Multimedia Communication Systems II

Review of Networking Basics

Yao Wang
Polytechnic University, Brooklyn, NY11201
http://eeweb.poly.edu/~yao
Overview of Telecommunication Networks and Internet

Based on
These slides are extracted from the slides made by authors of the book (J. F. Kurose and K. Ross), available from the publisher site for instructors and students. We would like to thank the authors for the excellent book and the slides.

The third edition website:
http://wps.aw.com/aw_kurose_network_3
Roadmap

- What is the Internet?
- Network edge
- Network core
- Internet structure and ISPs
- Packet delay and loss
- Protocol layers, service models
What's the Internet: “nuts and bolts” view

- **millions of connected computing devices:** *hosts, end-systems*
  - PCs workstations, servers
  - PDAs phones, toasters
  - running *network apps*

- **communication links**
  - fiber, copper, radio, satellite
  - transmission rate = *bandwidth*

- **routers:** forward packets (chunks of data)
What’s the Internet: “nuts and bolts” view

- **protocols** control sending, receiving of msgs
  - e.g., TCP, IP, HTTP, FTP, PPP

- **Internet:** “network of networks”
  - loosely hierarchical
  - public Internet versus private intranet

- **Internet standards**
  - RFC: Request for comments
  - IETF: Internet Engineering Task Force
A closer look at network structure:

- **network edge:** applications and hosts
- **network core:**
  - routers
  - network of networks
- **access networks, physical media:** communication links
**Network edge: connection-oriented service**

**Goal:** data transfer between end systems

- **handshaking:** setup (prepare for) data transfer ahead of time
  - Hello, hello back human protocol
  - *set up “state”* in two communicating hosts

- **TCP - Transmission Control Protocol**
  - Internet’s connection-oriented service

**TCP service** [RFC 793]

- **reliable, in-order byte-stream data transfer**
  - loss: acknowledgements and retransmissions

- **flow control:**
  - sender won’t overwhelm receiver

- **congestion control:**
  - senders “slow down sending rate” when network congested
**Network edge: connectionless service**

*Goal:* data transfer between end systems
- same as before!

- **UDP** - User Datagram Protocol [RFC 768]: Internet’s connectionless service
  - unreliable data transfer
  - no flow control
  - no congestion control

**App’s using TCP:**
- HTTP (Web), FTP (file transfer), Telnet (remote login), SMTP (email)

**App’s using UDP:**
- streaming media, teleconferencing, DNS, Internet telephony
The Network Core

- mesh of interconnected routers
- the fundamental question: how is data transferred through net?
  - circuit switching: dedicated circuit per call: telephone net
  - packet-switching: data sent thru net in discrete “chunks”
Network Core: Circuit Switching

End-end resources reserved for “call”

- link bandwidth, switch capacity
- dedicated resources: no sharing
- circuit-like (guaranteed) performance
- call setup required
Network Core: Circuit Switching

- Network resources (e.g., bandwidth) divided into “pieces”
  - Pieces allocated to calls
  - Resource piece *idle* if not used by owning call (*no sharing*)

- Dividing link bandwidth into “pieces”
  - Frequency division
  - Time division
Circuit Switching: FDMA and TDMA

FDMA

Example:
4 users

TDMA
Network Core: Packet Switching

- each end-end data stream divided into packets
  - user A, B packets share network resources
  - each packet uses full link bandwidth
  - resources used as needed

resource contention:
- aggregate resource demand can exceed amount available
- congestion: packets queue, wait for link use
- store and forward: packets move one hop at a time
  - transmit over link
  - wait turn at next link

Bandwidth division into “pieces”
- Dedicated allocation
- Resource reservation

EE4414 Networking Basics 14
Packet Switching: Statistical Multiplexing

Sequence of A & B packets does not have fixed pattern \(\Rightarrow\) statistical multiplexing.

In TDM each host gets same slot in revolving TDM frame.
Packet switching versus circuit switching

Packet switching allows more users to use network!

- 1 Mbit link
- each user:
  - 100 kbps when “active”
  - active 10% of time
- circuit-switching:
  - 10 users
- packet switching:
  - with 35 users, probability > 10 active less than .0004
Packet switching versus circuit switching

Is packet switching a “slam dunk winner?”

