Application Layer - Net Apps

Some common network applications

- Web servers (apache, nginx), web browsers (firefox, chrome, internet explorer)
- telnet (terminal communication between hosts), ssh (secure terminal communication between hosts)
- Email servers and clients
- Chat servers and clients
- Bit torrent, file sharing, P2P
- Audio/video streaming
- Many others...

Common Features in Network Apps

Focus is on Application Layer

Application Layer Features

- Processes communicating with other processes on or between hosts
- Utilize services of the Transport Layer
- Exchange messages and data via protocols

Common architectures found with Application Layer programs

- Client-Server communication
- Peer to Peer (P2P) communication
- Hybrids

Client-Server Architecture

Distinquishing characteristics:

Server

- Process that runs on remote machine
- Utilizes well-known (or known) IP address and port number
- May run forever, waiting for clients to connect to service

Client

- Process that runs on a local host
- Contacts servers at well known IP addresses and port numbers
- Comes and goes as needed

Choices in Network Applications

When developing network applications, what's important to consider?

- Do you need reliable data transfer?
- What are your throughput requirements?
- Does your application have any timing requirements?
- Do you need security, confidentiality, or authentication?

Application	Data Reliability	Bandwidth	Latency Concerns
file transfer	yes	variable	none
email	yes	variable	none
web	yes	variable	none
video conf	some loss ok	10kbps - 5Mbps	yes
video games	some loss ok	1 - 10kbps	yes

Recall services of the Internet!

Internet has a best-effort delivery service with NO guarantees!

Best-effort Service with Reliability

While service is best-effort, Transport Layer protocols provide additional services to developers!

- TCP Service - Transmission Control Protocol
  - Connection oriented service that provides reliable data transfer.
  - Acts a bit like virtual circuit - connection between client and server must be established first and torn down after communication happens
  - Also provides flow control and data congestion algorithms to *fairly* share bandwidth
- UDP Service - User Datagram Protocol
  - Lightweight transport protocol
  - Connectionless and unreliable - best-effort delivery service

Apps and Protocols

What transport layer protocol are used by the Application Layer:

Application	**Protocol**	**Transport Layer Protocol**
email	STMP [RFC 2821]	TCP
remote terminal	Telnet [RFC 854]	TCP
web	HTTP [RFC 2616]	TCP
streaming media	HTTP, RTP, etc...	TCP and/or UDP
audio/video conf	SIP, RTP, etc...	mostly UDP

UDP is often chosen to help reduce latency and the overhead associated with reliable data transfer!

What is a Protocol

Let's start by looking at Application Layer protocols

Protocol

A well-defined set of rules, syntax, messages, along with a means to pass information between two (or more) hosts.

Simple real-world example:

- Greeting someone you meet (Hello, How are you, What's up?, etc...)
- Handshakes between people

Simple Example

Protocols can take on many different forms:

- Humans speak protocols or use gestures
- Machines send character strings
- Machines also might send well defined sequences of bits (or bytes)

Examples:

Time Server

- To request time, send "REQUEST TIME" to Port 9045 at IP 131.212.102.101
- Response from time server will be: "CURRENT TIME: <hour> <minute> <second>"

Time Server Continued:

Time Server

- To request time, send "REQUEST TIME" to Port 9045 at IP 131.212.102.101
- Response from time server will be: "CURRENT TIME: <hour> <minute> <second>"

```
Client: sending ``REQUEST TIME'' to 131.212.102.101:9045
Server: response ``CURRENT TIME: 14 43 23''

Client: sending ``REQUEST TIME'' to 131.212.102.101:9045
Server: response ``CURRENT TIME: 14 54 03''
```

Let's look at HTTP

- Hyper-Text Transfer Protocol (HTTP) - protocol for transferring file data between different hosts
- Defined in RFC 1945 and RFC 2616 among a few others
- Uses TCP for reliable data transfer
- Works on well-known port 80
- Uses persistent and non-persistent connections
- Can pipeline requests

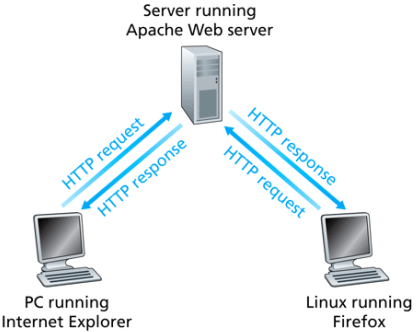


Figure 2.6 ♦ HTTP request-response behavior

HTTP

- Uses TCP, so must perform TCP 3-way handshake
- Notion of Round-Trip Time (RTT) plays into how fast you get your data
- Non-persistent connection
  - Establish multiple TCP connections for each request
- Persistent connection
  - Uses original TCP connections for all request
  - Can pipeline requests in this mode

