DNS: Domain Name System

People: many identifiers:
- SSN, name, Passport #

Internet hosts, routers:
- Always: IP address (32 bit) - used for addressing datagrams
- Often: "name", e.g., nif-c14.wsu.edu - used by humans

Q: How do we map between names and IP addresses?

A: Domain Name System (DNS):
- distributed database implemented in hierarchy of many name servers
- application-layer protocol hosts, name servers communicate to resolve names (name/address translation)
- note: this is a core Internet function implemented as application-layer protocol
- complexity at network's "edge"

Q: How do we map between names and IP addresses?

A: Domain Name System (DNS):
- distributed database implemented in hierarchy of many name servers
- application-layer protocol hosts, name servers communicate to resolve names (name/address translation)
- note: this is a core Internet function implemented as application-layer protocol
- complexity at network's "edge"

DNS: Domain Names

- Names are hierarchical
  - www.eecs.wsu.edu.

- Each suffix corresponds to a zone
  - .
  - edu.
  - wsu.edu.
  - eecs.wsu.edu.

- Each zone has at least two authoritative servers

DNS name servers

- no server has all name-to-IP address mappings

- local name servers:
  - each ISP, company has local (default) name server
  - host DNS query first goes to local name server

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

Why not centralize DNS?
- single point of failure
- traffic volume
- distant centralized database
- maintenance

- doesn't scale!

DNS: Root name servers

- contacted by local name server that can not resolve name

- root name server:
  - contacts authoritative name server if name mapping not known
  - gets mapping
  - returns mapping to local name server

- ~ dozen root name servers worldwide
**Simple DNS example**

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

1. Contacts its local DNS server, dns.eurecom.fr
2. dns.eurecom.fr contacts root name server, if necessary
3. root name server contacts authoritative name server, dns.umass.edu, if necessary

**DNS example**

Root name server:
- may not know authoritative name server
- may know intermediate name server: who to contact to find authoritative name server

**DNS: iterated queries**

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

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

**DNS: caching and updating records**

once (any) name server learns mapping, it *caches* mapping
- cache entries timeout (disappear) after some time

update/notify mechanisms under design by IETF
- RFC 2136
**DNS records**

- **DNS:** distributed db storing resource records (RR)
  
  **RR format:** (name, value, type, ttl)

- **Type=A**
  - name is hostname
  - value is IP address

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

- **Type=CNAME**
  - name is an alias name for some "cannonical" (the real) name
  - value is cannonical name

- **Type=MX**
  - value is hostname of mailserver associated with name

**DNS protocol, messages**

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

<table>
<thead>
<tr>
<th>Identification</th>
<th>Flags</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>number of questions</td>
</tr>
<tr>
<td></td>
<td>number of authority RRs</td>
</tr>
<tr>
<td>questions</td>
<td>variable number of questions</td>
</tr>
<tr>
<td>answers</td>
<td>variable number of resource records</td>
</tr>
<tr>
<td>authority</td>
<td>variable number of resource records</td>
</tr>
<tr>
<td>additional info</td>
<td>variable number of resource records</td>
</tr>
</tbody>
</table>

**msg header**

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

**Example: Java client (TCP)**

```java
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());
```

Example: Java client (TCP), cont.

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

String sentence = inFromUser.readLine();
outToServer.writeBytes(sentence + "\n");
modifiedSentence = inFromServer.readLine();
System.out.println("FROM SERVER: " + modifiedSentence);
clientSocket.close();
```

Example: Java server (TCP)

```java
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()));

            clientSentence = inFromClient.readLine();
            capitalizedSentence = clientSentence.toUpperCase() + "\n";
            outToClient.writeBytes(capitalizedSentence);
        }
    }
}
```

Example: Java client (UDP)

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

class UDPClient {
    public static void main(String args[]) throws Exception {
        BufferedReader inFromUser =
            new BufferedReader(new InputStreamReader(System.in));

        DatagramSocket clientSocket = new DatagramSocket();
        InetAddress IPAddress = InetAddress.getByName("hostname");

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

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

        // End of while loop, loop back and wait for another client connection
    }
}
```
Example: Java client (UDP), cont.

```java
DatagramPacket sendPacket = new DatagramPacket(sendData, sendData.length, IPAddress, 9876);
clientSocket.send(sendPacket);
DatagramPacket receivePacket = new DatagramPacket(receiveData, receiveData.length);
clientSocket.receive(receivePacket);
String modifiedSentence = new String(receivePacket.getData());
System.out.println("FROM SERVER:" + modifiedSentence);
clientSocket.close();
```
Chapter 2: Summary

Most importantly: learned about protocols

- typical request/reply message exchange:
  - client requests info or service
  - server responds with data, status code
- message formats:
  - headers: fields giving info about data
  - data: info being communicated
- control vs. data msgs
  - in-based, out-of-band
- centralized vs. decentralized
- stateless vs. stateful
- reliable vs. unreliable msg transfer
- “complexity at network edge”
- security: authentication