Lecture 4
Application Layer

Reading: Chapter 2
Chapter 1: Network Layers
• Network Layers

Chapter 2: Application layer
• Principles of network applications
• Web and HTTP
• FTP
• Electronic Mail
  • SMTP, POP3, IMAP
• DNS
• P2P Applications
• Socket programming with TCP
• Socket programming with UDP
Review Network Layers

• In Summary ....
  • Layers let us separate functionality, uses idea of modularity
    • Modules in Object oriented programs can be swapped
  • Network layers can be replaced when technology changes
    • Ex. Ethernet wires replaced by 802.11 Wireless
Chapter 2: Application Layer

Goals
• Look at network applications and protocols

Application Architecture
• Client-server
• Peer-to-peer
• Mix of Two

• Look at popular application-level protocols
  • HTTP
  • FTP or SFTP
  • SMTP / POP3 / IMAP
  • DNS
• Network applications
  • Use socket API
Popular Applications

• What network applications do you use?
Some Network Applications

- E-mail
- Web
- Instant messaging
- Remote login
  - PCAnywhere
- P2P file sharing
- Multi-user network games like WOW
- Streaming stored video clips
- Voice over IP
- Real-time video conferencing
- Grid computing
  - Seti-AT-Home
Application Architectures

• First, look at three most Popular
  • Client-server
  • Peer-to-peer (P2P)
  • Hybrid of client-server and P2P
Client-Server Architecture

Server
- Always-on machine
- Permanent IP address

Clients
- Communicate with server
- May be intermittently connected
- May have dynamic IP addresses
- Do not communicate directly with each other
Client Server Examples

- What are common Client-Server Applications?

Advantages and Disadvantages
Peer to Peer

• How about Peer to Peer?

Advantages and Disadvantages
Client Server
HTTP and the Web
Identify Structure of HTTP

• What type of architecture is HTTP?
• What Transport Protocol does HTTP use?
First Two Layers of Protocol Stack

- **Application**: Network applications
- FTP, SMTP, HTTP
- **Transport**: Process-process data transfer
- TCP, UDP

Applications depend on Transport Layer Which in turn depend on lower layers
What Exactly is HTTP?

- HTTP is the TCP/IP based application layer communication protocol
- Standardizes how browser **client and server** communicate with each other
- Defines how **content** is requested and transmitted across Internet
HTTP - Client/Server

HTTP Allows us to Manage Web Pages

• Web page consists of objects

• Object can be: HTML file, JPEG image, Java applet, Audio file, Video file...

• Web page consists of base HTML-file which includes several referenced objects

• Each object is addressable by a URL

• Example URL:

  www.someSchool.edu/Department/pic.gif

    host name           path name
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
• Three Versions HTTP
  • HTTP 1.0: RFC 1945 - 1996
  • HTTP 1.1: RFC 2068 – 1999
  • HTTP 2.0: RFC 7540 - 2015
HTTP overview (continued)

Uses TCP for transport protocol

1. Client initiates TCP connection
2. Creates connection to server
3. Server accepts TCP connection from client
4. HTTP messages exchanged between
   Browser (HTTP client) and
   Web server (HTTP server)
5. TCP connection closes
HTTP

HTTP is “Stateless”, What does that mean?

• Server maintains no information about past client requests
• Programs that maintain “state” remember past events and keep that information

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 Two Types of Connections

Nonpersistent HTTP
- At most one object is sent over a TCP connection
- HTTP/1.0 uses non-persistent 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
Nonpersistent HTTP

User enters URL
www.someSchool.edu/someDepartment/home.index

1a. HTTP client initiates TCP connection to HTTP server at www.someSchool.edu on port 80

2. HTTP client sends HTTP request message (containing URL) into TCP connection socket. Message indicates that client wants object: someDepartment/home.index

1b. HTTP server at host www.someSchool.edu waiting for TCP connection at port 80. “accepts” connection, notifying client

3. HTTP server receives request message, forms response message containing requested object, and sends it

(time contains text, references to 10 jpeg images)
Nonpersistent HTTP (cont.)