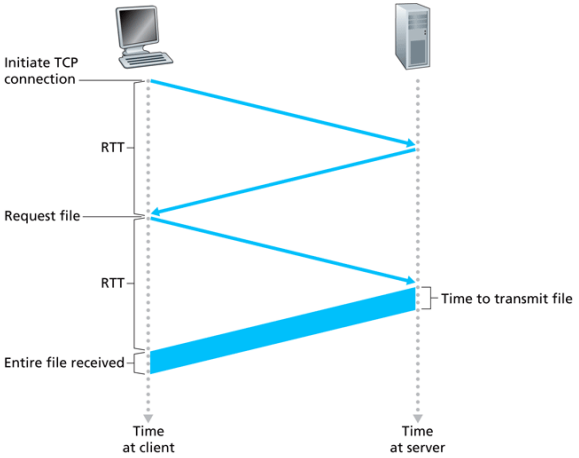


Figure 2.7 ♦ Back-of-the-envelope calculation for the time needed to request and receive an HTML file

HTTP

- HTTP is based on REQUESTS and RESPONSES
- HTTP Request Message
- Typical requests: GET. POST, HEAD, PUT, DELETE

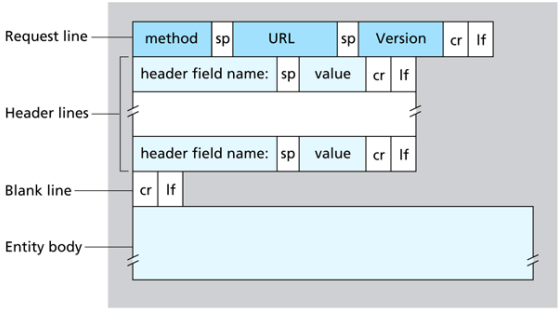


Figure 2.8 ♦ General format of an HTTP request message

GET Method HTTP Request

GET Request

```
GET <file> HTTP/1.1
HOST: <host address>
Connection: Close
User-Agent: Mozilla/4.0
Accept-language: en
```

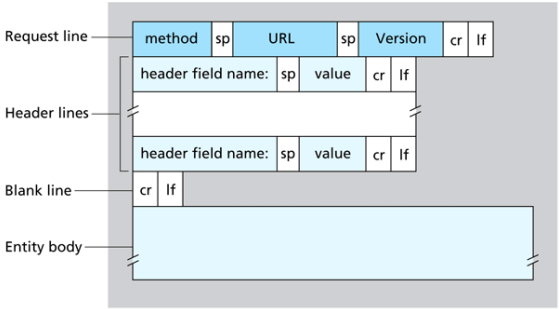


Figure 2.8 ♦ General format of an HTTP request message

HTTP Response Message

- What is returned from a request

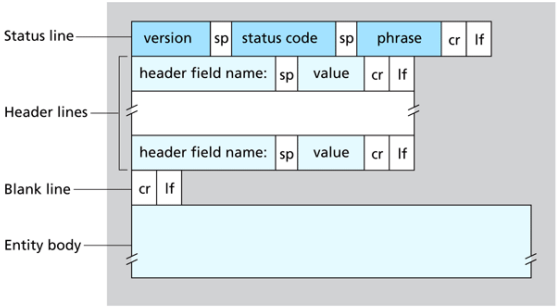


Figure 2.9 ♦ General format of an HTTP response message

Let's access HTTP!

Use *telnet* to access the HTTP port on web servers.

- telnet is a communication protocol for sending characters to a remote terminal. It runs on port 23 by default.
- However, you can have it access a different port! For HTTP, that is port 80.

- telnet is a communication protocol for sending characters to a remote terminal. It runs on port 23 by default.
- However, you can have it access a different port! For HTTP, that is port 80.

