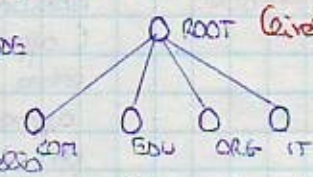
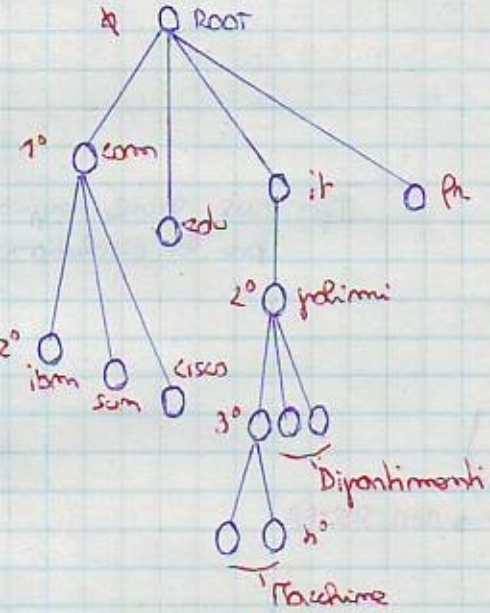
I domini di 1° livello sono facilmente riconoscibili:

- .COM per siti privati aziendali
- .EDU per le università americane
- .GOV per le unità governative
- .NET per enti di controllo di Internet
- .ORG per organizzazioni varie.
- .COUNTRY CODE (IT, FR...)

La gerarchia dei nomi è sia geografica che organizzativa.



I DNS sono disposti in livelli gerarchici e quelli più importanti sono quelli a livello di root. Sono molto sorvegliati e rappresentano la preda degli hacker. Quindi:



Si noti che i livelli possibili sono 127. Un nome è al massimo lungo 255 caratteri e ogni label ha un massimo di 63 caratteri.
Spesso le pagine dinamiche sfruttano questi limiti.

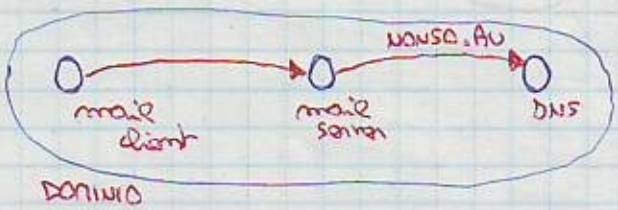
Il nome di dominio può rappresentare un gruppo di macchine, una macchina specifica o un particolare servizio. Quando si fa la richiesta di risoluzione, essa deve contenere in modo implicito o esplicito l'indicazione del tipo di risposta che viene richiesta. Per esempio:

? . ELET. POLIMI. IT → che servizio usa?

Consideriamo per esempio il seguente indirizzo di posta elettronica:

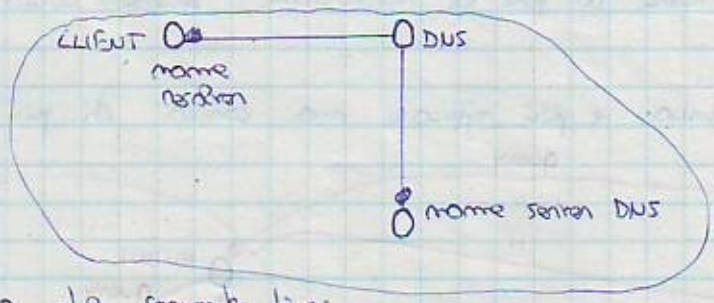
GIOVANNI@NONSO.AU

Si ha:

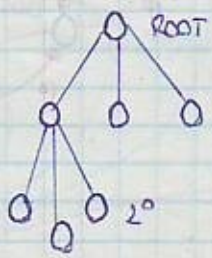


Il DNS fa la sua ricerca basandosi magari su altri DNS.

Il sistema DNS è di tipo CLIENT - SERVER. I DNS sono sparsi per il mondo in maniera gerarchica. La cosa importante è che ci deve essere un DNS in ogni dominio.



In una struttura del seguente tipo:



Il server ROOT conosce i domini di livello 1 che a loro volta conoscono quelli di livello 2 e così via. In realtà non è così perché così facendo l'albero dei server sarebbe stretto e lungo, ma in realtà esso è più largo. I DNS del primo livello conoscono quelli del secondo e magari anche del terzo livello.