4. HTTP server closes TCP connection.

5. HTTP client receives response message containing html file, displays html.
   Parses html file, finds 10 referenced jpeg objects

6. Steps 1-5 repeated for each of 10 jpeg objects
Non-Persistent HTTP: Response time

How long does this take?

Definition of RTT:
Time to send small packet to travel from client to server and back
Non-Persistent HTTP: Response time

**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 RTT + File Transmission Time
Persistent HTTP

Nonpersistent HTTP
• Requires 2 RTTs per object
• Overhead for *each* TCP connection
• Uses HTTP header
  Connection: Close

Persistent HTTP
• Server leaves connection open after sending response
• Subsequent HTTP messages between same client/server sent over open connection
• Uses HTTP header
  Connection: Keep-Alive
Persistent vs. Non-persistent

Non-persistent

multiple connections

client

open

RTT

close

RTT

close

Extra RTT's for 3 objects

server

Persistent

persistent connection

client

open

RTT

close

RTT

close

One RTT for all objects

server
Real Data on Persistent vs. Non-Persistent HTTP

Wrote Two Scripts using Venmo API
One uses same connection for 50 consecutive requests and
One initiates new connection for every request
to collect 50 public transactions on Venmo using its API
Ran both scripts five times and the result was very interesting

Average time with persistent connections: 7.00 seconds
Average time with new connections: 22.38 seconds

Difference is almost 3 orders of magnitude
with keep-alive/persistent connections, since three way
handshake is avoided

https://blog.insightdatascience.com/learning-about-the-http-connection-
keep-alive-header-7ebe0efa209d
Real Data on Persistent vs. Non-Persistent HTTP

Non-Persistent HTTP

Persistent HTTP
HTTP/2 Is Going to Speed up the Web

The Next Generation of Web Is Here
HTTP 2.0 Improvements
Standard as of 2015

1. Simultaneous connections, or multiplexing
HTTP/1.1 can only fetch one resource at a time, HTTP/2 can fetch multiple resources over a single connection simultaneously, requests are interleaved
Done for speedup of delivery

2. Header optimization
Each and every request over HTTP contains header information
With HTTP/1.1, many of these headers are repeated in a single session
HTTP/2 removes redundant headers while compressing the remaining headers
HTTP 2.0 Improvements

3. Binary format

HTTP/1.1 sends data in a textual format, whereas HTTP/2 sends data in a binary format.
While binary can't be read, it reduces amount of errors and inefficiencies present in textual formats.

4. Server Push

HTTP/1.1, servers have to wait for the client to initiate a connection.
HTTP/2, servers can push resources to the client pro-actively.
Supposed to improve performance.

FAQ on Github
https://http2.github.io/faq/
Ultimate Guide for HTTP/2
https://kinsta.com/learn/what-is-http2/
HTTP Messages

- Two types of HTTP messages
  1. Request
  2. Response

More Complete Reference
http://www.w3.org/Protocols/rfc2616/rfc2616-sec14.html

Tutorial Reference

Easy Reference
http://www.jmarshall.com/easy/http/
HTTP Request

- A request line has three parts, separated by spaces:
  1. Method name,
  2. Local path of the requested resource,
  3. Version of HTTP being used

Example

```
GET /path/to/file/index.html HTTP/1.0
```

Method name – Uppercase GET
path – path on web server to file, file name
HTTP Version – Uppercase of form, HTTP/x.x
HTTP Response

• Initial response line, called the Status Line, also has three parts separated by spaces:
  1. HTTP version,
  2. Response status code, result of request
  3. Phrase describes status code

Example

HTTP/1.0  200    OK    or
HTTP/1.0  404    Not Found

HTTP version is same format as in request line, "HTTP/x.x"
Status code is meant to be computer-readable
“Reason phrase” is meant to be human-readable
Header Lines

• Header lines provide information about request, response, or object sent in message body
• Header lines in text readable format with one line per header, of the form:
  • "Header-Name: value", ending with CRLF
  • HTTP 1.0 defines 16 headers
    • None are required
  • HTTP 1.1 defines 46 headers, and one
    • Host: is required in requests
More Possible Header Lines

- **User-Agent**: header identifies program that's making the request, in the form
  - "Program-name/x.xx", where x.xx is the (mostly) alphanumeric version of the program
  - Example, Older Netscape sends header "User-agent: Mozilla/3.0Gold".

