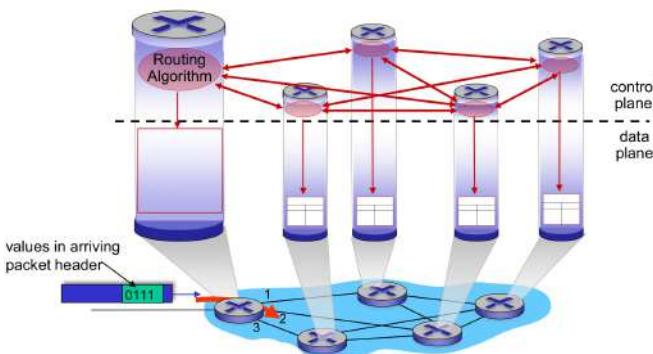
topologia della rete (impossibile fare da una sola persona) → stato di link e poi utilizzo di distanza

2. Software-defined networking:

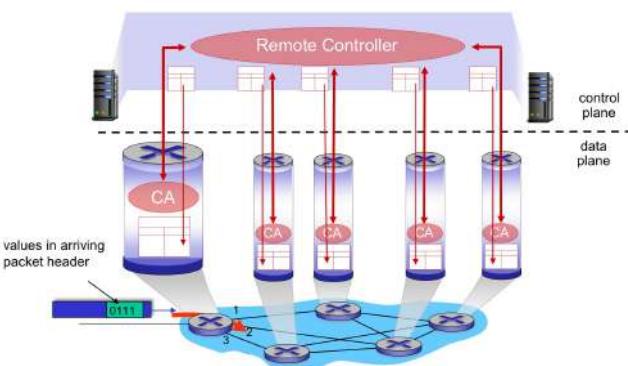
↓ Graficamente

routing centralizzato con entità che calcola percorsi a costo minimo. Essendo un metodo su

1.



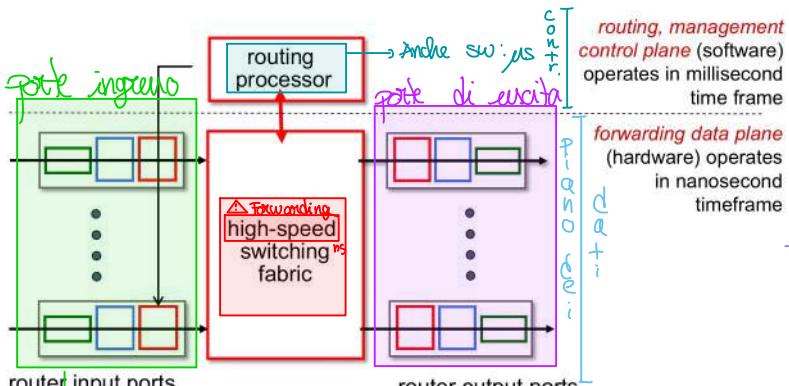
2.



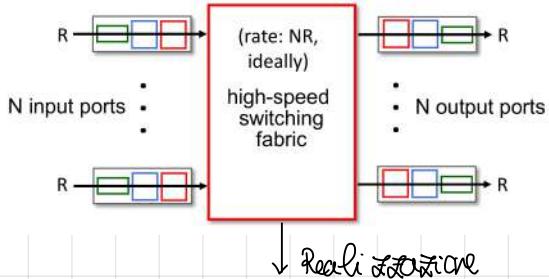
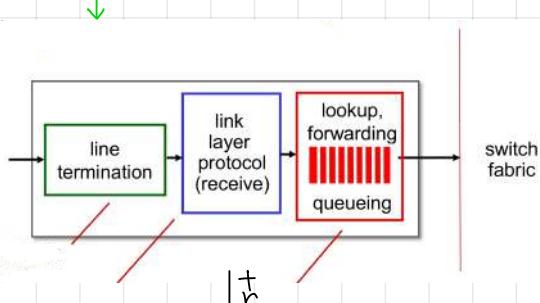
↓ Modelli di servizio per trasportare datagram

Utente può richiedere servizio garantito con rit. massimo oppure può richiedere una banda implementando IP best effort, sull'ordine e sulla banda

Architettura interna di un router



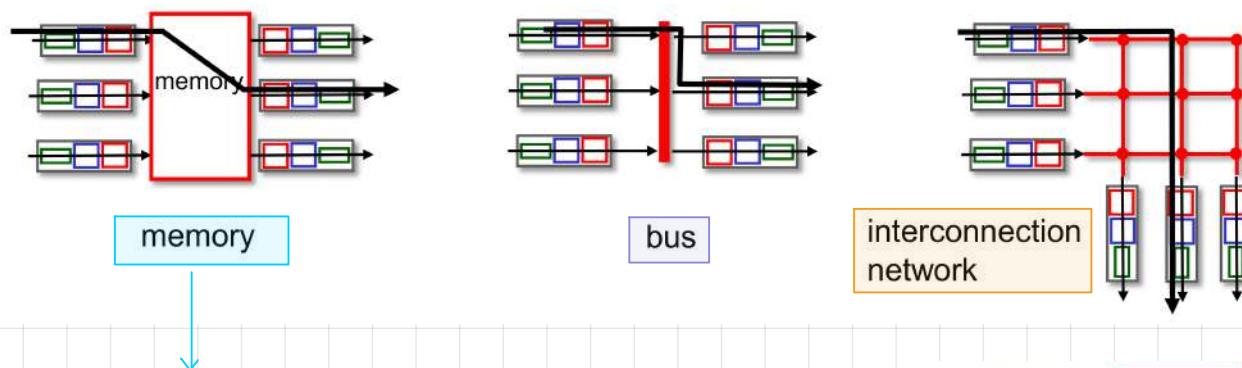
→ Computer "dedicati": ricevono e inviano datagram e gestiscono piano di controllo
 ↓ logica di commutazione
 decide dove mandare datagram
 implementata con processori dedicati + porta: sistema multiprocessore



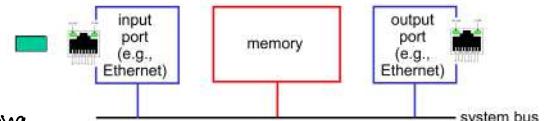
Indirizzi reti destinate (range per risparmio memoria)

Destination Address Range	Link Interface
11001000 00010111 00010000 00000000 through 11001000 00010111 00010111 11111111	0
11001000 00010111 00011000 00000000 through 11001000 00010111 00011000 11111111	1
11001000 00010111 00011001 00000000 through 11001000 00010111 00011111 11111111	2
otherwise	3

↓ se non c'è match per il longest prefix match: → num. hit = destinazione manovra TCAM: in ingresso stringa, in output risultato senza ricerca (→ velocità)



1^a generazione: Computer general purpose trasformati in router. Controllore in ingresso copia in memoria, guarda tab. routing e poi controllore in uscita invia in memoria e accedi in memoria

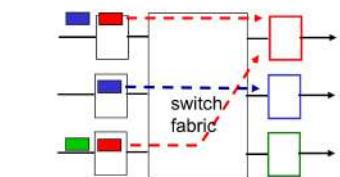


Switching a bus: Su ogni linea di ingresso bus condiviso utilizzata per per uscire in quanto bus più bus per trasferimenti contemporanei in pezetti per sfruttare e si possono dividere i

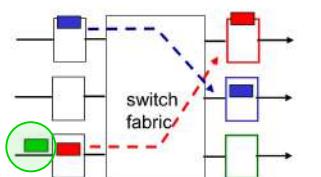
intressi: tempi elevati a causa di dati: c'è un processo e si ha trasferimento → in solo trasferimento e condiviso e si possono dividere i

Interconnection networks

Via



output port contention: only one red datagram can be transferred. lower red packet is **blocked**

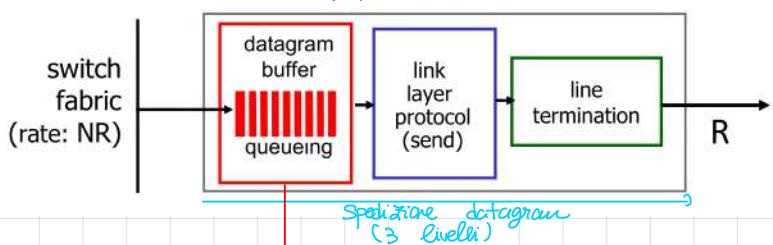


one packet time later: green packet experiences HOL blocking

Possono esserci code: 2 differenti da linee in ≠ vero out =

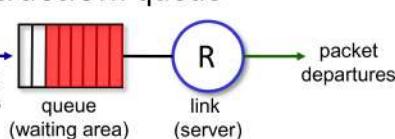
head-of-the-line blocking: primo pacchetto della coda sperimenta ritardo e di conseguenza anche i successivi (politica FCFS)

↓ parte di attesa



Se coda di attesa si riempie si ha **packet loss**. Si parla anche di **overflow**.

problema della gestione della coda



↓ **schéma di gestione delle code**

tail drop: butta via ultimo arrivato

priorità: butta via pacchettino intermedio e mette ultimo arrivato → ha senso se traffico è prioritizzato

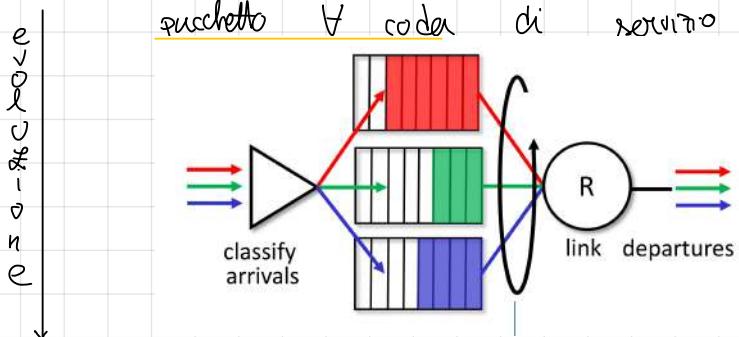
marking: avvisa sorgente di coda piena per farla rallentare (**Explicit Congestion Notification**): deve essere inviata per tempo non tutti uguali: informazioni di all'arrivo coda

FCFS: non necessariamente fair perché tipo diverso → prioritizzati in base al datagram

Priorità: priorità a pacchetti con differenza a seconda di

Round Robin: pacchetti accodati in coda di servizio

priorità più alta → coda di attesa

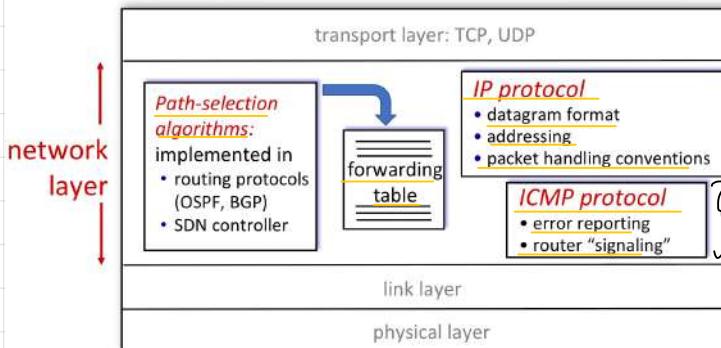


Weighted Fair Queuing (WFQ): RR con tempo di servizio ad ogni coda

↓ **Problema**

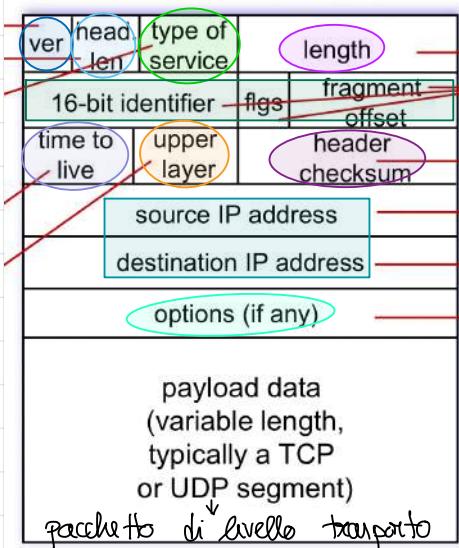
Network neutrality: modo in cui ISP dividono risorse tra utenti. **TCFS** la rispetta, **priorità no** perché la priorità è sostanzialmente decisa in base a quanto viene pagato, e' necessario che il protocollo lo faccia. Si hanno poi anche altri tempi regolati da stati.

- PROTOCOLLO IP
 - definisce formato dei datagram, definisce uno schema di indirizzamento (IP univoco)
 - e definisce regole di invio dei datagram
 - ↓ livello internetworking



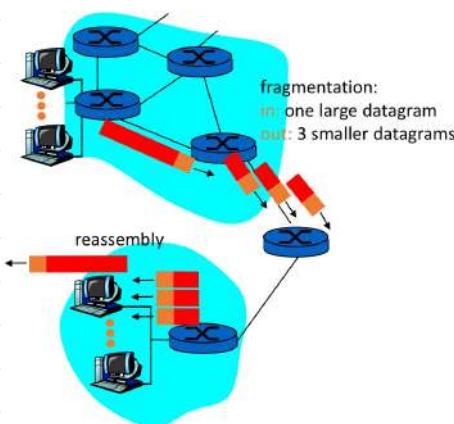
Protocollo di supporto: Notifica

↓ Formato datagram
32 bits → affineamento

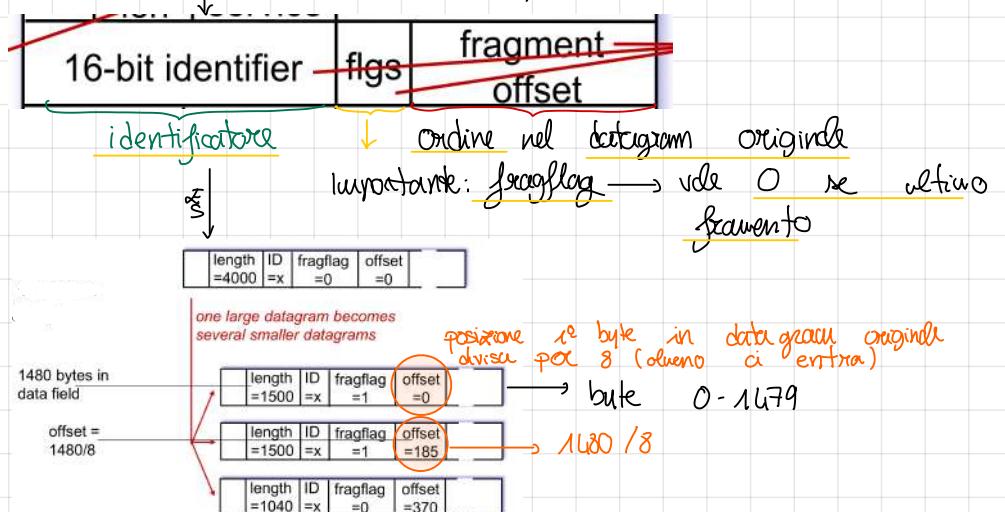


- Versione protocollo IP uscita (IPv4): 4 bit
- Lunghezza header: 4 bit
- Classe di servizio datagram: 0-5 differenti
- **prosità gestita da router**
- Lunghezza datagram complessivo: 16 bit
- Max 64KB
- Gestione datagram "separati" (Ethernet)
- Numero di hop: 8 bit. Aggiornato prima di indirizzo
- Serve nel caso in cui venga alterata informazione: pacchetto girato e non arriva a dest. **STOP**
- Livello superiore a cui sono forniti dati payload: 8 bit
- Error detection, calcolato su header: servizio best effort: nessuna garanzia → Tutto per evitare alterazione compi.
- IP pertinente e arrivo
- Usati da alcune applicazioni → Se non ci sono l'intestazione è di 20 bytes
- im frame com dimensione massima di 1500 byte nel caso quindi avere datagram > frame.

Il datagram è incapsulato
caso di Ethernet e PPP.



Il datagram viene frammentato (la parte payload) ed ogni frammento viene inviato separatamente e viene aggiunta intestazione per permettere riassemblo.



