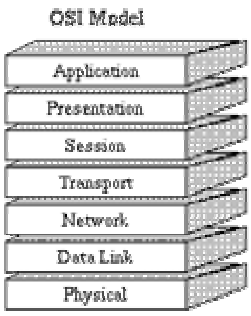
CSC 551: Web Programming
Spring 2004

Internet & World Wide Web Protocols

- network layers
- TCP/IP
  - ➢ domain name system, IP addresses, routing protocols
- HTTP
  - ➢ GET/POST, headers, caching, cookies

OSI 7-layer model

in the 70's, computer networks were ad hoc, vendor-specific



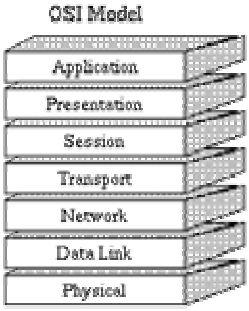
Open Systems Interconnection model

- developed by the ISO in 1984
- provides an abstract model of networking
- divides the tasks involved in moving info between networked computers into 7 task groups
- each task group is assigned a layer

Each layer is reasonably self-contained, so

- can be implemented independently
- changes/updates to a layer need not effect other layers

Protocol layers



Application layer
- describes how applications will communicate
  e.g., HTTP, FTP, Telnet, SMTP

Presentation layer
- describes the form of data being transferred & ensures that it will be readable by receiver
  e.g., floating point formats, data compression, encryption

Session layer
- describes the organization of large data sequences & manages communication session
  e.g., coordinates requests/responses

Transport layer
- describes the quality and nature of data delivery
  e.g., how retransmissions are used to ensure delivery

Network layer
- describes how a series of exchanges over various data links can deliver data across a network
  e.g., addressing and routing

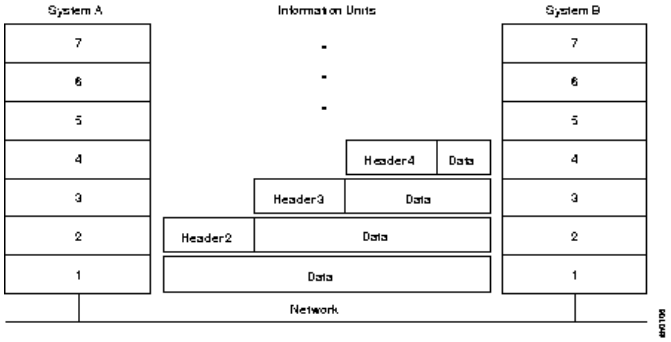
Data Link layer
- describes the logical organization of data bits transmitted on a particular medium
  e.g., frame sequencing, error notification

Physical layer:
- describes the physical & electrical properties of the communications media
  e.g., voltage levels, data rates, max distances

Layer protocols

across the network, processes at the same level can (seemingly) communicate
- e.g., Web server & browser run at the application level, communicate via HTTP

in reality, actual communication takes place at the physical layer
- upper layers can only communicate with those above and below
- *at the source*, as data is passed down the layers:
  the protocol for each layer adds control information to the data
- *at the destination*, as data is passed up the layers:
  the protocol for each layer strips and analyzes the control information for that layer



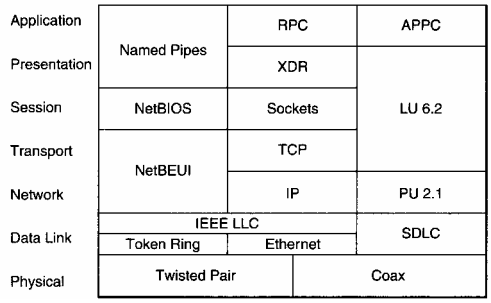
Internet protocol suite

Network layer: Internet Protocol (IP)

- provides generalized packet network interface
- handles routing through the Internet
- connectionless and unreliable

