RETI A CIRCUITO VS RETI A PACCHETTO

CIRCUITO	PACCHETTO
• Risorse DEDICATE	• Risorse CONDIVISE
• Servizio garantito: banda costante, ma non massima; no perdita; ritardo costante	• Si trasmette al massimo rate possibile, ma si possono avere perdite, ritardo, congestione
• Richiede connessione (FDM o TDM)	• No connection needed (servizio non garantito)
• Variazione del ritardo = 0	• Ritardo variabile, si possono creare code (STORE & FORWARD)
• Si sprecano risorse su trasmissioni intermittenti, servendo solo pochi utenti	• Si possono servire più utenti, a scapito di un po' di congestione occasionale che può portare a RITARDO e PERDITA DI PACCHETTI

DELAY

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

- d_{proc} = RITARDO DI PROCESSAMENTO → Controlla i bit errors e determina il link di output; è piccolo e quasi costante (< ms/μs)
- d_{queue} = RITARDO DI CODA → Ritardo aspettando in coda nel buffer del router, è molto VARIABILE e dipende dalla congestione, ha l'impatto maggiore!
- d_{trans} = RITARDO DI TRASMISSIONE → $d_{trans} = \frac{L}{R}$ con L = dim. pacchetto, R = link bandwidth
  tende ad essere costante e trascurabile
- d_{prop} = RITARDO DI PROPAGAZIONE → Dipende dalle distanze e dalle velocità di propagazione del link:
  $d_{prop} = \frac{d}{s}$ con $s \simeq 2 \cdot 10^8$ m/s
  Effetto importante a grandi distanze.

* Intensità del traffico = $\frac{L \cdot a}{R}$, con a = rate d'arrivo medio dei pacchetti.
  Si tenta di evitare che il canale superi il 50% di occupazione, altrimenti il ritardo di coda cresce esponenzialmente, e di conseguenza la perdita.

HTTP

- HTTP = Hypertext Transfer Protocol
- Use TCP sulla porta 80 → CONNECTION-ORIENTED e RELIABLE
- È un protocollo client-server
- È un protocollo STATELESS → I server non hanno memoria delle precedenti richieste dei clients.
- Messaggi in ASCII (in chiaro)

- NONPERSISTENT HTTP: usato in HTTP 1.0; può essere inviato al massimo 1 oggetto per ogni connessione TCP → Lento

- PERSISTENT HTTP: usato in HTTP 1.1; più veloce perché si possono inviare più oggetti sulla stessa connessione, uno dopo le rispettive richieste. È difficile da gestire → Quando chiudere la connessione?
  I server hanno dei timer
  - CON PIPELINING: Non bisogna attendere l'arrivo dell'oggetto precedente per una nuova richiesta, ma si possono inviare più richieste una dopo l'altra → ancora più veloce

- Messaggi di tipo: GET, POST, HEAD, PUT, DELETE
  Campi importanti → Nel GET: campo "Host"
  Nella risposta: campi "Last-Modified" e "Content-Length"

- COOKIES: Sono il modo per rimediare al fatto che HTTP è stateless!
  Ci sono 4 componenti:
  1) Cookie header nella risposta HTTP (Set-cookie: ID)
  2) Cookie header nella richiesta HTTP (cookie: ID)
  3) COOKIE FILE presente nell'host del client e gestito dal browser; file su cui vengono salvati i cookie ID assegnati.
  4) Database di back-end del Server, dove salvare gli ID associati con varie info.

Permettono, ad esempio, di:
- Salvare autenticazione e autorizzazioni
- Carte di credito
- Suggerimenti e pubblicità (mirate)
- Salvare lo stato della sessione utente

* NOTA: I cookies permettono ai siti di sapere molte cose sugli utenti; attenzione alla PRIVACY!

- **CACHING:**

Ha l'obiettivo di soddisfare le richieste dei client senza coinvolgere il server d'origine.

Per questo scopo, si usa come una Web Cache, un **PROXY SERVER**, spesso installato dagli stessi ISP's nelle LAN.