- These headers help Webmasters troubleshoot problems
- They also reveal information about the user
The Message Body

- An HTTP message may have a body of data sent after the header lines
  - **Response**, this is where requested resource is returned to client
  - **Request**, this is where user-entered data or uploaded files are sent to the server

**Content-Type**: Header    MIME-type of data in body,
Examples:    **text/html or image/gif**
(MIME = Multi-Purpose Internet Mail Extensions)

**Content-Length**: Header    Number of bytes returned
HTTP Request Message: General format

```
<table>
<thead>
<tr>
<th>method</th>
<th>sp</th>
<th>URL</th>
<th>sp</th>
<th>version</th>
<th>cr</th>
<th>lf</th>
</tr>
</thead>
<tbody>
<tr>
<td>header field name</td>
<td>:</td>
<td>value</td>
<td>cr</td>
<td>lf</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
```
Example of Uploading User Input

Post Method
• Web page often includes form input
• Input is uploaded to server in **entity body**

URL Method:
• Uses GET method
• Input is uploaded in URL field of request line:

www.somesite.com/animalsearch?monkeys&banana
Example of HTTP Post Form Request

http://csd4.cslabs.ewu.edu/helpdesk/

The HTTP commands submitted by the Browser:

POST /helpdesk/login.asp HTTP/1.1\nHost: csd4.cslabs.ewu.edu\nUser-Agent: Mozilla/5.0 (X11; U; Linux i686; en-US; rv:1.9.0.14) Gecko/2009090216 Ubuntu/8.04 (hardy) Firefox/3.0.14\nContent-Type: application/x-www-form-urlencoded\nContent-Length: 42
Request Method Types by HTTP Version

HTTP/1.0
- GET
- POST
- HEAD
  - Asks server to leave requested object out of response

HTTP/1.1 and HTTP/2
- GET, POST, HEAD
- PUT
  - Uploads file in entity body to path specified in URL field
- DELETE
  - Deletes file specified in the URL field
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:)

304 Not Modified
  Conditional Get or Heat request and resource not modified

400 Bad Request
  • request message not understood by server

404 Not Found
  • requested document not found on this server

505 HTTP Version Not Supported
HTTP Request Example

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 Response Example

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 ...

data, e.g., requested
HTML file

status line
(protocol
status code
status phrase)

header
lines
Trying out HTTP (client side) for yourself

1. Telnet to your favorite Web server:

   telnet cis.poly.edu 80
   \[\text{Opens TCP connection to port 80 (default HTTP server port) at cis.poly.edu. Anything typed in is sent to port 80 at cis.poly.edu}\]

2. Type a GET HTTP request:

   GET /~ross/ HTTP/1.1
   Host: cis.poly.edu
   \[\text{By typing this in (hit carriage return twice), you send this minimal (but complete) GET request to HTTP server}\]

3. Look at response message sent by HTTP server!
What is telnet?

• **How does telnet work?**
  - Lets you remotely log on as regular user with access to all data and programs installed on that computer

• Telnet client sends request to Telnet server
• **Server replies**
  • User name and password
• If accepted, Telnet client establishes connection to host, thus making your computer a virtual terminal

• Telnet client also establishes connection to text based servers which allows commands in Ascii

• **We can use telnet to test HTTP servers ...**
Let’s look at HTTP in action

• telnet example from penguin.ewu.edu

telnet penguin.ewu.edu 80

GET /cscd330/ HTTP/1.1
Host: penguin.ewu.edu

Note: This won't work anymore with HTTP/2
Cookies and First Conditional Gets
HTTP Caching

- HTTP allows Conditional Requests
  - Has to do with caching for speedup
  - Results dependent on value of a validator
  - Used to validate contents of a cache
  - All conditional headers check if resource stored on server matches a specific version
HTTP Caching