Poi non è detto che ogni dominio possieda un server. Si può usare un server del dominio di livello superiore. Si consideri ora la seguente situazione:

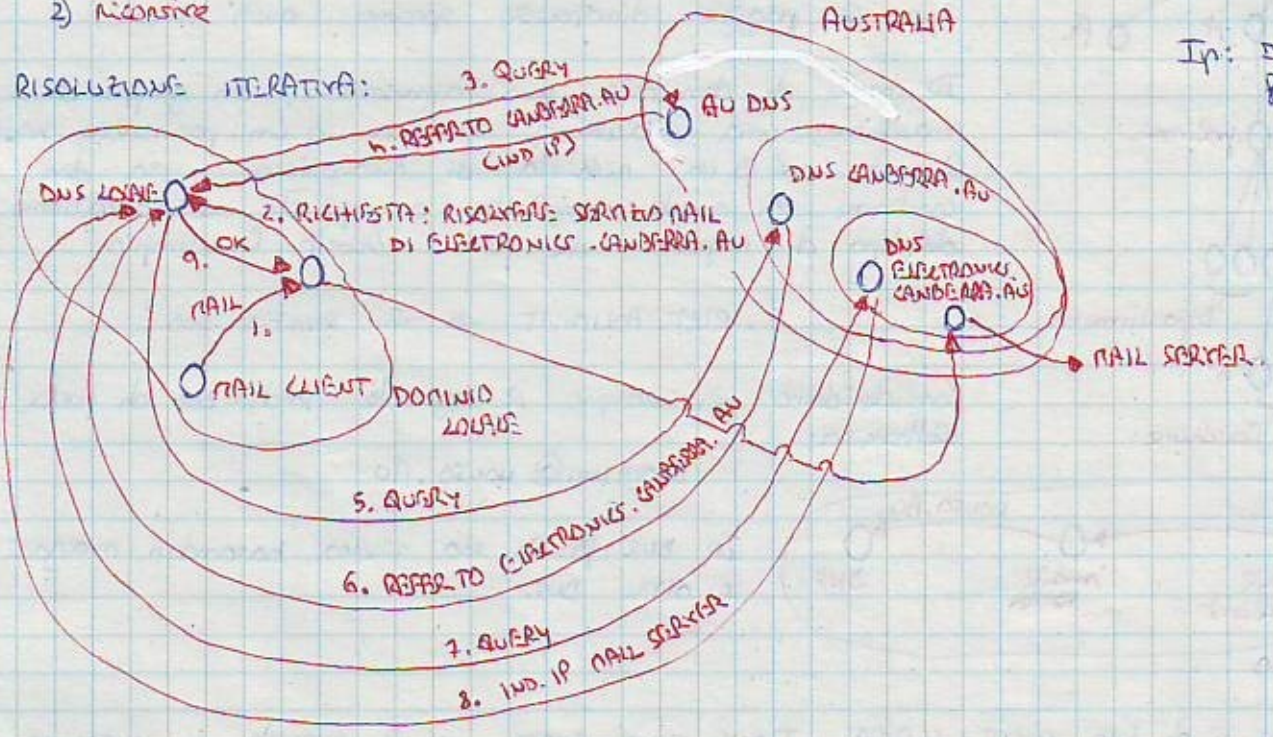
Abbiamo detto che ogni dominio ha un DNS. (approccio decentralizzato). In realtà è può essere un secondo approccio denominato approccio centralizzato in cui un unico DNS di livello superiore gestisce entrambi i domini.

L'approccio centralizzato è poco scalabile ma meno costoso. Vediamo ora le risoluzioni. Esse possono essere:

1) iterative

2) ricorsive

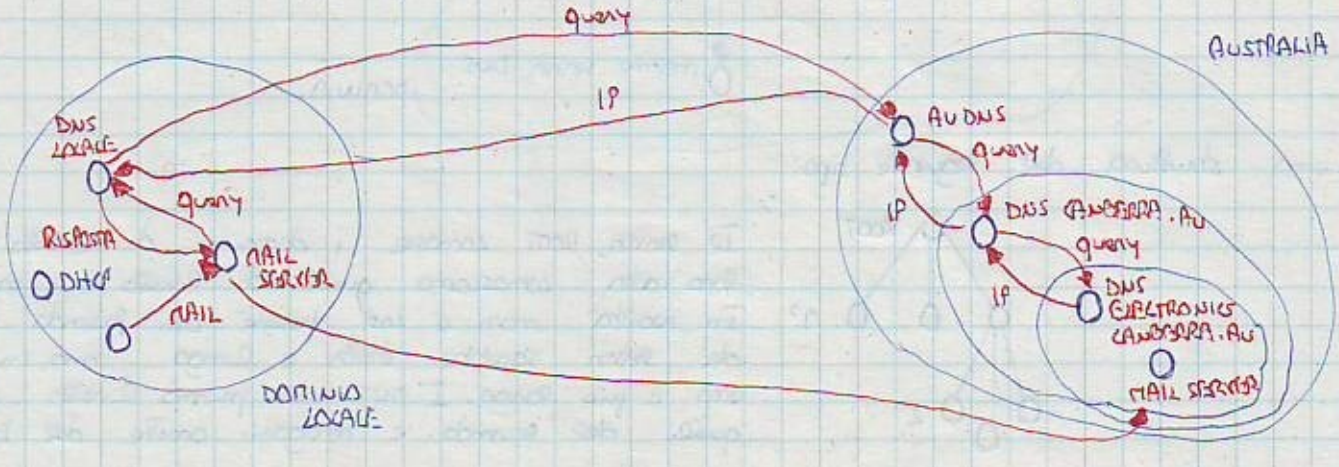
RISOLUZIONE ITERATIVA:



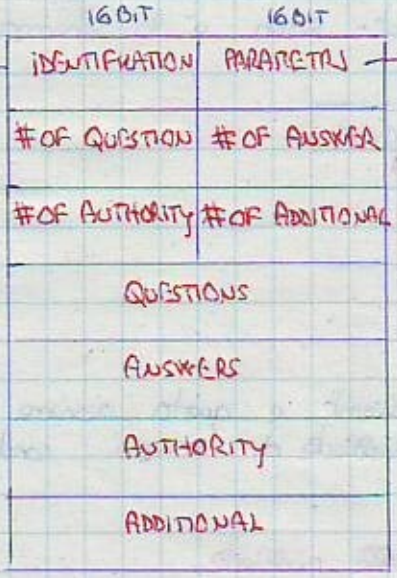
Ip: DNS locale sappia fare la risoluzione.

Chiaramente se un DNS non riesce a fare la risoluzione, chiede aiuto ai DNS di livello superiore.

RISOLUZIONE RICORSIVA: è più furba e più efficace ma carica di più i server.



Si ricordi che il server DHCP ci fornisce i vari indirizzi dei servizi tra cui il DNS e ci fornisce anche il DEFAULT GATEWAY. I problemi del DNS sono legati al carico. Il carico aumenta all'aumentare del livello gerarchico. Per diminuire questo carico si può fare del **caching** delle risoluzioni già fatte cioè si possono memorizzare le stesse per un po' di tempo. Si noti che il router può ripetere le risoluzioni precedenti dalla cache. Si noti inoltre che il client può anche richiedere all'intera cache del DNS, in modo da agire autonomamente. Vediamo la struttura di un messaggio DNS:

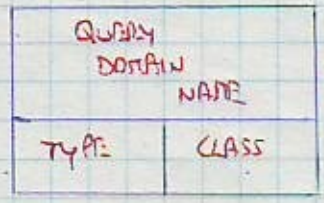


Per corrispondere domande e risposte.

→ ci sono tanti flags:
- REQUEST / REPLY
- RIS. ITERATIVA / RICORSIVA
- SEGNALAZIONI DI ERRORI
- RIS STANDARD (nome → IP)
  " INVERSA (IP → nome)

sezioni di 32 bit

Vediamo anche la struttura di una query:



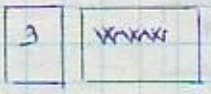
Nella query domain name c'è un nome simbolico codificato nel seguente modo:

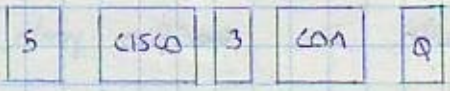
↳ lunghezza label seguente.

↳ STOP.

