DNS: Domain Name System

Persone: identificatori:
- CF, nome, Numero di Passaporto

Host e router Internet:
- Indirizzo IP (32 bit) - usato per instradare i pacchetti
- "nome", per es., massimotto.diiie.unisa.it - usati dagli esseri umani

D: corrispondenza tra indirizzo IP e nome ?

Domain Name System:
- database distribuito realizzato con una gerarchia di molti name server
- protocollo application-layer host, router, name server comunicano per risolvere i nomi (traduzione indirizzo/ nome)
  - nota: una funzione "core" di Internet realizzata come protocollo application-layer

I name server DNS

Perchè non centralizzare il DNS?
- Singolo punto critico
- volume di traffico
- Database centralizzato lontano
- manutenzione

In una parola non è possibile scalare!

- nessun server ha tutte le corrispondenze nome-indirizzo IP

name server locali:
- ciascun ISP o ente ha un local (default) name server
- Le query DNS degli host vanno prima al local name server

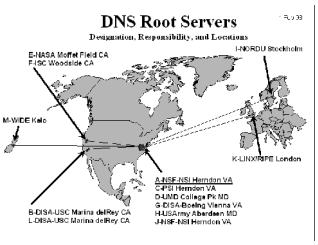
name server authoritative:
- Per un host: memorizza l'indirizzo IP e il nome di quell'host
- Può effettuare la traduzione nome/ indirizzo di quel nome di host

DNS: Root name server

Vengono contattati dal local name server che non sa risolvere un nome

root name server:
- Contatta un authoritative name server se il name mapping è sconosciuto
- Ottiene il mapping
- Invia il mapping al local name server

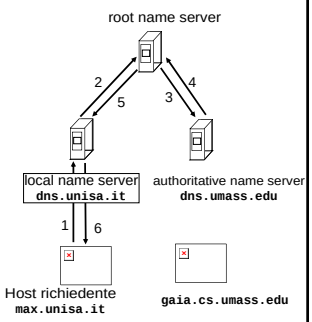
~ una dozzina di root name server a livello mondiale



Esempio di DNS

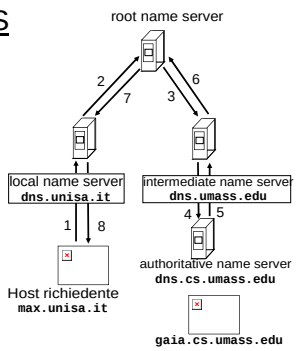
L'host **max.unisa.it** vuole l'indirizzo IP di **gaia.cs.umass.edu**

1. Contatta il suo DNS locale, **dns.unisa.it**
2. **dns.unisa.it** contatta il root name server, se necessario
3. Il root name server contatta l'authoritative name server, **dns.umass.edu,** se necessario



Esempio di DNS

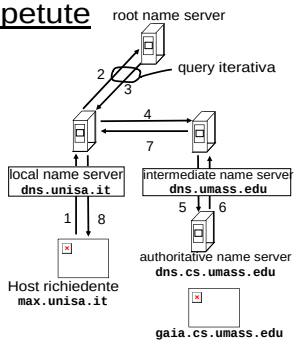
Il Root name server:
- Può non conoscere l' authoritative name server
- Potrebbe conoscere un intermediate name server: chi contattare per trovare l' authoritative name server



DNS: queries ripetute

Query ricorsiva:
- Molla il problema della risoluzione del nome al name server contattato
- sovraccarico?

Query iterativa:
- Il server contattato risponde con il nome del server da contattare
- "Non conosco questo nome, ma chiedi a questo server"



DNS: caching/aggiornam. records

- appena (un qualsiasi) nameserver apprende un mapping, lo memorizza (caches)
  - Le voci della cache "scadono" (scompaiono) dopo un certo tempo
- La IETF ha allo studio meccanismi di update/notify
  - RFC 2136
  - http://www.ietf.org/html.charters/dnsind-charter.html

DNS records

DNS: db distribuito che memorizza resource records (RR)

formato RR : **(name, value, type,ttl)**

Type=A
- **name** è l' hostname
- **value** è l'IP address

Type=NS
- **name** è il dominio (es., foo.com)
- **value** è l'IP address dell' authoritative name server per quel dominio

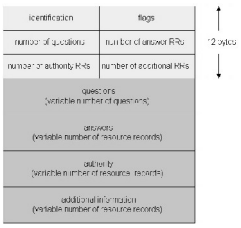
Type=CNAME
- **name** è un alias per un dato nome "canonico" (il vero nome)
- **value** è il nome canonico