Run telnet on your machine to access UMD's Webserver:

```
telnet www.d.umn.edu 80
```

Once connected, send a GET message:

```
GET /index.php HTTP/1.1
HOST: www.d.umn.edu
<CRLF>
```

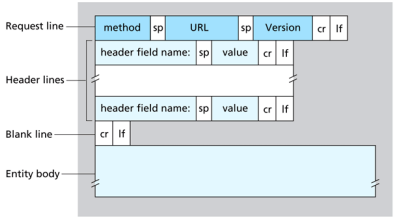


Figure 2.8 ♦ General format of an HTTP request message

HTTP Response Error Codes

Possible Error codes returned from a Request

- 200: OK
- 301: Moved Permanently
- 400: Bad Request
- 404: Not Found
- 505: HTTP version not supported

GET Request

```
GET <file> HTTP/1.1
HOST: <host address>
Connection: Close
User-Agent: Mozilla/4.0
Accept-language: en
```

HTTP Continued

- HTTP is stateless! Does not keep state about connections on server.
- State can be added with *cookies*
- Cookies are sent and received in the headers of the Response and Request messages of HTTP
- HTTP Response: `Set-cookie: <ID>`
- HTTP Request: `cookie: <ID>`

Find cookies in your own telnet based HTTP queries!

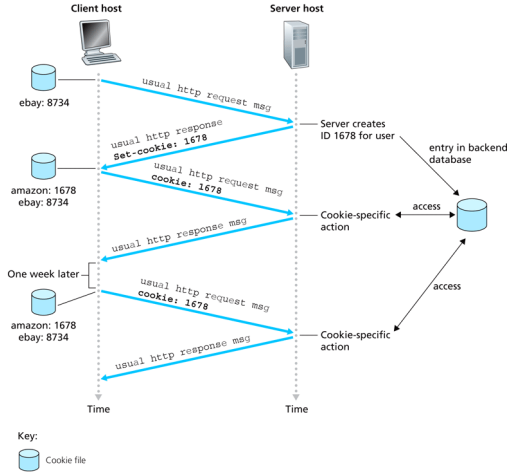
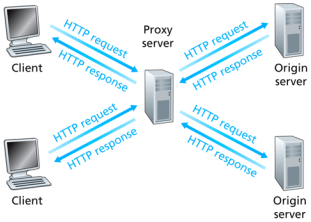


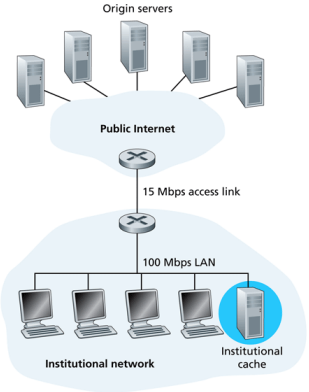
Figure 2.10 ♦ Keeping user state with cookies

Central Content vs. Distributed Content

Web Cache - Distribute data to make access local

- Attempts to maintain local copies of HTTP-based data
- Instead of acquiring data from original server, get it from cache
- Data is fetched and stored in cache on demand





How Web Cache is Maintained with HTTP

Conditional GET

- How does local HTTP server make sure data in cache is relevant and timely?
- HTTP Request uses GET with optional `If-Modified-Since` header

`If-Modified-Since`

```
GET /index.php HTTP/1.1
Host: www.d.umn.edu
If-modified-since: Wed, 4 Jan 2012 09:00:30
```

- This relates to the `Last-Modified` header in the Response!
- Try it out!

Another useful HTTP header: Range

`Range:`

- Allows client to request specific byte ranges from the server
- Useful because it clients can suspend data transfer and pick up where they left off!

Set of Bytes

```
GET /index.php HTTP/1.1
Host: www.d.umn.edu
Range: bytes=0-12
```

Ranges of Bytes

```
GET /index.php HTTP/1.1
Host: www.d.umn.edu
Range: bytes=0-212,290-300
```

Review - HTTP

By this point, you should be starting to feel comfortable with the following:

- telnet - access open ports on hosts using different protocols
- HTTP - basic GET request and response messages to access and transfer files between web servers and web clients
- DNS - general working at least in terms of the questions, but not what is happening behind the scenes.

Today, we will

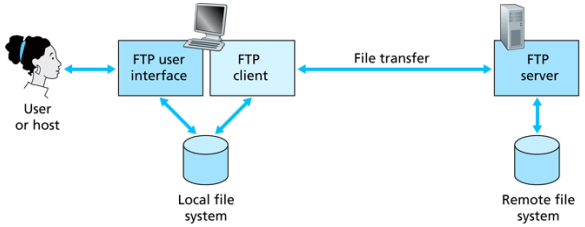
- Cover a couple additional protocols
- Start into the Socket API

Protocol Case Study - FTP

FTP

File Transfer Protocol - insecure file transfer system that predates HTTP

- Details found in RFC 959
- Uses 2 parallel TCP connections
- Control information is sent out of band (meaning separate from data transfer)
- Port 21 is used for control commands
- Port 20 is used for data transfer



Protocol Case Study - FTP

ASCII protocol (try telnet'ing to port 21 on an ftp server)

- `USER <username>`, for example `USER anonymous`
- `PASS <password>`, for example `PASS myemail@mydomain.com`
- `LIST`, ask the server to send a list of files (which goes to the data connection port)
- `RETR <filename>`, retrives a specific file to the data connection
- `STOR <filename>`, pushes a local file to the FTP server

Active FTP mode can determine the data port, which may not be the default port 20.