Se qualche frammento viene perso la ricostruzione non è possibile: database

Scritto dom' finisceit (servizio best-effort)

indirizzi host

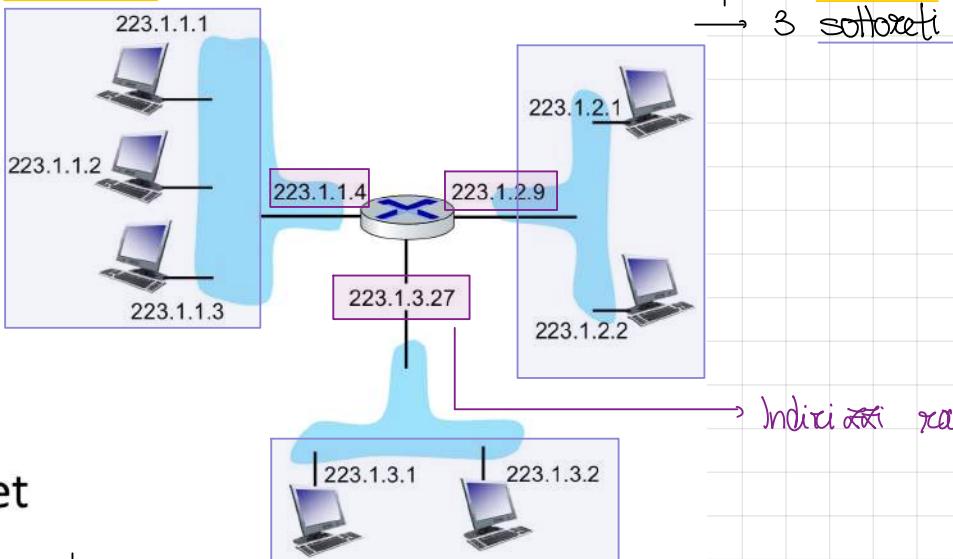
Ad ogni host viene assegnato un IP su 32 bit → una \neq interfaccia: il router deve avere più indirizzi per potere funzionare, gli host tipicamente ne hanno solo 1 → 32 bit interpretati su 4 octetti interpretati come decimali separati da '.'.

↓
Subnet: Sottorete di internet, "rete fisica" comincia a internet. Insieme di host tra cui poter comunicare senza dover passare per un router.

↓
IP strutturato: parti indirizzo forniscono informazioni su posizione geografica host

↓ 2 parti
subnet part: prefisso = identifica sottorete in cui si trova host

host part: seconda parte individua host → parte univoca



Se ci vogliamo rivolgere a tutta la sottorete vanno tutti
fatti 1 nella parte host → insieme ad indirizzo sottorete si mette
anche nr. bit indirizzo: /nr
↓ traduzione

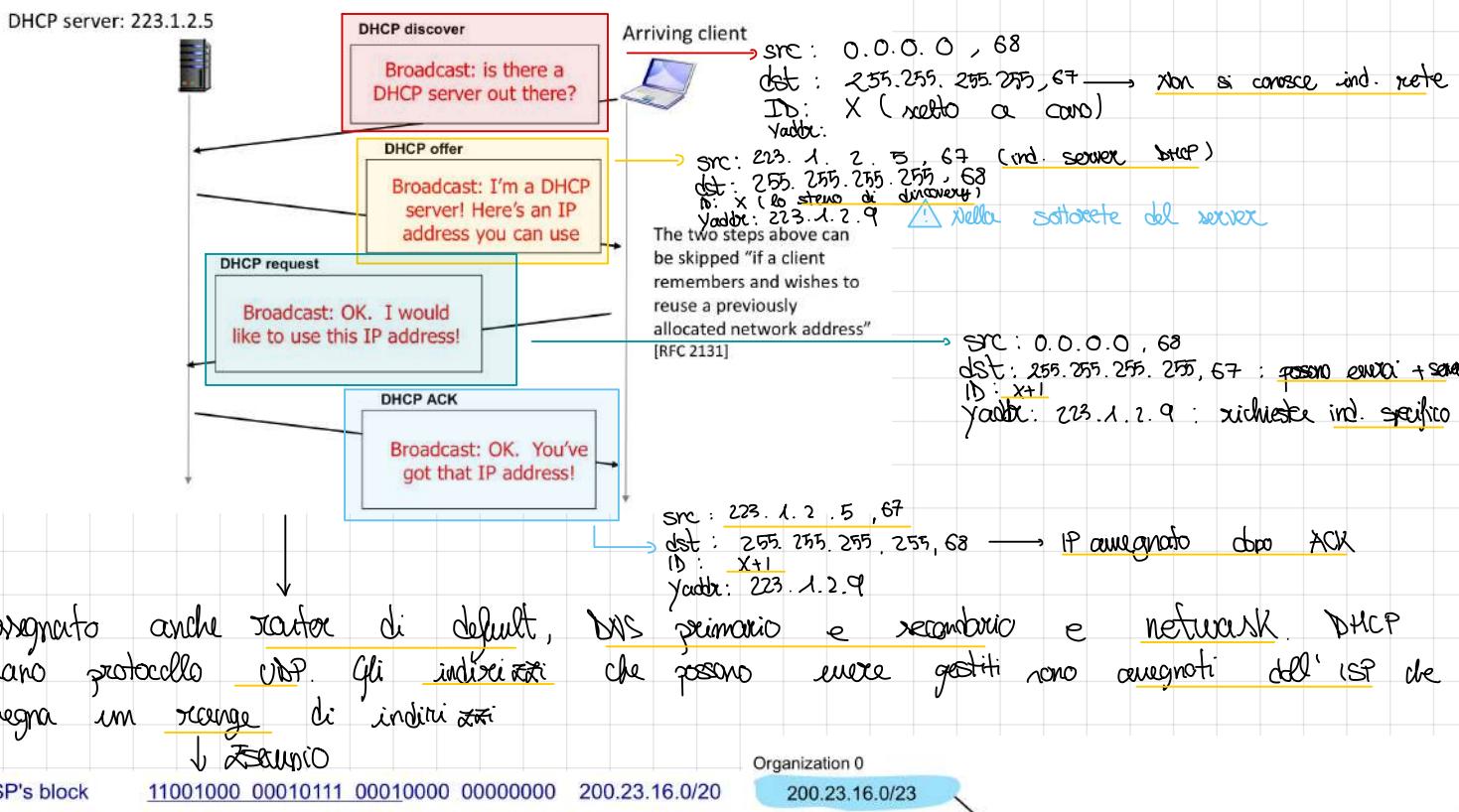
Stringa dei 32 bit con nr MSB = 1 e 32-nr LSB=0: subnet mask. Per
ottenere indirizzo sottorete: IP host AND subnet mask

Forse CBR: Classless Interdomain Routing → Inizialmente IP divisi in classi
(A,B,C,D,E) in base ad estensione in bit parte host. È molto limitante, questo
approccio è stato superato

↓ Ottenimento indirizzo

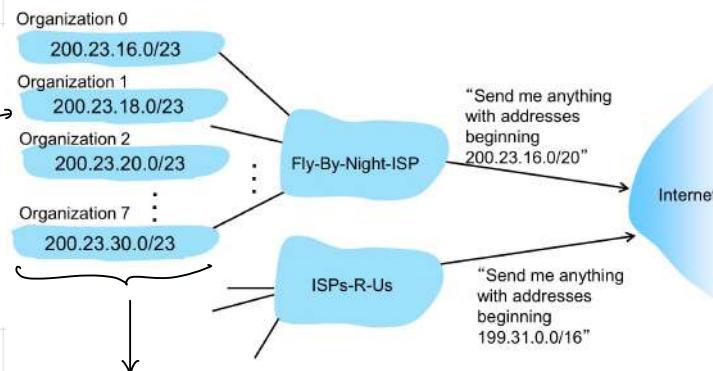
Può essere richiesto alla organizzazione dopo aver dato nome al computer: assegnazione
puntuante → salvato in un file del sistema. L'approccio è poco efficiente in quanto
l'IP è poco sfruttato. Questo è un problema quando gli indirizzi scadranno.

Gli IP vengono quindi assegnati soltanto al momento del bisogno e quindi ne
vengono molti meno. Questo semplifica molto anche l'acquisizione dell'IP per nuove
connessioni → DHCP: plug and play. È un'applicazione client - server che il server
assegna tutti i parametri di configurazione di rete. L'indirizzo ha una scadenza
e può essere rinnovato alla scadenza → 4-way handshake: 4 messaggi scambiati. Si
suppone che il servizio DHCP sia disponibile (presente server)



Assegnato anche range di default, DNS primario e secondario e netmask. DHCP
mano protocollo CIDR. Gli indirizzi che possono essere gestiti sono assegnati dall'ISP che
assegna un range di indirizzi
↓ scansio

ISP's block 11001000 00010111 00010000 00000000 200.23.16.0/20



Efficienza nel scouting: i router nella rete del provider sono divisi in blocki per poter instradare più facilmente. Le informazioni con questo livello di dettaglio non devono essere memorizzate nei router del core di internet, rendendo tutto più veloce riducendo le entrate nelle felcielle di forwarding. L'ISP locale ha al di sopra altri ISP di livello più alto. Quello più in alto ottiene gli indirizzi dell'ICANN. Gli indirizzi a 32 bit sono praticamente esauriti. Per questo si usa IPv6 che ha indirizzi a 128 bit.

o NAT

Si designa un indirizzo ad un gruppo di host. Per fare questo si usano indirizzi "speciali" (riservati). Tra questi vi sono gli indirizzi privati, che non servono quindi essere esposti nella rete internet pubblica. Questi possono ad esempio essere usati all'interno di un'organizzazione che ne garantisce l'privacy.

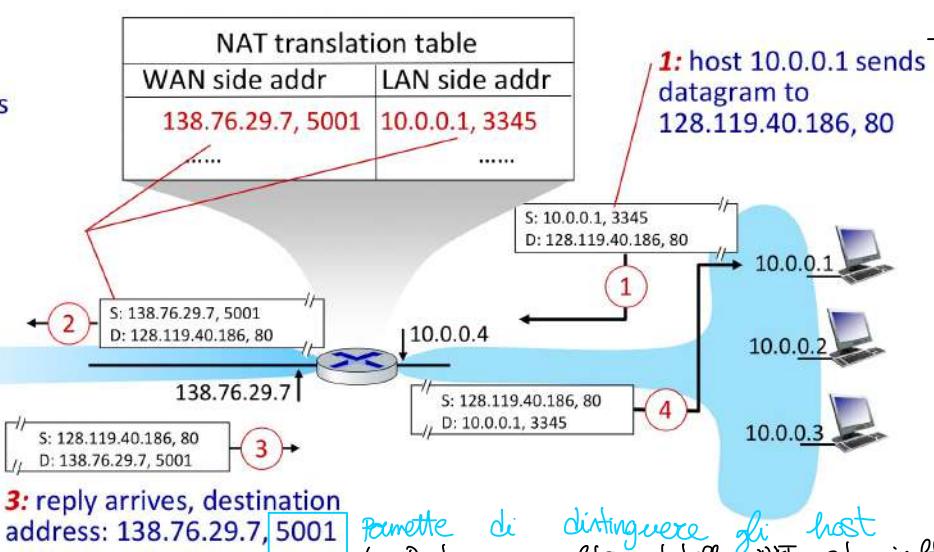
all datagrams leaving local network have same source NAT IP address: 138.76.29.7, but different source port numbers

datagrams with source or destination in this network have 10.0.0/24 address for source/destination (as usual)

host si scambiano dati all'interno della stessa rete. Vogliamo però costruire una soffietto dell'internet pubblico. Si manca quindi indirizzi privati all'interno della rete ed un router che risituisce l'IP privato con quello pubblico rendendo il datagram routeabile all'esterno. **⚠ problema:** se tutti hanno lo stesso indirizzo la risposta arriverà al router. è necessario quindi differenziare tra gli host → Si mette anche la porta (XAT PXT)

2: NAT router changes datagram source address from 10.0.0.1, 3345 to 138.76.29.7, 5001, updates table

NAT translation table	
WAN side addr	LAN side addr
138.76.29.7, 5001	10.0.0.1, 3345
.....



NAT server cambia porta sorgente con porta da lui scelta univocamente & host

Permette di distinguere di host
Router consulta tabella NAT ed invia ad host
grado sostituendo IP pubblico con privato e porta
tradotta con porta originale

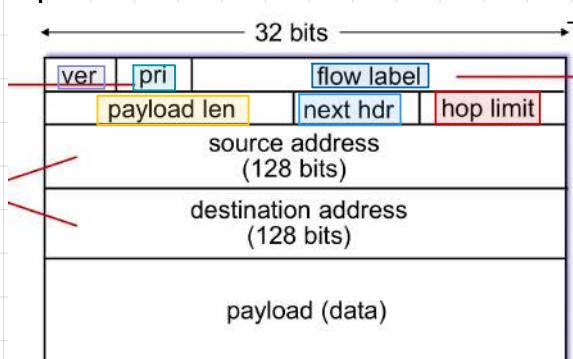
Non è possibile tradurre solo l'IP e non la porta: se 2 host usano la stessa porta si avrebbe ambiguità di risposta

⚠ Controverse: il router dovrebbe implementare solo i livelli 1, 2, 3. Il router cambia però la porta che è per di livello 4. Il NAT infatti si dice avere una soltuzione temporanea in quanto la soluzione long-term è data da IPv6. Inoltre vede il principio di end-to-end argumenting perché il router si mette nel metà sostituendo IP e porta. Non è possibile mettere un server nella rete privata in quanto il suo IP deve essere noto. Per permettere questo si pubblica l'indirizzo del router e si riserva una porta. Questo prevede però che la tavola NAT sia preimpostata.

⚠: posso usare NAT e DHCP insieme (è vantaggioso farlo per avere indirizzi IP privati automatici)

• IPv6

Noto per sopportare a mancanza indirizzi e per aumentare la qualità del servizio. Si vuole infine aumentare la velocità dei router. Con IPv4 il router doveva guardare lunghezza intestazione per verificare presenza di campi opzionali.

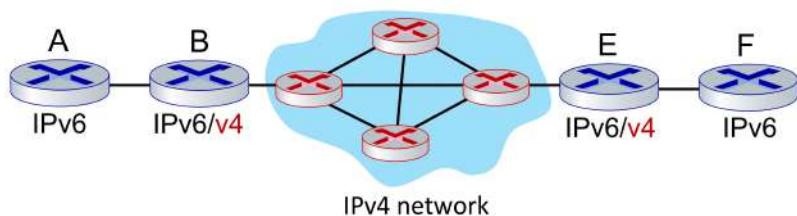


→ lunghezza fina + indirizzi a 120 bit
→ versione protocollo usata (6)
→ identifica datagram nello stesso flusso
→ priorità
→ lunghezza payload in byte
→ = TTL IPv4
→ Protocollo liv. superiore a cui trasmettere dati
⚠: Non c'è checksum: < ritardo in quanto ogni router deve fare TTL- e poi ric算算checksum. La procedura non è vantaggiosa in quanto viene fatta ora defenziva solo nell'intestazione

⚠ ⚡: No frammentazione: perdita di tempo per il router. Se il datagram è troppo lungo questo viene scartato e chi lo invia riceve una notifica. C'è però una dimensione minima (1200 byte circa). In IoT si usa una versione IPv6 che non frammenta (pachetti gicali). Non ci sono più i campi opzionali.

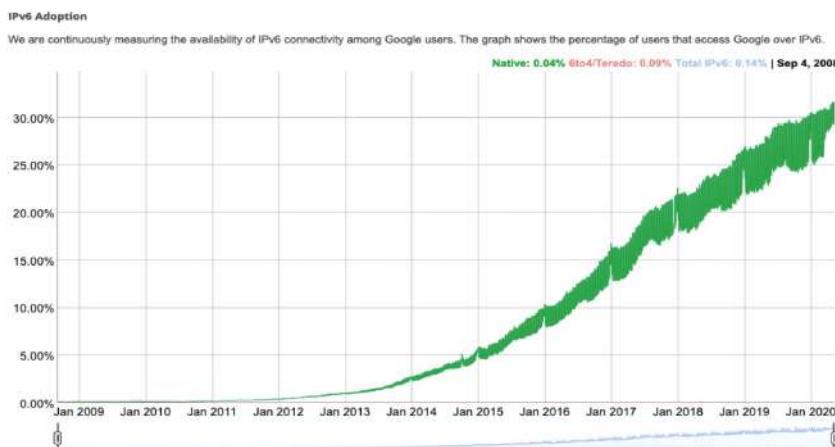
transizione

I router possono usare sia IPv4 che IPv6



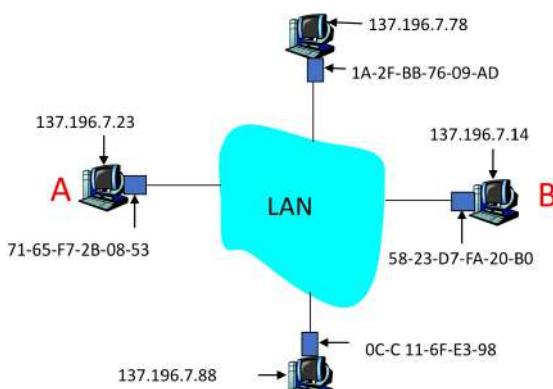
Oppure si usa tunneling dove si mette il datagramm IPv6 in un datagramm IPv4. «at» a questo punto fa la decapsulazione e prende il datagramm IPv6.