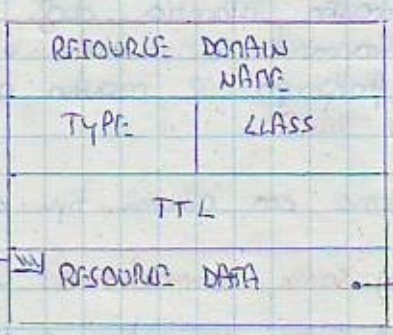
Per esempio:

www.cisco.com →





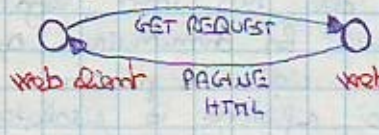
Vediamo infine la ANSWER RECORDS:



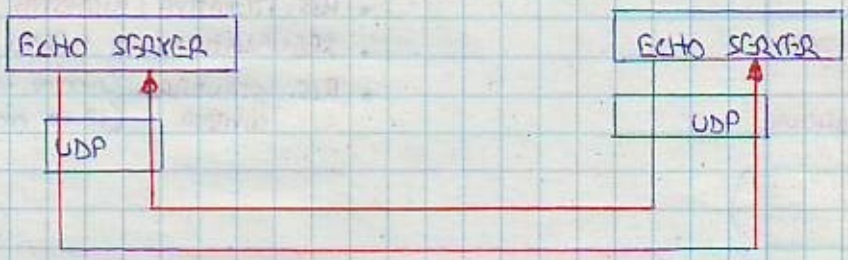
→ qui dentro per esempio c'è l'indirizzo IP.

Settando TYPE e CLASS posso chiedere la cpu della macchina, quale sistema operativo gira e così via (interrogazioni particolari). Il DNS gira su TCP sulla porta 53.

Parliamo ora di applicazioni client-server. Consideriamo:



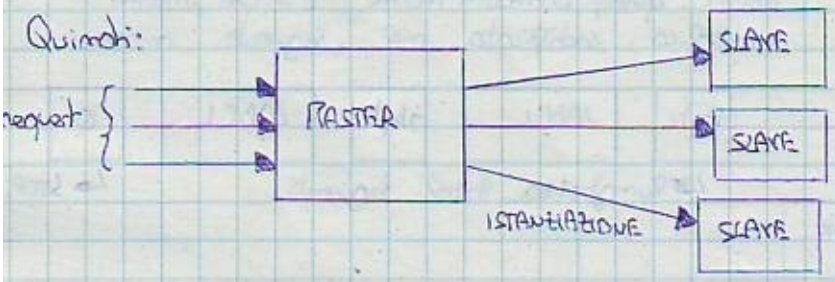
Di solito lo schema che analizzeremo sarà quello, ma può accadere che una macchina funzioni sia da client che da server. Tipicamente i client e i server comunicano tramite TCP e UDP. Il più semplice sistema client-server è lo schema UDP ECHO SERVER.



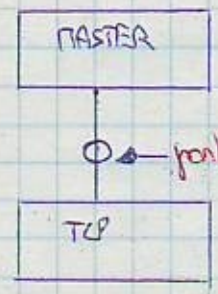
In genere i server sono più complicati dei client e questo avviene perché in genere un server può dover gestire molteplici richieste di servizio contemporaneamente. Le richieste possono essere fatte in due modi:

1) seriale mente attraverso l'uso di una coda delle richieste;
2) parallelamente dove le richieste vengono suddivise da un'unità master che riceve le richieste dal mondo esterno;

Quindi:



Il master sta in ascolto sulla sua specifica porta, cioè ha eseguito una PASSIVE OPEN.



Il master istanzia degli slave i quali poi gestiscono effettivamente le richieste e comunicano con il master nel frattempo il master attende sulla sua porta.

Vediamo ora alcuni tipi di server:

1) server connectionless → funzionano su UDP.
2) server connection oriented → funzionano su TCP.
3) sequenziale
4) parallelo

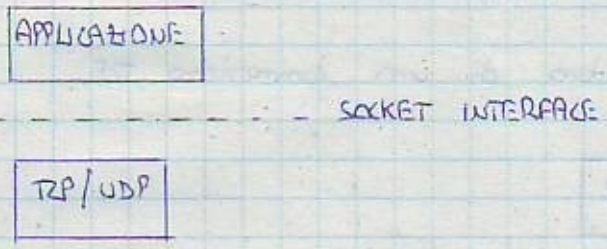
Quindi:

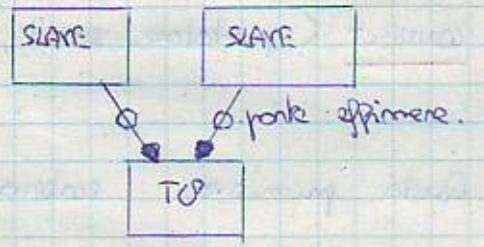
	SEQUENZIALI	PARALLELI
Server connectionless	X	
Server connection-oriented		X

I server sequenziali usano sempre la stessa porta sia per effettuare la richiesta di servizio sia per elaborare i dati. I server paralleli invece non usano la stessa porta.

Chiamiamo porta effimera una porta assegnata dinamicamente tra slave e TCP.

Consideriamo ora la seguente situazione:





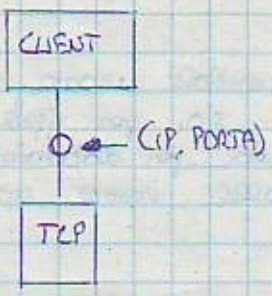
Quando arriva un segmento di byte che gestisce la richiesta ci si appoggia alla demultiplazione del TCP. Si noti però che la demultiplazione è a carico del TCP e non dell'applicazione. Abbiamo già visto cosa è la socket interface. Esiste uno standard di base quando si lavora con la socket interface, ma ci sono significative differenze tra i diversi sistemi operativi. La socket interface ha delle chiamate di base verso il sistema operativo, ma la stessa può arricchirsi anche di funzioni di libreria speciali. Queste ultime fornisono del valore aggiunto. Le chiamate di base ricalcano la struttura delle chiamate UNIX del tipo OPEN, READ, WRITE, CLOSE (per quanto riguarda i file, i device, e i socket). Un socket è una struttura con più parametri per cui le chiamate sono più specifiche. Per esempio:

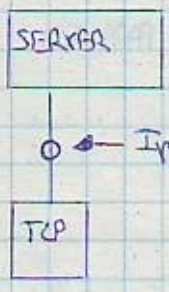
- SOCKET è una primitiva che apre un socket con i seguenti parametri:
  - PF: Protocol Family che altro non è che un codice per identificare una famiglia di protocolli (TCP/IP).
  - Type → SOCK_STREAM (TCP)
    → SOCK_DATAGRAM (UDP)
    → SOCK_RAW (IP o su interfacce di rete)
    ⋮
  - Protocol che mi dice il tipo di protocollo usato

  NB: la primitiva socket ci restituisce un puntatore al socket.

  Questa primitiva viene usata in fase preliminare.
- CLOSE (puntatore_socket). Chiude il socket.
- BIND (puntatore_socket, Local Address, Address Length). Questa primitiva lega il socket alla porta locale. Il Local Address consiste nell'indirizzo IP più la porta locale.

Quindi:

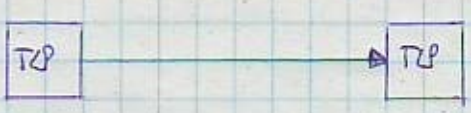




Dobbiamo ora creare la connessione logica. Per fare ciò usiamo la seguente primitiva:

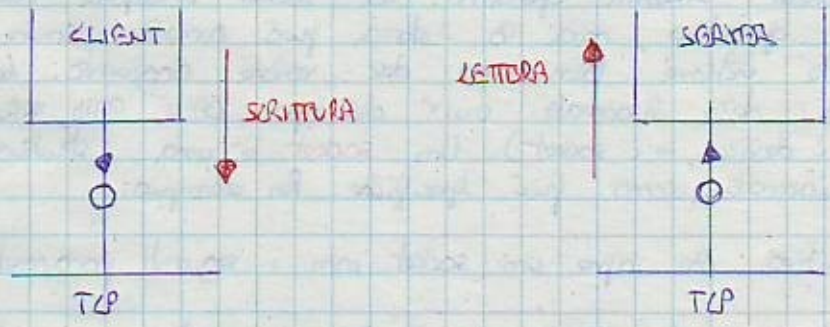
- CONNECT (puntatore_socket, D.A, Address length)
  IP + PORTA

Questa primitiva scatena l'apertura di una connessione TCP.



Se la connect ci ritorna un risultato positivo allora posso inviare i dati come per esempio la richiesta.

