Network Applications: The Web and High-Performance Web Servers

Y. Richard Yang

http://zoo.cs.yale.edu/classes/cs433/

9/24/2013

Outline

- Admin and recap
  - HTTP details
  - High-performance Web server
Admin

- Assignment One solution posted on the class homepage
- Assignment Two due one week from today

Recap: FTP

- Two types of TCP connections opened:
  - a control connection;
  - data connections
    - two approaches to open a data connection: PASV or PORT

- FTP is called a stateful protocol
  - state established by commands such as
    - USER/PASS
    - CWD
    - TYPE
Recap: HTTP Message Flow

FTP client

USER xxx
PASS xxx
PORT clientip:cport

HTTP server

GET /home/index.html

Server sends file on same connection

HTTP client

CWD home
RETR index.html

Server initiates TCP data connection
server:20
clientip:cport

HTTP server

Recap: HTTP Req. Msg Format

- ASCII (human-readable format)

<table>
<thead>
<tr>
<th>method</th>
<th>sp</th>
<th>URL</th>
<th>sp</th>
<th>version</th>
<th>cr</th>
<th>if</th>
</tr>
</thead>
<tbody>
<tr>
<td>header field name : value</td>
<td>cr</td>
<td>if</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>header field name : value</td>
<td>cr</td>
<td>if</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>cr</td>
<td>if</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Entity Body
Examples

- Access BasicWebServer using common browsers and observe the request headers

Recap: Dynamic Content Pages

- There are multiple approaches to make dynamic web pages:
  - Embedding code into pages (server side include)
    - http server includes an interpreter for the type of pages
  - Invoke external programs (http server is agnostic to the external program execution)

http://www.cs.yale.edu/index.shtml
http://www.cs.yale.edu/cgi-bin/ureserve.pl
http://www.google.com/search?q=Yale&sourceid=chrome
Example SSI

- See index.shtml, header.shtml, ...

CGI: Invoking External Programs

- Two issues
  - Input: Pass HTTP request parameters to the external program
  - Output: Redirect external program output to socket
Example: Typical CGI Implementation

- Starts the executable as a child process
  - Passes HTTP request as environment variables
    - http://httpd.apache.org/docs/2.2/env.html
    - CGI standard: http://www.ietf.org/rfc/rfc3875
  - Redirects input/output of the child process to the socket

http://httpd.apache.org/docs/2.2/howto/cgi.html

Example: CGI

- Example:
  - GET /search?q=Yale&sourceid=chrome HTTP/1.0
  - setup environment variables, in particular
    - $QUERY_STRING=q=Yale&sourceid=chrome
  - start search and redirect its input/output

http://docs.oracle.com/javase/1.5.0/docs/api/java/lang/ProcessBuilder.html
http://docs.oracle.com/javase/1.5.0/docs/api/java/lang/Process.html
Example

- Exec
  - http://www.cs.yale.edu/homes/yry/courses/cs433/cgi/price.cgi?appl

```perl
#!/usr/bin/perl -w

$company = $ENV{'QUERY_STRING'};
print "Content-Type: text/html\r\n";
print "\r\n";
print "<html>";
print "<h1>Hello! The price is \n";
if ($company =~ /appl/) {
  my $var_rand = rand();
  print 450 + 10 * $var_rand;
} else {
  print "150";
}
print "</h1>";
print "</html>";
```

Client Using Dynamic Pages

- See ajax.html for client code example

http://www.cs.yale.edu/homes/yry/courses/cs433/cgi/ajax.html
Discussions

- What features are missing in HTTP that we have covered so far?

HTTP: POST

- If an HTML page contains forms or parameter too large, they are sent using POST and encoded in message body

```
request line

{ header lines

  method  sp  URL  sp  version  cr  if

  header field name : value  cr  if

  header field name : value  cr  if

  cr  if

Entity Body
```
HTTP: POST Example