Google stima che il 30% dei client accede ai servizi tramite IPv6



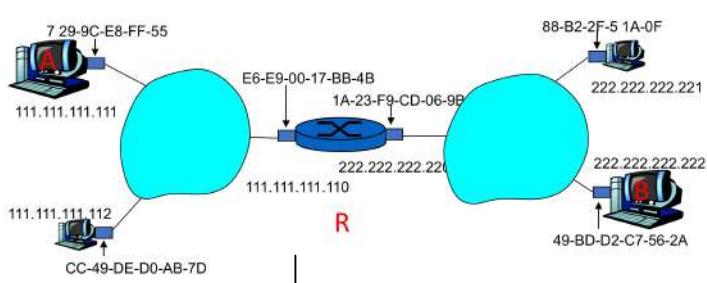
• Protocolli di SUPPORTO

ARP



Suggeriamo che A voglia spedire un datagramm a B. Per fare questo A si rivolge al livello data link. Il data link mette come source il MAC di A (che possiede) ma non può mettere come destination il MAC di B perché non lo conosce (nuovo livello IP)

Oppure



Voglio mandare datagramm in altra rete. Non posso mandare frame perché non conosco MAC router

Penso ricavare MAC da IP? Ogni volta che c'è receita di convenzione IP → MAC viene invocato ARP: ricevendo un broadcast con IP di cui si vuole la traduzione. L'host con IP corrispondente invia un messaggio di reply con il suo MAC. Ogni host mantiene una cache (ARP table) in cui vengono mantenute le associazioni trovate fino a quel momento. Questa viene consultata prima del messaggio broadcast ed ha un TTL per permettere aggiornamento informazioni.

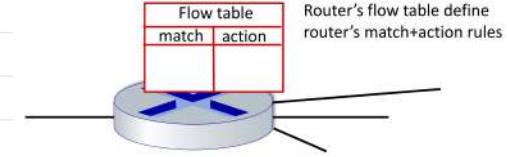
ICMP

Mando del comando PING, funzionalmente è di livello network ma gira sopra IP in quanto usa i suoi datagramm. Un messaggio ICMP contiene

tipo, codice e header con i primi 8 byte del messaggio IP di esempio nel caso di messaggi di errore.

• GENERALI ED FORWARDING

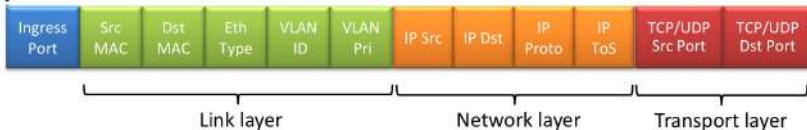
match plan action: In flow table si trova match che dica cosa fare → il match è destination header. È possibile fare un matching generalizzato (quando altri campi)? voglio personalizzare le azioni possibili in base a determinati campi dell'intestazione → tabella di forwarding estesa: flow table guardando table i vari campi dell'intestazione. Nella flow table si può anche avere un campo priority: + match con azioni diverse. Si specifica quindi una priorità per disambiguare. Si possono anche avere dei campi utili a fini statisticci. Nella realtà, per fare flow generalizzato si usa openflow. Tramite openflow è possibile implementare un firewall in quanto permette di bloccare i pacchetti. È possibile anche implementare uno switch. Openflow permette di unificare i tipi di dispositivo.



Packet + byte counters

1. Forward packet to port(s)
2. Drop packet
3. Modify fields in header(s)
4. Encapsulate and forward to controller

Header fields to match:



Link layer

Network layer

Transport layer

→ dispositivi programmati per fare diverse funzioni

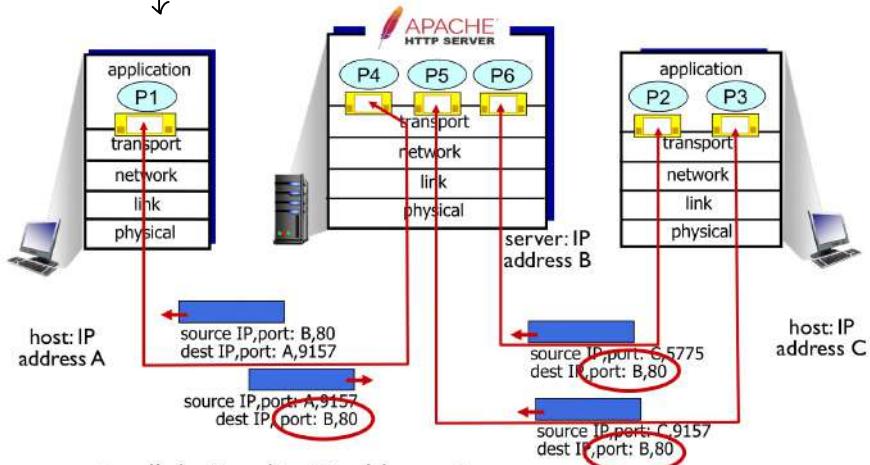
• MIDDLE BOXES

Qualunque dispositivo, escluso router, posizionato tra sorgente e destinazione. Abbiamo tra questi il NAT server, il firewall, gli IDS, i load balancers, le cache e anche le middleboxes application-specific. Inizialmente erano soluzioni low proprietarie ma la loro diffusione ha comportato verso una maggiore apertura.

Process-to-Process Data delivery

Per adesso siamo in grado di mandare datagram dal host sorgente al destinatario. Sono necessarie operazioni di multiplexing (out) e demultiplexing (in) per differenziare dati tra processi in esecuzione su host. Il livello transport crea un'astrazione di un "filo" diretto tra process sorgente e destinatario. I pacchetti vengono chiamati segmenti (idea di continuità).

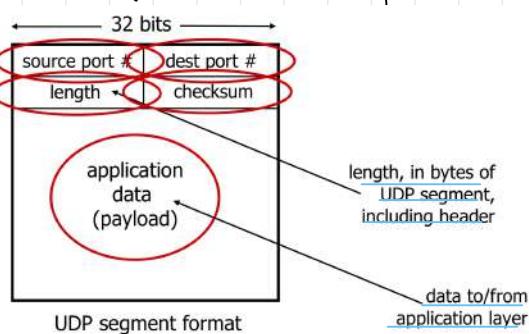
2 casi di demultiplexing: connectionless e connection oriented. Nel primo caso il destinatario è identificato da IP e porta ed i messaggi vengono gestiti in modo individuale. Nel caso connection-oriented si ha una "connessione" tra sorgente e destinatario che si immagina come un "canale" tra s e d. Per identificare la connessione dobbiamo avere gli IP degli host e le porte dei processi. Ogni server può avere più socket, ciascuno identificato dalla propria quadrupla.



Three segments, all destined to IP address: B, dest port: 80 are demultiplexed to **different** sockets

• UDP

Protocollo best effort connectionless (non conviene di apertura e chiusura). Il protocollo produce poco overhead ed è veloce. UDP è usato da DNS, app di streaming, loss tolerant ed altri servizi.
↓ Formato del pacchetto



→ Checksum utilizzato per error detection e per calcolarlo si usa somma in complemento ad 1

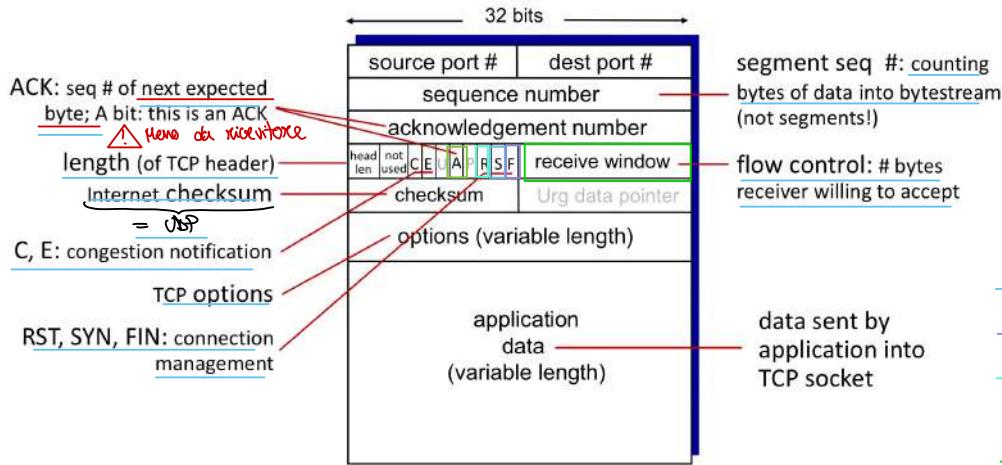
example: add two 16-bit integers

1 1 1 1 0 0 1 1 0 0 1 1 0 0 1 1	1 1 0 1 0 1 0 1 0 1 0 1 0 1 0 1
wraparound	1 1 0 1 1 1 0 1 1 1 0 1 1 1 0 1
sum	1 0 1 1 1 0 1 1 1 0 1 1 1 1 0 0
checksum	0 1 0 0 0 1 0 0 0 1 0 0 0 0 1 1

• PROTOCOLLO TCP

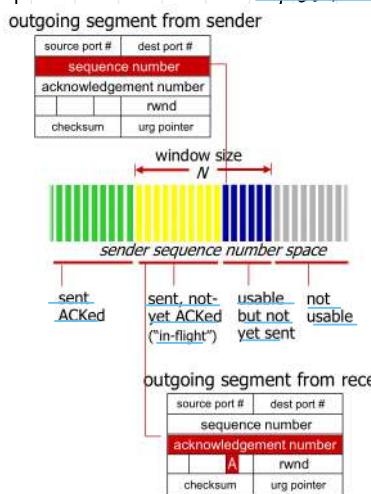
Crea un link point-to-point affidabile e "in-order". Il flusso è full-duplex e si ha una maximum segment size. C'è una fase di handshake iniziale (inizializzazione) e finale (fine connessione). Il protocollo implementa il controllo di flusso e utilizza ACK cumulativi.

Struttura segmento



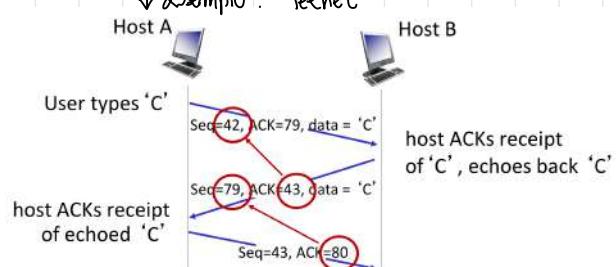
orientato al byte: num seq.
1° byte segmento → meno da trasmettere
Dice se campo acknowledgement number è significativo
Word per attivare connessione
Chiudere connessione
Reset della connessione in caso di problemi
Riavvio del ricevitore nel evento di ACK

Gli ACK sono piggybacked: "confezionati" nel segmento in direzione opposta



! I numeri di sequenza si riferiscono ai byte: relativi al primo byte del segmento e quindi inviare remane al prossimo byte del segmento e così via

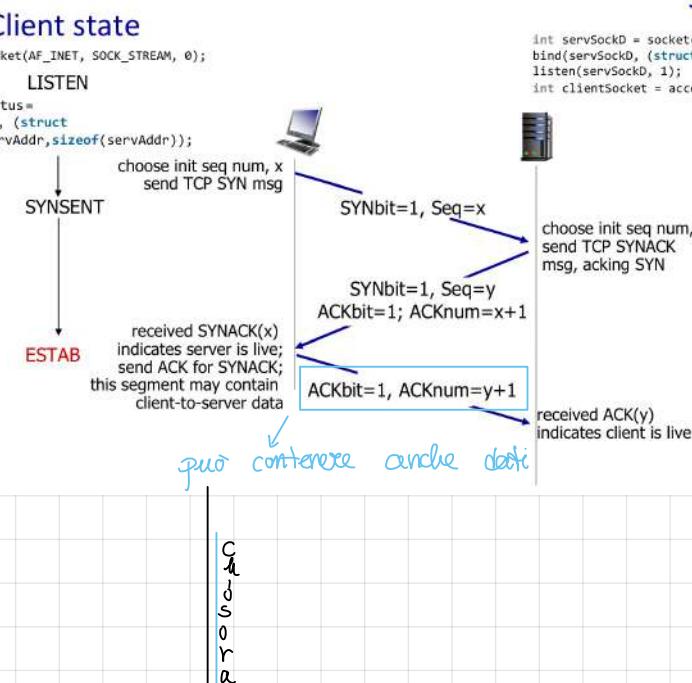
ARQ: automatic request per affidabilità d'ACK. È cumulativo per tutti i byte precedenti → nel cumulo ACK numero viene messo il numero del prossimo byte che ci si aspetta di ricevere (Ho ricevuto correttamente dati fino a quel punto)



TCP è connection oriented: pratica di trasferire i dati si ha la fine di connection setup → TCP lato A e B allocano struttura dati (verificabili di stato) per gestire la comunicazione (seq. number, ACK number, buffer). Vengono deallocate nella fine di connection teardown (chiudere connessione), da richiesta di connect a lato client.

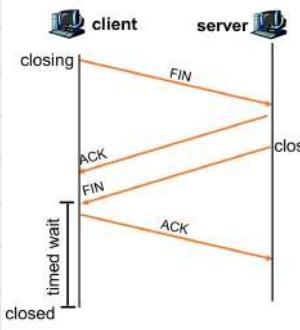
Server state

```
int servSockD = socket(AF_INET, SOCK_STREAM, 0);
bind(servSockD, (struct sockaddr*)&servAddr, sizeof(servAddr));
listen(servSockD, 1);
int clientSocket = accept(servSockD, NULL, NULL);
```



Segmenti in fase di comunicazione hanno SYN bit = 1

→ 3-way handshake
X e Y sono caselli per evitare confusione (sarebbe più ricevere pacchetti vecchi se il client ha effettuato più connessioni)



Si userà il flag di FIN elaborato da parte del client. Si risponde inviando solo un ACK per rendere chiudere la connessione perché deve prima trasmettere i dati rimanenti. Quindi poi la connessione è chiusa da FIN. A questo punto C manda ACK e rimane in attesa fin quando l'ACK non è stato ricevuto. Se l'ACK si riceve il server rimanda FIN ed il client invia nuovamente ACK. Se C non riceve nuovo FIN dopo un certo timeout si suppone che l'ACK sia arrivato.

• TRASPORTO dei DATI

Reliable data transfer

Link virtuale tra C ed S (punto-punto) → Si usa protocollo ARQ con ACK, timeout e ritrasmissione → **⚠ Precauzione** avevamo detto che TimeOut>RTT. Quanto poteva essere fatto in quanto poteva essere calcolato grazie alla presenza di link fisici. Non può essere fatta la stessa cosa in quanto cavo perché il queuing delay è imprevedibile. Non va molto troppo corto per evitare sprechi di banda ed energia né troppo lungo in quanto il throughput in tel cavo tende a 0. Si fanno quindi delle stime basandosi sul punto

Utilizziamo RTT di pacchetti inviati precedentemente e per la prima volta si mette un valore costante (non influisce dato che i pacchetti inviati sono tanti). Si può utilizzare la media degli RTT. Questo è svantaggioso a causa della presenza di valori vecchi. Usiamo una media mobile: con n piccoli la stima è "reattiva" ma poco accurata, viceversa per n grande. Si dice perciò la media esponenziale mobile che, in generale, non oscilla e dà più importanza ai valori più recenti → $ERTT_1 = RTT_0$ velocità veloce al cavo

$$ERTT_2 = \alpha \cdot RTT_1 + (1 - \alpha) \cdot RTT_0$$

$$ERTT_3 = \alpha \cdot RTT_2 + (1 - \alpha) \cdot RTT_1 + (1 - \alpha)^2 \cdot RTT_0$$

.....

$$\begin{aligned} ERTT_{n+1} &= \alpha \cdot RTT_n + \alpha(1 - \alpha) \cdot RTT_{n-1} \\ &+ \alpha(1 - \alpha)^2 \cdot RTT_{n-2} + \dots + (1 - \alpha)^n \cdot RTT_0 \end{aligned}$$



Iterativa

$$ERTT_{n+1} = \alpha \cdot RTT_n + (1 - \alpha) \cdot [\alpha \cdot RTT_{n-1} + \alpha(1 - \alpha) \cdot RTT_{n-2} + \dots + (1 - \alpha)^{n-1} \cdot RTT_0]$$

$$ERTT_{n+1} = \alpha \cdot RTT_n + (1 - \alpha) \cdot ERTT_n$$

Se α tende ad 1 la stima tende all'ultimo valore → Stima oscillante

Se α tende a 0 la stima privilegia i valori vecchi → Stima stationaria

⚠ Limite: è necessario ricordare tutti i ritardi precedenti. Per questo la formula viene risolta in maniera iterativa

Nelle prime versioni veniva scelto Timeout = 2RTT. Quanto è però svantaggioso in quanto può non funzionare in quanto non è adattivo alla situazione.

$$\text{Timeout Interval} = \text{Estimated RTT} + \alpha * \text{Dev RTT}$$

→ Ricostiamo RTT

Deviazione rispetto al valore medio: "safety margin"

$$\text{DevRTT} = (1 - \beta) * \text{DevRTT} + \beta * |\text{SampleRTT} - \text{EstimatedRTT}|$$

Eventi: invio

TCP riceve messaggio da TCP a.

Timeout scaduto b.

