Justin Ciocoi

Oct. 14, 2023

CSCI 379: Computer Networking

Midterm Review

Chapter 2: Application Layer

- In chapter 2, a variety of key application-layer concepts are defined, including:
  - *Network services* required by applications
  - *Clients and Servers*
  - *Processes*
  - *Transport-Layer Interfaces*

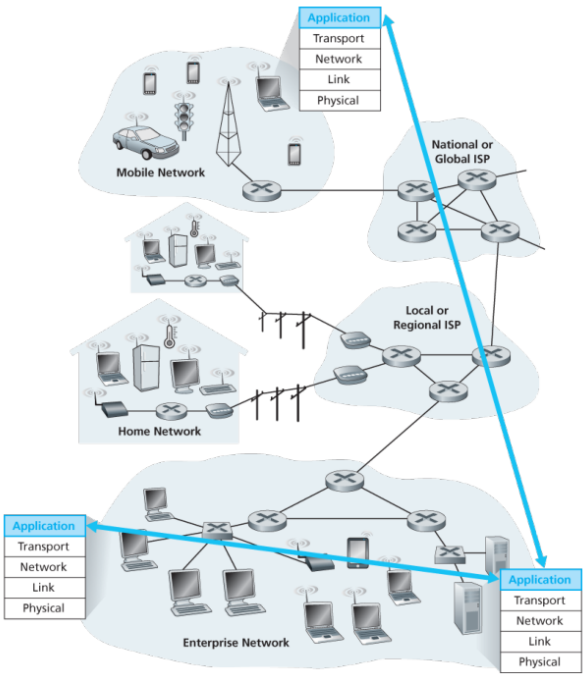


Figure 2.1 Communication for a network application takes place between end systems at the application layer

2.1: Principles of Network Applications

- **The Client-Server Architecture**
    - In the client-server model, there is an *always-on* server that can be accessed by a client at the client's convenience
    - The *server* has a *unique, fixed* ***IP Address***
    - Many of the most popular web services operate using this architecture
- **P2P Architecture**
    - In this model, there is no reliance on a central server which can be accessed, and instead network tasks are achieved through intermittently connected hosts called ***peers***
    - This model is used in things like torrenting, where a file housed on another end system of the same variety is downloaded, and in internet telephony applications like Skype and VOIP
    - The P2P model provides a much better level of scalability, as it is *self-scalable* since an additional user will also provide additional resources and thus the resource pool *should* grow at a rate proportional to resource use

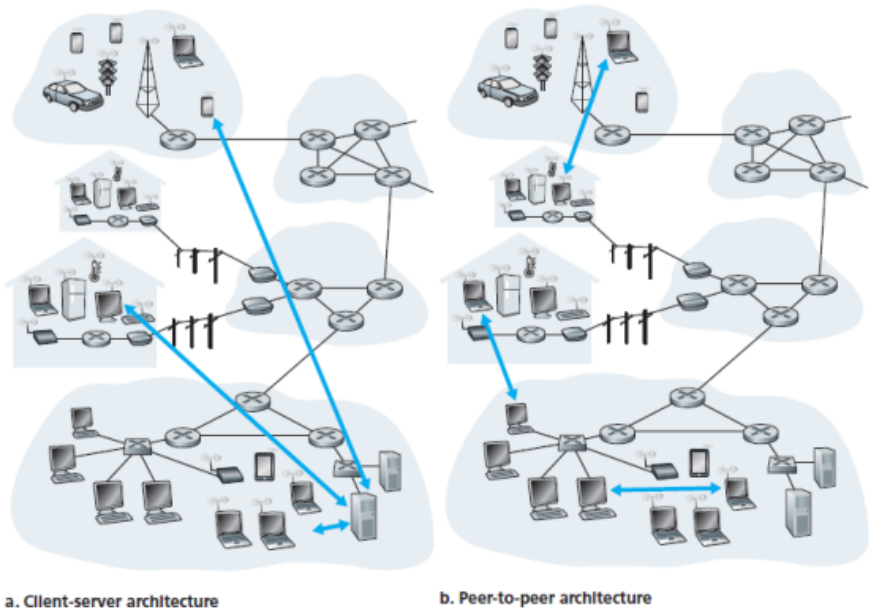


Figure 2.2 (a) Client-server architecture; (b) P2P architecture

- **Inter-Process Communication**
    - When two processes on *different* end systems want to communicate, they must do so over the network

