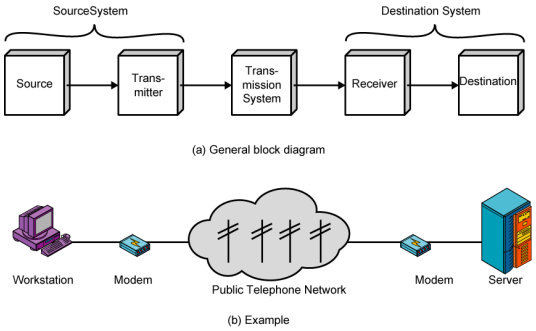
Data Communications *v.* Networking

- Communication is concerned with the transmission of data over a communication medium/channel between two entities. Here we are more concerned about EE issues such as physical properties of communication medium, physical characteristics of signals and interfaces, format, and timing of signals, etc.…

- Networking is concerned with the physical topology of two or more communicating entities and the logical topology of data transmission. Issues such as addressing, routing, reliability, etc become important.

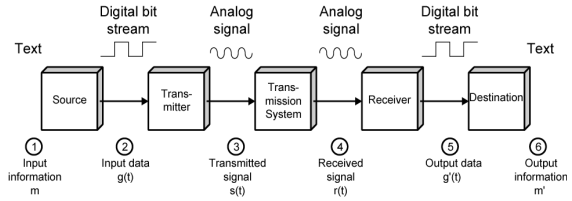
Simplified Communication Model



Major Communication tasks

- Transmission system utilization (DC+Net)
- Addressing (DC+Net)
- Interfacing (DC)
- Routing (Net)
- Signal generation (DC)
- Recovery (DC+Net)
- Synchronization (DC+Net)
- Message formatting (Net)
- Security (Net)
- Error detection and correction (DC+Net)
- Congestion control (Net)
- Flow control (DC+Net)

Simplified Communications Model



Layered Model

- Systems communicate over a shared communication medium according to an agreed upon convention (standard).
- Several sets of standards currently exist:
  - DoD: TCP/IP
  - ISO: OSI model
  - Commercial: SNA, IPX (Novell)
  - Proprietary
- In this module, we will basically follow the 7 layer approach defined by ISO:OSI.

DoD Model

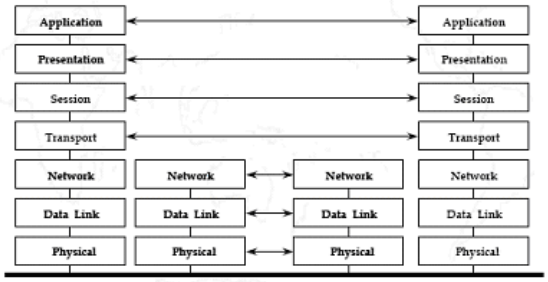
- DARPA (Defence Advanced Research Projects Agency)
- ARPANET => Internet
- TCP/IP Transmission Control Protocol/Internet Protocol
- TCP/IP developed concurrently with ISO model. TCP/IP does not contain protocols relating to all ISO layers. Most of the functionalities of ISO are embedded in TCP/IP.

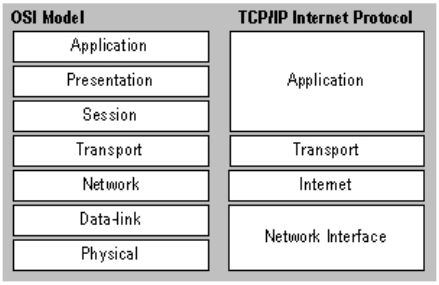
ISO/OSI Model

- Communication functions are partitioned into a vertical set of seven layers.
- each layer performs a related subset of functions required for communication.
- each layer provides services to next higher layer while depending on the previous lower layer to do more primitive functions.
- decomposes one problem into a number of more manageable sub-problems.
- communication is achieved by having corresponding (peer) entities in the same layer in two different systems communicate via a protocol.
- each protocol entity sends data down to the next lower layer so as to get data across to its peer entity.
- each entity communicates with entities in the layers above it and below it, across an interface.