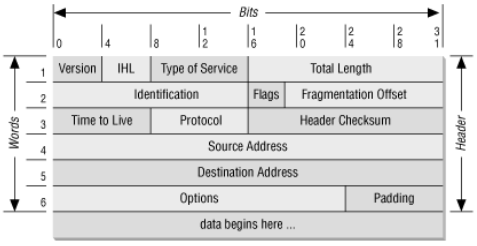
Transport layer: Transmission Control Protocol (TCP)

- provides a virtual circuit over which two processes can communicate
- supplies logic to give reliable, connection-oriented session
- FTP (file transfer) and HTTP are built on top of TCP



Internet Protocol (IP)

the IP protocol adds packet routing info (20 bytes)



Time-to-live (TTL):
indicates number of hops packet is allowed to take before being discarded

Source address:
IP address of host sending the packet

Destination address:
IP address of host to receive the packet

Options:
options such as sender-specified routing or security

IP addresses

IP addresses are 32 bits long

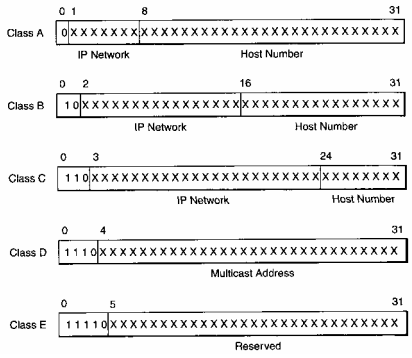
10010011 10000110 00000010 00010100

↓ *written as a dotted sequence*

147.134.2.20

divided into 5 classes

- class A: start with 0, then 7-bit code
  2^{24} = 16,777,216 hosts in subnetwork
- class B: start with 10, then 14-bit code
  2^{16} = 65,536 hosts in subnetwork
- class C: start with 110, then 21-bit code
  2^{8} = 256 hosts in subnetwork
- class D: start with 1110
  used for multicasting
- class E: start with 11110
  reserved for future use



IPv6 extends address size to 128 bits

- extensions support authentication, data integrity, confidentiality

Domain name system

rarely do applications deal directly with IP addresses

- a hierarchical system of domain names can be used instead
- top level domains: edu, com, gov, org, net, …

commonly: *hostname.subdomain.domain* (possibly many subdomains)
e.g., `bluejay.creighton.edu`

a domain name server (DNS) is a machine that keeps a table of names and corresponding IP addresses

- there are 13 root servers in the world (mirrored)
- when an application specifies a host name,
  - ➢ go to local domain name server and try lookup
  - ➢ if not stored there, then local DNS requests address from a root server
  - ➢ root server determines appropriate name server & forwards request

Routing protocols

routers (or *gateways*) are special purpose machines on the Internet that determine the path for packets from source to destination

- when a router receives a packet, inspects the destination address
- looks up that address in a routing table
- based on the contents of the table, forwards the packet to another router (or to its final destination if possible)

Routing Information Protocol (RIP)

- describes how routers exchange routing table information
- uses hop-count as the metric of a path's cost

Open Shortest Path First Protocol (OSPF)

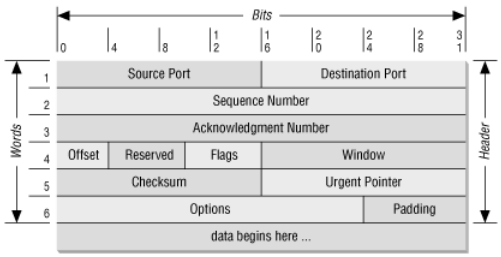
- more robust, scalable protocol than RIP
- doesn't exchange entire tables, only updates changed links

Internet Control Message Protocol (ICMP)

- adjunct to IP, notifies sender (or other router) of abnormal events
  e.g., unreachable host, net congestion

Transmission Control Protocol (TCP)

the TCP protocol adds info for providing a virtual circuit, including message formatting, circuit management, flow control, error correction