- Un Proxy Server è un Server che fa da intermediario tra un gruppo di client e degli altri server
  - ↳ Fa da { Server per i client; Client per gli altri server } → È sia Server che Client!
- I client inviano le richieste al Server Proxy:
  - Se il Proxy ha l'oggetto in cache, lo invia al client
  - Se non ha l'oggetto in cache, lo richiede ai Server, lo salva in cache e lo invia al client
- Anche ogni client ha una propria Web Cache che sfrutta allo stesso modo, ma in maniera locale.

VANTAGGI:
1. Migliori prestazioni: il Server Proxy è più vicino ai client
2. Molto efficace su client che fanno le stesse richieste (Aziende, compagnie, università, ...)
3. I Server d'origine hanno meno richieste da servire, quindi possono servire più utenti
4. Riduce la CONGESTIONE sia nella LAN che nella rete Internet.

SVANTAGGIO: I contenuti presenti in cache potrebbero non essere aggiornati! Come fare a sapere se i contenuti sono o meno aggiornati?

↳ **CONDITIONAL GET:**

- Sfrutta il campo "Last-Modified" della risposta HTTP!
- Ogni volta che il Proxy Server riceve una richiesta di un oggetto presente in cache, invia un Conditional GET al Server d'origine, con il campo "If-modified-since: <datetime>", dove <datetime> è il valore del campo "Last-Modified" dell'oggetto in cache.
- Se l'oggetto è stato modificato, il Server d'origine risponde con 200 OK, inviando l'oggetto aggiornato; altrimenti risponde con: 304 NOT MODIFIED.

FTP

- FTP = File Transfer Protocol ; permette il trasferimento di files da/verso host remoti
- Utilizza TCP perché ha bisogno di comunicazione AFFIDABILE ; Server su porta 21
- CONTROLLO "OUT OF BAND" (FUORI BANDA) : usa 2 connessioni separate , una per i comandi di controllo (porta 21 del server) e un'altra per la trasmissione dei dati (porta 20 del client).
- È un protocollo client - server
- I server FTP mantengono lo STATO della connessione : autenticazione, directory corrente, ... → Limitano il numero di utenti massimi connessi (HTTP NO), perché gestendo lo stato , si limita la capacità.

Funzionamento :

1. L' FTP client contatta l' FTP server sulla porta 21 usando TCP e AUTENTICANDOSI con username e password
2. Il server invia l'autorizzazione sulla connessione di controllo e il client può navigare tra le directory del server cui ha accesso
3. Quando il server riceve una richiesta per il trasferimento di un file, apre una connessione TCP sulla porta 20 del client
4. Il Server trasmette il file e CHIUDE la connessione dati
5. Il client , se non ha altre richieste , può chiudere la connessione di controllo.

ELECTRONIC MAIL

Si base su 3 componenti principali :

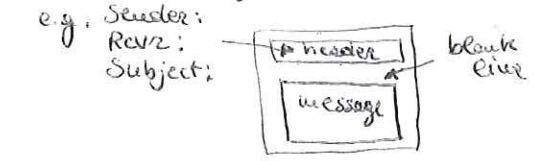
1. User Agent (UA) : mail reader , usati per leggere, editare i messaggi
2. Mail Servers : server mail che si scambiano i messaggi tra loro per farli viaggiare
3. Protocollo SMTP : Simple Mail Transfer Protocol

- Gli user agent gestiscono la posta entrante ed uscente presente nella mailbox dell'utente nel Mail Server
- I Mail Server hanno :
  - Una MAILBOX per ogni UA associato , in cui il server fa storage delle mail ricevute
  - Una MESSAGE QUEUE in cui sono inseriti i messaggi mail che il server deve inviare
  - Il protocollo SMTP , che il Server deve eseguire sia come Client (quando invia) che come Server (quando riceve).