ISO/OSI provides a common basis for coordination of standards and is based on a hierarchical model:

- Application Layer
- Presentation Layer
- Session Layer
- Transport Layer
- Network Layer
- Data Link Layer
- Physical Layer





TCP/IP vs. ISO-OSI

Application Layer

Goals:

- conceptual, implementation aspects of network application protocols
  - transport-layer service models
  - client-server paradigm
  - peer-to-peer paradigm
- learn about protocols by examining popular application-level protocols
  - HTTP
  - FTP
  - SMTP / POP3 / IMAP4
  - DNS

Application: communicating, distributed processes
- e.g. Email, Web, P2P file sharing, IM
- running in end systems (hosts)
- exchange messages to implement application

Application-layer protocols:
- one "piece" of an application
- define messages exchanged by apps and actions taken
- use communication services provided by lower layer protocols (TCP, UDP)

Application-layer protocols defines:

- Types of messages exchanged, e.g. request or response messages
- Syntax of message types: what fields in messages & how fields are delineated
- Semantics of the fields, i.e. meaning of information in fields
- Rules for when and how processes send & respond to messages

Public-domain protocols:
- defined in RFCs
- allows for interoperability
- eg, HTTP, SMTP

Proprietary protocols:
- eg, Skype, Viber, etc…

Client-Server Paradigm

Client:
- initiates contact with server ("speaks first")
- typically requests service from server,
- Web client implemented in browser; email client implemented in mail reader

Server:
- provides requested service to client e.g., Web server sends requested Web page, mail server delivers e-mail

Which type of service does an application need?

Data Loss and Timing
- some apps (e.g., audio) can tolerate some loss
- other apps (e.g., file transfer, telnet) require 100% reliable data transfer
- some apps (e.g., Internet telephony, interactive games) require low delay to be "effective"

Bandwidth
- some apps (e.g., multimedia) require minimum amount of bandwidth to be "effective"
- other apps ("elastic apps") make use of whatever bandwidth they get

Application	Data loss	Bandwidth	Time Sensitive
file transfer	no loss	elastic	no
e-mail	no loss	elastic	no
Web documents	no loss	elastic	no
real-time audio/video	loss-tolerant	audio: 5kbps-1Mbps video:10kbps-5Mbps	yes, 100's msec
stored audio/video	loss-tolerant	same as above	yes, few secs
interactive games	loss-tolerant	few kbps up	yes, 100's msec
instant messaging	no loss	elastic	yes and no

Transport protocol Services

TCP service:

- *connection-oriented:* setup required between client and server processes
- *reliable transport* between sending and receiving process
- *flow control:* sender won't overwhelm receiver
- *congestion control:* throttle sender when network overloaded
- *does not provide* timing or minimum bandwidth guarantees

UDP service:

- unreliable data transfer between sending and receiving process
- does not provide: connection setup, reliability, flow control, congestion control, timing, or bandwidth guarantee

Q: why bother? Why is there a UDP?

Application	Application layer protocol	Underlying transport protocol
e-mail	SMTP [*RFC 2821*]	TCP
remote terminal access	Telnet [*RFC 854*]	TCP
Web	HTTP [*RFC 2616*]	TCP
file transfer	FTP [*RFC 959*]	TCP
streaming multimedia	proprietary (e.g. RealNetworks)	TCP or UDP
Internet telephony	proprietary (e.g., Dialpad)	typically UDP

WEB and HTTP

- Web page consists of objects
- Object can be HTML file, JPEG image, Java applet, audio file,…
- Web page consists of base HTML-file which includes several referenced objects
- Each object is addressable by a URL
- Example URL:

http://pages.intnet.mu/rhh/index.html

HTTP overview

HTTP: HyperText Transfer Protocol

- Web's application layer protocol
- client/server model
  - *client:* browser that requests, receives, "displays" Web objects
  - *server:* Web server sends objects in response to requests

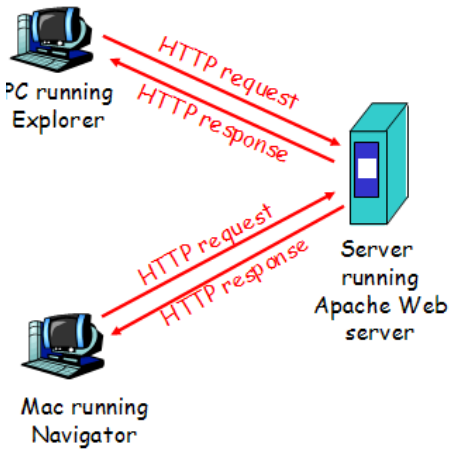
- HTTP 1.0: RFC 1945
- HTTP 1.1: RFC 2068

HTTP Uses TCP:

- client initiates TCP connection (creates socket) to server, *port 80*
- server accepts TCP connection from client
- HTTP messages (application-layer protocol messages) exchanged between browser (HTTP client) and Web server (HTTP server)
- TCP connection closed