- To do this, a process on an end system will *send and receive* messages through a software interface called ***socket***, also called a *port*
- A socket is the interface between the application layer and the transport layer within a host
- It can also be referred to as the *Application-Programming Interface*, or ***API***
- A destination process can be identified using an IP address to identify the end system, and then a socket, or port number specifying the destination process on said end system

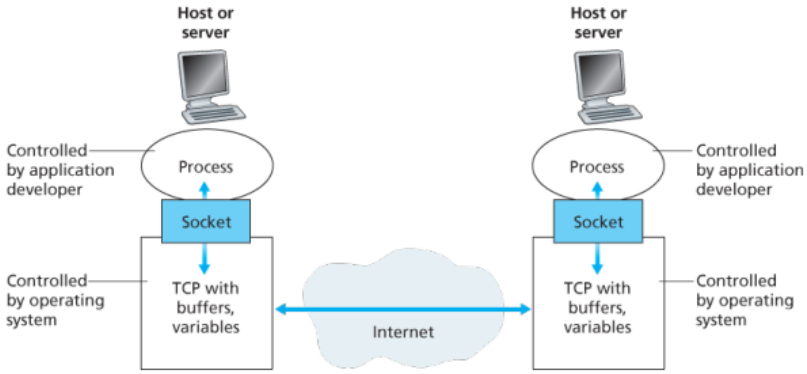


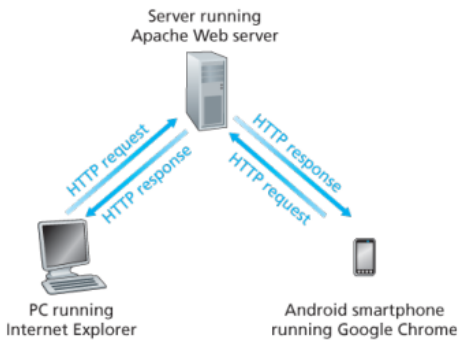
Figure 2.3 Application processes, sockets, and underlying transport protocol

- **Properties of Transport Services**
  - *Reliable Data Transfer*
    - There is always the potential for packets to get lost when traveling across a network
    - Therefore, it can be advantageous for a transport protocol to provide a guarantee of data delivery, which we can call reliable data transfer
    - Certain applications are *loss-tolerant* which means they can tolerate the loss of some packets
      - This includes things like music or video streaming, where a lost packet can manifest in buffering or bit-rate lowering
    - Other applications, like document downloading or email require a reliable data transfer
  - *Throughput*
  - *Timing*
  - *Security*

- **Transport Services Provided by the Internet**
  - *TCP Services*
    - *Connection-Oriented Service*, which refers to the three way handshake involved in TCP, initiating a connection until it is told to be severed
    - *Reliable Data Transfer*, which refers to TCP's guarantee of data delivery
  - *UDP Services*
    - UDP is a *best-effort, connectionless* protocol

2.2: The Web and HTTP

- **Hyper Text Transfer Protocol (HTTP)**
  - HTTP is implemented in both a client-side and server-side program, which run on different end systems and communicate using HTTP messages
  - HTTP uses TCP as its underlying protocol rather than UDP
  - Once HTTP sends the message into a socket, TCP takes over and therefore HTTP does not have to worry about doing the job of reliable data transfer
  - HTTP is a *stateless protocol,* which means the server does not store any state information about the client



- **Persistent and Non-Persistent Connections**
  - HTTP with non persistent connections will establish a new connection for each HTTP message that is being sent back and forth between client and server
  - This can be inefficient due to the need for a TCP handshake for each web object, whereas a persistent connection can send multiple objects over the same TCP