- Great for bursty data
  - resource sharing
  - simpler, no call setup
- Excessive congestion: packet delay and loss
  - protocols needed for reliable data transfer, congestion control
- Q: How to provide circuit-like behavior?
  - bandwidth guarantees needed for audio/video apps
  - still an unsolved problem (chapter 6)
Packet-switched networks: forwarding

- **Goal**: move packets through routers from source to destination
  - we’ll study several path selection (i.e. routing) algorithms (chapter 4)

- **Datagram network**:
  - *destination address* in packet determines next hop
  - routes may change during session
  - analogy: driving, asking directions

- **Virtual circuit network**:
  - each packet carries tag (virtual circuit ID), tag determines next hop
  - fixed path determined at *call setup time*, remains fixed thru call
  - *routers maintain per-call state*
Network Taxonomy

- Telecommunication networks
  - Circuit-switched networks
    - FDM
    - TDM
  - Packet-switched networks
    - Networks with VCs
    - Datagram Networks

- Datagram network is *not* either connection-oriented or connectionless.
- Internet provides both connection-oriented (TCP) and connectionless services (UDP) to apps.
Internet structure: network of networks

- roughly hierarchical
- at center: “tier-1” ISPs (e.g., UUNet, BBN/Genuity, Sprint, AT&T), national/international coverage
  - treat each other as equals

Tier-1 providers also interconnect at public network access points (NAPs)

Tier-1 providers interconnect (peer) privately
Tier-1 ISP: e.g., Sprint

Sprint US backbone network

- Seattle
- Tacoma
- Stockton
- San Jose
- Anaheim
- New York
- Pennsauken
- Wash. DC
- New York
- Pensauken
- Relay
- Wash. DC
- Orlando
- Atlanta
- Roachdale
- Kansas City
- Chicago
- Fort Worth
-芰
- OC12 (622 Mbps)
- OC48 (2.4 Gbps)
- DS3 (45 Mbps)
- OC3 (155 Mbps)

Legend:
- Black: OC48 (2.4 Gbps)
- Red: OC12 (622 Mbps)
- Blue: OC3 (155 Mbps)
- Purple: DS3 (45 Mbps)
Internet structure: network of networks

- "Tier-2" ISPs: smaller (often regional) ISPs
  - Connect to one or more tier-1 ISPs, possibly other tier-2 ISPs

Tier-2 ISP pays tier-1 ISP for connectivity to rest of Internet
- tier-2 ISP is customer of tier-1 provider

Tier-2 ISPs also peer privately with each other, interconnect at NAP
Internet structure: network of networks

- "Tier-3" ISPs and local ISPs
  - last hop ("access") network (closest to end systems)
Internet structure: network of networks

- a packet passes through many networks!
Delay and Loss in Packet Switched Networks
Four sources of packet delay

1. nodal processing:
   - check bit errors
   - determine output link

2. queueing
   - time waiting at output link for transmission
   - depends on congestion level of router
Delay in packet-switched networks

3. Transmission delay:
   - $R = \text{link bandwidth (bps)}$
   - $L = \text{packet length (bits)}$
   - time to send bits into link = $L/R$

4. Propagation delay:
   - $d = \text{length of physical link}$
   - $s = \text{propagation speed in medium (} \approx 2 \times 10^8 \text{ m/sec)}$
   - propagation delay = $d/s$

Note: $s$ and $R$ are very different quantities!
Packet loss

- queue (aka buffer) preceding link in buffer has finite capacity
- when packet arrives to full queue, packet is dropped (aka lost)
- lost packet may be retransmitted by previous node, by source end system, or not retransmitted at all
- For applications with delay bounds, packets arriving too late are effectively lost! (more on this later!)
Internet protocol stack

- **application**: supporting network applications
  - FTP, SMTP, STTP

- **transport**: host-host data transfer
  - TCP, UDP

- **network**: routing of datagrams from source to destination
  - IP, routing protocols

- **link**: data transfer between neighboring network elements
  - PPP, Ethernet

- **physical**: bits “on the wire”
Layering: logical communication

Each layer:
- distributed
- "entities" implement layer functions at each node
- entities perform actions, exchange messages with peers
Layering: *logical* communication

**E.g.: transport**
- take data from app
- add addressing, reliability check info to form “datagram”
- send datagram to peer
- wait for peer to ack receipt
- analogy: post office
Layering: physical communication
Protocol layering and data encapsulation

Each layer takes data from above
- adds header information to create new data unit
- passes new data unit to layer below
Summary