HTTP is "stateless"

- i.e. server maintains no information about past client requests



HTTP Connections

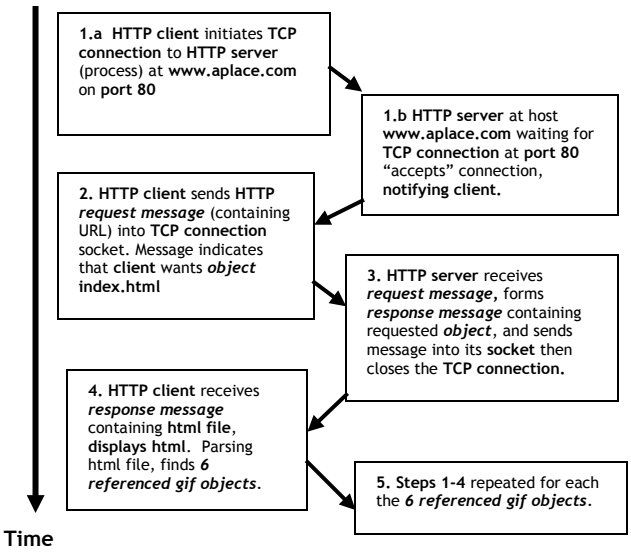
1. Non-persistent HTTP:
   - At most one object is sent over a single TCP connection.
   - HTTP/1.0 uses non-persistent HTTP.
2. Persistent HTTP:
   - Multiple objects can be sent over a single TCP connection between client and server.
   - HTTP/1.1 uses persistent connections in default mode.

Non-persistent HTTP

Suppose a user enters the following URL:

http://www.aplace.com/index.html and that this homepage contains some text and references to 6 gif images in total.

The following interactions will take place:



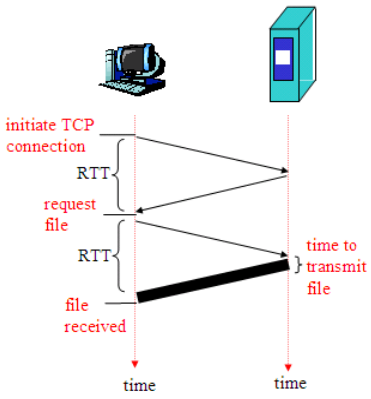
Response Time Modeling

Definition of Round-Trip Time (RTT) is the time to send a small packet to travel from client to server and back.

Response time:

- one RTT to initiate TCP connection
- one RTT for HTTP request and first few bytes of HTTP response to return
- file transmission time

Total = 2 x RTT+ transmit time



Non-persistent HTTP issues:

- requires 2 RTTs per object.
- OS must work and allocate host resources for each TCP connection.
- but browsers often open parallel TCP connections to fetch referenced objects.

Persistent HTTP issues

- Server leaves connection open after sending response.
- Subsequent HTTP messages between same client/server are sent over same connection.

Persistent HTTP without pipelining

- client issues a new request only when previous response has been received.
- one RTT for each referenced object.

Persistent HTTP with pipelining

- default in HTTP/1.1
- client sends requests as soon as it encounters a referenced object.
- as little as one RTT for all the referenced objects.

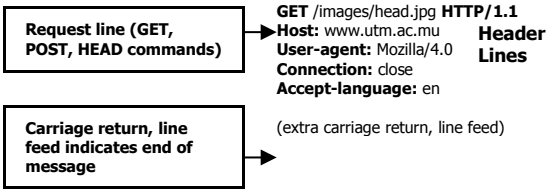
HTTP messages

- There are 2 types of HTTP messages: Request and Response.

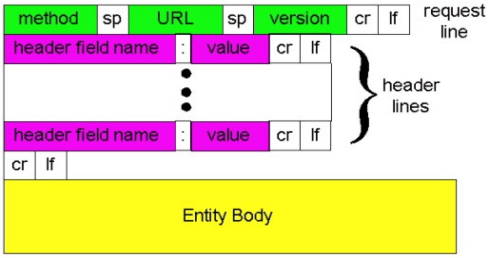
HTTP Request message

- ASCII (Human readable format)

e.g.



General Format



Uploading Form Input

POST method:

- Web page often includes form input.
- Input is uploaded to server in entity body.