Source & destination ports
a port is analogous to a mailbox

Sequence number:
identifies its place in sequence
(byte # in overall message)

Acknowledgement number:
specifies the next byte # in sequence,
if destination does not receive it in X
amount of time, will notify sender

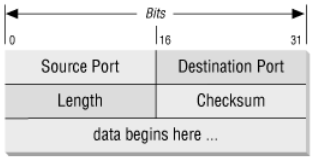
Control flags:
used to set up connection (3-way handshake: request, ack, ack),
mark as urgent, terminate connection, …

User Datagram Protocol (UDP)

UDP protocol is a simple, connectionless alternative to TCP

used in many Internet applications that require only simple query/response

e.g., `time`



Source & destination ports
same as in TCP

Length:
number of bytes in the packet

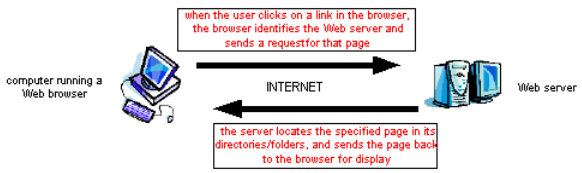
Checksum:
rudimentary error detection

World Wide Web

the Web is the world's largest client/server system

communication occurs via message passing

- within browser, select URL of desired page
- browser requests page from server
- server responds with message containing
  - type of page (HTML, gif, pdf, zip, …)
  - page contents
- browser uses type info to correctly display page
- if page contains other items (images, applets, …),
  browser must request each separately



HTTP

Hypertext Transfer Protocol (HTTP):

application-level protocol for distributed, collaborative, hypermedia information systems

- generic, stateless, object-oriented
- can be used for many tasks, such as name servers & distributed object management systems
- underlying language of the Web

HTTP/1.0 allowed only connectionless message passing

- each request/response required a new connection
- to download a page with images required multiple connections
  can overload the server, require lots of overhead

HTTP/1.1 provides persistent connection by default

- once client & server connect, remains open until told to close it *(or timeout)*
  reduces number of connections, saves overhead
- client can send multiple requests without waiting for responses
  e.g., can request all images in a page at once

GET request

most URL's have the form: `protocol://serverName URI`

e.g., `http://www.creighton.edu/~davereed/index.html`

to retrieve a document via HTTP from the server, issue a GET request

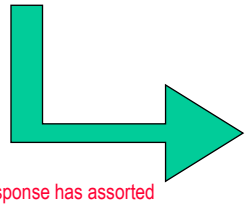
```
GET URI HTTP/1.1
Host: serverName
```

Web server only knows the contents of the GET request message

- automatically generated by browser when you select a URL
- could also come from a link checker, a search engine robot, …
- can come directly from a telnet connection using port 80

GET example

```
bluejay> telnet www.creighton.edu 80
Trying...
Connected to swift.creighton.edu.
Escape character is '^]'.
GET /~davereed/index.html HTTP/1.1
Host: www.creighton.edu
```



server response has assorted header information, followed by the page

```
HTTP/1.1 200 OK
Date: Thu, 22 Jan 2004 18:35:24 GMT
Server: Apache/1.3.27 (Unix) PHP/4.1.2
Last-Modified: Tue, 13 Jan 2004 17:38:00 GMT
ETag: "155005-1a4-40042cf8"
Accept-Ranges: bytes
Content-Length: 420
Content-Type: text/html

<HTML><HEAD><TITLE>Dave Reed's Home Page</TITLE>

<!--- Dave Reed          index.html        10/5/00 -->
<!------------------------------------------------->

<SCRIPT language=JavaScript>
     if (self!=top) top.location.href=self.location.href;
</SCRIPT>

</HEAD>

<FRAMESET border=0 cols="175,*">
  <FRAME  name="menu" src="menu.html" />
  <FRAME name="main" src="info.html" />
</FRAMESET>

</HTML>
```