- `PORT h1,h2,h3,h4,p1,p1` - Active FTP which sets up the data port as a function of `p1,p2` to host `h1.h2.h3.h4`
- Data connection port $= p1 * 256 + p2$
- Note that the client must listen for the server! Server will connect to client on the port to send the data! We'll find out later this isn't so great for firewalls and NATs!

Passive FTP

- `PASV` - Passive FTP which tells server to listen for client's data connection to the server

Protocol Case Study - FTP

Example

1. `telnet ftp.kernel.org 21`
2. `USER anonymous`
3. `PASS <password>`
4. `PASV`
5. `LIST`

Meanwhile, in another terminal after Step 5 above:

1. `telnet h1.h2.h3.h4 <p1 * 265 + p2>`

Did you get the file list?

Protocol Case Study - E-Mail

Various protocols associated with email:

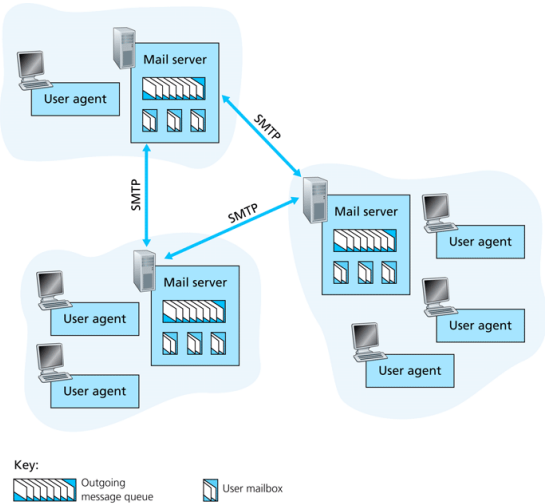
- SMTP
- POP
- IMAP

Based on idea of User Agents and Mail Servers interacting!

SMTP

Simple Mail Transfer Protocol

- RFC 2821 (for starters)
- Originally, port 25 for everyone, but much more restricted now



Protocol Case Study - E-Mail SMTP

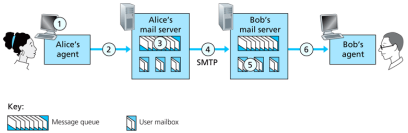


Figure 2.17 ♦ Alice sends a message to Bob

Basic interaction:

- HELO, MAIL FROM, RCPT TO, DATA

```
nomad41-249:~ willemsn$ telnet 131.212.41.105 25
Trying 131.212.41.105...
Connected to umd41-105.d.umn.edu.
Escape character is '^]'.
220 csnetlab.d.umn.edu ESMTP Postfix (Ubuntu)
HELO csnetlab.d.umn.edu
250 csnetlab.d.umn.edu
MAIL FROM: <willemsn@d.umn.edu>
250 2.1.0 Ok
RCPT TO: umdcs@localhost
250 2.1.5 Ok
DATA
354 End data with <CR><LF>.<CR><LF>
Hello. How are you.
.
250 2.0.0 Ok: queued as D99A03FA1A0
QUIT
221 2.0.0 Bye
Connection closed by foreign host
```

Protocol Case Study - E-Mail POP and IMAP

Getting your email

- Various application layer protocols can help here, including HTTP
- POP3 - Post Office Protocol, Version 3 (RFC 1939)
  - Simple mail access
  - Authorize, transact, update cycles of interaction
  - Also, download and delete common with POP
  - Port 110
- IMAP - Internet Mail Access Protocol (RFC 3501)
  - Fine grained access to mail
  - Mail folder state preserveed

Protocol Case Study - E-Mail POP

Basic interaction:

- `user`, `pass`, `list`, `retr <N>`, `dele <N>`

```
nomad41-249:~ willemsn$ telnet 131.212.41.105 110
Trying 131.212.41.105...
Connected to umd41-105.d.umn.edu.
Escape character is '^]'.
220 csnetlab.d.umn.edu ESMTP Postfix (Ubuntu)
+OK POP3 server ready
user bob
+OK
pass hungry
+OK user successfully logged on
list
1 498
2 912
.
retr 1
<message 1 contents>
.
dele 1
retr 2
<message 1 contents>
.
dele 2
quit
+OK POP3 server signing off
```