GET method:

- Uses URL method
- Input is uploaded in URL field of request line: e.g. www.xxx.com/indexsearch?engineering

Method Types
HTTP/1.0:

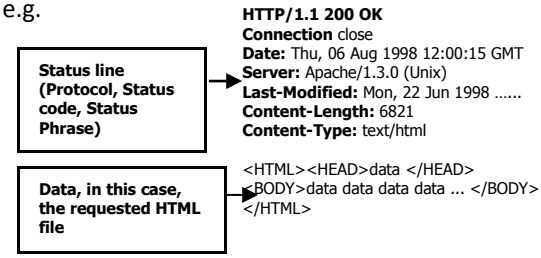
- GET, POST
- HEAD: asks server to leave requested object out of response.

HTTP/1.1:

- GET, POST, HEAD
- PUT: uploads file in entity body to path specified in URL field.
- DELETE: deletes file specified in URL field.

HTTP Response message

e.g.



HTTP Response Status Code

- Found on the first line of the client-server response message.

Some sample status codes:

200 OK: request succeeded, requested object later in this message
301 Moved Permanently: requested object moved, new location later in same message
400 Bad Request: request message not understood by server
404 Not Found: requested document not found on this server
505 HTTP Version not supported: self explanatory.

Trying out HTTP (client side) for yourself

Telnet to Telecom Plus personal pages web server:

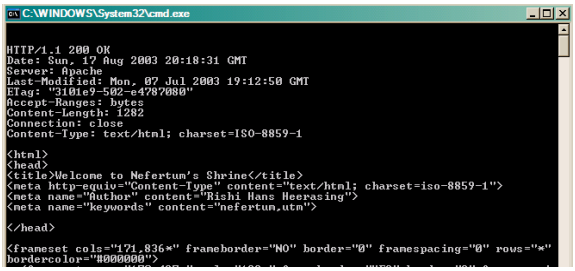
When you are online on your PC, open the Command Prompt Window (or choose **Run**, type **cmd**) and type the following as is: (Pay attention to the spaces)

telnet(space)pages.intnet.mu(space)80(enter)

The command you just typed has open a TCP connection on port 80 on the server **pages.intnet.mu**. The screen should go blank. Now everything you type in will get sent to *port 80* of the server. Next type your request message: (e.g. try to get my site's homepage)

GET(space)/rhh/index.htm(space)HTTP/1.1(enter)(enter)

If you typed well, you should get something like this:



Otherwise, you will get a 400 or 404 error codes.

Note: you might not see your request message on the screen while typing. Don't worry, just type it blindly.

Try it with other referenced objects as well.

Client-Server Interaction: Authorization

Authorization: control access to server content.

- *authorization credentials*: typically username, password
- *stateless*: *client* must present authorization in *each* request
- *authorization*: header line in each request
- if no authorization: header, server refuses access, sends WWW authenticate: header line in response

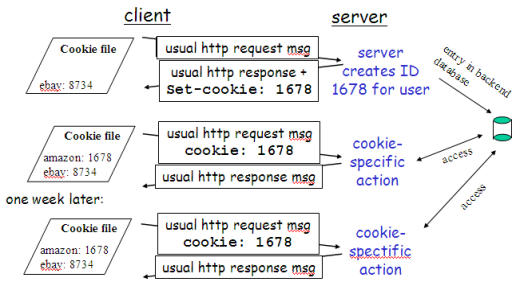
Cookies: Keeping "State"

Many major websites use cookies nowadays.

Four components:

1) cookie header line in the HTTP response message
2) cookie header line in HTTP request message
3) cookie file kept on user's host and managed by user's browser
4) Database at Web site

e.g.
Susan always accesses the Internet from the same PC. She visits an Ecommerce site for first time e.g. Amazon. When initial HTTP request arrives at site, site generates a unique ID and creates an entry in database for ID.



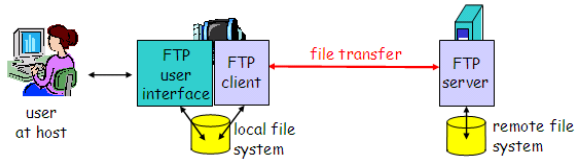
When and where cookies can be used:

authorization, shopping carts, recommendations
user session state (Web e-mail e.g. Hotmail)

Cookies and Privacy:

cookies permit sites to learn a lot about you
you may supply name and e-mail to sites
search engines use redirection & cookies to learn yet more
advertising companies obtain info across sites

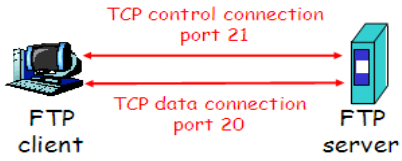
FTP: File Transfer Protocol



- transfer file to/from remote host
- client/server model
  client: side that initiates transfer (either to/from remote)
  server: remote host
- ftp: RFC 959
- ftp server: port 21