ACK ricevuto c.

a. Creo segmenti con # seq. adeguato (contatore aggiornato). Dopo poi evoco fatto partire il timer se questo non è ancora setto → un solo timer relativo a pacchetto più vecchio in attesa di ACK. Il timer viene settato con Time Out Interval.

b. Se si verifica TimeOut si ritrasmette quel segmento

c. Se si riceve ACK per segmenti che non lo avevano ancora ricevuto si aggiorna l'ACK number ed anche il timer

Eventi:
ricezione

Arrivo di segmento in ordine con # seq. atteso con ACK già ricevuto fino a # seq. atteso a.

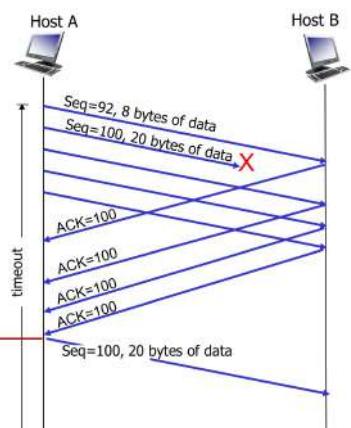
Arrivo di segmento in ordine con # seq. atteso. Altre segmento attende ACK b.

Arrivo di segmento con # seq. più alto (gap) c.

Arrivo di segmento che sta nel gap d.

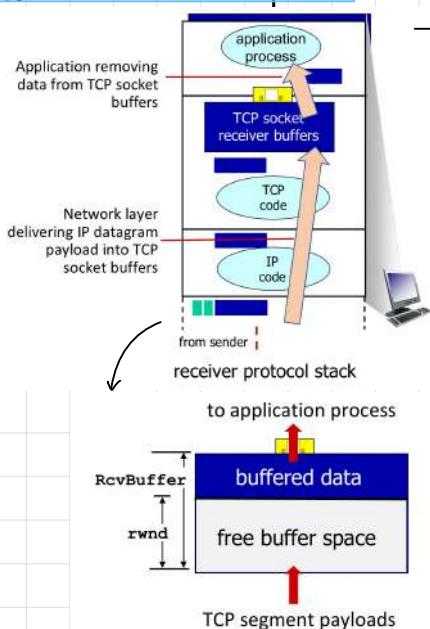
- a. ACK postposto di 100 ms per arrivare prox. segmento. Se non arriva invio ACK
- b. ACK cumulativo per entrambi i segmenti in ordine
- c. ACK duplicato con # byte successivo atteso
- d. ACK immediato se segmento parte da immedio gap.

Fast retransmit



A riceve 3 ACK duplicati: il pacchetto 100 è stato perso (3 indiri si fanno la prova). Non si aspetta quindi il timeout ma A ritrasmette subito il pacchetto (fast retransmission)

Controllo di flusso



→ Il processo prende i dati ad una velocità diversa da quella di arrivo dei dati. Per evitare che la velocità di arrivo sia più alta del processo applicativo.
I buffer si accumulano nel buffer TCP ricevitore fino a "tranciare": dati che si perdono devono essere ritrasmessi creando over-head. Il ricevitore perciò avverte il trasmettitore della situazione chiedendogli di rallentare la trasmissione

Viene calcolato lo spazio libero consentendo la dimensione del buffer, il numero di seq. dell'ultimo byte prelevato e quello dell'ultimo bufferizzato: Free space = dim - (# bp - # bh). Comunica poi lo spazio libero al trasmettitore tramite il campo receive window. In ACK # si mette il # del 1° byte che mi permette di ricevere in ordine. Il segmento arriva al trasmettitore

dopo un po', quindi lo stato del ricevitore è cambiato dall'invio del segmento. Devo quindi mandare meno di $rwnd$ byte: last byte sent - last byte ACK sono i byte mandati nel frattempo, da cui si ottiene che il numero di byte da mandare è dato da $rwnd - \Delta S.A$

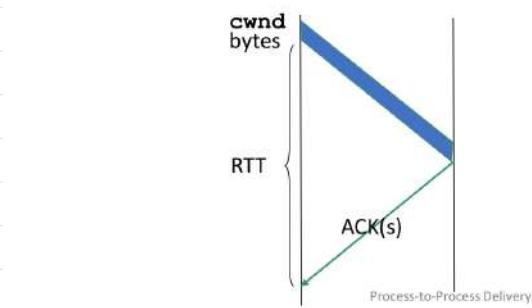
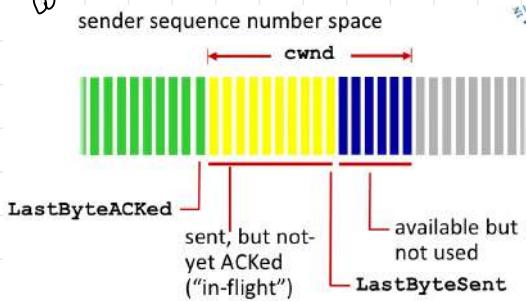
Controllo della congestione

+ da controllo di flusso per luogo dove si verifica packet loss. Nel caso del flusso si verificava al buffer del ricevitore. Eviduiamo però questo caso. Il problema può verificarsi lungo il "cammino" in caso di "strutture". In ogni caso si necessita però di retransmissione. La congestione si verifica nei router intermedi in quanto ricevono più abbagli di pacchetti provenienti. Questi accade in quanto i router gestiscono traffico da più sorgenti, tutte le quali vogliono trasmettere al massimo rate. L'ideale è che le sorgenti diminuiscano il rate di invio dei dati prima di generare packet loss.

2 appross: **Network-admitted** → Router manda feedback a sorgente quando buffer comincia a riempirsi per far diminuire il rate alle sorgenti. Nel nostro approccio però il router impedisce i livelli fino a 3 e quindi questa tecnica non funziona

End-end congestion control

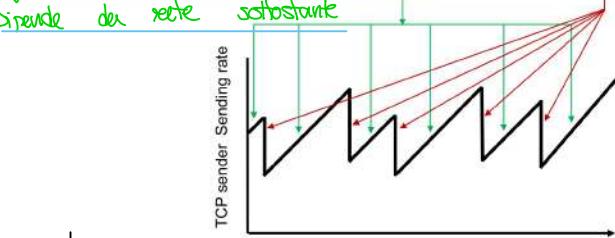
Aggiustamento rate trasmissione



↓ Additive Increase, multiplicative Decrease

Additive Increase
increase sending rate by 1 maximum segment size every RTT until loss detected: *a poco fino a congestione*

Multiplicative Decrease
cut sending rate in half at each loss event



AIMD sawtooth behavior: *probing* for bandwidth

Mecanismo implicato: TCP mittente capisce da domani sintomi che c'è congestione (ritardi, packet loss). Il pacchetto è considerato perso al triplice ACK duplicato o allo scadere del timeout. Non è detto che sia congestione (può essere anche errori) ma TCP sa che lo sia senza.

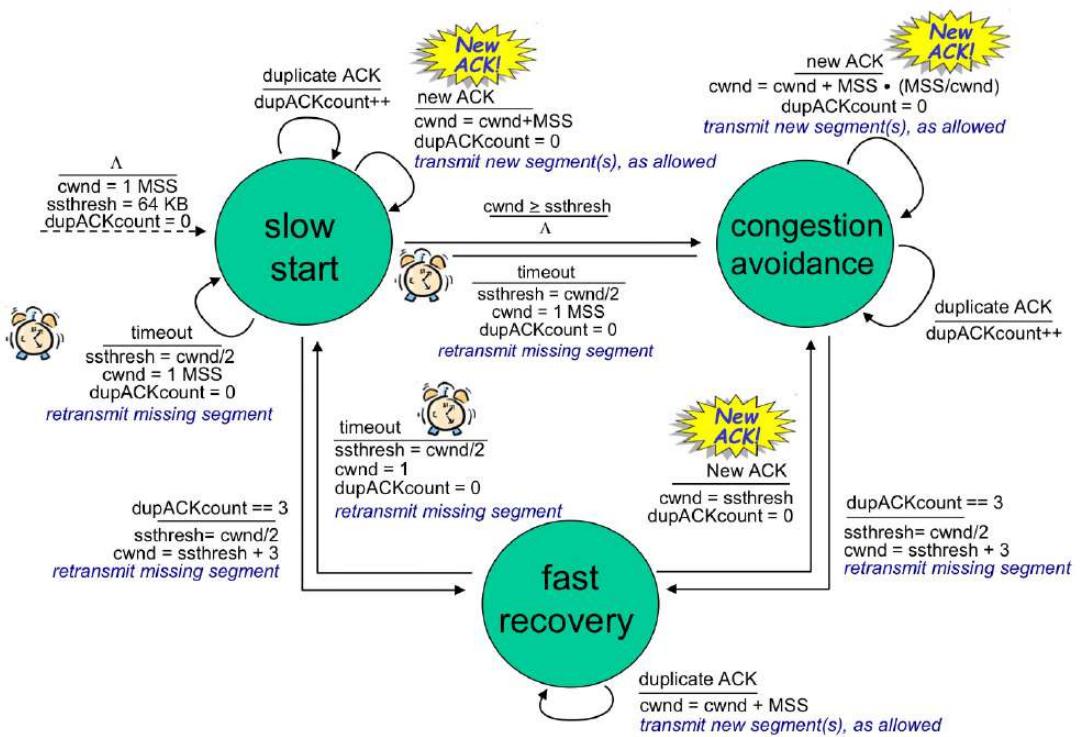
T manda cwnd bytes impiegando $\frac{t_{trans} + RTT}{transciclo}$. Il rate è dato da: $rate = \frac{cwnd}{RTT}$

Si capisce quindi su cwnd avendo che RTT rimanga lo stesso. Si avranno 2 finestre: $rwnd$ e $cwnd$. Si prende la più piccola, anche se nel nostro caso trascuriamo (considerandola grande) $rwnd$. Vorremo tenere il rate più alto possibile compatibilmente con la rete. Non ha senso calcolare "offline" il rate ottimale in un contesto dinamico come internet. Devo utilizzare un approccio adattivo. Se ricevo 3 ACK duplicati ho continuato a ricevere pacchetti, mentre nel caso di timeout la congestione è più grave.

- 3 fasi:
1. slow start
 2. Congestion avoidance
 3. Reaction to loss event

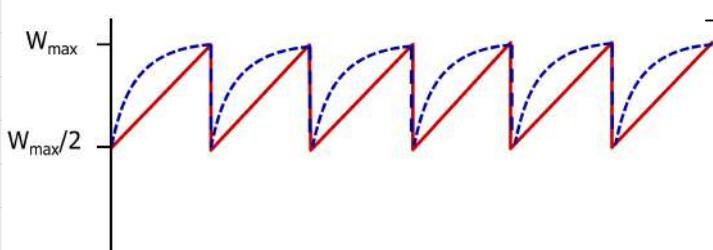
Inizio mettendo cwnd = 1 MSS. Se ricevo ACK posso aumentare e ricevere parto buono gcdoppio la cwnd. se ricevo di nuovo ACK gcdoppio di nuovo e così via ($2 \rightarrow 4$). Quanto prosegue fino al raggiungimento di una soglia (ssthresh). Si passa ora alla fase di Congestion avoidance in cui si incrementa in maniera additiva. ssthresh vell di default 64KB. Alla prima perdita si hanno 2 modi di reagire a seconda del tipo della perdita: con triplo ACK duplicato si doppia cwnd e si ripete da lì (congestione non grave). Se si ha invece timeout si riporta cwnd a 1 MSS e si riparte con slow start. In entrambi i casi si fa ssthresh = cwnd / 2

↓ Macchina a stati finiti

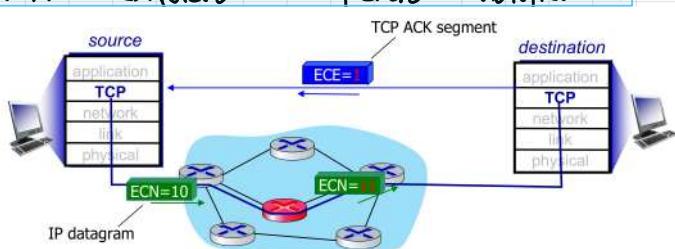


↓ Migliore algoritmo per ottenere a threshold

→ Nella fase di Congestion avoidance l'aumento avviene in maniera cubica (TCP cubic)



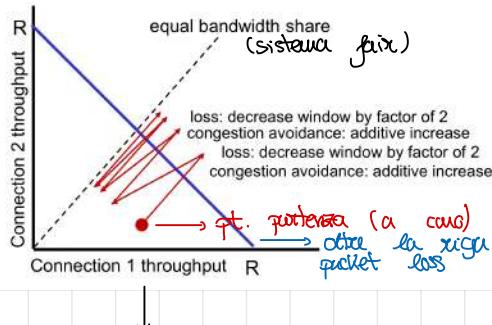
ECN : explicit congestion notification



→ Possibile approccio network centred: router setta alcuni bit del datagram quando ci sono (2 bit di default 00). 10 = moderata, 11 = congestione elevata. Quando dest riceve pacchetto manda un RCE=1 per fargli ridurre cwnd.

TCP fairness

In presenza di più connessioni le tratta tutte allo stesso modo o ne privilegia qualcuna? Se K connessioni condividono una connessione con rate R, ciascuna dovrebbe ottenere una rate di R/K



→ Se si trova sulla linea il sistema sarebbe inefficiente (perciò badabu). Per avere efficiente le connessioni insieme devono prendere R badabu. Per avere efficiente il intervento deve operare nella retta blu

↓
TCP deve "rappresentarsi" con UDP. Questo non ha alcun meccanismo di controllo e quindi viene a mancare anche la fairness (vuol dire ha badabu a TCP). Per ottenere rate più alti chi utilizza TCP fa uso di connessioni in parallelo.

Network Security

confidenzialità: solo il destinatario dovrebbe poter leggere il messaggio ed eventuali terzi
 ↓
 → nome che lo protegono non devono leggerlo

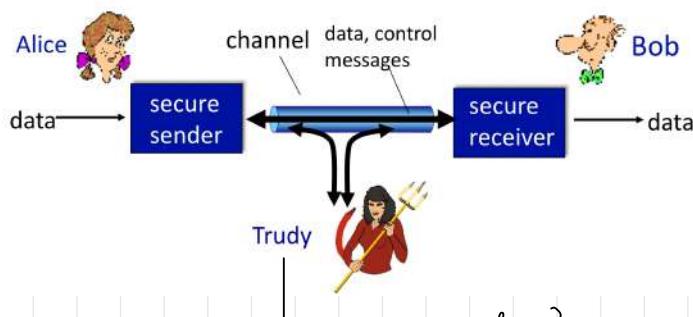
Non sempre necessaria

autenticazione: l'utente richiede che destinatario si identifichi

messaggio integrità: messaggio non deve poter essere combinato da un intruso

controllo degli accessi e della disponibilità dei servizi

↓ Scenario



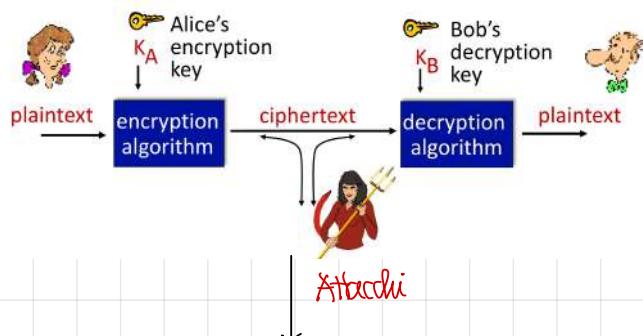
eavesdrop: spia messaggi

spoofing: messaggi con indirizzo falso

manjacking: si sostituisce ad una delle 2 sorgenti

denial of service: server non responde in grado di mettere a disposizione servizi

↓ utilizzo della crittografia



Metodo蛮力破解: devo provare tutte le chiavi → sicurezza è data dal tempo

Analisi statistiche: non esaminano tutto lo spazio delle chiavi

chiave simmetrica: la chiave è la stessa per la cifratura e la decifratura

↓ Attack

Trudy può intercettare il messaggio ma non decifrarlo perché non si conosce ks

Cifrario di Caesar (a sostituzione)

plaintext: abcdefghijklmnopqrstuvwxyz

ciphertext: mnbvcxzasdghjklopuiytrewq

e.g.: Plaintext: bob. i love you. alice
 ciphertext: nkn. s gktc wky. mgsbc

↓ Appuccio più resistente

▪ n substitution ciphers, M_1, M_2, \dots, M_n

▪ cycling pattern:

• e.g., n=4: $M_1, M_3, M_4, M_2; M_3, M_1, M_2, M_4; \dots$

▪ for each new plaintext symbol, use subsequent substitution pattern in cyclic pattern

• dog: d from M_1 , o from M_3 , g from M_4

🔑 **Encryption key:** n substitution ciphers, and cyclic pattern

• key need not be just n-bit pattern

Cifrari a stream: Cifrano un bit per volta

Cifrari a blocco: Blochi cifrati indipendentemente l'uno dall'altro il più utilizzato è

DES: chiave di 56 bit con blochi di 64 bit. → Test sicurezza (DES challenge)

poco di decifratura e risultato in un giorno → Sostituito prima da

Trudy può prelevare, modificare, cancellare e aggiungere messaggi.