Two kinds of validators typically used

1. Date of last document modification,
   - `last-modified` date

2. A string, uniquely identifying each version,
   - **Entity tag** or `etag`

An ETag is an identifier for a specific version of a resource

If the resource at a given URL changes, a new Etag value is generated

Nice explanation of HTTP Conditional Requests and Caching

HTTP Caching, Conditional Headers

**If-Match**

Succeeds if ETag of distant resource is equal to one listed in this header. By default, unless etag is prefixed with 'W/', it performs a strong validation - byte by byte identical.

**If-None-Match**

Succeeds if ETag of distant resource is different to each listed in this header. By default, unless etag is prefixed with 'W/', it performs a strong validation.

**If-Modified-Since**

Succeeds if the Last-Modified date of the distant resource is more recent than the one given in this header.

**If-Unmodified-Since**

Succeeds if the Last-Modified date of the distant resource is older or the same than the one given in this header.

**If-Range**

Similar to If-Match, or If-Unmodified-Since, but can have only one single Etag, or one date. If it fails, the range request fails and, instead of a 206 Partial Content response, a 200 OK is sent with the complete resource.
HTTP Cache Example

Normal Get and document retrieval

Client                                Server

GET /doc HTTP/1.1

HTTP/1.1 200 OK
Last-Modified: date
Etag: “xyz”
HTTP Cache Example

Conditional Get and results from Server, not changed

Client                                Server

GET /doc HTTP/1.1
If-Modified-Since: *date*
If-None-Match: “xyz”

HTTP/1.1 304 Not Modified

Since document has not changed, client uses the cached document
More efficient than re-sending document
HTTP Cache Example

Conditional Get and results from Server, doc changed

Client                      Server
GET /doc HTTP/1.1
If-Modified-Since: date
If-None-Match: “xyz”

HTTP/1.1 200 OK
Last-Modified: date2
Etag: “xyz2”

Since document has changed, it is resent
HTTP

- Doesn't keep any state ...
- Each time you connect, it's like you are a completely new user, no persistent connections
- How do browsers deal with this?
  - Cookies ....
User State - Cookies

Many major Web sites use cookies

**Four components**

1) Cookie header line of HTTP *response* message
2) Cookie header line in HTTP *request* message
3) Cookie file kept on user’s host, managed by user’s browser
4) Back-end database at Web site

**Example**

- Susan always accesses Internet from one PC
- Visits e-commerce site for first time
- When initial HTTP requests arrives at site, site creates:
  - Unique ID
  - Entry in backend database for ID
User State - Cookies

http://en.wikipedia.org/wiki/HTTP_cookie

client

- eBay 8734
- Cookie file
- eBay 8734, Amazon 1678

One week later:

- eBay 8734
- Amazon 1678

server

- Usual HTTP request msg
- Amazon server creates ID 1678 for user
- Create entry
- Usual HTTP response msg
- Set-cookie: 1678

Access
- Backend database

Usual HTTP response msg
- Cookie: 1678

Usual HTTP request msg
- Cookie: 1678

Usual HTTP response msg
- Cookie: 1678

Usual HTTP response msg
- Cookie: 1678
HTTP Cookie Commands

1. HTTP Commands for Cookies
   GET /index.html HTTP/1.1
   Host: www.example.org

   ----->

   browser       server

2. HTTP/1.0 200 OK
   Content-type: text/html
   Set-Cookie: name=value
   Set-Cookie: name2=value2;
   Expires=Wed, 09 Jun 2021 10:18:14 GMT

   ---->

   browser       server

3. GET /spec.html HTTP/1.1
   Host: www.example.org
   Cookie: name=value;
          name2=value2
   Accept: */*

   <----

   browser       server
Cookies

What cookies do for you

- Authorization
- Shopping carts
- Recommendations
- User session state
  (Ex. Web e-mail)

How to keep “state”

- Protocol endpoints: maintain state at sender/receiver over multiple transactions
- Cookies: http messages carry state

Cookies and Privacy

- Cookies permit sites to learn a lot about you
- You supply name and e-mail to sites
How Cookies are Used

• Used to determine the number of visitors
  • Sites can determine:
    • How many visitors arrive
    • How many are new versus repeat visitors
    • How often a visitor has visited

• Stores User Preferences
  • Site can look different for each visitor
  • For example, if you visit msn.com, allows you to enter your zip code and get customized weather information ...