Response header fields

the first line of the server's response contains a status code

- 200 OK — request was processed successfully
- 301 Moved permanently — document has been moved
- 304 Not modified — if cached version is up-to-date
- 400 Bad request — syntax error in client's request
- 403 Forbidden — client is not allowed access (e.g., protected)
- 404 Not found — file could not be found
- 500 Internal server error — server failed
- 503 Service unavailable — server is overloaded

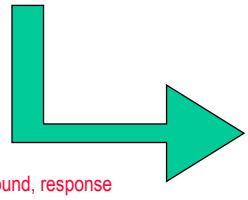
Other response header fields

in addition to the status code, the server's response may include

- Date — response time (in GMT)
- Server — identification info on the server
- Last-modified — time document was last changed (in GMT)
- Content-length — size of document, in bytes
- Content-type — file format (e.g., html, gif, pdf)
- Expires — prevents browser from caching beyond date

File not found

```
bluejay> telnet www.creighton.edu 80
Trying...
Connected to swift.creighton.edu.
Escape character is '^]'.
GET /~davereed/foo.html HTTP/1.1
Host: www.creighton.edu
```

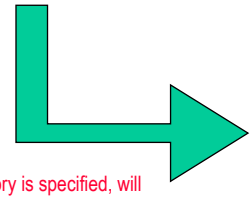


```
HTTP/1.1 404 Not Found
Date: Thu, 22 Jan 2004 18:37:29 GMT
Server: Apache/1.3.27 (Unix) PHP/4.1.2
Content-Type: text/html

<head>
<META HTTP-EQUIV=refresh
    CONTENT="50;URL=http://www.creighton.edu/">
<title>Requested Page Not Found!</title>
</head>
<body bgcolor=white>
<font face="Arial">
<h1>Requested Page Not Found!</h1>
<hr>

<p><b>
The URL you requested was not found on <a
href="http://www.creighton.edu">this server</a>. (Error 404)
</p>
<p>
.
.
.
```

if file not found, response includes 404 status code and generic error page

Directories as URI's

```
bluejay> telnet www.creighton.edu 80
Trying...
Connected to swift.creighton.edu.
Escape character is '^]'.
GET /~davereed/ HTTP/1.1
Host: www.creighton.edu
```

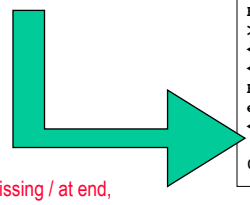


if a directory is specified, will look for a file named `index.html`

```
HTTP/1.1 200 OK
Date: Thu, 22 Jan 2004 18:41:17 GMT
Server: Apache/1.3.27 (Unix) PHP/4.1.2
Last-Modified: Tue, 13 Jan 2004 17:38:00 GMT
ETag: "155005-1a4-40042cf8"
Accept-Ranges: bytes
Content-Length: 420
Content-Type: text/html

<HTML><HEAD><TITLE>Dave Reed's Home Page</TITLE>

<!--- Dave Reed          index.html        10/5/00 -->
<!-------------------------------------------------->

<SCRIPT language=JavaScript>
     if (self!=top) top.location.href=self.location.href;
</SCRIPT>

</HEAD>

<FRAMESET border=0 cols="175,*">
  <FRAME  name="menu" src="menu.html" />
  <FRAME name="main" src="info.html" />
</FRAMESET>

</HTML>
```


Redirection

```
bluejay> telnet www.creighton.edu 80
Trying...
Connected to swift.creighton.edu.
Escape character is '^]'.
GET /~davereed HTTP/1.1
Host: www.creighton.edu
```