Alice e Bob possono avere 2 qualsiasi attenti che vogliono comunicare: web browser / norme online banking sono e client, server DNS, router

m composto in algoritmo pubblico che utilizzando una chiave privata

m: msg in chiaro

KA(m): ciphraumato cifrato con chiave A

m = KB(KA(m)) ove KA: chiave cifratura

KB: chiave decifratura

mapping lettere da m a c. lettera di c (V lettera)

le chiavi possibili sono 26!. Non importa provare tutte ma noi

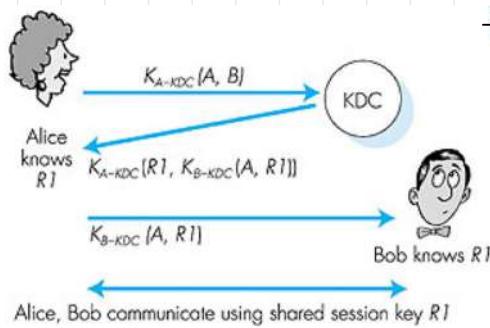
possiamo fare cose così.

SCAM
BY
SYN

triste che c'è poi YES che una chiave di 120, 102, 256 bit < block

Incontro di persona, crittografia a chiave pubblica per lo scambio della chiave ognuno c'è di distribuzione delle chiavi.

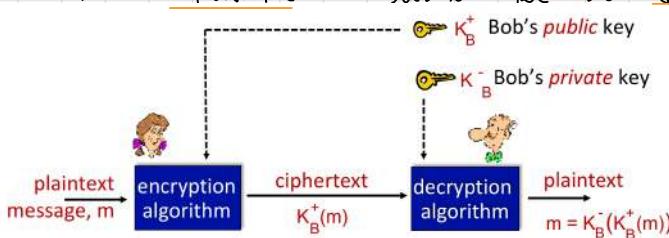
↓ Funzionamento



Alice. Bob conoscono R1. Alice invia K_A-KDC(A, B) al KDC. Il KDC risponde con K_A-KDC(R1, K_B-KDC(A, R1)) a Alice e K_B-KDC(A, R1) a Bob. Alice e Bob comunicano utilizzando R1.

→ Chiave pubblica:

2 chiavi: pubblica → usata per la cifratura
privata → usata per la decifratura



↓ Requisiti chiave

- Dato $K_B^+(\cdot)$ e $K_B^-(\cdot)$, si deve avere $K_B^-(K_B^+(m)) = m$
- Dato $K_B^+(\cdot)$ deve essere impossibile calcolare $K_B^-(\cdot)$
- $K_B^-(K_B^+(m)) = m = K_B^+(K_B^-(m))$: può scambiare l'ordine in cui applico le chiavi

⚠ Svantaggio: Molto più lenta della crittografia simmetrica. La lentezza aumenta aumentando la lunghezza dei messaggi.

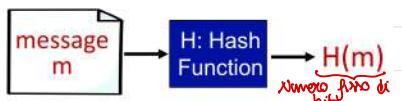
Scambiamo la chiave vis con la crittografia a chiave pubblica e poi utilizziamo la crittografia a chiave simmetrica.

• INTEGRITÀ

Chi riceve il messaggio deve verificare la presenza di alterazioni, replicazioni e se abbiamo inviato una sequenza di messaggi l'ordine deve rimanere inalterato. Deve infine autenticare il mittente.

↓ messaggio digest (impostato):

"Riassunto" del messaggio. Utilizzo una funzione hash che, dato un impostato un messaggio produce un curta un digest.

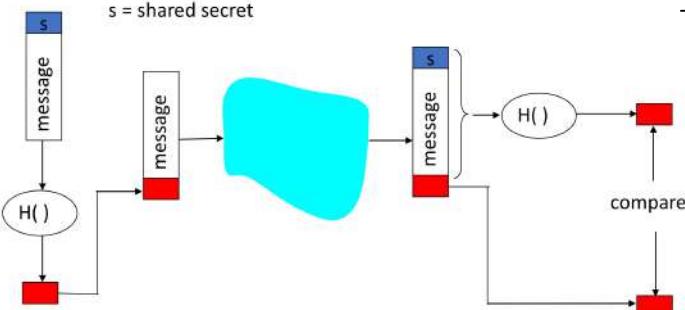


↓ proprietà

- Facile da calcolare
- Impossibile: da $H(m)$ non si può rintracciare m
- Resistenza alle collisioni: Dati m ed m' deve essere quasi impossibile che $H(m) = H(m')$
- L'uscita deve essere apparentemente casuale.

Algoritmi utilizzati: MD5 e SHA-1

MAC (garantisce integrità)



Alice e Bob condividono una stringa di bit (secret). Quando Alice vuole inviare il messaggio "attacca" il MAC, generato tramite l'hashing del messaggio + il segreto condiviso concatenati. Bob quando lo riceve separa il messaggio dal MAC, ci concatena il segreto e fa l'hashing e poi confronta i 2 MAC generati. Autentica il messaggio se i MAC sono uguali. Ero sicuro che sia facile ad avere

Se il segreto rimane segreto, Bob può inviare il messaggio e non Trudy.

↓ Algoritmo **HMAC**

1. Concatena $s, m \rightarrow [s|m]$
2. Hash di $[s|m] \rightarrow H([s|m])$
3. concatene hash con messaggio $\rightarrow [H([s|m])|m]$
4. Fa di nuovo hash $\rightarrow H[H([s|m])|m]$

↓ **playback attack**

Trudy si sostituisce ad Alice nella comunicazione scrivendo il messaggio e poi inviandolo successivamente. Questo accade perché non viene verificato che il messaggio sia "falso" e non "punto". Inoltre, nella generazione del MAC non abbiamo usato nessuna crittografia. Per evitare il problema precedente si usano gli OTP (One-Time pad), generati per ciascun comunicazione.

⚠ Come garantiscono che i messaggi arrivino in sequenza? Lo vedremo più avanti

• FIRMA DIGITALE

- Cari attributi:
1. Verificabile → Alice's frasi vengono tenute per referimento
 2. Non falso/fake
 3. Non ripudiable
 4. Garantisce integrità del messaggio
↓ possiamo usare il MAC?

Garantisce l'integrità. Non è però verificabile perché il segreto è condiviso tra Alice e Bob e quindi potrebbero aver firmato entrambi. Non c'è quindi unico MAC.
↓ utilizziamo la crittografia a chiave pubblica:
da chiave privata non deve avere nulla pubblico: viene utilizzata per generare la firma su quel particolare messaggio

Bob's message, m

Dear Alice
Oh, how I have missed you. I think of you all the time! ...(blah blah blah)
Bob

K_B^- Bob's private key

Public key encryption algorithm

$m, K_B^-(m)$

Dear Alice
Oh, how I have missed you. I think of you all the time! ...(blah blah blah)
Bob
 $K_B^-(m)$
firma

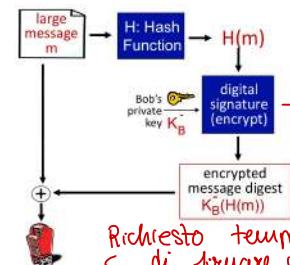
→ Cifratutto con chiave privata

MSI firmato = hash + firma

→ Per la verifica della firma usiamo la proprietà $K_B^-(K_B^+(m)) = m = K_B^+(K_B^-(m))$

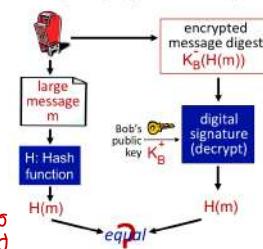
- suppose Alice receives msg m , with signature: $m, K_B^-(m)$
- Alice verifies m signed by Bob by applying Bob's public key K_B^+ to $K_B^-(m)$ then checks $K_B^+(K_B^-(m)) = m$.
- If $K_B^+(K_B^-(m)) = m$, whoever signed m must have used Bob's private key

Bob sends digitally signed message:



Richiesto tempo molto
di firmare messaggi

Alice verifies signature, integrity of digitally signed message:

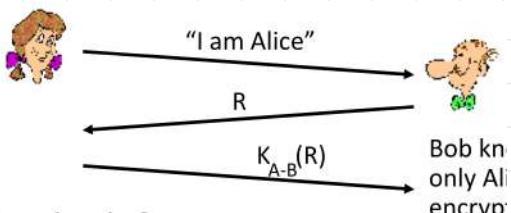


Ahhiamo fino ad ora avuto che la chiave pubblica sia sempre quella corretta in quanto la chiediamo sempre a Bob. Questo ci espone ad un'eventuale presenza di Trudy al posto di Bob. → Utilizziamo i certificati digitali (firmati dall'ente che li ha rilasciati) → Campi: nome ente che rilancia, nome persona certificata, chiave pubblica persona, firma digitale CA. La CA è trustable e quindi la chiave pubblica può essere autenticata senza problemi. Anche a Bob in alcuni casi può essere richiesto un certificato.

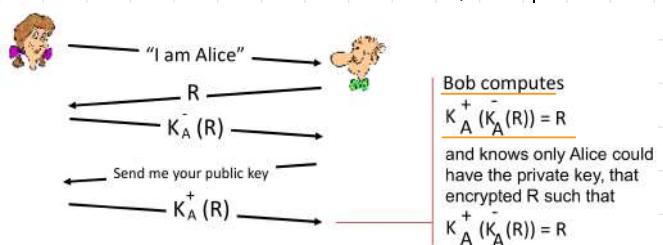
• AUTENTICAZIONE

C'è nowe un nonce: numero utilizzato una sola volta per evitare attacchi second and playback.

↓ utilizzo chiave simmetrica

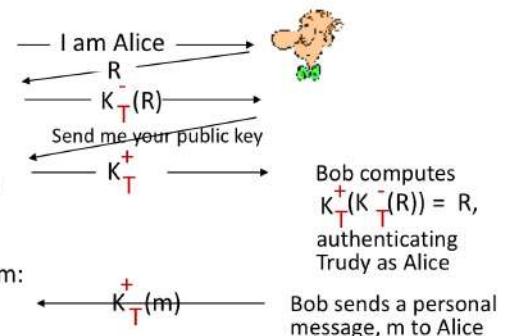
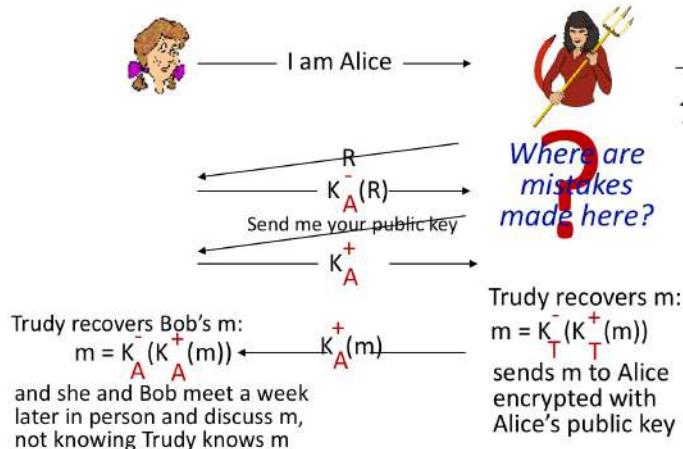


↓ utilizzo chiave pubblica



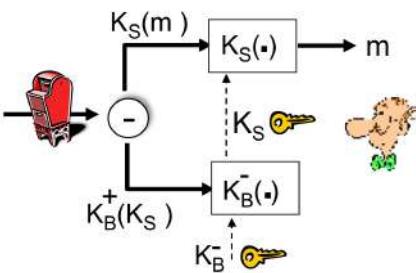
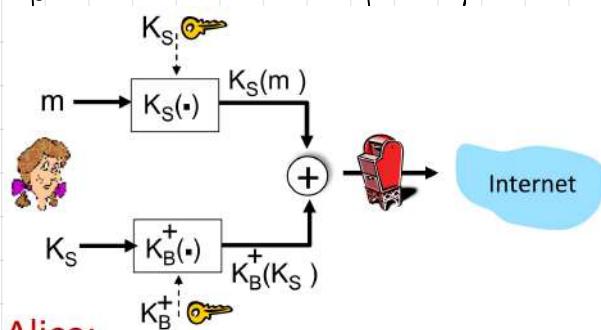
Rimane sempre il problema dell'attacco

METH. Cose non utilizziamo certificati digitali)



• SICUREZZA e-mail

Autenticazione con parola d'avverti e messaggi in chiaro. Per sicurezza è importante l'integrità e la certezza dell'entità. Si utilizza algoritmo PGP (pretty good privacy). Questo fornisce alcuni servizi per implementare alcuni servizi a livello applicazione.



→ L'utilizzo della chiave pubblica per cifrare il messaggio è già troppo grande. Viene generata una chiave di remoto K_S da Alice e cifrata con K_B+(Bob). K_S viene usata per cifrare il messaggio con un

chiave pubblica non è troppo perante. Utilizzando questo tipo di cifratura Trudy non può ricavare m perché non conosce K_B+(Bob) e quindi non ha m. Per garantire l'autenticazione e l'integrità viene utilizzata la firma digitale.

↓ Sicurezza al livello di trasporto

TLS: transport layer security → Si applica solo a connessioni TCP

Protocollo molto usato e supportato da tutti i browser (HTTPS = HTTP + TLS). Garantisce confidenzialità, integrità e autenticazione.

- Handshaking: scambio certificati, utilizzano K_{priv} per autenticarsi e si scambiano master secret
- Key derivation function: Entrambi fanno KDF per derivare chiavi
- Data transfer: Stream
- Chiusure connesive: chiavata in maniera sicura
 - Client invia hello al server e server manda certificato (si autentica). Il client genera il master secret e lo cifra con K_s e glielo manda.
 - Server genera le chiavi con la KDF per avere > sicurezza rispetto ad utilizzo 1 chiave:

K _C	→ Chiavata	per dati da C a S
K _S	→ Chiavata	per dati da S a C
M _C	→ Segreto condiviso	HAC da C a S
M _S	→ Segreto condiviso	HAC da S a C

Anche C può usare la KDF in quanto è nota ed il master secret è condiviso

- Per TCP i dati sono uno stream. Secondo questa logica il controllo di integrità deve essere fatto alla fine ma questo non conviene. TLS "inventa" un nuovo pacchetto: record → $K_C \left(\begin{array}{c|c|c|c} \text{length} & \text{data} & \text{MAC} \\ \hline \end{array} \right)$

Sono "divisi a seconda della 'direzione'"

generato con 2 elementi precedenti (chiavata M_S o M_C a seconda della direzione dei dati)

Per evitare che Trudy riordini i pacchetti si usa numero di sequenza e nonce. Per evitare inoltre chiavi da parte di Trudy si introduce il campo type nel record (incluso nel MAC): data o else → invalid se type = 1 e non si ha la chiusura della connessione TCP.

⚠️ Gli handshake di TCP e TLS fanno perdere tempo: si face a protocollo QUIC che utilizza il protocollo UDP

Attacco reord and playback: funziona se viene registrata intera sessione → In fase di handshake nel client hello specifica le cipher suite supportate e mette un nonce. Il server riceve un algorithm da parte e manda indietro la suite e il certificato ed in più manda un nuovo nonce.

Network layer security: IPsec

A livello di datagram si garantisce confidenzialità, crittografia, autenticazione e identificazione → 2 modelli: transport → solo il payload è cifrato e autenticato
tunnel → l'intero datagram è cifrato e autenticato

sicurezza

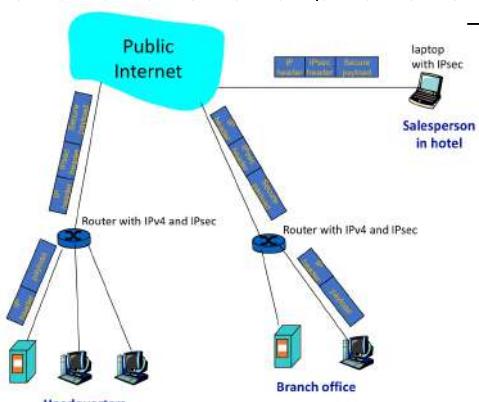
2 tipi: Authentication Header

Encapsulation Security Protocol

Non può più viaggiare perché non è scartabile. Si incapsula in un nuovo datagram con un nuovo header IP e si spedisce quello.