Nice page of cookie info

http://en.wikipedia.org/wiki/HTTP_cookie
Cookie Management

• The following are common options
  1. To enable or disable cookies completely
     Always accepted or  Always blocked
  2. Allow the user to see the cookies that are active
     with respect to a given page by typing
     \texttt{javascript:alert(document.cookie)}
     in browser URL field, Firefox use Shift-F4

• Most (all) browsers incorporate a cookie manager for
  the user to see and selectively delete the cookies
  currently stored in the browser
Browser Cookies

Two types of cookies are used

- **Session cookies,**
  - Temporary cookies remain in cookie file of your browser until you leave the site
  - These cookies only stored in memory

- **Persistent cookies,**
  - Remains in cookie file
  - Have an expiration date
Browser Cookies

http://en.wikipedia.org/wiki/HTTP_cookie

Each cookie has values for six fields
• Name - Name of the cookie
• Value - ID string set by Web site
• Domain - Of Web site issuing cookie
• Path - “/” means the cookie is valid anywhere on that domain
• Expires - Cookie expires on that date
• Send for – Any connection or secure connection

Example from my cookie stash ....
Privacy Problems with Cookies

• What are they? Or, is it not a problem?
  • Cookies Allow Targetting
    • Specific Web site, site can track not only your purchases
      • Pages you read, Ads you click etc.
      • Purchase something enter your name and address,
        • Site potentially knows more about you than a traditional mail order company
    • Cookies can track you cross multiple sites
      • DoubleClick most infamous example of this
      • Many companies use DoubleClick to serve banner ads on their sites.
        • DoubleClick puts (1x1 pixels) GIF files on site lets DoubleClick load cookies on your machine
        • Can track your movements across multiple sites!!
Cookie Management and Privacy

• Manage Your Cookies
  • Cookies can be managed to protect your privacy
  • Can delete them regularly when you close your browser
  • This site gives an excellent recommendation for managing cookies

http://www.allaboutcookies.org/manage-cookies/
Peer-to-Peer and Hybrids
What is a P2P Network?

- Define it ...
- A *distributed* network architecture may be called a P2P network if participants share a part of their own resources
- These *shared resources* are necessary to provide service offered by network
- Participants of such network are both
  1. Resource providers
  2. Consumers
What is P2P?

• Various definitions seem to agree on
  • Sharing of resources
  • Direct communication between equals (peers)
  • No centralized control
Client/Server Architecture

- Well known, powerful, reliable server is data source
- Clients request data from server
- Very successful model
  - WWW (HTTP), FTP, Web services, etc.
Client/Server Limitations

- Scalability is hard to achieve
- Presents single point of failure
- Requires administration
- Unused resources at network edge
  - Client machines

- P2P systems try to address these limitations
P2P Architecture

All nodes are both clients and servers
  Provide and consume data
  Any node can initiate a connection

No centralized data source
“The ultimate form of democracy on the Internet”

“The ultimate threat to copyright protection on the Internet”
P2P Network Characteristics

- Clients are also servers and routers
  
  Nodes contribute **content, storage, memory, CPU**
- Nodes are **autonomous** (no administrative authority)
- Network is **dynamic**: nodes enter and leave the network “frequently”
- Nodes **collaborate directly** with each other (not through well-known servers)
- Nodes have widely **varying capabilities**
P2P Goals and Benefits

- **Efficient use of resources**
  - Unused bandwidth, storage, processing power at the “edge of the network”

- **Scalability**
  - No central information, communication and computation bottleneck
  - Aggregate resources grow naturally with utilization