POST /path/script.cgi HTTP/1.0
User-Agent: MyAgent
Content-Type: application/x-www-form-urlencoded
Content-Length: 15

item1=A&item2=B

Example using nc

Stateful User-server Interaction: Cookies

Goal: no explicit application level session

❖ Server sends “cookie” to client in response msg
  Set-cookie: 1678453

❖ Client presents cookie in later requests
  Cookie: 1678453

❖ Server matches presented-cookie with server-stored info
  ❖ authentication
  ❖ remembering user preferences, previous choices

client

usual http request msg

usual http response +
  Set-cookie: #

server

usual http request msg

Cookie: #

usual http response msg

cookie-specific action

usual http request msg

Cookie: #

usual http response msg

cookie-specific action
Cookie Example

- Modify BasicHTTPSwever.java to set Cookie

Authentication of Client Request

- **Authentication goal:** control access to server documents
- **Stateless:** client must present authorization in each request
- **Authorization:** typically name, password
  - Authorization: header line in request
  - if no authorization presented, server refuses access, sends
    WWW-authenticate: header line in response

Browser caches name & password so that user does not have to repeatedly enter it.
Example: Amazon S3

- Amazon S3 API

HTTP/1.0 Delay

- >= 2 RTTs per object:
  - TCP handshake --- 1 RTT
  - client request and server responds --- at least 1 RTT (if object can be contained in one packet)

- Discussion: how to reduce delay?
HTTP Message Flow: Persistent HTTP

- Default for HTTP/1.1
- On same TCP connection: server parses request, responds, parses new request, ...
- Client sends requests for all referenced objects as soon as it receives base HTML
- Fewer RTTs

Browser Cache and Conditional GET

- **Goal:** don’t send object if client has up-to-date stored (cached) version
- client: specify date of cached copy in http request
  - If-modified-since: <date>
- server: response contains no object if cached copy up-to-date:
  - HTTP/1.0 304 Not Modified
- server: response contains modified object:
  - HTTP/1.1 200 OK
  - <data>
Web Caches (Proxy)

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

- User sets browser: Web accesses via web cache
- Client sends all http requests to web cache
  - if object at web cache, web cache immediately returns object in http response
  - else requests object from origin server, then returns http response to client

---

Two Types of Proxies

(FORWARD) PROXY server

- Server 1
- Server 2
- Server 3

HTTP request to Proxy/Server

- Cache for static data
- Filter
- Access control

Client

(REVERSE-) PROXY server

- Server 1
- Server 2
- Server 3

HTTP request from Proxy/Server

- Load-balancing
- Filter
- Cache for static data

The reverse-proxy is in the same network as the server

Client

DMZ

The proxy is in the same network as the client

Server 1

Server 2

Server 3

INTERNET

http://www.celinio.net/techblog/?p=1027
Summary: HTTP

- HTTP message format
  - ASCII (human-readable format) requests, header lines, entity body, and responses line

- HTTP message flow
  - stateless server
    - each request is self-contained; thus cookie and authentication, are needed in each message
  - reducing latency
    - persistent HTTP
      - the problem is introduced by layering!
    - conditional GET reduces server/network workload and latency
    - cache and proxy reduce traffic and/or latency

WebServer Implementation

Create ServerSocket(6789)

connSocket = accept()

read request from connSocket

read local file

write file to connSocket

close connSocket

Discussion: what does each step do and how long does it take?
Many socket/IO operations can cause a process to block, e.g.,
- `accept`: waiting for new connection;
- `read` a socket waiting for data or close;
- `write` a socket waiting for buffer space;
- `I/O read/write` for disk to finish

Recap: Server Processing Steps

Want to be able to process requests concurrently.
**Goal: Limited Only by the Bottleneck**

Before

![Diagram showing CPU, DISK, and NET before implementing multi-threads.](image)

After

![Diagram showing CPU, DISK, and NET after implementing multi-threads.](image)

**Using Multi-Threads for Servers**

- A thread is a sequence of instructions which may execute in parallel with other threads.

- A multi-thread server is a concurrent program as it has multiple threads that are active at the same time, e.g.,
  - we can have one thread for each client connection
  - thus, only the flow (thread) processing a particular request is blocked
Multi-Threaded Server

- A multithreaded server might run on one CPU
  - The CPU alternates between running different threads
  - The scheduler takes care of the details
  - Switching between threads might happen at any time

- Might run in parallel on a multiprocessor machine

Java Thread Model

- Every Java application has at least one thread
  - The “main” thread, started by the JVM to run the application’s main() method
  - Most JVM’s use POSIX threads to implement Java threads

- main() can create other threads
  - Explicitly, using the Thread class
  - Implicitly, by calling libraries that create threads as a consequence (RMI, AWT/Swing, Applets, etc.)
Thread vs Process

- Concurrency is introduced through objects of the class `Thread`
  - Provides a ‘handle’ to an underlying thread of control

- Threads are organized into thread groups
  - A thread group represents a set of threads
    - `activeGroupCount()`;
  - A thread group can also include other thread groups to form a tree
  - Why thread group?