VPN: sicurezza rete privata ma non costa quanto rete privata

→ non hanno garanzie sicurezza assoluta anche comuniendo fuori dall'edificio

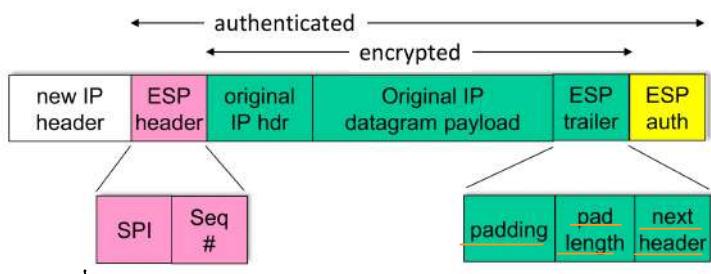


Cifrano tutti i datagram in uscita e li facciamo viceversa in modo sicuro. Viene decifrato quando arriva alla rete di destinazione

Security associations (SAs)

IP è connectionless → SA è una "connessione" half duplex tra 2 endpoint. Questo rende IPsec connection oriented. Attraverso il flusso uni-directionale in cui un endpoint di inizio e fine e la SA cambia se i 2 si scambiano. I router memorizzano i parametri che caratterizzano la SA: IP sorgente e destinazione SA, identificatore della SA all'interno del router di origine, tipo di cifratura, chiave, tipo di integrity check e reperto condotto.

↓ Datagramm IPsec



Viene cifrato il payload insieme all'header utilizzando algoritmi di cifratura a blocchi. Ci sono esp trailer per evitare una divisione multipla del blocco.

A quale SA appartiene datagramma

↓

Su parte viola il verde cifrata si calcola il MAC (ESP auth). Si mette tutto dentro a nuovo data frame IPsec come payload e viene inviato ed instradato. La destinazione si accorge che è IPsec quindi guarda SPI e cerca nel database la SA. Nell'entrata corrispondente alla SPI trova tutte le informazioni delle prime e per prime cosa verifica l'integrità. Lo decifra poi con la chiave presente nel dh ottenendo la parte verde decifrata e si ricorda il trailer. Si destina poi all'host destinatario.

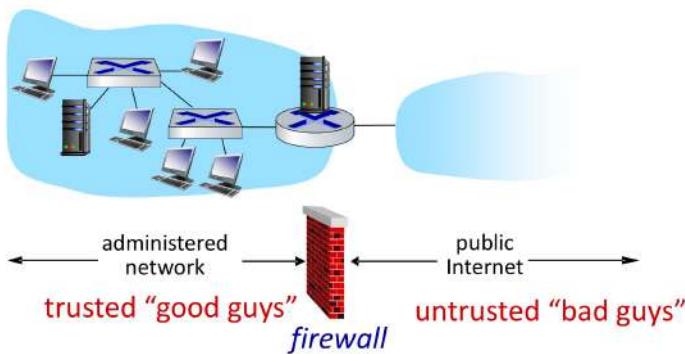
↓ 2 db

Security Association Database: Ci dice cose fare

Security Policy Database: Politica da avere: lo traffico come IPsec o lo lascio inviato?

- **SICUREZZA OPERAZIONALE (a livello di sistema)**

Firewall: voglio proteggere la mia rete da attacchi esterni e voglio anche limitare alcuni accessi verso l'esterno (siti non importanti d esempio)



Possiamo progettare il firewall alle oura di diversi: è indispensabile avere le porte per comunicare con l'esterno ma bisogna avere le guardie per fare i controlli. Il firewall crea un muro per controllare gli accessi alla rete aziendale. Si ha ovviamente doveva un punto di accesso: router di frontiera.

↓

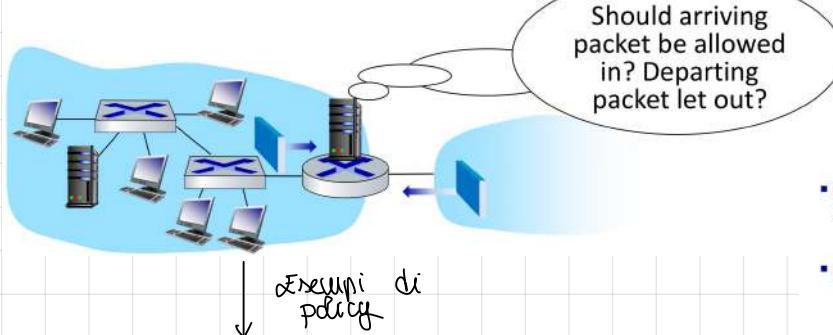
In questo caso vengono inseriti i pacchetti buoni e non buoni. Il firewall ha quindi la funzione di denial of service come ad esempio il SYN flooding: molti attaccanti inviano richieste di apertura di connessione TCP che il server accetta e allora riceve per permettere la comunicazione. Avendo molte richieste, le riceve prima o poi terminano ed il server non può più accettare richieste. Il firewall infatti previene accessi illegitti ai dati e fornisce accessi autorizzati dall'esterno all'interno (il sito web non deve essere rifiutato).

↓ diversi approcci firewall

Filtraggio di pacchetti: può essere stateless o stateful
↳ Solo selezione su pacchetti e non su utenti (mantiene informazioni su connessioni TCI create al momento)

Firewall a livello applicazione: Ogni applicazione deve passare attraverso il gateway

Filtraggio stateless



→ Firewall implementato su router di frontiera ed implementati alcuni campi IP source / dst, TCP/UDP source / dst / port, ICMP, TCP SYN e ACK hits.

▪ example 1: block incoming and outgoing datagrams with IP protocol field = 17 and with either source or dest port = 23

- result: all incoming, outgoing UDP flows and telnet connections are blocked

▪ example 2: block inbound TCP segments with ACK=0

- result: prevents external clients from making TCP connections with internal clients, but allows internal clients to connect to outside

Policy	Firewall Setting
no outside Web access	drop all outgoing packets to any IP address, port 80
no incoming TCP connections, except those for institution's public Web server only.	drop all incoming TCP SYN packets to any IP except 130.207.244.203, port 80
prevent Web-radios from eating up the available bandwidth.	drop all incoming UDP packets - except DNS and router broadcasts.
prevent your network from being used for a smurf DoS attack	drop all ICMP packets going to a "broadcast" address (e.g. 130.207.255.255)
prevent your network from being tracerouted	drop all outgoing ICMP TTL expired traffic

ACL
Access Control List

action	source address	dest address	protocol	source port	dest port	flag bit
allow	222.22/16	outside of 222.22/16	TCP	> 1023	80	any
allow	outside of 222.22/16	222.22/16	TCP	80	> 1023	ACK
allow	222.22/16	outside of 222.22/16	UDP	> 1023	53	---
allow	outside of 222.22/16	222.22/16	UDP	53	> 1023	----
deny	all	all	all	all	all	all

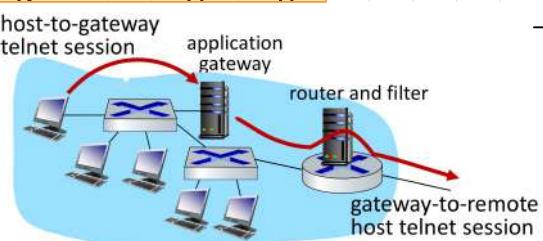
Chiamata di default (deve essere presente sempre)

Filtraggio stateful

Alcuni pacchetti vengono ammessi ma non hanno senso → ACK commissione TCP quando non c'è connessione TCP attiva: può essere molti altri quando non si deve fare entrare. Se il gateway lavora in modo stateless non può "concatenare" le due connessioni.
 ↓ Stateful

Si "prende nota" di connessioni TCP aperte e chiuse: il pacchetto ha senso solo se fa connessione TCP a cui fu riferito è attivo. Altrimenti il pacchetto viene scartato.

Application gateways



→ permette di scegliere cosa gli utenti possono e non possono fare. Deve essere implementato all'interno della rete e l'utente deve prima aprire la connessione verso il gateway che si occupa, se l'utente è autorizzato, di aprire la connessione verso l'esterno. Questa procedura deve essere ripetuta per ogni applicazione.

Limitazione: controllo di pacchetti → pacchetti buoni possono comunque intrarre la rete (IP spoofing) → Se viene cambiato l'IP sorgente il firewall può essere ingannato

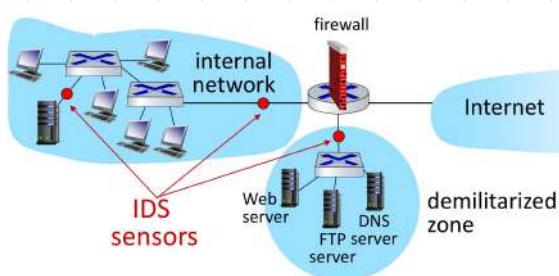
Ispezione "superficiale" dei singoli pacchetti → In alcune case ne guardano anche il contenuto: può anche utile controllare la port scanning (individua porta non protetta), e attacchi DDoS

Si utilizzano intrusion detection systems.

↓

Firewall in acciaio con eventualmente un gateway applicativo. Le rete vengono divise in accessibili facilmente e non (dell'esterno). Quelle accessibili facilmente vengono messe nella zona demilitarizzata, con controlli meno restrittivi.

Nella zona militarizzata vengono messi più DS per garantire > sicurezza.



Wireless and mobile networks

2 aspetti importanti: connettività wireless → utenti si connettono a reti wireless tramite dispositivi mobili

mobilità → dispositivi che cambiano posizione nel tempo

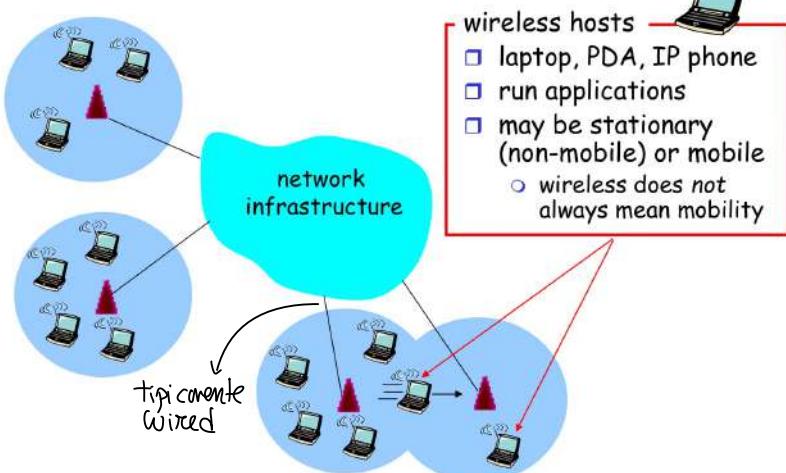
→ Host di tipo diverso: laptop, smartphone, IoT. Indipendentemente dal tipo, ogni host grava le applicazioni. Gli host possono essere stationary o mobili.

↓ collegamento a rete

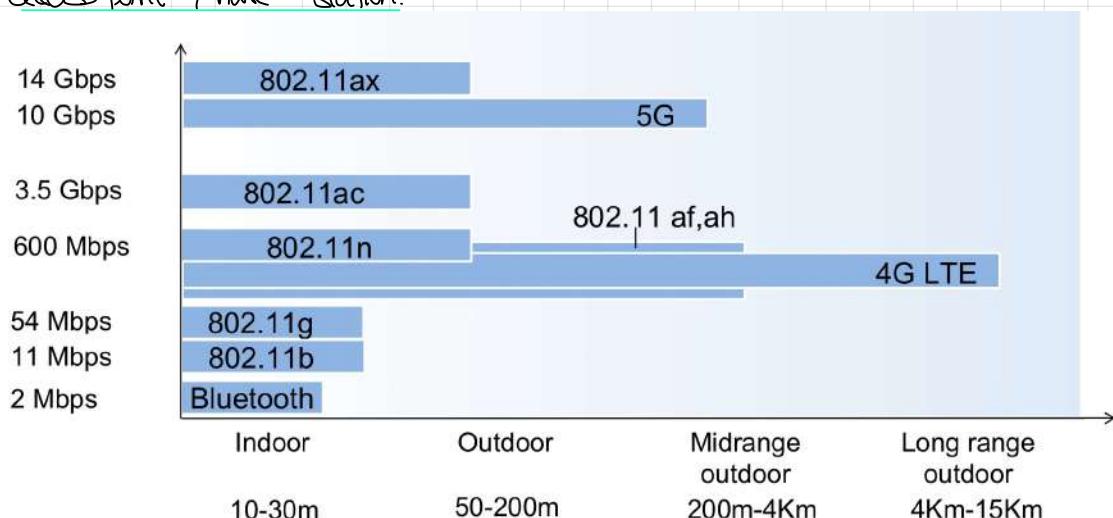
Access point (reti Wi-Fi)

Base Station (reti cellulari)

che coprono un certo hotspot oppure cella. Il utero wireless è broadcast e cumdivo, quindi ci sono un protocollo di accesso



→ il bitrate varia anche a seconda della distanza tra host e access point / base station.



Classificazione (hanno o no infrastruttura)

Le reti con infrastruttura prevedono un'infrastruttura wired con AP che fornisce accesso wireless nell'hotspot. Se il dispositivo è mobile si può spostare da un AP/BS ad altri → handoff. trasferimento ruote in modo transparente. L'accesso wireless è di fatto un'estensione dell'infrastruttura wired.

Le reti senza infrastruttura (ad hoc) prevedono device nodi che generano automaticamente la rete (all'interno del ragno di copertura degli host).

↓

Le reti Wi-Fi e cellulari hanno infrastruttura mentre Bluetooth viene utilizzato senza infrastruttura. In queste reti il collegamento è a singolo hop. Esistono le reti di sensori: sono le reti estese. Possono avere multi-hop anche reti wireless ad hoc (comunicazione tra veicoli: VANET)

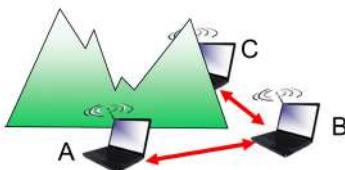
Caratteristiche

1) link wireless differiscono da quelli wired. potenza del segnale diminuisce quando si propaga (con la distanza). path loss. Il utero wireless è sensibile alle interferenze nella stessa frequenza (alcune bande sono già assegnate). Si dice infine multipath propagation: segnale si propaga con percorsi diversi. Più quindi "shuttering" sui percorsi e riflessi. Possono quindi ricevere sbagliati con un effetto che può essere migliorativo o peggiorativo.

Minimizzazione qualità: SNR = rapporto tra potenza del segnale e potenza del rumore. Se è più alto la qualità è migliore. Per le nostre applicazioni ci interessa il BER (Bit Error Rate): lo vogliamo più basso possibile. Fissato il livello fisico, se si aumenta la potenza di trasmissione aumenta il SNR e diminuisce di conseguenza il BER. Alcune tecniche di codifica sono più robuste e quindi hanno un BER più basso → le tecniche di codifica più robuste hanno hitrate più basse (ri una più energia & singolo bit). A parità di BER nello quindi la codifica con hitrate più elevate. Conviene scegliere codifiche dinamicamente in base alla qualità della connessione.

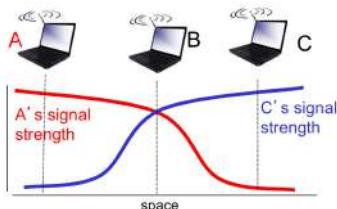
↓ altri problemi

Problema del nodo nascosto



→ A trasmette a B e B comunica con C. C è però invisibile ad A e vice-versa. Se C vuole trasmettere a B trae il mezzo libero e lo stesso accade se A trasmette a B. Si verifica una collisione al nodo B.

Signal attenuation



→ Il segnale diminuisce con l'aumentare della distanza del trasmittitore: può avere confuso da una certa distanza in poi come rumore di fondo. A vuole trasmettere a B e anche C vuole trasmettere a B. C interpreta il segnale di A come rumore di fondo e quindi inizia a trasmettere: collisione.

• Wi-Fi (Wireless dan standard IEEE 802.11)

Evoluzione:

IEEE 802.11 standard	Year	Max data rate	Range	Frequency
802.11b	1999	11 Mbps	30 m	2.4 Ghz
802.11g	2003	54 Mbps	30m	2.4 Ghz
802.11n (WiFi 4)	2009	600 Mbps	70m	2.4, 5 Ghz
802.11ac (WiFi 5)	2013	3.47Gbps	70m	5 Ghz
802.11ax (WiFi 6)	2020	14 Gbps	70m	2.4, 5 Ghz
802.11af	2014	35 – 560 Mbps	1 Km	unused TV bands (54-790 MHz)
802.11ah	2017	347Mbps	1 Km	900 Mhz

Viene evata generalmente nella versione con infrastruttura: switch ethernet + access point. In questo caso il home service set è costituito dagli AP e dagli host mentre nel caso di utilizzo ad hoc ci sono solo gli host. Ogni banda è divisa in 11 canali che possono essere usati per la trasmissione. Si cambia canale per non creare interferenza. I device del BSS devono usare lo stesso canale: fase di associazione iniziale in cui il device ascolta su tutti i canali fino a trovare quello dell'AP. Il device rimane in ascolto fino a beacon: frame speciale con cui AP comunica presenza, SSID e MAC address. Un device può ricevere beacon da più AP nelle vicinanze e deve scegliere in base a dei criteri. La procedura di associazione può coinvolgere anche di una fase di autenticazione. Una volta conclusa l'associazione il device acquisisce l'IP (tipicamente DHCP). Lo scanning iniziale può essere passivo, nel quale il device si avvicina sul canale, riceve il beacon, sceglie l'AP con una richiesta diretta e attende la risposta oppure attivo in cui si invia un probe request dal device ed una probe response dell'AP e poi si fa l'associazione standard.