- **Generic networks**
  - Two types of services:
    - connection-oriented and connectionless
  - Two types of switching (routing)
    - Circuit switching and packet switching (datagram and VC)
    - Internet uses datagram packet switching, and can offer both connection-oriented (through TCP) and connectionless services

- **Internet structure and protocol layers**
  - Edges (workstations, servers), core (routers), physical links (access links and backbone links), coordinated by protocols
  - Protocol layers of the Internet
    - Application (HTTP, email), transport (TCP, UDP), network (IP), link, physical
Internet Protocols: TCP/IP

Based on Slides by Jorg Liebeherr for EL536
and
Slides by Kurose/Ross, Chapter III
(Transport Layer)
Roadmap

- Overview of IP protocol stack
  - Network vs transport layers
- Network layer
  - IP addresses
  - IP datagrams
- Transport layer: UDP
- Transport layer: TCP
  - Session management
  - TCP segment format
  - Retransmission
  - Flow control
IP Protocol Stack

- IP (Internet Protocol) is a Network Layer Protocol

![IP Protocol Stack Diagram]
Transport vs. network layer

- **network layer**: logical communication between hosts
  - Responsible mainly for routing
  - Network layer packets are called datagrams
  - Routing is based on datagram headers

- **transport layer**: logical communication between processes
  - Relies on, enhances, network layer services
  - Responsible for
    - Session management (establish, disconnect)
    - Error detection and control
    - Flow control
  - Transport layer packets are called segments
  - Routers do not look into segment header
IP Overview

- IP is the highest layer protocol that is implemented at both routers and hosts:

**Analogy between the transport of a datagram with the delivery of a letter:** each datagram has a source and destination address, and each intermediate router forwards an entering datagram to the next intermediate router or the final host based on the destination address.
Internet Addresses

- Each network interface on the Internet has a unique global address, called the IP address.
- An IP address:
  - is 32 bits long.
  - encodes a network number and a host number.
- IP address has a hierarchical structure:
  - e.g. 128.238.42.112

Diagram:
- Most US universities
- Poly
- Subnet
- host
IP Datagram Format

- 20 bytes ≤ Header Size ≤ $2^4 \times 32$ bit-words = 60 bytes
- 20 bytes ≤ Total Length ≤ $2^{16}$ bytes = 65536 bytes

---

<table>
<thead>
<tr>
<th>version (4 bits)</th>
<th>header length</th>
<th>Type of Service/TOS (8 bits)</th>
<th>Total Length (in bytes) (16 bits)</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td>Identification (16 bits)</td>
<td>flags (3 bits)</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Protocol (8 bits)</td>
<td>Fragment Offset (13 bits)</td>
</tr>
<tr>
<td>TTL Time-to-Live (8 bits)</td>
<td>Protocol (8 bits)</td>
<td>Source IP address (32 bits)</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>Destination IP address (32 bits)</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>Options (if any, &lt;40 bytes)</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>DATA</td>
<td></td>
</tr>
</tbody>
</table>
Fields of the IP Header

- **Protocol:** Specifies the higher-layer protocol. Used for demultiplexing to higher layers.

- **Header checksum:** verifies correctness of header.
  - 16 bit ones complement addition of all 16-bit words in the header, verified and recomputed at each router.
Transport Protocols in the Internet
Transport Protocols in the Internet

**UDP - User Datagram Protocol**
- datagram oriented
- Unreliable (best-effort), connectionless
- simple
- unicast and multicast
- Low-delay, hence good for multimedia applications
- used a lot for services
  - network management (SNMP), routing (RIP), naming (DNS), etc.

**TCP - Transmission Control Protocol**
- stream oriented, in sequence
- reliable, connection-oriented
- complex
- only unicast
- used for most Internet applications:
  - web (http), email (smtp), file transfer (ftp), terminal (telnet), etc.
UDP - User Datagram Protocol

- UDP supports unreliable transmissions of datagrams
- UDP merely extends the host-to-host delivery service of IP datagram to an application-to-application service
- The only thing that UDP adds is multiplexing and demultiplexing (encapsulation)
UDP Format

- **Port numbers** identify sending and receiving applications (processes). Maximum port number is $2^{16}-1=65,535$

- **Message Length** is at least 8 bytes (i.e., Data field can be empty) and at most 65,535