Separate Control and Data Connections

- FTP client contacts FTP server at port 21, specifying TCP as transport protocol.
- Client obtains authorization over control connection.
- Client browses remote directory by sending commands over control connection.
- When *server* receives a command for a file transfer, the server opens a *TCP data connection* to client.
- *After transferring one file, server closes connection*.
- *Server* opens a second *TCP data connection* to transfer another file.
- Control connection: "*out of band*"
- *FTP* server maintains *"state"*: current directory, earlier authentication.



Sample commands: sent as ASCII text over control channel

- USER username, PASS password, LIST return list of file in current directory
- RETR filename retrieves (gets) file, STOR filename stores (puts) file onto remote host.

Sample status codes and phrase: (as in HTTP)

- 331 Username OK, password required, 125 data connection already open; transfer starting, 425 Cannot open data connection, 452 Error writing file.

Electronic Mail

Three major components:
- user agents
- mail servers
- simple mail transfer protocol: SMTP

User Agent
- a.k.a. "email reader"
- composing, editing, reading and sending email messages
- e.g., Eudora, Outlook Express, Thunderbird,elm
- outgoing, incoming messages stored on server.

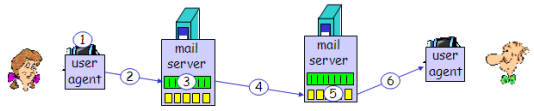
Mail Servers
- mailbox contains incoming messages for user
- message queue of outgoing mail messages
- SMTP protocol between mail servers to send email messages
- client: sending mail server
- server: receiving mail server

SMTP [RFC 2821]

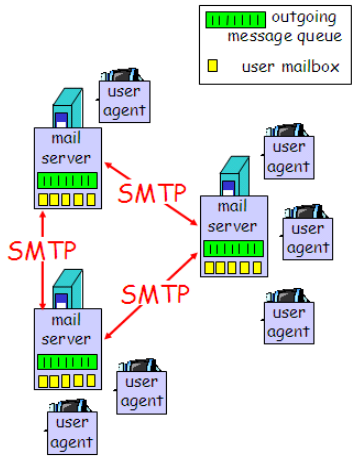
- uses TCP to reliably transfer email message from client to server, port 25
- direct transfer: sending server to receiving server
- Three phases of transfer:
  handshaking (greeting)
  transfer of messages
  closure
- command/response interaction
  commands: ASCII text
  response: status code and phrase
- messages must be in 7-bit ASCII

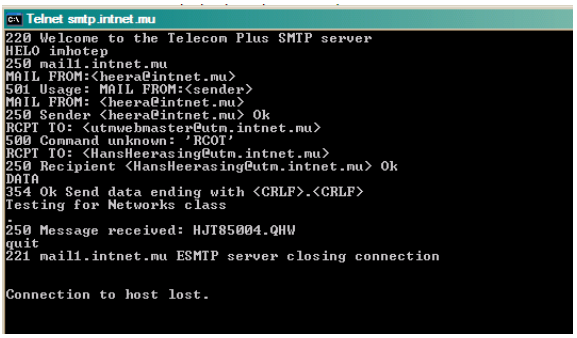
Example: Alice sends a message to Bob

1) Alice uses UA to compose message and "to" bob@utm.intnet.mu
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



Typical SMTP Interaction





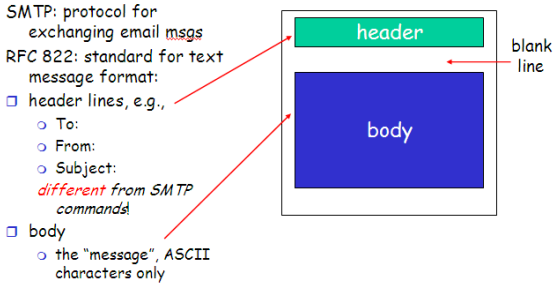
TRY sending a mail to yourself using the Command Prompt alone.

- telnet smtp.intnet.mu 25
- see 220 reply from server, Enter HELO, MAIL FROM, RCPT TO, DATA, QUIT.