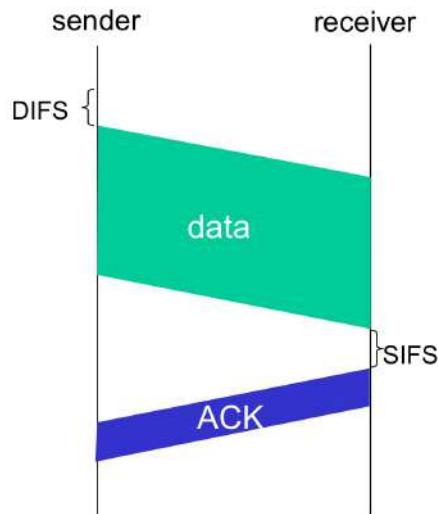
↓ Si forma BSS

La comunicazione è wireless ed il mezzo è condiviso: necessario protocollo per l'accesso multiplo. Non è possibile usare CSMA/CD in quanto il device dovrebbe trasmettere

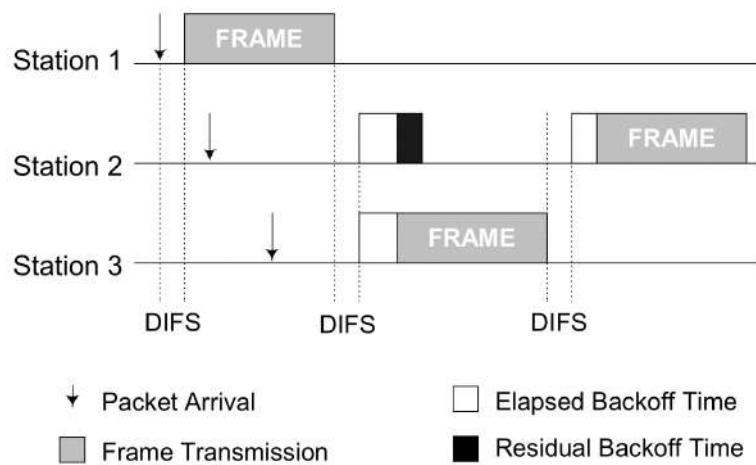
è difficile contemporaneamente, il che è impossibile con l'antenna. Anche duplicando l'antenna rimane però il problema del nodo nascosto in cui la collisione avviene nel nodo destinatario: CSMA/CD non riesce a rilevare la collisione (della sorgente)

↓ Soluzione

CSMA/CA (Collision Avoidance): Il ricevitore si accorge della collisione e lo comunica al trasmettitore.



↓ Supponiamo che ci sono più nodi (stationi)



1. Trasmettitore "sente" il metto e attende che il metto sia libero per DIFS. Se è libero il frame viene trasmesso tutto intero Messo libero

2. Receiver manda, dopo SIFS, TCK. SIFS va atteso in quanto il receiver deve "transformarsi" da ricevitore a trasmettitore per mandare ACK. Per lo standard SIFS > SIFS per avere > priorità ad ACK.

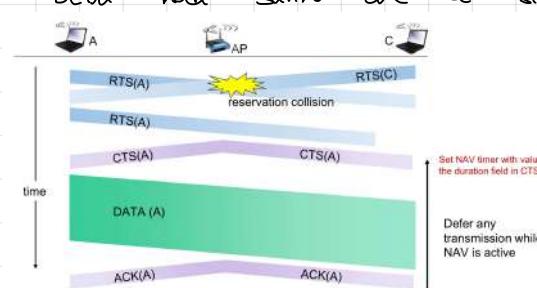
Esempio ricevuto correttamente = no collisioni e no errori. Se non viene ricevuto l'ACK qui verrà verificata una collisione oppure c'è stato un errore rincontrato durante il controllo di correttezza. È utile ricordare i 2 casi in quanto solo nel 1° ha senso attendere. Di fatto assumiamo poi che si sia verificata una collisione (scelta conservativa).

Inizialmente le station 2 e 3 vogliono trasmettere ma il metto è occupato. Se il nodo attende solo DIFS la probabilità di collisione è molto elevata. Si fa quindi una cassa cassola: generato intervallo di backoff (come ethernet). Il timer viene attivato dopo che il metto è stato libero per almeno un DIFS. Quando il timer scade può avviare la trasmissione. Se stanno fanno gli altri nodi di avviso di eventuali pacchetti.

→ Come scegliere l'intervalllo? Si sceglie un intervallo casuale nella finestra $[0, CW-1]$ ovvero CW è uno slot di tempo. Inizialmente $CW = CW_{min}$. Se dopo che si è atteso il backoff time non si è ricevuto l'ACK si assume che ci sia stata una collisione e quindi ottengono $CW = 2 * CW$ fin quando $CW = CW_{max}$ per evitare ritardi eccessivamente lunghi.

Non dicono collisioni: 2 nodi possono generare lo stesso frame e rimanere sempre il problema del nodo nascosto

Virtual Carrier Sensing: Vengono inviati frame RTS e CTS. RTS viene trasmesso da trasmettitore in broadcast ed il destinatario manda un CTS che lo autorizza a trasmettere. Il vero destinatario del CTS sono i nodi ad esso intorno in quanto gli altri nodi sanno che è stato autorizzato un altro nodo a trasmettere.



Finché il NAV è attivo il metto è occupato

Non rendere il problema delle collisioni in quanto possono avvenire di RTS e CTS. Annicato però diminuito la probabilità in quanto i pacchetti sono piccoli e quindi di conseguenza si riduce anche il danno. Il meccanismo non ha vantaggi se si tratta di pacchetti piccoli.

Formato del frame Wi-Fi

byte	2	2	6	6	6	2	6	0 - 2312	4
	frame control	duration	address 1	address 2	address 3	seq control	address 4	payload	CRC

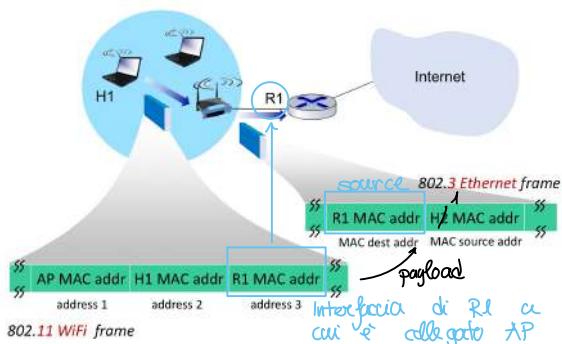
Address 1: MAC address of wireless host or AP to receive this frame

Address 2: MAC address of wireless host or AP transmitting this frame

Address 4: used only in ad hoc mode

Address 3: MAC address of router interface to which AP is attached

duration: utilizzato per iniziare timer AIFS



frame control:

2	2	4	1	1	1	1	1	1	1
protocol version	type	subtype	to AP	from AP	more frag	retry	power mgt	more data	WEP

direzione frame
frame type (RTS, CTS, ACK, data)

Gestione della mobilità

Un host può muoversi da un access point ad un altro. Supponiamo che gli AP siano collegati tramite uno switch e questo capisce dove iniettare i pacchetti con un meccanismo di self learning: inizialmente continua ad iniettare i frame dal primo AP fin quando non trova un porto di esterno in modo che lo switch sappia dove iniettare i pacchetti.

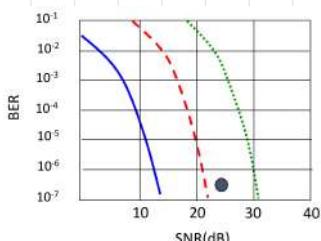
Advanced capabilities

1. Rate Adaptation

La qualità varia di molto, principalmente a causa dell'instabilità dell'etere come metto di trasmissione. Il BER sperimentato dal ricevitore può risultare troppo elevato portando a retransmissioni che risultano più inefficaci. Si parla quindi di codifica più robusta che produce un BER più basso avendo perciò un hitrate più basso → automaticamente deciso da interfaccia di serie.

2. Power Management

I dispositivi wireless funzionano generalmente a batteria. Non è un grosso problema per telefoni / tablet mentre è un problema per sensori. Vogliamo trovare un modo per trattenere e ricevere tenendo "svegli" l'interfaccia wireless. Un modo non ha recenti di tenere l'interfaccia wireless sempre accesa. L'AP "lavora" al posto dei nodi wireless: i pacchetti ricevuti vengono soffrievati nell'AP quando il nodo dorme ed il nodo li scatta periodicamente → beacon periodico da AP: pacchetti disponibili per nodi destinatari. Se un nodo non vede pacchetti a lui destinato continua a dormire. Questo approccio introduce un ritardo dovuto al beacon time.



• BLUETOOTH

Personal Area Network: raggio di trasmissione < Wi-Fi. Inizialmente pensata come tecnologia di cable replacement. È principalmente usata in modalità ad hoc e lavora sulla banda 2.4 GHz. Ha un hitrate di circa 3Mbps. La rete è organizzata come master-slave: il master è colui che "inizia" la rete mentre gli slave sono i client di cui alcuni possono essere parked (radio spenta) periferici, rolling: master organizza trasmissione.

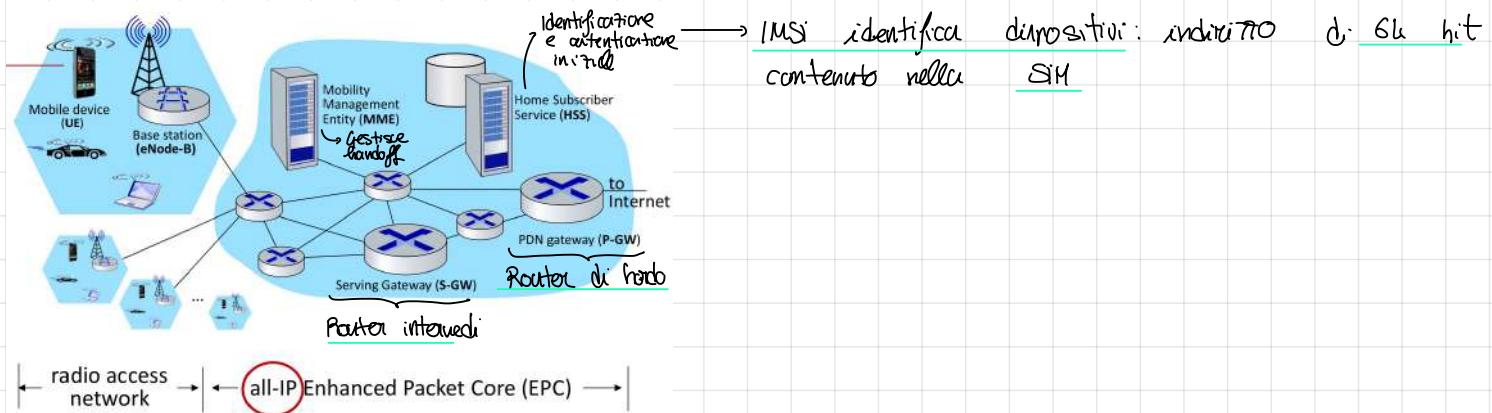
Vedere metti TDM con un tempo di 625 μsec ed avendo che il canale è one-way, gli slot possono essere usati via via dai client che dal master ce ne torino. La divisione non è equis in quanto il master prende metti hand a, la stessa che tutti gli slave devono diedersi.

Frequency hopping: se sulla frequenza c'è un'interruzione il ricevitore non riceve. Allora alle bande 2.4 - 2.5 GHz vengono creati \Rightarrow canali che vengono tutti utilizzati a rotazione. Il canale è scattizzato in base al tempo (slot durata = master, pari = slave) e quindi viene utilizzato un canale + slot di tempo. Tutti i nodi cambiano canale in maniera coordinata per rendere possibile la comunicazione. Tuttavia la desincronizzazione è sempre possibile e le interferenze possono esserci ancora separate mitigati. Tutti devono condividere la sequenza di hopping (condotta da master a slave). Comunque manca una sequenza predefinita per evitare interferenze di master vicini o di multi-attivazioni che potrebbero interfare tutte le comunicazioni seguendo la sequenza predefinita.

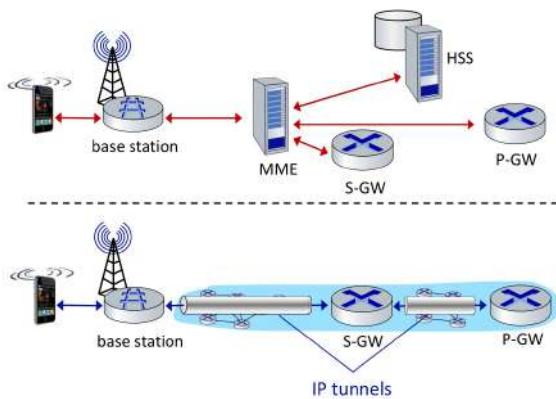
• RETI CELLULARI

Rete maggiormente utilizzata da dispositivi mobili ed ha un bit rate di circa 100Mbps. La rete 4G ha una parte edge e core appartenenti alla stessa società ed è una rete globale che usci i protocolli già visti ed è interconnessa alla rete pubblico. A differenza di internet, il wireless ha un different link layer, si tratta della mobilità e si ha un identity con la SIM. Il contratto con un operatore fornisce l'accesso alla propria rete home oppure permette di fare roaming con gestori con cui hanno contratto

↓ Struttura rete LTE



↓ Divisione tra piano controllo e piano dati

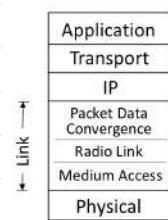


control plane

- new protocols for mobility management , security, authentication

data plane

- new protocols at link, physical layers
 - extensive use of tunneling to facilitate mobility



LTE link layer protocols:

- Packet Data Convergence: header compression, encryption
 - Radio Link Control (RLC) Protocol: fragmentation/reassembly, reliable data transfer
 - Medium Access: requesting, use of radio transmission slots



Come abbiamo visto per Wi-Fi e BT si hanno anche in questo caso meccanismi di power saving:
 - light sleep: ci si risveglia ogni 100 ms
 - deep sleep: 5-10 s di inattività

• MOBILITÀ

no mobility

high mobility

device moves between networks, but powers down while moving

Norma di coda: minimizza mobilità

device moves within same access net in one provider network

protocolli di liv. superiore non si ricongiungono

device moves among access nets in one provider network

device moves among multiple provider networks, while maintaining ongoing connections

Wireless and Mobile Networks: 8-55

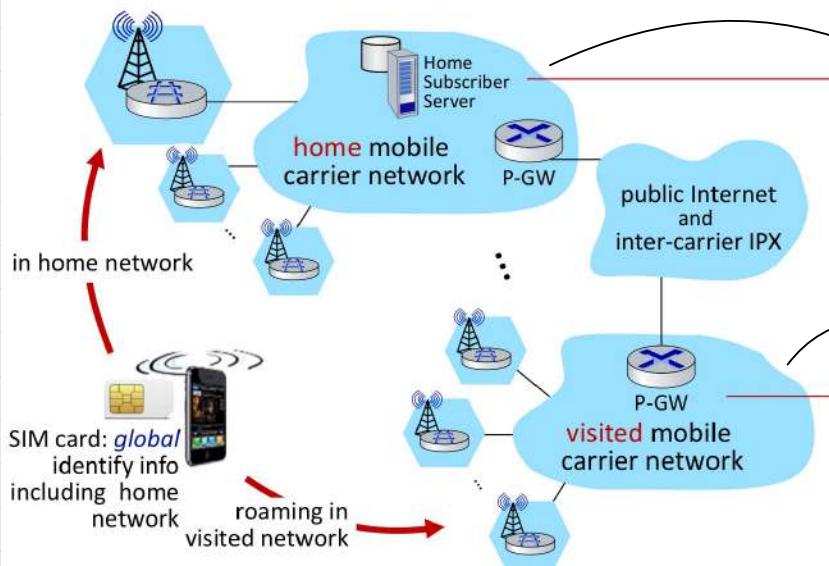
Approccio

Funzioni di filo ad es: routing indiretto

→ se si vuole comunicare con un host mobile si fa riferimento alla sua home network.

routing diretto

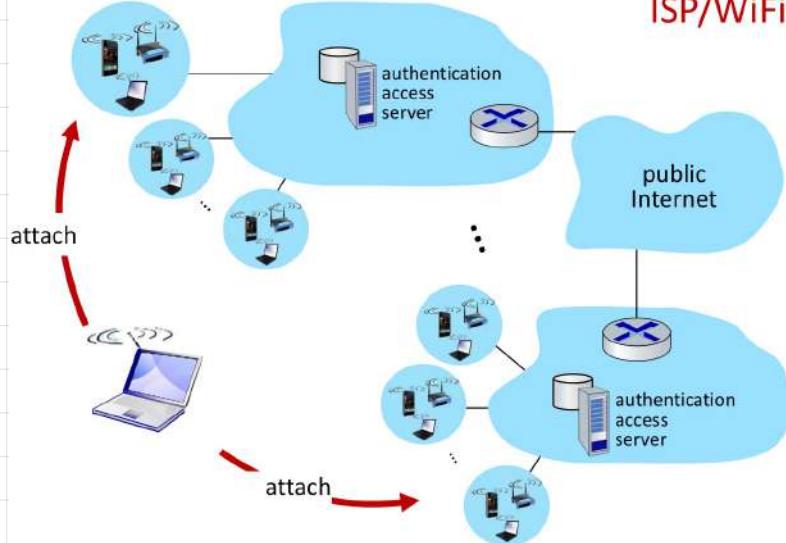
→ si rivolge all'indirizzo della home network ma questa volta avere l'indirizzo con cui comunicare direttamente con l'host mobile.