- **SMTP:**
- SMTP = Simple Mail Transfer Protocol
- Utilizza (TCP) sulla porta 25 per la connessione tra i Mail Servers; i client mail di solito inviano le mail in uscita al server sulla porta 587 (la 465 è deprecata)
- Il trasferimento di mail tra i server prevede 3 fasi:
  1. Handshaking
  2. Trasferimento dei messaggi
  3. Chiusura connessione
- L'interazione è di tipo comando/risposta:
  - COMANDI: testo in chiaro in ASCII a 7 bit
  - RISPOSTE: status code + frase

* I messaggi, header compreso, DEVONO essere codificati in (ASCII a 7 bit)!

- SMTP use connessioni persistenti!
- E' un protocollo PUSH (si inviano messaggi; invece HTTP è pull)
- Si usa CRLF.CRLF per terminare il messaggio (praticamente 1 punto in una riga)
- Si possono inviare più oggetti su 1 unico messaggio (in HTTP NO!), con il Content-Type: multipart/mixed.



Formato MIME:

MIME = Multimedia Mail Extension

E' dichiarato con alcune linee nell'header

- Content-Type: type/subtype; parameters → Usato per inviare dati multimediali
- Per inviare più oggetti in 1 unico messaggio, si usa:

  Content-Type: multipart/mixed; boundary = StartOfNextPart

  dove boundary è una stringa generata pseudo-random che fa da separatore tra i vari oggetti
- Spesso, si usa anche multipart/alternative per inviare 2 volte lo stesso oggetto, ma una volta in formato testo e una volta in formato Html!

MAIL ACCESS PROTOCOLS:

Per il recupero delle mail dai Mail Server agli User Agent, si usano protocolli diversi da SMTP; i principali sono:

- POP3
- IMAP
- HTTP

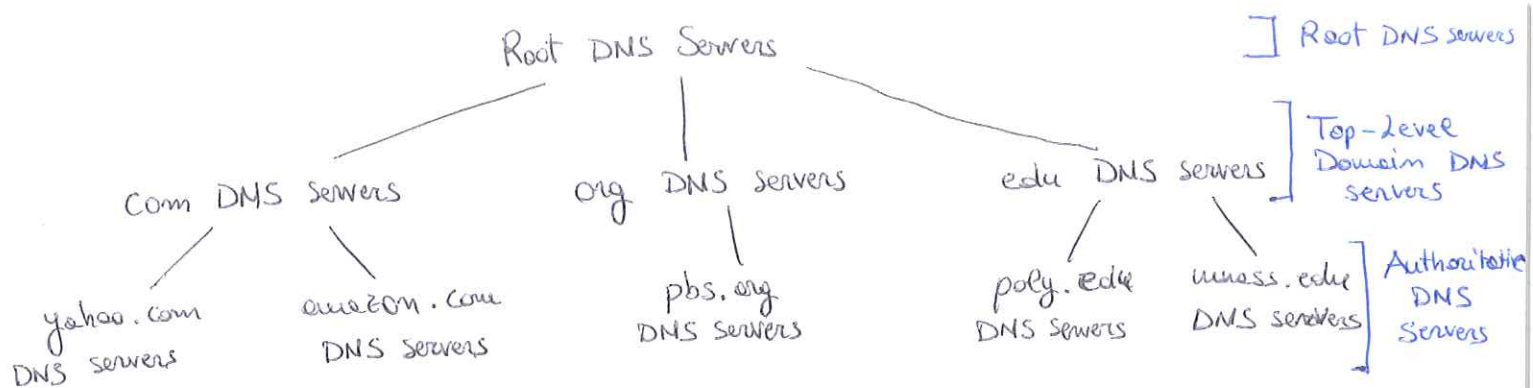
- **POP3:**
- POP = Post Office Protocol
- Connessione (TCP) col mail server su porta 110
- Si sviluppa in 3 fasi l'interazione:
  1) AUTENTICAZIONE : tramite username e password
  2) TRANSAZIONE : scaricare le mail dal server
  3) UPDATE : cancellare dal server le mail segnate come delete e chiudere
- Tipicamente usato in modalità "DOWNLOAD AND DELETE" → (STATELESS tra le sessioni)
  ↳ Non si potranno rileggere i messaggi cambiando client
- Aggiunte in seguito modalità "DOWNLOAD AND KEEP"

- **IMAP:**
- IMAP = Internet Mail Access Protocol
- Mantiene tutti i messaggi nel Server stesso!
- Permette all'utente di organizzare i messaggi in folders direttamente sul server
- È (STATEFUL) → mantiene lo stato attraverso le sessioni: ad esempio i nomi dei folders e il mapping tra ID delle mail e il nome del folder.