Java Thread Class

http://java.sun.com/javase/6/docs/api/java/lang/ThreadGroup.html
Some Main Java Thread Methods

- **Thread(Runnable target)**
  Allocates a new Thread object.
- **Thread(String name)**
  Allocates a new Thread object.
- **Thread(ThreadGroup group, Runnable target)**
  Allocates a new Thread object.
- **start()**
  Start the processing of a thread; JVM calls the run method

Creating Java Thread

- **Two ways to implement Java thread**
  - Extend the Thread class
    - Overwrite the run() method of the Thread class
  - Create a class C implementing the Runnable interface, and create an object of type C, then use a Thread object to wrap up C
- **A thread starts execution after its start() method is called, which will start executing the thread’ s (or the Runnable object’ s) run() method**
- **A thread terminates when the run() method returns**

http://java.sun.com/javase/6/docs/api/java/lang/Thread.html
Option 1: Extending Java Thread

class PrimeThread extends Thread {
    long minPrime;
    
    PrimeThread(long minPrime) {
        this.minPrime = minPrime;
    }
    
    public void run() {
        // compute primes larger than minPrime . . .
    }
}

PrimeThread p = new PrimeThread(143);
p.start();

Option 1: Extending Java Thread

class RequestHandler extends Thread {
    RequestHandler(Socket connSocket) {
        // ...
    }
    
    public void run() {
        // process request
    }
    
    ...
}

Thread t = new RequestHandler(connSocket);
t.start();
Option 2: Implement the Runnable Interface

class PrimeRun implements Runnable {
    long minPrime;
    PrimeRun(long minPrime) {
        this.minPrime = minPrime;
    }

    public void run() {
        // compute primes larger than minPrime . . .
    }
}

PrimeRun p = new PrimeRun(143);
new Thread(p).start();

Option 2: Implement the Runnable Interface

class RequestHandler implements Runnable {
    RequestHandler(Socket connSocket) { ... }
    public void run() {
        //
    }

    ...
}
RequestHandler rh = new RequestHandler(connSocket);
Thread t = new Thread(rh);
t.start();
Benefits of Web Caching

Assume: cache is “close” to client (e.g., in same network)

- smaller response time: cache “closer” to client
- decrease traffic to distant servers
  - link out of institutional/local ISP network often bottleneck

Diagram:
- Public Internet
- Origin servers
- 1.5 Mbps access link
- Institutional network
- 10 Mbps LAN
- Institutional cache
Cache Sharing: Internet Cache Protocol (ICP)

When one proxy has a cache miss, send queries to all siblings (and parents): “do you have the URL?”

Whoever responds first with “Yes”, send a request to fetch the file

If no “Yes” response within certain time limit, send request to Web server

Discussion: where is the performance bottleneck of ICP?
Summary Cache

- Basic idea:
  - let each proxy keep a directory of what URLs are cached in every other proxy, and use the directory as a filter to reduce number of queries

- Problem: storage requirement
  - solution: compress the directory => imprecise, but inclusive directory

The Problem

Diagram showing the interaction between Proxy A and Proxy B, with a compact representation of the directory.
Bloom Filters

- Support membership test for a set of keys

\[ URL \: u \quad Bit \: Vector \: V_B \]

\[ H_1(u) = P_1 \]
\[ H_2(u) = P_2 \]
\[ H_3(u) = P_3 \]
\[ H_4(u) = P_4 \]

- To check if URL x is at B, compute \( H_1(x), H_2(x), H_3(x), H_4(x) \), and check \( V_B \)

No Free Lunch: Problems of Web Caching

- The major issue of web caching is how to maintain consistency
- Two ways
  - pull
    - Web caches periodically pull the web server to see if a document is modified
  - push
    - whenever a server gives a copy of a web page to a web cache, they sign a lease with an expiration time; if the web page is modified before the lease, the server notifies the cache