Home network: tutta la rete del provider di cui abbiamo la SIM

Visited network: rete in cui è spostato → Roaming: origine ha un contratto permette redirezione facile

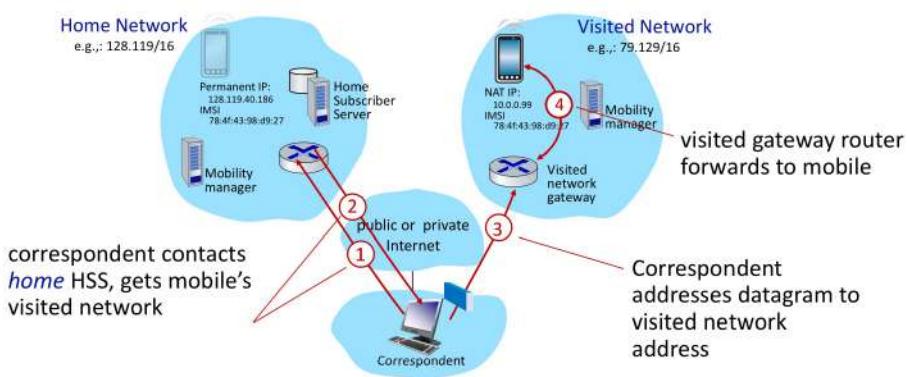
ISP/WiFi: no notion of global "home"



- credentials from ISP (e.g., username, password) stored on device or with user
- ISPs may have national, international presence
- different networks: different credentials
 - some exceptions (e.g., eduroam)
 - architectures exist (mobile IP) for 4G-like mobility, but not used

All'intorno della home network si ha un IP ad essa relativa che cambia nella rete visited e può essere pubblico o privato. D'insi rimane però lo stesso. Il home subscriber server traccia l'host nella rete home mentre i mobility manager gestiscono la mobilità. Il correspondent vede comunicare con l'host mobile con continuità (senza interruzione notizia). Quando l'host mobile entra nella rete visitata si registra con il mobility manager che lo fa sapere rete home dell'host tramite l'home subscriber server. Il correspondent manda il messaggio all'IP iniziale dell'host e l'home gateway, sapendo che l'host non è sulla rete home, fa il forward verso il gateway della rete visitata che lo instrada correttamente all'host. Quando l'host risponde il datagramma può fare il percorso inverso oppure può essere inviato direttamente dal router della visited network al correspondent.

⚠ Problema: triangolare → Il messaggio può fare giri inutili se chi deve comunicare sta sulla stessa rete in quanto si parla comunque dell'home network



L'ultimo avviene con il routing indiretto ma il gateway fornisce l'indirizzo dell'host sulla visited network a cui il correspondent invia il datagramma: la mobilità non è più trasparente e non ci ha più triangolazione

⚠ Cosa succede in caso di mobilità verso altre reti visitate?

Routing indiretto: Il mobility manager della seconda rete visitata avvisa l'home subscriber server che reconfigura l'home network gateway per fare il forward in tale rete visitata da continuità viene interrotta solo quando viene fatto di configurazione è in corso e non è ancora completa: perdita di datagramma ma non è un problema in quanto IP è best effort.

Routing diretto: Viene semplicemente comunicato l'indirizzo nella nuova rete. Se commutare TCP viene però interrotta in quanto cambia l'IP: non è garantita la continuità. Poi il correspondent comunica con mobility manager della RPL senza modifiche per la MN: catena di instradamenti.

Impatto sui livelli superiori

Durante le fasi di configurazione alcuni datagrammi vanno persi. Inoltre, anche il protocollo di congestione è impattato: nelle reti wireless capitano perdite che il protocollo considera causate da congestione e le perdite sono considerate anche dalla mobilità. Viene attivato il protocollo di controllo di congestione impattando il throughput. Si usa perciò il meccanismo explicit control notification.

Algoritmi di routing

che percorre su un pacchetto per andare da un host ad un altro? Problema di ottimizzazione sul grafo

↓ 2 approcci

1. Qualcuno mi fornisce la topologica della rete: matrice di incidentata ($n \times n$ ove $n = \text{node}$). Abbiamo poi bisogno dei costi delle connessioni (archi): può essere espresso in termini di tempo di transito, di distanza e di congestione (quanto traffico ci sta già passando). A differenza di ricerca operativa però capiamo bene che i costi non sono fissi. Si parla in questo caso di Dynamic Optimization. Non si ha più $C_{i,j}$ ma $C_{i,j,t}$. Rimane invece costante la distanza $d_{i,j}$ mentre varia il tempo di transito (dipendente dalla congestione).

⚠ d_{i,j} è costante in quanto i router sono fissi.

2. Conosciamo solo la topologica locale (il vicinato). Si ha poi la programmazione dinamica per arrivare all'ottimo: path di router di costo minimo. Ogni router viene detto hop → ROUTE OTTIMA = rute di costo minimo. Per l'implementazione vengono utilizzate le tabelle di scouting e i grafi in cui i nod sono gli hop (router) che si occupano di fare infradamento. I grafi sono stazionari ($C_{ij} = C_{ji}$).

Algoritmi:

Approccio globale: topologia completa e costi → Dijkstra: link state algorithm

Approccio distribuito: distance vector: informazioni solo sui router vicini con cui comunicano

↓ possono essere informazioni per raggiungere path ottimo.

Statici: Nel tempo le rotte cambiano lentamente (ci hanno ad esempio sulla distanza)

dinamici: Le rute cambiano velocemente (ci hanno sulla congestione)

• ALGORITMI DISTANCE-VECTOR

Problema diviso in sottoproblemi → alg. Bellman-Ford: $d_x(y)$ = costo cumulativo minimo da x ad y. Allora $d_x(y) = \min_y \{C(x,y) + d_y(y)\}$ [3 elementi se x ha 3 vicini]

↓ descrizio vicini di x: comm. due have

Bellman-Ford example



$$d_v(z) = 5, d_x(z) = 3, d_w(z) = 3$$

$$d_u(z) = \min \{C(u,v) + d_v(z), C(u,x) + d_x(z), C(u,w) + d_w(z)\}$$

$$\downarrow = \min \{2+5, 1+3, 5+3\} = 4$$

u^* = nodo che raggiunge il minimo: next hop.

$$u^* = \arg \min_y \{C(x,y) + d_y(y)\}$$

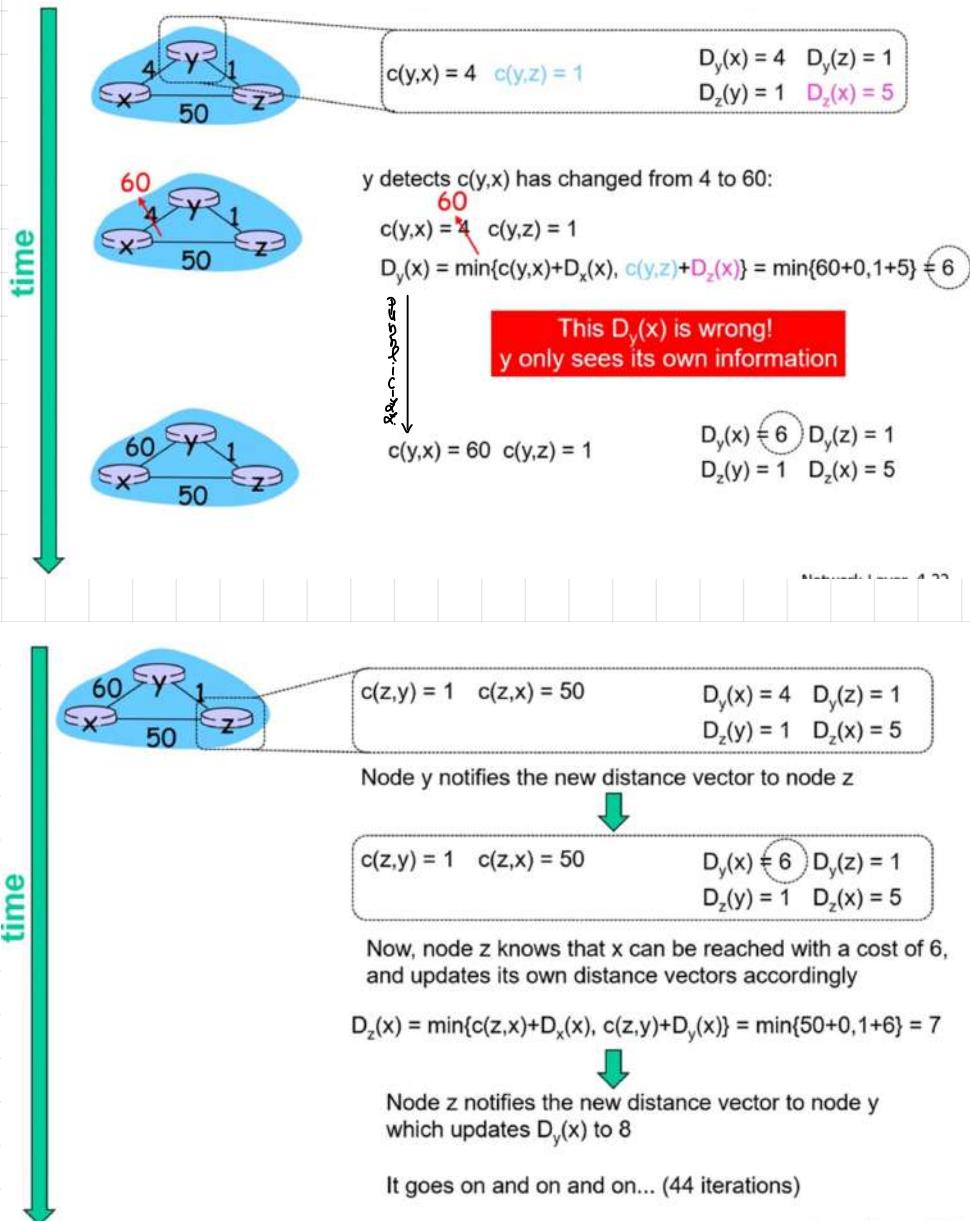
Supponiamo $d_x(y)$ = strada costo min → X manda via un vettore di distanze $D_x = [d_x(y) : y \in N]$ ove X = insieme dei nodi. X quindi conosce $c(x,y)$ e $\forall y \in N$ ha $D_y = [d_y(y) : y \in N]$

Idea: Ogni tanto ogni nodo invia il proprio DV ai vicini (lo pubblicizza). Distribuiscono quindi la complessità dell'algoritmo centralizzato. Quando il nodo riceve il DV da un vicino aggiorna il proprio con BF. Questo ha senso in quanto la congestione può variare molto: $d_x(y) \leftarrow \min_y \{C(x,y) + d_y(y)\}, \forall y \in N$. Facendo così la strada converge all'effettivo costo minimo.

↓ Funzionamento

Algoritmo iterativo asincrono event-based: evento = cambio capacità di flusso sul link. \rightarrow indirizzi distribuiti (vicini informati quando DV cambia → aggiorni → cambia) dono l'aggiornamento del DV. ⚠ Problema: quando cambiano i costi sui link il nodo se ne accorge, aggiorna DV e comunica ai vicini. Se il costo era distribuito i nodi vicini vengono a saperlo subito e la tecnica torna velocemente. Nel caso di costo aumentato si ha il problema di congelio di ∞ : 4th iteration neanche per rendersi conto dell'aumento del costo. Questo accade perché i nodi vedono solo i vicini e non tutta la rete

N	S
E	E
O	C
O	P



Confronto tra i DS e il DV: scambio msg

DS \rightarrow n nodi ed è a cerchi: $O(n^2)$ messaggi

DV \rightarrow Scambio msg solo fra vicini

Velocità convergenza

DS \rightarrow $O(n^2)$

DV \rightarrow può variare a causa del controllo dell'infinito
e robustezza (questi)

DS \rightarrow pubblicizzato costo shaglato ma ogni router calcola la propria tabella di costi: link cost errato

DV \rightarrow gli shaghi impattano anche i vicini secondivamente path cost errato

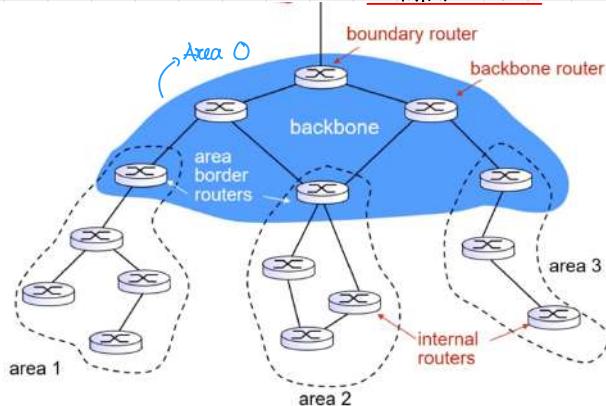
Utilizziamo un approccio hybrid

• APPROCCIO GERARCHICO

Perdizionamento non più flat, in cui tutti i router hanno lo stesso ruolo. Si prende spunto dalla definizione di Internet: rete di reti in cui ciascuna è un "unità amministrativa" ciascuna con le proprie regole. I router vengono quindi aggregati in autonomous system, ciascuno con la propria policy (es. rete TIM, Vodafone, Google, Apple,...). Avremo quindi un routing intra-AS indipendentemente dagli ASes. Ci serviranno poi altri protocolli per il trasferimento tra i diversi ASes: inter-AS. Per uscire da un AS per prima cosa si individua l'uscita (router di frontiera) e lo shortest path per arrivareci. A questo punto l'algoritmo inter-AS mi porta all'AS di destinazione, in particolare al router di frontiera. Questo applica un algoritmo intra-AS per

area verso alla destinazione.

Protocollo intra-AS: OSPF → Open shortest path first: Algoritmo link state con topologia distribuita. Viene usato Dijkstra. La topologia viene divisa in tre livelli con messaggi che hanno una entry + node: flooding → Dijkstra ha bisogno delle informazioni su tutta la rete: i cambiamenti devono avere diffusi a tutti. I router hanno commiti diversi per evitare flooding eccessivo →



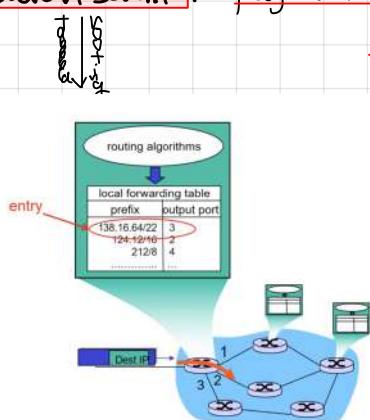
I router vengono partitionati in aree in cui si cerca un deposito di S. Quando l'area cambia i pacchetti fluiscano verso i router di back bone che sostanziano il traffico. Si divide tra aree e back bone riduce di molto le informazioni necessarie. Per evitare ci sono i area border routers. Per trovare lo shortest path ovviamente i router di area (+0) e quelli di back bone collaborano.

Protocollo inter-AS: BGP → Border Gateway Protocol

eBGP: Ottiene raggiungibilità da AS vicini (dove posso andare)
iBGP: Propaga informazioni a router interni. Questo avviene perché gli AS comunicano e poi andare dall'interno all'esterno si deve sapere quel è l'uscita più veloce.

Hanno anche capretti politici: un AS avrebbe potrebbe rifiutarsi di trasportare pacchetti generati da un AS esterno verso un AS esterno pur avendo sullo shortest path → Può essere richiesto un pagamento: servizio di transito.

advertisements: prefisso + attributi = route



AS-PATH: AS da attraversare per arrivare a destinazione

NEXT-HOP: router che ci serve per cambiare AS

→ router interni: si trovano su iBGP e sull'esecuzione di OSPF

→ Entrata nella tabella: il router diventa consente di un prezzo, determina l'uscita per tale prezzo, mette la copia nella forwarding table ed aggiorna la tabella di routing.
Se un router riceve più pubblicazioni per lo stesso indirizzo guarda l'AS-PATH e viene scelto quello minimo.

Skewtaggio: gli AS potrebbero avere sotto grandi e questo non viene considerato