since URI is missing / at end, browser must do 2 requests

```
HTTP/1.1 301 Moved Permanently
Date: Thu, 22 Jan 2004 18:42:53 GMT
Server: Apache/1.3.27 (Unix) PHP/4.1.2
Location: http://www.creighton.edu/~davereed/
Transfer-Encoding: chunked
Content-Type: text/html; charset=iso-8859-1

166
<!DOCTYPE HTML PUBLIC "-//IETF//DTD HTML 2.0//EN">
<HTML><HEAD>
<TITLE>301 Moved Permanently</TITLE>
</HEAD><BODY>
<H1>Moved Permanently</H1>
The document has moved <A
HREF="http://www.creighton.edu/~davereed/">here</A>.<P
>
<HR>
<ADDRESS>Apache/1.3.27 Server at <A
HREF="mailto:webmaster@creighton.edu">www.cr
eighton.edu</A> Port 80</ADDRESS>
</BODY></HTML>

0
```

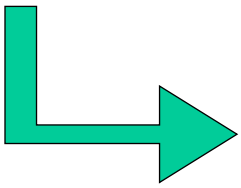
Request header fields

the client can specify additional information in the request

- User-Agent specifies the browser version
- Referer — tells server where the user came from
  useful for logging and customer tracking
- From — contains email address of user
  generally not used for privacy reasons
- Authorization — can send username & password
  used with documents that require authorization
- If-Modified-Since — only send document if newer than specified date
  used for caching

Conditional GET

```
bluejay> telnet www.creighton.edu 80
Trying...
Connected to swift.creighton.edu.
Escape character is '^]'.
GET /~davereed/ HTTP/1.1
Host: www.creighton.edu
If-Modified-Since: Wed, 21 Jan 2004 14:00:00 GMT
```



```
HTTP/1.1 304 Not Modified
Date: Thu, 22 Jan 2004 18:45:25 GMT
Server: Apache/1.3.27 (Unix) PHP/4.1.2
ETag: "155005-1a4-40042cf8"
```

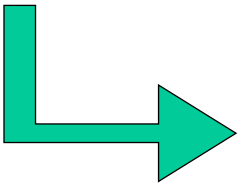
since the document has not been modified since the specified date, the page is not sent by the server (status code 304)

Other request methods

HEAD	similar to GET, but requests header information only *useful for checking to see if a document exists, how recent*
POST	similar to GET, but encodes inputs differently *useful for submitting form contents to a CGI program*
PUT	upload a document to the server *new in HTTP/1.1*
DELETE	delete a document from the server *new in HTTP/1.1*

HEAD example

```
bluejay> telnet www.creighton.edu 80
Trying...
Connected to swift.creighton.edu.
Escape character is '^]'.
HEAD /~davereed/index.html HTTP/1.1
Host: www.creighton.edu
```



server does not send the page, only the header information

```
HTTP/1.1 200 OK
Date: Thu, 22 Jan 2004 18:47:16 GMT
Server: Apache/1.3.27 (Unix) PHP/4.1.2
Last-Modified: Tue, 13 Jan 2004 17:38:00 GMT
ETag: "155005-1a4-40042cf8"
Accept-Ranges: bytes
Content-Length: 420
Content-Type: text/html
```

Caching

browsers cache pages to save downloading

- maintain temporary storage (cache) for recent pages
- when a page is requested, check to see if already in cache
- if not in the cache, issue GET request
  - when response message arrives,
    - display page and store in cache (along with header info)
- if already stored in the cache, send GET request with If-Modified-Since header set to the data of the cached page
  - when response message arrives,
    - if status code 200, then display and store in cache
    - if status code 304, then display cached version instead

Cookies

HTTP message passing is transaction-based, stateless

- many e-commerce apps require persistent memory of customer interactions

e.g., amazon.com
remembers your name, credit card, past purchases, interests

Netscape's solution: cookies

- a *cookie* is a collection of information about the user
- server can download a cookie to the client's machine using the "Set-cookie" header in a response

```
Set-cookie: CUSTOMER=Dave_Reed; PATH=/; EXPIRES=Thursday, 29-Jan-04 12:00:00
```

- when user returns to URL on the specified path, the browser returns the cookie data as part of its request

```
Cookie: CUSTOMER=Dave_Reed
```