- **Reliability**
  - Replicas
  - Geographic distribution
  - No single point of failure
P2P Goals and Benefits

- **Ease of administration**
  - Nodes self-organize
  - Built-in fault tolerance, replication, and load balancing
  - Increased autonomy

- **Anonymity – Privacy**
  - not easy in a centralized system

- **Dynamism**
  - Highly dynamic environment
  - Ad-hoc communication and collaboration
Hybrid P2P

• First attempt at P2P
  • Napster
    • 1999 redefined Internet, music industry and way we all think about intellectual property
    • It allowed people to share MP3 songs with each other ... OMG!
    • First major implementation of P2P network
    • Used central Database to keep track of songs
Napster Steps

When you look for Song

1. **Open Napster**, Napster checked for an Internet connection

2. If it found a connection, Napster logged you onto central server. **Purpose** - kept an index of all Napster users currently online and connect them to each other. **It does not contain any of the MP3 files**

3. Type in title or artist of song you are looking for

4. Napster utility on your computer queries index server for other Napster computers online that have song you requested
Napster Steps

5. When found match, Napster server informs your computer where to find requested file, Napster shows list of systems

6. You click file(s) that interest you and then Download

7. Your copy of Napster attempts to establish a connection with system hosting file you selected, a connection is successfully made, file begins downloading
Napster Architecture

• Server plus peers architecture
Napster

• Problem music industry had with Napster
  1. Automated way to copy copyrighted material
  2. Thousands of people were, making thousands of copies of copyrighted songs,
     • Neither music industry nor artists got any money in return for those copies
• However, people argued that Audio Home Recording Act of 1992 allowed buyer of CD or cassette
  • Right to not only make a copy for their own personal use, but also to make copies for friends’ if original owner is not selling copies
Next P2P Architecture
Pure P2P

• Napster's trouble with legal issues prompted a re-design of P2P
  • If there is a centralized database of songs, can find the owner/administrator
  • If no database of songs, but millions of Peer/Clients all sharing with each other .... much harder to shut down
• Thus, Pure P2P was born
• An example of pure P2P?
  • Gnutella
“Pure” P2P Architecture

- No server
- Arbitrary end systems directly communicate
- Peers are intermittently connected and change IP addresses
- Example: Gnutella

Highly scalable but can be difficult to manage

How do peers find each other?
How do they exchange file information?
Gnutella

• Two big differences between Gnutella and older Napster

  1. No central database that knows all files available on Gnutella
     • All machines tell each other about available files using a distributed query approach

  2. Many client applications available to access the Gnutella network
Pure P2P - Gnutella

• Some of the Popular Gnutella Clients
  BearShare
  Gnucleus
  LimeWire
  Morpheus
  WinMX
  XoloX

"Gnutella" today isn't a single project or piece of software, but open protocol used by various clients
Pure P2P - Gnutella

- Type in the name of the song
- Your machine knows of at least one other Gnutella machine somewhere on the network
  - Machines search to see if file is on local hard disk
  - If so, they send back file name and IP address to requester
  - At the same time, these machines send same request to machines they are connected to, process repeats.
  - Request has a TTL (time to live) limit placed on it
  - Request might go out six or seven levels deep before it stops propagating
Pure P2P - Gnutella

• Advantages or Disadvantages of this Approach?

• Disadvantages
  • No guarantee that file you want is on any of 8,000 machines you can reach
  • Queries for files can take some time to get a complete response
  • Your machine is part of this network, give up some of your bandwidth to handle requests from all the other users

• Advantages
  • Record industry can't easily shut down file trades
  • Pure P2P, no centralized server to go down
Hybrid of Client-server / P2P

Skype

- Voice-over-IP - P2P application
- Centralized server
  Stores address of remote party
- Client-client connection
  Direct (not through server)
Hybrid of Client-server / P2P

**Instant Messaging**

• Chatting between two users is P2P, but uses Directory Server to log-in and authenticate

  • User registers IP address with central server, provides user name
  • Central server authenticates user and provides contact list
Reading: Still Chapter 2
Lab this week, Wireshark

End