connection

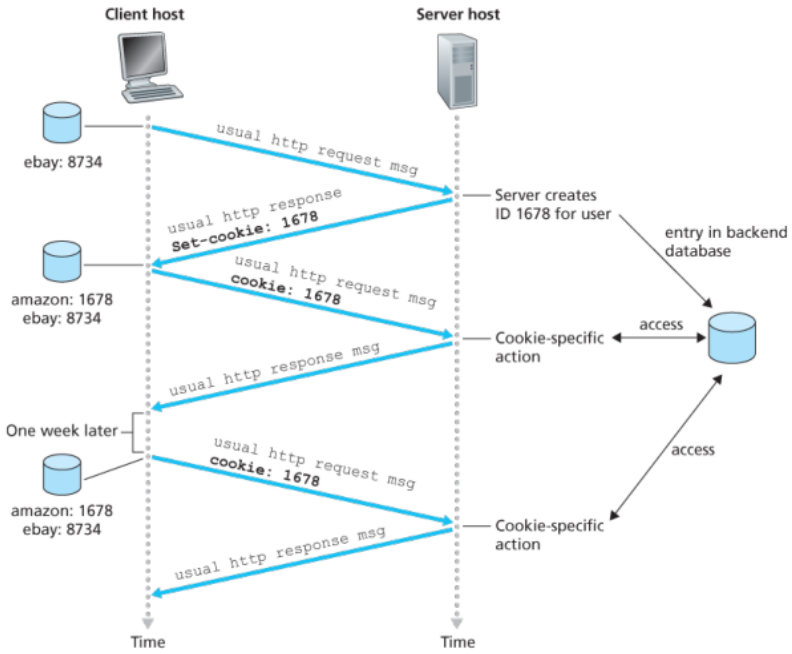
    - A persistent connection will be closed after a variable *time-out* time during which the connection in question is not used

- **HTTP Message Format**
    - *HTTP Request Message*
- ```
  GET /somedir/page.html HTTP/1.1
  Host: www.someschool.edu
  Connection: close
  User-agent: Mozilla/5.0
  Accept-language: fr
  ```
    - The first line includes the *Method*, in this case `GET`, the *File URL*, and the *HTTP Version*
    - The second line specifies the *host IP Address*
    - The third line identifies whether it is a *persistent or non-persistent connection*
    - The fourth line specifies *the program operating on the client side*
    - The last line indicates *language preference* for the response message
- *HTTP Response Message*
    - The response message is very similar to the request message, but it contains a status code and message corresponding to the site's status
    - ```
      HTTP/1.1 200 OK
      Connection: close
      Date: Tue, 18 Aug 2015 15:44:04 GMT
      Server: Apache/2.2.3 (CentOS)
      Last-Modified: Tue, 18 Aug 2015 15:11:03 GMT
      Content-Length: 6821
      Content-Type: text/html

      (data data data data data ...)
      ```
    - Common status codes include
        - 200 OK
        - 301 Moved Permanently
        - 400 Bad Request
        - 404 Not Found

        - 505 HTTP Version Not Supported

- **Cookies**
    - Since HTTP is stateless, it does not directly store any information about the user on the server side
    - However, for many sites, having the site remember prior user interaction is clearly beneficial, and therefore a system is in place to facilitate this
    - *Cookies* use a unique identification number for each user that is sent in an HTTP request messaged which is indexed in a backend server database that is used to store information corresponding to that same unique identifier



- **Web Caching**
    - A *web cache* or *proxy server* is an intermediate server where HTTP information is cached for future use
    - When an HTTP request goes out, it is first sent to the proxy server
        - If the proxy server has the desired information, it can return the information and bypass the origin server entirely, leading to more efficient navigation of the web
    - The proxy server will then forward the request to an origin server to service the user
    -

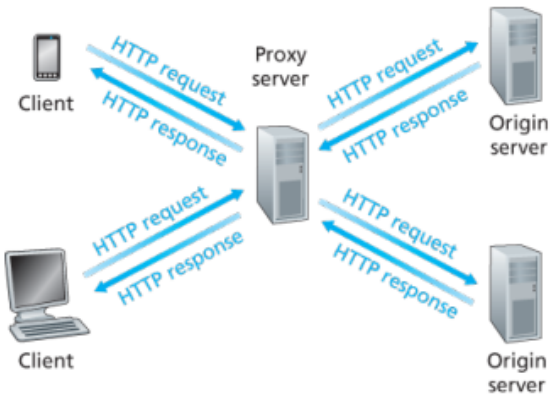


Figure 2.11 Clients requesting objects through a Web cache

- *The Conditional GET*
  - The conditional `GET` command allows a proxy server to check if a file has been updated on an origin server since the last cache of that file
  - It allows proxy servers to check if the information they have is the most up to date or not and respond to users with the most recent version of the file

2.3: Electronic Mail in the Internet

- The internet mail system has three main components
  - *User agents*
  - *Mail servers*
  - *Simple Mail Transfer Protocol (SMTP)*