- **Checksum** is for header (of UDP and some of the IP header fields)
Port Numbers

- UDP (and TCP) use port numbers to identify applications.
- A globally unique address at the transport layer (for both UDP and TCP) is a tuple \(<\text{IP address, port number}\>\).
- There are 65,535 UDP ports per host.
**TCP = Transmission Control Protocol**

- Connection-oriented protocol
- Provides a reliable unicast end-to-end byte stream over an unreliable internetwork.
TCP is Connection-Oriented

- Before any data transfer, TCP establishes a connection:
  - One TCP entity is waiting for a connection ("server")
  - The other TCP entity ("client") contacts the server
- Use “three way handshake” for setting up connections
- Each connection is full duplex
Reliable Data Transfer

- Byte stream is broken up into chunks which are called **segments**
  - Receiver sends acknowledgements (ACKs) for segments
  - TCP maintains a timer. If an ACK is not received in time, the segment is retransmitted

- **Detecting errors:**
  - TCP has checksums for header and data. Segments with invalid checksums are discarded
  - Each byte that is transmitted has a sequence number
TCP Segment Format

- TCP segments have a 20 byte header with >= 0 bytes of data.

<table>
<thead>
<tr>
<th>IP header</th>
<th>TCP header</th>
<th>TCP data</th>
</tr>
</thead>
<tbody>
<tr>
<td>20 bytes</td>
<td>20 bytes</td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Sequence number (32 bits)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Acknowledgement number (32 bits)</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Header length</th>
<th>Flags</th>
<th>Window size</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>TCP checksum</th>
<th>Urgent pointer</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Options (if any)</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
</tr>
</tbody>
</table>

DATA
TCP segment structure

- **Source port #**
- **Destination port #**
- **Sequence number**
- **Acknowledgement number**
- **Receive window**
- **Checksum**
- **Urg data pointer**

**Options (variable length)**

- **URG**: urgent data (generally not used)
- **ACK**: ACK # valid
- **PSH**: push data now (generally not used)
- **RST, SYN, FIN**: connection estab (setup, teardown commands)
- **Internet checksum** (as in UDP)

**Counting by bytes of data (not segments!)**

# bytes receiver willing to accept

**Internet checksum** (as in UDP)
TCP seq. #'s and ACKs

Seq. #'s:
- byte stream “number” of first byte in segment’s data

ACKs:
- seq # of next byte expected from other side
- cumulative ACK

Q: how receiver handles out-of-order segments
- A: TCP spec doesn’t say, - up to implementor

Simple telnet scenario:
- User types ‘C’
- Host A receives and sends back
- Host B echoes back
- Time:
  - Seq=42, ACK=79, data = ‘C’
  - Seq=79, ACK=43, data = ‘C’
  - Seq=43, ACK=80
  - Seq=43, ACK=80
TCP: retransmission scenarios

Host A
Seq=92, 8 bytes data
ACK=100

Host B
Seq=92, 8 bytes data
ACK=100

lost ACK scenario

SendBase = 100

Host A
Seq=92, 8 bytes data
ACK=100

Host B
Seq=92, 8 bytes data
ACK=120

lost ACK scenario

SendBase = 100

Host A
Seq=100, 20 bytes data
ACK=100

Host B
Seq=92, 8 bytes data
ACK=120

premature timeout

SendBase = 120

SendBase = 120
TCP retransmission scenarios (more)

Host A
Seq=92, 8 bytes data
ACK=100

Host B
Seq=100, 20 bytes data
ACK=100

X
loss

SendBase = 120

Cumulative ACK scenario
TCP Flow Control

- receive side of TCP connection has a receive buffer:

  
  - "recvWindow"
  - "recvBuffer"
  - "data from IP"
  - "spare room"
  - "TCP data in buffer"
  - "application process"

- app process may be slow at reading from buffer

- flow control

  - sender won't overflow receiver's buffer by transmitting too much, too fast

- speed-matching service: matching the send rate to the receiving app's drain rate
TCP Flow control: how it works

- Rcvr advertises spare room by including value of RcvWindow in segments
- Sender limits unACKed data to RcvWindow
  - guarantees receive buffer doesn’t overflow

(Suppose TCP receiver discards out-of-order segments)

- spare room in buffer
  - $RcvWindow = RcvBuffer - \text{[LastByteRcvd - LastByteRead]}$
Principles of Congestion Control