Type=MX
- **value** è l' hostname del mail server associato al **nome**

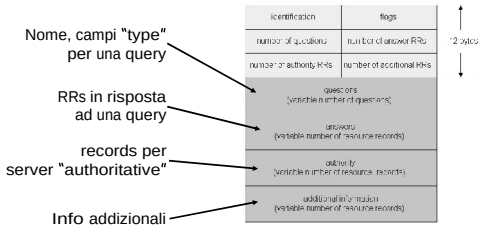
protocollo DNS, messaggi

protocollo DNS : messaggi di query/reply, con lo stesso formato

msg header
- identificazione: 16 bit # per le query, la reply a una query usa lo stesso #
- flags:
  - query o reply
  - recursion desired
  - recursion available
  - reply is authoritative



protocollo DNS, messaggi



Programmazione Socket

Obiettivo: imparare come costruire applicazioni client/server comunicanti tramite socket

Socket API
- Introdotte nel BSD4.1 UNIX, 1981
- create, usate, rilasciate esplicitamente dalle applicazioni
- paradigma client/server
- due tipi di servizi di trasporto disponibili con la socket API:
  - unreliable datagram
  - reliable, byte stream-oriented

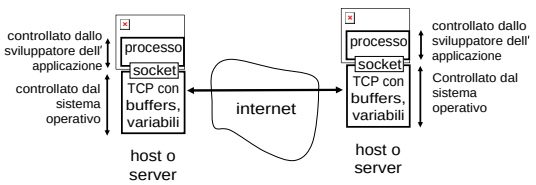
socket

Un interfaccia (una "porta") locale all'host, creata e posseduta dall'applicazione, controllata dal S.O. tramite la quale il processo applicativo può sia inviare che ricevere messaggi da/per un altro processo (remoto o locale)

Programmazione Socket con il TCP

Socket: una porta tra il processo applicativo e il protocollo di trasporto end-to-end (UDP o TCP)

servizio TCP : trasferimento affidabile di bytes da un processo all'altro



Programmazione Socket con il TCP

Il client contatta il server
- Il processo server deve prima essere attivato
- Il server deve aver creato una socket (porta) che accetta il contatto del client

Il client contatta il server:
- Creando una socket TCP locale al client
- Specificando un indirizzo IP, un numero di porta del processo server

Quando il client crea la socket: il client TCP stabilisce una connessione con il server TCP

Quando viene contattato dal client, il server TCP crea una nuova socket per consentire la comunicazione processo server - processo client
- Il server può parlare con più di un client

application viewpoint

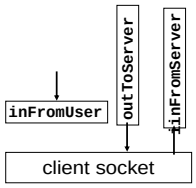
Il TCP fornisce un trasferimento affidabile e ordinato di bytes tra client e server

Programmazione Socket con il TCP

Esempio di appl. client-server:
- Il client legge una linea dallo standard input (**`inFromUser`** stream), la invia al server via socket (**`outToServer`** stream)
- Il server legge una linea dal socket
- Il server converte la linea in maiscole, la re-invia al client
- Il client legge dalla socket (**`inFromServer`** stream) e stampa la linea modificata

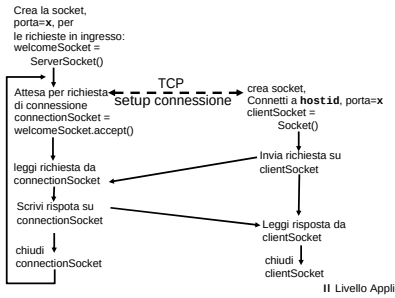
Input stream: sequenza di bytes nel processo

Output stream: sequenza di bytes dal processo



Interazione socket client/server: TCP

Server (in esecuzione su **`hostid`**) Client



Esempio: Java client (TCP)

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

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

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

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

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

Crea input stream

Crea client socket, connetti al server

Crea output stream connesso a socket

Esempio: Java client (TCP), cont.

