HTTP and HTML

Internet Technologies

2009.9.29
Application layer

• learn about protocols by examining popular application-level protocols
  – HTTP
  – FTP
  – SMTP / POP3 / IMAP
  – DNS
  – .... etc

• client-server paradigm
Protocol Stack

- **Application**: HTTP, SMTP, FTP, TELNET, DNS
- **End-to-End**: TCP, UDP
- **Network**: IP
- **Link Level**: Ethernet, token ring
Creating a network app

write programs that

No need to write software for network-core devices
Application architectures

- Client-server
- Peer-to-peer (P2P)
- Hybrid of client-server and P2P
Client-server architecture

server:

clients:

client/server
Pure P2P architecture
Hybrid of client-server and P2P

Skype
– voice-over-IP P2P application
– centralized server: finding address of remote party:
– client-client connection: direct (not through server)

Instant messaging
– chatting between two users is P2P
– centralized service: client presence detection/location
  • user registers its IP address with central server when it comes online
  • user contacts central server to find IP addresses of buddies
Processes communicating

**Process:** program running within a host.

**Client process:**

**Server process:**
Sockets

- process sends/receives messages to/from its socket

host or server

process

socket

TCP with buffers, variables

controlled by app developer

Internet

host or server

process

socket

TCP with buffers, variables

controlled by OS

host or server

process

socket

TCP with buffers, variables

controlled by OS
Socket programming

**Goal:** learn how to build client/server application that communicate using sockets

**Socket API**

- introduced in BSD4.1 UNIX, 1981
- explicitly created, used, released by apps
- client/server paradigm
- two types of transport service via socket API:
  - unreliable datagram
  - reliable, byte stream-oriented
Socket-programming using TCP

**Socket:** a door between application process and end-end-transport protocol (UCP or TCP)

**TCP service:** reliable transfer of *bytes* from one process to another
Socket programming *with TCP*

Client must contact server

- When contacted by client, **server** TCP creates new socket for server process to communicate with client

Client contacts server by:
Client/server socket interaction: TCP

Server (running on hostid)

- create socket, port=x, for incoming request:
  \[
  \text{welcomeSocket} = \text{ServerSocket()}
  \]

- wait for incoming connection request
  \[
  \text{connectionSocket} = \text{welcomeSocket.accept()}
  \]

- read request from connectionSocket
  \[
  \text{write reply to connectionSocket}
  \]

- close connectionSocket

Client

- create socket, connect to hostid, port=x
  \[
  \text{clientSocket} = \text{Socket()}
  \]

- send request using clientSocket
  \[
  \text{read reply from clientSocket}
  \]

- close clientSocket

TCP connection setup
Socket programming with TCP

Example client-server app:
1) client reads line from standard input (\texttt{inFromUser} stream), sends to server via socket (\texttt{outToServer} stream)
2) server reads line from socket
3) server converts line to uppercase, sends back to client
4) client reads, prints modified line from socket (\texttt{inFromServer} stream)
Example: Java client (TCP)

```java
import java.io.*;
import java.net.*;
class TCPClient {
    public static void main(String argv[]) throws Exception {
        String sentence;
        String modifiedSentence;

        BufferedReader inFromUser = 
            new BufferedReader(new InputStreamReader(System.in)) ;
        Socket clientSocket = new Socket("hostname", 6789);
        DataOutputStream outToServer = 
            new DataOutputStream(clientSocket.getOutputStream());

        BufferedReader inFromUser = 
            new BufferedReader(new InputStreamReader(System.in));
        Socket clientSocket = new Socket("hostname", 6789);
        DataOutputStream outToServer = 
            new DataOutputStream(clientSocket.getOutputStream());
    }
}
```
Example: Java client (TCP), cont.

```java
BufferedReader inFromServer =
    new BufferedReader(new InputStreamReader(clientSocket.getInputStream()));

sentence = inFromUser.readLine();

outToServer.writeBytes(sentence + '\n');

modifiedSentence = inFromServer.readLine();

System.out.println("FROM SERVER: " + modifiedSentence);

clientSocket.close();
```

Create input stream attached to socket

Send line to server

Read line from server
Example: Java server (TCP)

```java
import java.io.*;
import java.net.*;

class TCPServer {
    public static void main(String argv[]) throws Exception {
        String clientSentence;
        String capitalizedSentence;

        ServerSocket welcomeSocket = new ServerSocket(6789);
        BufferedReader inFromClient = new BufferedReader(new InputStreamReader(connectionSocket.getInputStream()));

        String clientSentence;
        String capitalizedSentence;

        while(true) {
            Socket connectionSocket = welcomeSocket.accept();
            BufferedReader inFromClient = new BufferedReader(new InputStreamReader(connectionSocket.getInputStream()));
        }
    }
}
```
Example: Java server (TCP), cont

Create output stream, attached to socket

DataOutputStream outToClient =
   new DataOutputStream(connectionSocket.getOutputStream());

Read in line from socket

clientSentence = inFromClient.readLine();

capitalizedSentence = clientSentence.toUpperCase() + '\n';

Write out line to socket

outToClient.writeBytes(capitalizedSentence);

End of while loop, loop back and wait for another client connection
App-layer protocol defines

- Types of messages exchanged,
- Message syntax:
- Message semantics
- Rules for when and how processes send & respond to messages

Public-domain protocols:
- defined in RFCs
- allows for interoperability
- e.g., HTTP, SMTP

Proprietary protocols:
- e.g., Skype
Transport service requirements of common apps

<table>
<thead>
<tr>
<th>Application</th>
<th>Data loss</th>
<th>Throughput</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 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>HTTP (eg Youtube), RTP [RFC 1889]</td>
<td>TCP or UDP</td>
</tr>
<tr>
<td>Internet telephony</td>
<td>SIP, RTP, proprietary (e.g., Skype)</td>
<td>typically UDP</td>
</tr>
</tbody>
</table>
Web and HTTP
Web and HTTP

First some jargon

Example URL:
www.someschool.edu/someDept/pic.gif

- host name
- path name
Addressing processes

• to receive messages, process must have identifier
• host device has unique 32-bit IP address
• identifier includes both IP address and port numbers associated with process on host.
• Example port numbers:
  – HTTP server: 80
  – Mail server: 25
• to send HTTP message to NTPU web server:
  – IP address: 120.126.125.166
  – Port number: 80
HTTP overview

HTTP: hypertext transfer protocol
HTTP overview (continued)

Uses TCP: HTTP is “stateless”
HTTP connections

Nonpersistent HTTP
- At most one object is sent over a TCP connection.