Summary

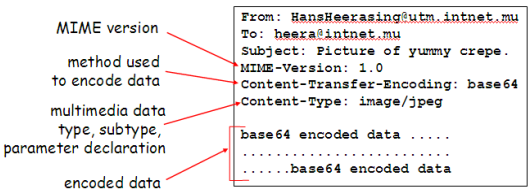
- SMTP uses persistent connections.
- SMTP requires message (header & body) to be in 7-bit ASCII.
- SMTP server uses CRLF.CRLF to determine end of message
- HTTP: pull SMTP: push
- Both have ASCII command/response interaction, status codes.
- HTTP: each object encapsulated in its own response message.
- SMTP: multiple objects sent in multipart messages.

Mail Message Format



Message format: Multipurpose Internet Mail Extensions

- MIME: multipurpose Internet Mail Extension, RFC 2045, 2056
- Additional lines in message header declare MIME content Type and Version.



MIME Types:

Content-type: type/subtype, parameters

Text: example subtypes: plain, html
Image: example subtypes: jpeg, gif
Audio: example subtypes: basic (8-bit μ-law encoded), 32kbps PCM (32 kbps coding)
Video: example subtypes: mpeg, qt (QuickTime)
Application: example subtypes: msword, octet-stream

Mail Access Protocols

- SMTP: delivery/storage to receiver's server
- Mail access protocol: retrieval from server
  - POP: Post Office Protocol [RFC 1939]
    authorization (agent <-->server) and download
  - IMAP: Internet Mail Access Protocol [RFC 1730]
    more features (more complex)
    manipulation of stored messages on server
  - HTTP: Hotmail, Yahoo! Mail, etc.

Properties of POP3

- Previous example uses "download and delete" mode.
- Bob cannot re-read e-mail if he changes client
- "Download-and-keep": copies of messages on different clients
- POP3 is stateless across sessions

IMAP Protocols

- Keep all messages in one place: the server.
- Allows user to organize messages in folders.
- IMAP keeps user state across sessions:
  names of folders and mappings between message IDs and folder name.

DNS- Domain Name System

People – Many Identifiers: Social Security #, National ID #...

What about internet hosts, routers, etc…?

- IP address (32 bit) – used for addressing datagrams.
- Domain Name, e.g. wtlab.utm.ac.mu used by us.

What is responsible for mapping IP Address to Domain Names? DNS

DNS

- Distributed Database - implemented in hierarchy of many *name servers.*
- Application-Layer Protocol – Host, routers, name servers to communicate to *resolve* names (Address/Name Translation) *Note: Complexity at Network's "edge"*

Why not have a centralized DNS?

- Single point of failure
- Traffic volume
- Distant centralized database
- Maintenance
- No one server has all name-to-IP address mappings.

LOCAL NAME SERVERS

- Each ISP, company has a local default name server e.g. TPlus: DNS server is dns1.intnet.mu at IP 202.123.2.6
- Host DNS query first goes to local name server.

AUTHORITATIVE NAME SERVERS

- For a host: stores that host's IP address, name.
- Can perform name/address translation for that host's name.

ROOT NAME SERVERS

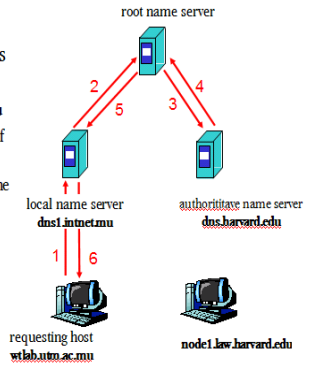
- contacted by local name server that can not resolve name
- root name server:
  - contacts authoritative name server if name mapping not known
  - gets mapping
  - returns mapping to local name server

13 root name servers worldwide

Simple DNS example

Host **wtlab.utm.ac.mu** wants IP address of **node1.law.harvard.edu**

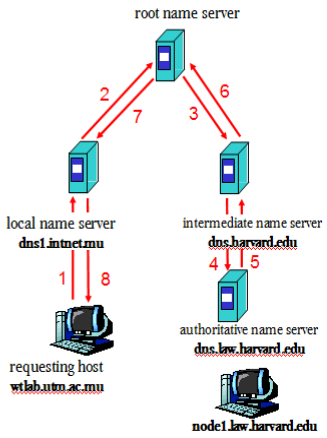
1. contacts its local DNS server, **dns1.intnet.mu**
2. **dns1.intnet.mu** contacts root name server, if necessary.
3. root name server contacts authoritative name server, **dns.harvard.edu**, if necessary



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: Caching and updating records.

- Once (any) name server learns mapping, it caches mapping
- Cache entries timeout and gets refreshed after some time (24-48 Hours)