DNS

- E' un'applicazione che permette di risolvere il mapping tra NOMI DI DOMINIO – IP address,
- E' sia un:
  DATABASE DISTRIBUITO: implementato in una gerarchia di diversi name servers
  PROTOCOLLO DI LIVELLO APPLICATIVO: host, routers e name servers comunicano per risolvere il mapping tra DOMAIN NAME ⟷ Indirizzo IP
  → COMPLEXITY AT THE "EDGE"!
- DNS offre i seguenti servizi: traduzione tra hostname e IP address, host ALIASING (nomi canonici e alias), Mail Servers aliasing, Distribuzione di carico.
- Perché non CENTRALIZZARE il servizio DNS?
  - Singolo punto di rottura
  - Volume del traffico enorme
  - Svantaggio utenti distanti dal server remoto
  - Manutenzione difficile
  - Zero SCALABILITA' (peggiora all'aumento degli utenti)

DNS = Domain Name System



- **DNS NAME SPACE:**

Bisogna pensare ad Internet come logicamente suddiviso in domini.

- DOMINIO = Insieme di host correlati (stesso tipo, paese, organizzazione, ...); Un dominio può essere suddiviso in sottodomini
- Ogni nodo ha una LABEL che lo rappresenta
- DOMAIN NAME: sequenze di labels da un nodo fino alla radice, separate da un punto (e.g. lopresti.di.univaq.it)
- Ogni dominio ha "AUTORITÀ" per i nomi nel suo dominio; tipicamente delega l'autorità ai suoi sottodomini.

SERVER AUTORITATIVI:

Ogni server è autoritativo (responsabile) di una "ZONA"; una zone è un sottografo connesso di un albero (tipicamente, sottografo connesso di un dominio, che forma un sottodominio).

Ci sono 2 casi:

1. Se Zone = SOTTOALBERO:
   - Il server, per ogni host nella zone, conserva l'indirizzo IP e il suo nome nella zone
   - Può fornire il mapping per quegli host tra hostname e IP address
2. Se Zone ≠ SOTTOALBERO
   - Il server delega autorità per alcuni dei suoi sottodomini a dei server di livello più basso
   - Le informazioni dei nodi sono in questi server di sottodominio
   - Il server mantiene solo dei riferimenti ai LOWER LEVEL SERVERS → Struttura GERARCHICA

LOCAL NAME SERVER:

Non appartiene strettamente alla gerarchia, ma ogni ISP ne ha uno; anche chiamato "default name server".
È da PROXY tra gli host e i DNS servers per le query DNS.
Ogni query di un host locale viene inviata al Local Name Server!

La comunicazione può avvenire in maniera:

① RICORSIVA: il Local Name Server interroga il Root DNS Server che si fa carico di procurargli il mapping richiesto, andando ad interpellare i server di livello inferiore e così via a catena.
Bisogna mantenere lo STATO delle richieste → ONEROSO!

② ITERATIVA: il root DNS server, così come gli altri, se conoscono il mapping lo comunicano, altrimenti restituiscono l'indirizzo del prossimo server (nella gerarchia) a cui chiedere.
E' il Local Name Server a fare tutte le richieste! → STATELESS

- **DNS CACHING:**

Ogni volta che un DNS server impara un mapping, lo salva in CACHE con un TTL associato dopo il quale invalidare l'informazione (SOFT STATE).
Ad esempio, il Local Name Server può salvare in cache anche gli indirizzi IP dei DNS server intermedi.