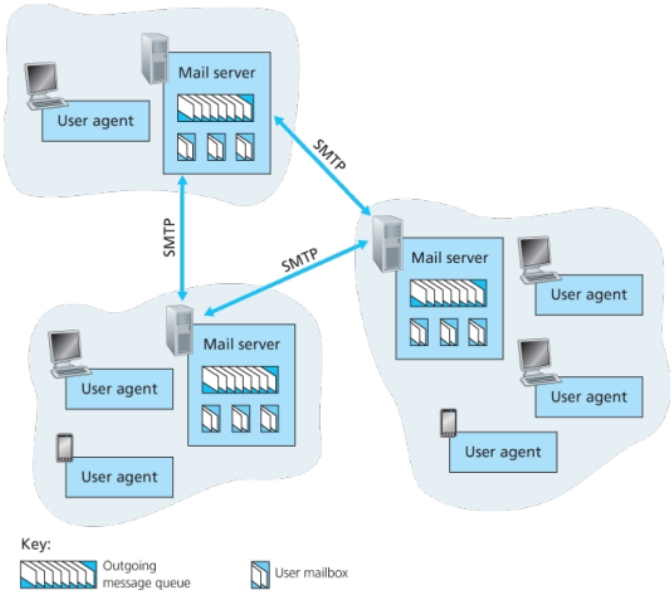


Figure 2.14 A high-level view of the Internet e-mail system

- **Comparison to HTTP**
  - HTTP is a *pull protocol*, whereas SMTP is a *push protocol*
  - Also, SMTP imposes the restriction of 7-bit ASCII encoding in the body and header of all messages, whereas HTTP does not

2.4: DNS - The Internet's Directory Service

- Hosts can be identified in different ways including a hostname, like *google.com* or an IP address like *121.7.106.83*
- For people, remembering phonetically and mnemonically easy names, like *google .com* is easier than remembering IP addresses, but for routers, the hierarchical structure of IP addresses is highly beneficial
- The *DNS* refers to two things
  - A distributed database implemented in a hierarchy of DNS servers spread across the world
  - An application-layer protocol that allows hosts to query this distributed database
- Usually, DNS servers are UNIX machines running the Berkeley Internet Name Domain (BIND) software
- DNS runs over UDP and uses port 53

- DNS is often used by other application-layer protocols, like HTTP and SMTP, to translate user-supplied hostnames to their corresponding IP addresses
- DNS provides aliasing for hosts and mail servers allowing for easier user access
- DNS also provides load distribution among multiple replicated servers which can help improve overall performance of the DNS
- **The Hierarchical Structure of DNS Servers**
-

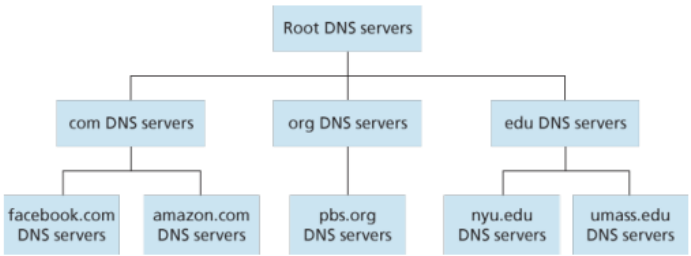


Figure 2.17 Portion of the hierarchy of DNS servers

  - **Root DNS servers** contain the IP addresses of the top-level domain (TLD) servers
    - There are about 400 of these servers worldwide
  - **Top-Level Domain Servers** contain the IP addresses of authoritative DNS Servers
    - Each top level domain (.com, .org, .net, .uk, .fr) has its own TLD server, or cluster of servers
  - **Authoritative DNS Servers** which contain the IP addresses of any publicly accessible files from an organization
    - Each organization has its own DNS server or pays a third-party to provide the service for them
- **Path That a DNS Request Takes**
  - A host sends a request to a Local DNS server, which sends it to a root DNS server which returns the IP address of the corresponding top-level domain server
  - The TLD server responds with the IP address of the authoritative DNS which sends back the file to the local DNS server
  - Finally, the local DNS server returns the file to the requesting host)

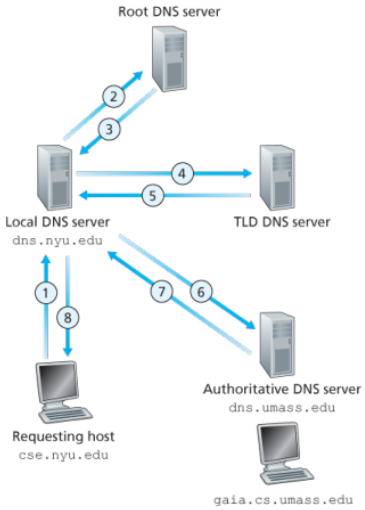


Figure 2.19 Interaction of the various DNS servers

- **DNS Caching**
  - The above method can be inefficient since every level of the DNS hierarchy must be contacted for every DNS request
  - To remedy this inefficiency, we can use *DNS Caching*
  - With DNS caching, any level of the DNS hierarchy can cache information and bypass contacting the higher levels of the hierarchy
  - This can be seen below