Persistent HTTP
- Multiple objects can be sent over single TCP connection between client and server.
Nonpersistent HTTP

Suppose user enters URL

www.someSchool.edu/someDepartment/home.index

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

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

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

3. HTTP server receives request
message, forms response
message containing requested
object, and sends message into
its socket
Nonpersistent HTTP (cont.)

4. HTTP server closes TCP connection.

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

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

Definition of RTT: time for a small packet to travel from client to server and back.

Response time:
Persistent HTTP

Nonpersistent HTTP issues:
- requires 2 RTTs per object
- OS overhead for *each* TCP connection
- browsers often open parallel TCP connections to fetch referenced objects

Persistent HTTP
- server leaves connection open after sending response
- subsequent HTTP messages between same client/server sent over open connection
- client sends requests as soon as it encounters a referenced object
- as little as one RTT for all the referenced objects
HTTP request message

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

```plaintext
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
Method types

HTTP/1.0
• GET
• POST
• HEAD

HTTP/1.1
• GET, POST, HEAD
• PUT
• DELETE
HTTP response message

status line
(protocol
status code
status phrase)

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, e.g.,
requested
HTML file
data data data data data data ...
HTTP response status 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
User-server state: cookies

Many major Web sites use cookies

**Four components:**

- Susan always access Internet always from PC
- visits specific e-commerce site for first time
- when initial HTTP requests arrives at site, site creates:
  - unique ID
  - entry in backend database for ID

**Example:**

2: Application Layer
Cookies: keeping “state” (cont.)

client

- eBay 8734
- cookie file
- eBay 8734 Amazon 1678

usual http request msg

Amazon server creates ID 1678 for user

create entry

backend database

usual http response msg

Set-cookie: 1678

cookie-specific action

server

usual http request msg

cookie: 1678

usual http response msg

cookie: 1678

usual http request msg

cookie: 1678

usual http response msg

cookie: 1678

one week later:

eBay 8734 Amazon 1678
Web caches (proxy server)

**Goal:** satisfy client request without involving origin server

- user sets browser: Web accesses via cache
- browser sends all HTTP requests to cache
  - object in cache: cache returns object
  - else cache requests object from origin server, then returns object to client
More about Web caching

Why Web caching?
Conditional GET

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

## Diagram

- **Cache:**
  - HTTP request msg
    - \text{If-modified-since: <date>}
  - HTTP response
    - \text{HTTP/1.0 304 Not Modified}

- **Server:**
  - HTTP request msg
    - \text{If-modified-since: <date>}
  - HTTP response
    - \text{HTTP/1.0 200 OK <data>}

2: Application Layer
Client and Server
Applications

• Traditional PC applications

• Client/server applications
Thin vs. Thick Clients

- Web Apps are “Thin”
- Server does processing
- Client does presentation
  + Simple! (Browser)
  - Limited GUI (HTML)
Thin vs. Thick Clients

• Software is “Thick” (AIM)
• Client does processing and presentation
  + GUI not limited by HTML
  + Fewer Latency Problems
  - People need to download & install client
Thick Email Client
Thin Email Client
Basic Security

• Authentication (Prove who you are)
• Traditional Authentication
  – Shared Secret
    • Server & client both know password
  – Password Demand (Server asks client for it)
    • Client presents it
    • Server checks against its own password DB
One-way Hash Function

- Combine, or “hash” bits of a string together to produce a “hash value”
- Function of the input
- Not invertible
- Hash’s should be unique
  - Strings A & B should not have same hash
Sample Hash Functions

• Bad Hash: Add Up Byte Values
  – FOOBAR = 70 + 79 + 79 + 66 + 65 + 82 = 441

• OK Hash: Linear Hash
  – Mathematical Function of Bits %
  – SOME_BIG_NUMBER

• Good Hash: MD5 (128 bit hash values)

• Better: SHA-1 (160 bit values)
HTML
Document Structure

<html>
<head><title>My First Web Page</title></head>
<body bgcolor="white">
<p>A Paragraph of Text.</p>
</body>
</html>
Nested Tags

- Like a tree, each element is contained inside a parent element
- Each element may have any number of attributes
Basic Tags

• `<hr>` horizontal rule
• `<br>` new line
• `<b>...</b>` bold
• `<i>...</i>` italicize text in between
Advanced Tags

• `<ul><li>First Item</li></ul>`
• `<li>Second Item</li></ul>`
• Also, `<ol>...</ol>`
• `<img src="URL of image file">`
Image File Types

- JPEG
- GIF
- PNG
- SVG
Tables

- `<table>...</table>`
- `<tr>...</tr>` for each row
- `<td>...</td>` for each element in a row
Comments

• <!-- This is a comment -->
• <!--
    This paragraph, 
is also a
comment...
-->
Special HTML

- `&lt;` → `<`
- `&gt;` → `>`
- `&amp;` → `&`
- `&nbsp;` → `space`