- **DNS RECORDS:**

Il DNS è un database distribuito che mantiene dei record detti RESOURCE RECORDS (RR):

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

- Type = A → name è l'hostname e value è l'indirizzo IP
  - * Type = AAAA → IPv6 address
- Type = NS → name è il DOMAIN NAME e value è l'indirizzo IP del server autoritativo del dominio
- Type = CNAME → name è l'ALIAS name di qualche name canonico, value è il name canonico (real)
- Type = MX → ~~name~~ value è il NOME del MAILSERVER associato con name.

- **DNS PROTOCOL:**

Query e response messages sono dello stesso formato.

- Use UDP sulla porta 53 → deve essere veloce (NO congestion control e slow start)
- DNS message = 12 byte HEADER + 4 sezioni dati variabili

HEADER: identificativo a 16 bit per query/response, 16 bit flags { query or reply; recursion desired; recursion available; reply is authoritative

+ 16 bit per ogni campo: # of questions, # of answer RR's, # of authority RRs, # of additional RRs } Numero di byte dei 4 campi dati

* Il TTL dei resource records (RR) è utile per evitare di dover comunicare a tutti eventuali cambiamenti di ALIAS name o indirizzi IP. D'altronde, o si comunica broadcast (!!), oppure sarebbe oneroso ricordare chi ha richiesto le info.
Così, invece, chi ne ha bisogno chiede e avrà gli aggiornamenti quando ci saranno!

ARCHITETTURE P2P

In termini di tempo ed efficienza, le architetture P2P sono molto migliori di quelle Client-Server, ma sono difficili da realizzare, per diversi motivi tra cui:

- "server" non sempre connessi
- end-systems arbitrari connessi direttamente tra loro (peers)
- i peers possono collegarsi e intermittenza e/o cambiare indirizzo IP.

- **FILE DISTRIBUTION: BitTorrent:**

TORRENT = Gruppo di peers che si scambiano chunks di un file. Il file torrent contiene metadati sui files e le cartelle da distribuire/scambiare

TRACKER = E' un SERVER che tiene traccia dei peers partecipanti ad un torrent

- File suddiviso in CHUNKS da 256 KB
- Un peer che si unisce al torrent parte senza chunks, ma li accumulerà nel tempo; quando si unisce, si registra con il Tracker e ottiene una lista di peers, connettendosi ad un sottoinsieme di peers (e.g. connessione TCP)
- Ogni peer, mentre fa il download, fa anche l'UPLOAD dei chunks che ha verso altri peers che glieli richiedono.
* I peers possono disconnettersi, una volta che hanno l'intero file possono abbandonare o, altruisticamente, restare connessi e condividere (conviene anche a loro!)

- **Quali chunks richiedere?**
Periodicamente, un peer chiede ai suoi vicini le liste dei loro chunks, e per scegliere quali richiedere segue la politica del RAREST FIRST, così si massimizza la probabilità che tutti i chunks siano disponibili!

- **A chi inviare i chunks?**
Si applica la politica TIT-FOR-TAT (do ut des): si inviano chunks ai 4 vicini che al momento stanno inviandoni chunks al PIU' ALTO RATE, si rivalutano ogni 10 secondi.
Inoltre c'è la politica detta' OPTIMISTICALLY UNCHOKE: ogni 30 secondi, si seleziona random un altro vicino e gli si inviano chunks, tale vicino potrebbe diventare uno dei 4!

* Questo meccanismo favorisce utenti con alta banda di upload; inoltre tende a far accoppiare peers con banda simile, evitando rallentamenti.

* Inoltre, incentiva gli utenti a trasmettere files e a non disconnettersi, fornendogli vantaggi nel download dei files, dato che, trasmettendo chunks, è molto più probabile riceverne.

- **SEARCHING FOR INFORMATION:**

  Come faccio a sapere chi ha quello che cerco?

- INDEX in P2P: mappa le informazioni con la locazione (IP addr + port #) dei peers.