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

      sentence = inFromUser.readLine();

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

      modifiedSentence = inFromServer.readLine();

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

      clientSocket.close();

   }
}
```

Crea input stream connesso a socket

invia linea al server

Leggi linea dal server

Esempio: Java server (TCP)

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

class TCPServer {

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

     ServerSocket welcomeSocket = new ServerSocket(6789);

     while(true) {

       Socket connectionSocket = welcomeSocket.accept();

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

Crea socket di attesa sulla porta 6789

attendi, sulla socket di attesa il contatto dal client

Crea input stream, connesso alla socket

Example: Java server (TCP), cont

Crea output stream, connesso alla socket → `DataOutputStream outToClient = new DataOutputStream(connectionSocket.getOutputStream());`

leggi linea dalla socket → `clientSentence = inFromClient.readLine();`

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

Scrivi linea sulla socket → `outToClient.writeBytes(capitalizedSentence);`

```
        }
    }
}
```

Fine del while loop, cicla indietro e aspetta per un altra richiesta di connessione da client

Programmazione Socket con l'UDP

UDP: nessuna "connessione" tra client e server
- nessun handshaking
- Il mittente aggiunge esplicitamente un indirizzo IP e una porta di destinazione
- Il server deve estrarre l'indirizzo IP address e la porta del mittente dal pacchetto ricevuto

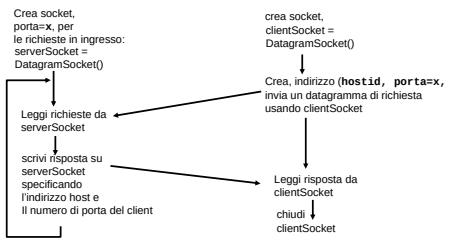
UDP: I dati trasmessi possono essere persi o ricevuti fuori ordine

application viewpoint

UDP fornisce un trasferimento inaffidabile di gruppi di bytes ("datagrammi") tra client e server

Interazione socket client/server: UDP

Server (in esecuzione su **`hostid`**) Client



Esempio: Java client (UDP)

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

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

Crea input stream → `BufferedReader inFromUser = new BufferedReader(new InputStreamReader(System.in));`

Crea client socket → `DatagramSocket clientSocket = new DatagramSocket();`

Traduci da hostname a IP usando il DNS → `InetAddress IPAddress = InetAddress.getByName("hostname");`

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

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


Example: Java client (UDP), cont.

Crea datagramma con lunghezza dati, IP, porta → `DatagramPacket sendPacket = new DatagramPacket(sendData, sendData.length, IPAddress, 9876);`

Invia il datagramma al server → `clientSocket.send(sendPacket);`

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

leggi il datagramma dal server → `clientSocket.receive(receivePacket);`

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

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


Esempio: Java server (UDP)

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

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

Crea datagram socket alla porta 9876 → `DatagramSocket serverSocket = new DatagramSocket(9876);`

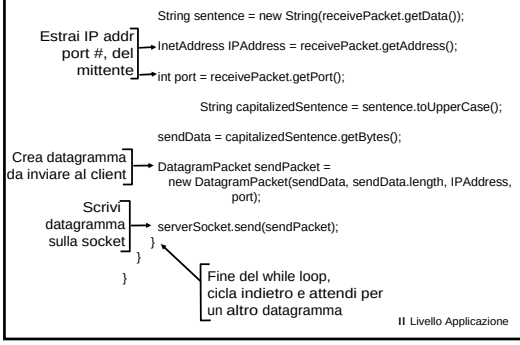
```
byte[] receiveData = new byte[1024];
byte[] sendData  = new byte[1024];

while(true)
 {
```

Crea spazio per il datagramma → `DatagramPacket receivePacket = new DatagramPacket(receiveData, receiveData.length);`

Ricevi il datagramma → `serverSocket.receive(receivePacket);`

Esempio: Java server (UDP), cont



II Livello Applicazione: Riepilogo

Abbiamo studiato un certo numero di applicazioni di rete!

- Requisiti del servizio di applicazione:
  - affidabilità, larghezza di banda, ritardo
- paradigma client-server
- Modello di servizio di trasporto Internet
  - connection-oriented, affidabile: TCP
  - inaffidabile, datagrammi: UDP

- protocolli specifici:
  - http
  - ftp
  - smtp, pop3
  - dns
- Programmazione socket
  - Implementazione client/server
  - Usando le socket tcp, udp

II Livello Applicazione: Riepilogo

Ma il nostro obiettivo principale è stato di: studiare le caratteristiche dei protocolli

- Tipico scambio di messaggi request/reply:
  - il client chiede info o servizi
  - il server risponde con dati, status code
- formati del messaggio:
  - header: campi che danno info sui dati
  - dati: info comunicate

- Messaggi dati o controllo
  - in-based, out-of-band
- Centraliz./decentralizzati
- Con o senza stato
- Trasferimento affidabile/ inaffidabile
- "complexity at network edge"
- sicurezza: autenticazione