Analizziamo ora la fase dati:



Usiamo la primitiva SEND (puntatore_socket, buffer, length, flags) per scrivere sul socket mentre usiamo la primitiva READ (puntatore_socket, buffer, length) per prelevare dal buffer di ricezione del TCP un certo numero massimo di byte da leggere. Si noti che la lettura avviene grazie all'applicazione che preleva i byte dal TCP. È possibile che l'applicazione possa leggere 4 byte. Infine citiamo due primitive che permettono di configurare un socket che sono:

OPTION = GET SOCK PORT (socket)
SET SOCK PORT (socket, options)

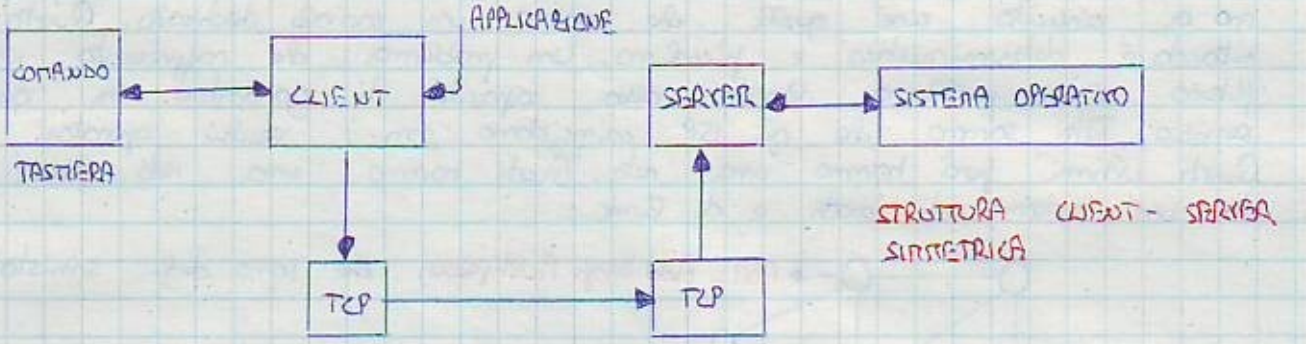
Per quanto riguarda il server, esso deve poter accettare le richieste che gli giungono. Quindi esso rimane fermo in attesa e questo viene specificato attraverso la primitiva:

NEW SOCKET = ACCEPT (puntatore_socket, Address, Address length);

Vediamo infine alcune funzioni di libreria più elaborate:

- Libreria per la costruzione di query DNS. Per esempio TELNET è un'applicazione di terminale virtuale remoto. Il client TELNET si appoggia su TCP e va a creare sul server TELNET una connessione.

Graficamente:



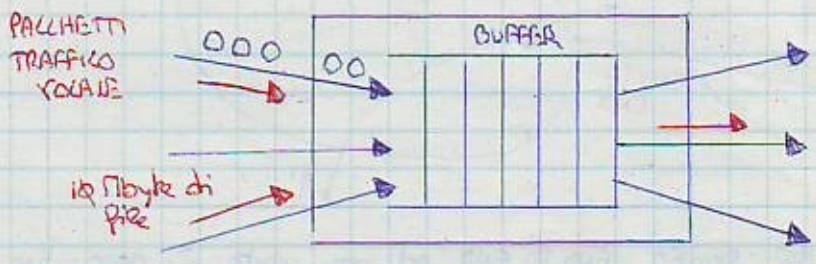
Consideriamo questa parte sull'architettura CLIENT-SERVER parlando del protocollo FTP. FTP = File Transfer Protocol si appoggia sul TCP.

L'FTP è un protocollo di tipo simmetrico cioè l'FTP può essere usato da parte nostra, o da parte di altre applicazioni. Il server FTP è di tipo connection oriented ed è di tipo parallelo.

Il master crea un set di slave, uno per ogni richiesta, e poi le connessioni passano allo slave che mantiene aperte due connessioni:

1) una connessione per la gestione dello stesso e per il controllo.
2) una connessione per i dati.

Torniamo ora a parlare di Internet a livello generico. L'unico servizio che "gira" su Internet è, come abbiamo già detto, il servizio Best Effort. Con il servizio Best Effort si riesce a fare bene la comunicazione dei dati, FTP... ma le applicazioni real-time o applicazioni con flussi di traffico che comportano ritardi non vanno bene con tale servizio. Consideriamo per esempio il traffico vocale che ha un bit rate di 32 kbit/s.



I pacchetti vengono accodati nel buffer (FIFO).

Cosa succede?