- **INDEX CENTRALIZZATO:**

  Server "Napster"; sorta di client-server:

  1. Quando un peer si connette, comunica al directory server centralizzato il proprio indirizzo IP e i contenuti che mette a disposizione
  2. Il peer richiede un contenuto e riceve una lista di indirizzi IP di peers che lo posseggono
  3. Ci si connette ai peers, richiedendo chunks del contenuto

  PROBLEMI
  - Singolo punto di rottura
  - Performance basse (alto numero di richieste da gestire) e poca scalabilità
  - Violazione del COPYRIGHT (!); server possibile di causa e rintracciabile facilmente

- **QUERY FLOODING:**

  Sistema DISTRIBUITO; il Server centrale, detto "Nodo di BOOTSTRAP", mantiene solo gli indirizzi IP dei peers, non il contenuto → No copyright!

- Ogni peer indicizza i files che lui mette a disposizione.
- Un peer che si unisce, riceve dal Nodo di Bootstrap solo una lista di indirizzi IP di peers.
- Se si cerca qualcosa, si fa QUERY FLOODING: si inviano richieste ai peers vicini, i quali a loro volta me fanno il forwarding ai propri vicini, e così via. Gli utenti creano una RETE DI OVERLAY, tramite connessioni logiche (e.g. TCP). Ovviamente non ci si connette a tutti i nodi, ma solo ad alcuni; tipicamente, dalla Teoria dei Grafi, $\log_2(N)$.

  I nodi che hanno l'oggetto rispondono con una Query Hit sul reverse path.

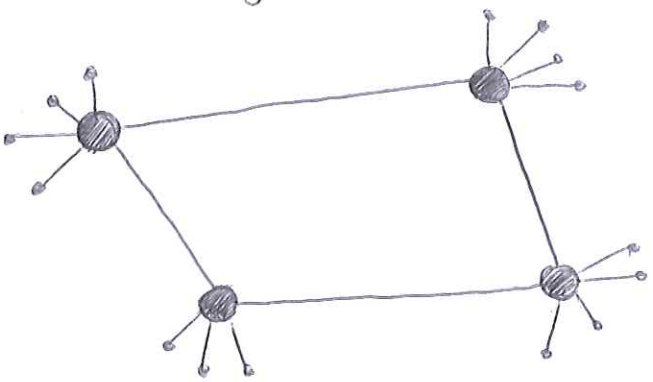
* Inattaccabile dal punto di vista legale; il server fornisce solo un servizio di localizzazione tramite indirizzo IP!

PROBLEMI
- Poco scalabile: flooding a scope limitato, altrimenti il numero delle query esploderebbe
- Tempo di ricerca molto elevato!

- **HIERARCHICAL OVERLAY:**

Soluzione a metà strada tra l'index centralizzato e l'approccio con query flooding.

- Ci sono peer particolari, detti SUPER NODE
- Tutti gli altri peers sono connessi ad 1 solo super node e, quando vi si connettono, gli comunicano il proprio file di index.
- I super nodes conoscono tutti i contenuti dei suoi peer figli
- I super nodes sono collegati tra loro in una normale rete P2P con query flooding



- L'architettura è DISTRIBUITA, ma i super peers agiscono come il server "Napster"
* Tuttavia, non è attaccabile legalmente perché i super peers cambiano di continuo!
- Il numero di richieste, soprattutto per il flooding, da inviare è molto minore, perché vanno inviate solo tra i super nodes
- Ricerca molto efficiente e ALTA SCALABILITÀ
- C'è sempre il nodo di bootstrap che fa partire l'app, mantenendo la lista dei super-peers!

- E se il nodo di bootstrap va giù?

Assumendo che l'indirizzo IP del nodo di bootstrap sia noto, i super-peer mandano periodicamente dei messaggi di ALIVE al nodo di bootstrap, così che possa ricostruire lo stato!

* I ROUTER NAT "mascherano" gli indirizzi IP degli host! Se entrambi i peers sono dietro una NAT, si usa da tramite un NODO DI RELAY per comunicare.

- Esempio: Skype, BitTorrent, GNutella, ...