Congestion:
- informally: “too many sources sending too much data too fast for network to handle”
- different from flow control!
  - sender sending to fast for the receiver to handle
- manifestations:
  - lost packets (buffer overflow at routers)
  - long delays (queueing in router buffers)
Approaches towards congestion control

Two broad approaches towards congestion control:

End-end congestion control:
- no explicit feedback from network
- congestion inferred from end-system observed loss, delay
- approach taken by TCP

Network-assisted congestion control:
- routers provide feedback to end systems
  - single bit indicating congestion (SNA, DECbit, TCP/IP ECN, ATM)
  - explicit rate sender should send at
**TCP Congestion Control**

- end-end control (no network assistance)
- sender sets transmission rate based on the current value of CongestionWindow (CongWin)
- CongWin is dynamic, function of perceived network congestion

**How does sender perceive congestion?**

- loss event = timeout or 3 duplicate acks
- TCP sender reduces rate (CongWin) after loss event

**three mechanisms:**
- AIMD
- slow start
- conservative after timeout events
What you should know

- Difference between network and transport layer functions
- IP protocol
  - IP address scheme
  - IP datagram format
- UDP
  - Socket function
  - UDP datagram format
- TCP
  - TCP segment format, why using sequence number
  - Retransmission
  - Flow control
  - Congestion control
Application Layer Functions and Protocols with focus on Content Distribution/Retrieval on the Web

Based on Kurose/Ross, Chapter II --- Application Layer
**Roadmap**

- Principles of app layer protocols
  - Client-server paradigm
  - Transport service requirements of common app.
- Web and HTTP
- Content distribution networks
Applications and application-layer protocols

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

Application-layer protocols
- one “piece” of an app
- define messages exchanged by apps and actions taken
- use communication services provided by lower layer protocols (TCP, UDP)
App-layer protocol defines

- Types of messages exchanged, e.g., request & 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, KaZaA
Client-server paradigm

Typical network app has two pieces: client and server

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

**Server:**
- provides requested service to client
- e.g., Web server sends requested Web page, mail server delivers e-mail
What transport service does an app need?

**Data loss**
- Some apps (e.g., audio) can tolerate some loss
- Other apps (e.g., file transfer, telnet) require 100% reliable data transfer

**Timing**
- 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
# Transport service requirements of common apps

<table>
<thead>
<tr>
<th>Application</th>
<th>Data loss</th>
<th>Bandwidth</th>
<th>Time Sensitive</th>
</tr>
</thead>
<tbody>
<tr>
<td>file transfer</td>
<td>no loss</td>
<td>elastic</td>
<td>no</td>
</tr>
<tr>
<td>e-mail</td>
<td>no loss</td>
<td>elastic</td>
<td>no</td>
</tr>
<tr>
<td>Web documents</td>
<td>no loss</td>
<td>elastic</td>
<td>no</td>
</tr>
<tr>
<td>real-time audio/video</td>
<td>loss-tolerant</td>
<td>audio: 5kbps-1Mbps video:10kbps-5Mbps</td>
<td>yes, 100’s msec</td>
</tr>
<tr>
<td>stored audio/video</td>
<td>loss-tolerant</td>
<td>same as above</td>
<td>yes, few secs</td>
</tr>
<tr>
<td>interactive games</td>
<td>loss-tolerant</td>
<td>few kbps up</td>
<td>yes, 100’s msec</td>
</tr>
<tr>
<td>instant messaging</td>
<td>no loss</td>
<td>elastic</td>
<td>yes and no</td>
</tr>
</tbody>
</table>
Internet transport protocols 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 providing: timing, 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?
# Internet apps: application, transport protocols

<table>
<thead>
<tr>
<th>Application</th>
<th>Application layer protocol</th>
<th>Underlying transport protocol</th>
</tr>
</thead>
<tbody>
<tr>
<td>e-mail</td>
<td>SMTP [RFC 2821]</td>
<td>TCP</td>
</tr>
<tr>
<td>remote terminal access</td>
<td>Telnet [RFC 854]</td>
<td>TCP</td>
</tr>
<tr>
<td>Web</td>
<td>HTTP [RFC 2616]</td>
<td>TCP</td>
</tr>
<tr>
<td>file transfer</td>
<td>FTP [RFC 959]</td>
<td>TCP</td>
</tr>
<tr>
<td>streaming multimedia</td>
<td>RTSP or proprietary (e.g., RealNetworks)</td>
<td>TCP or UDP</td>
</tr>
<tr>
<td>Internet telephony</td>
<td>SIP, H.323, or proprietary (e.g., Dialpad)</td>
<td>typically UDP</td>
</tr>
</tbody>
</table>
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 overview (continued)