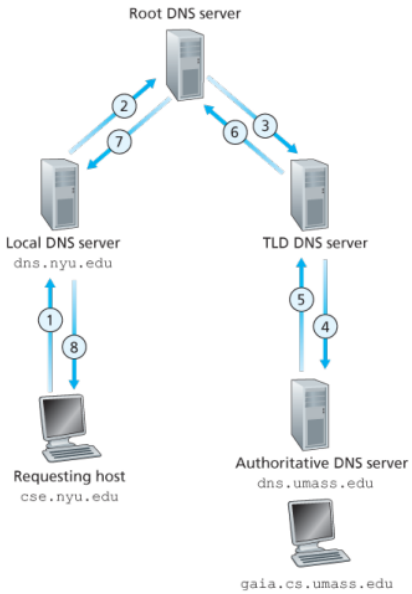
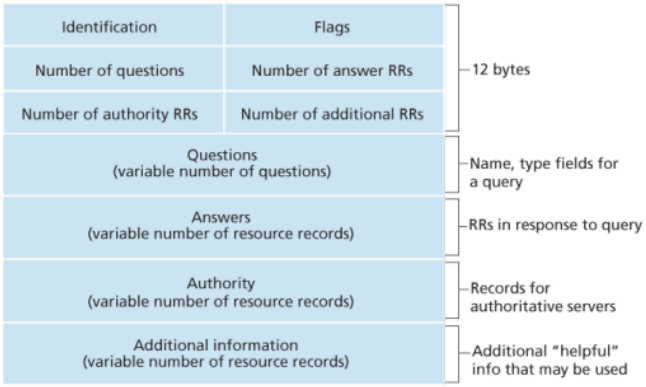


Figure 2.20 Recursive queries in DNS

- **DNS Message Format**



- **Entering Records into the DNS Database**
  - To enter data into the DNS database, an IP address and corresponding domain name must be registered
  - A *registrar* is a commercial entity that verifies the uniqueness of the domain name and enters it into the DNS database

Chapter 3: Transport Layer

3.1: Introduction and Transport-Layer Services

- A transport-layer protocol provides for *logical communication* between application processes running on different hosts
- These protocols are implemented in end systems, but do not need to be implemented in network routers

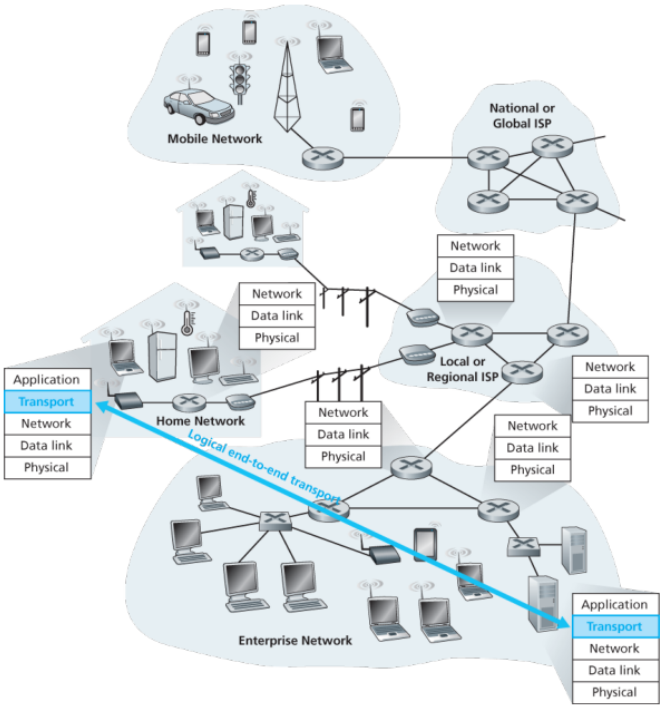


Figure 3.1 The transport layer provides logical rather than physical communication between application processes

3.2: Multiplexing and Demultiplexing

- **Multiplexing**
  - When a segment is sent from the application layer into the network layer, it is multiplexed such that the network layer can understand it
- **Demultiplexing**
  - Similarly, a segment coming from the transport layer must be demultiplexed before it enters the application-layer