**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”**
- server maintains no information about past client requests

---

Aside

Protocols that maintain “state” are complex!
- past history (state) must be maintained
- if server/client crashes, their views of “state” may be inconsistent, must be reconciled
HTTP connections

**Nonpersistent HTTP**
- At most one object is sent over a TCP connection.
- HTTP/1.0 uses nonpersistent HTTP

**Persistent HTTP**
- Multiple objects can be sent over single TCP connection between client and server.
- HTTP/1.1 uses persistent connections in default mode

When you request a HTML file that contains links to 10 JPEG images, with non-persistent HTTP, after getting the HTML file, the connection is closed, and reopened again to obtain each JPEG image. But with persistent HTTP, the connection remains open until all linked objects are retrieved.
**HTTP request message**

- two types of HTTP messages: *request, response*
- HTTP request message:
  - ASCII (human-readable format)

```
GET /somedir/page.html HTTP/1.1
Host: www.someschool.edu
User-agent: Mozilla/4.0
Connection: close
Accept-language: fr
```

(extra carriage return, line feed)
HTTP request message: general format

- **Request Line**
  - **Method**
  - **URL**
  - **Version**

- **Header Lines**
  - **Header Field Name**
  - **Value**

- **Entity Body**
HTTP response message

HTTP/1.1 200 OK
Connection close
Date: Thu, 06 Aug 1998 12:00:15 GMT
Server: Apache/1.3.0 (Unix)
Last-Modified: Mon, 22 Jun 1998 ...
Content-Length: 6821
Content-Type: text/html

data data data data data data data ...

The last modified date allows “conditional GET” (see later)
HTTP response status codes

In first line in server->client response message.
A few sample codes:

200 OK
  ◦ request succeeded, requested object later in this message

301 Moved Permanently
  ◦ requested object moved, new location specified later in this message (Location:)

400 Bad Request
  ◦ request message not understood by server

404 Not Found
  ◦ requested document not found on this server

505 HTTP Version Not Supported
**Helper application (media player)**

- When the requested HTML file contains non-text objects, the files are first downloaded from the server to the client, then a corresponding helper application is launched to open the object. For multimedia objects (e.g., Mp3 sound, JPEG images, avi/mpeg video), the helper application is a media player (e.g., Windows MediaPlayer, RealPlayer).
Conditional GET: client-side caching

- **Goal:** don’t send object if client has up-to-date cached version
- **client:** specify date of cached copy in HTTP request
  
  ```
  If-modified-since: <date>
  ```
- **server:** response contains no object if cached copy is up-to-date:
  
  ```
  HTTP/1.0 304 Not Modified
  ```

- Large cache shortens response time, but also occupies more local disk space
Content distribution networks (CDNs)

- The content providers are the CDN customers.

**Content replication**

- CDN company installs hundreds of CDN servers throughout Internet
  - in lower-tier ISPs, close to users
- CDN replicates its customers’ content in CDN servers. When provider updates content, CDN updates servers

![Diagram of CDN distribution network]

- Origin server in North America
- CDN distribution node
- CDN server in S. America
- CDN server in Europe
- CDN server in Asia
More about CDNs

Routing requests
- CDN creates a “map”, indicating distances from leaf ISPs and CDN nodes
- When query arrives at authoritative DNS server:
  - Server determines ISP from which query originates
  - Uses “map” to determine best CDN server

Not just Web pages
- Streaming stored audio/video
- Streaming real-time audio/video
  - CDN nodes create application-layer overlay network
What you should know

- Application layer functionalities and protocols
  - Client-server model
  - Application layer protocol specifies client-server interactions (request and response)
  - Makes use of underlying transport protocol
- Web applications
  - Web browser and server in the client-server model
  - HTTP (uses TCP, stateless, request and response)
    - Why using client-side caching? What are the trade-offs when determining the cache size?
- Why would a content provider employ a CDN?
References

  - The book website (http://wps.aw.com/aw_kurose_2/) contains useful demos and links for further exploration.

- 3rd edition: