15-213 Recitation 13: Proxylab – Network and Web

11 April 2016

Ralf Brown and the 15-213 staff

Agenda

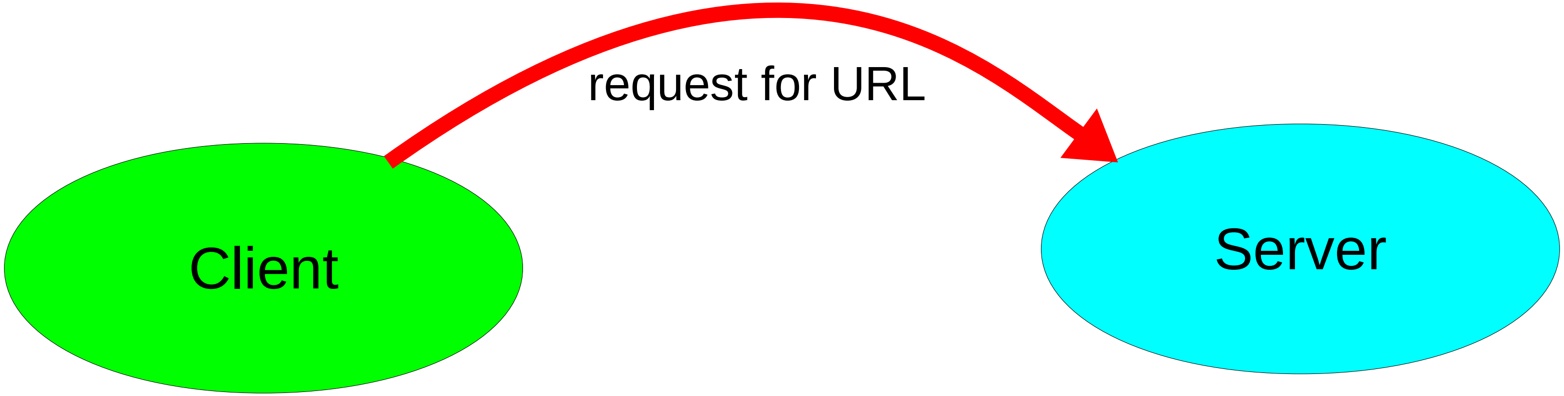
- Reminders
- Complex Web Pages
- Proxies
- Threads
- Appendix: HTTP

Reminders

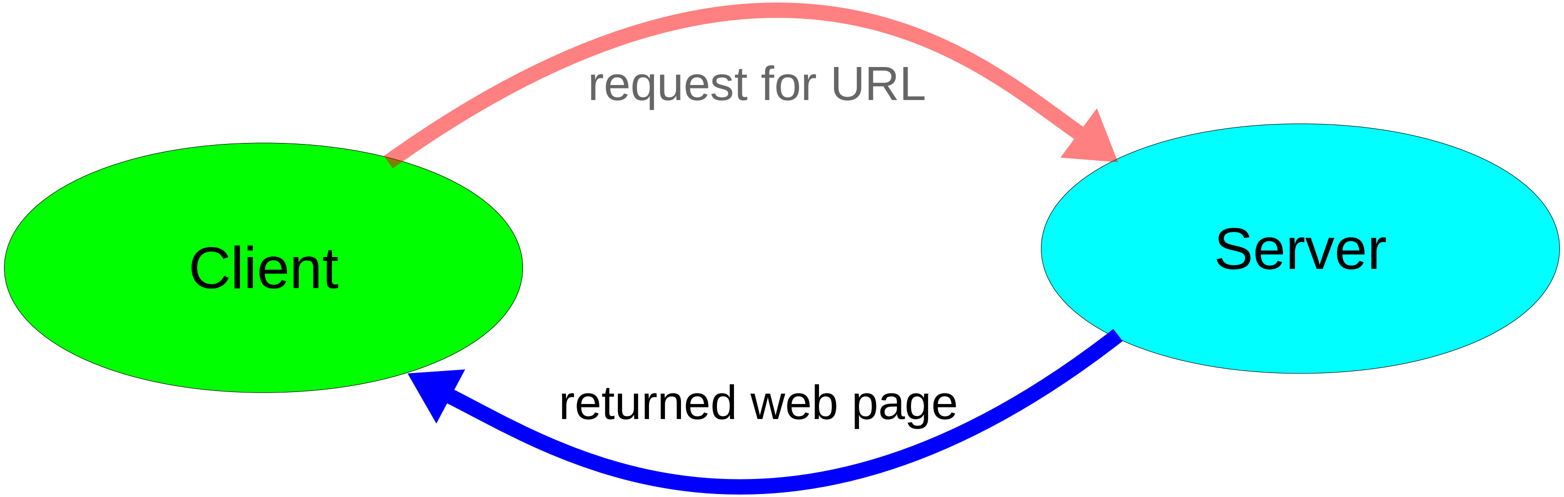
- Start working on Proxylab **now** – there are no grace days!

How the Web Works in Textbooks

How the Web Works in Textbooks



How the Web Works in Textbooks



How the Web Really Works

- An HTML page may depend on tens or even hundreds of support files
  - images, style sheets, scripts, etc.
- Excerpt from www.cmu.edu/index.html

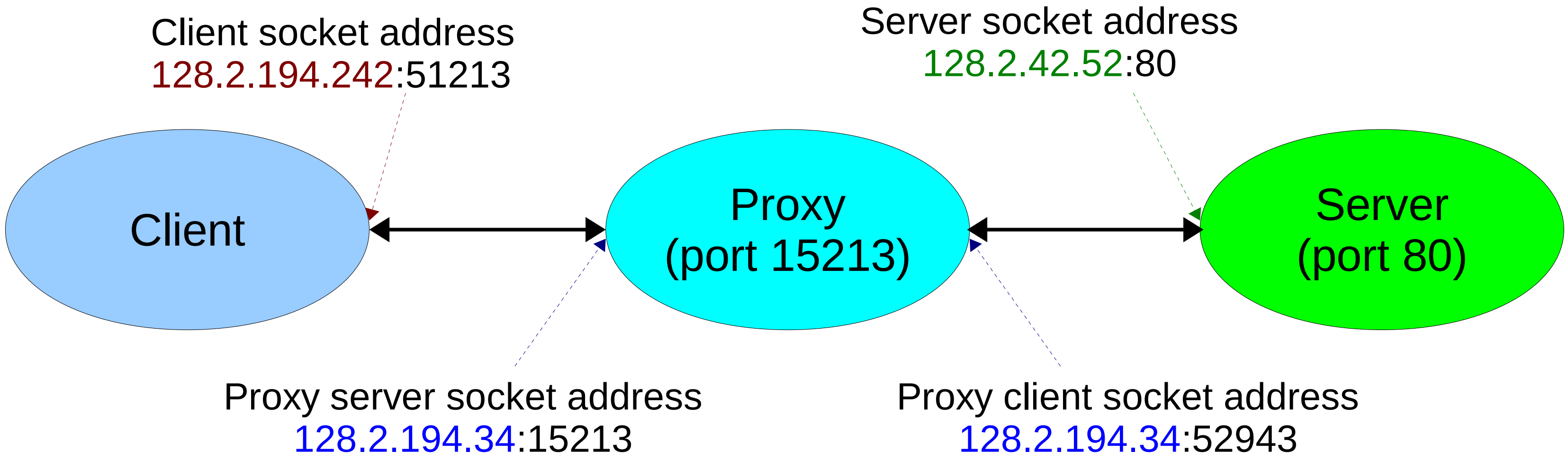
```
<link href="//fonts.googleapis.com/css?family=Open+Sans:400,400italic,
    300italic,300,700,700italic" rel="stylesheet" type="text/css"/>
<link href="//www.cmu.edu/cmu-design-2015/css/main.min.css"
    rel="stylesheet" type="text/css"/>
<link href="//www.cmu.edu/favicon.ico" rel="icon"/>
<link href="http://www.cmu.edu/news/feeds/news.rss" rel="alternate"
    title="CMU News" type="application/rss+xml"/>

<script src="//www.cmu.edu/cmu-design-2015/js/jquery-1.10.1.min.js"
    type="text/javascript"></script>

<div aria-label="Costume Designs by Susan Tsu" class="hero "
    data-img="assets/images/homepage/2016/costume_awards_1400x700-min.jpg"
    style="background-image:url(assets/images/homepage/2016/costume_awards2_
           600x400-min.jpg.jpeg)">
```

How the Web Really Works

- The large number of subfiles is a strong argument for concurrent servers
  - Sequential retrieval means waiting for tens of back-to-back requests
  - network I/O is usually slower than processing
- Caching is simplified by using separate support files
  - each object already has a unique URL that can be used as a key
  - many pages can share a single support object
  - if only part of the page changes, there's no need to re-fetch the unchanged bits

Proxy

- Proxies are special because they are both client **and** server
  - act as server for the computer making a request
  - act as client to the server the request is directed at
- The overall control flow will look like a server, but the proxy will have to act as a client to complete the request



Proxy - Functionality

- Should work on the vast majority of sites
    - exception 1: sites which require POST operation
        - logging in to websites, sending Facebook messages, ...
    - exception 2: HTTPS is not expected to work
        - Google, YouTube, and a number of other popular websites try to force HTTPS for security – watch out for that
- Cache previous requests
    - LRU eviction policy
    - must allow concurrent reads while remaining consistent
    - details are in the write up

Proxy - Caching

- Need a multi-threaded cache
    - sequential cache would be a bottleneck on a parallel proxy
    - safe for multiple threads to read cached contents at the same time
        - two threads can read from the same cache block
    - writing content is trickier
        - what if another thread is reading the content being overwritten?
        - what if two threads write the same block at once?

Proxy - Caching

- Need a multi-threaded cache
  - sequential cache would be a bottleneck on a parallel proxy
  - safe for multiple threads to read cached contents at the same time
    - two threads can read from the same cache block
  - writing content is trickier
    - what if another thread is reading the content being overwritten?
    - what if two threads write the same block at once?

We'll cover this in the lectures on synchronization

Implementing and Testing Your Proxy

- Go step-by-step
    - start with a sequential proxy
    - then make it concurrent by adding threads
    - finally, add caching
- Ensure that you can deal with arbitrary binary data
    - strcpy won't work, try memcpy instead
- Make yourself an automated test suite
    - use cURL in a script or makefile to retrieve a set of URLs
- Or, set your browser to use your proxy and see what breaks

Telnet / cURL

- Telnet is much like our echoserver demo
  - it passes its input to the server, and displays whatever the server sends back
  - using it to get a web page means building the HTTP request manually
- cURL: “URL transfer library” with a commandline program
  - builds valid HTTP requests for you!

```
curl http://www.cs.cmu.edu/~ralf
```

```
<!DOCTYPE HTML PUBLIC "-//IETF//DTD HTML 2.0//EN">
<html><head>
<title>301 Moved Permanently</title>
</head><body>
<h1>Moved Permanently</h1>
<p>The document has moved <a href="http://www.cs.cmu.edu/~ralf/">here</a>.</p>
</body></html>
```

  - can also be used to generate HTTP proxy requests

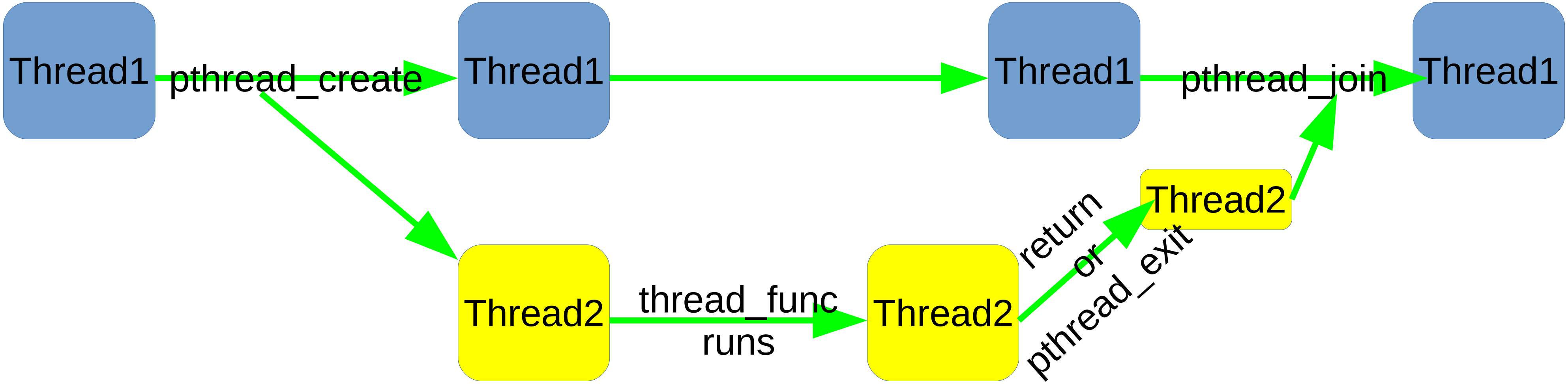
```
curl --proxy lemonshark.ics.cs.cmu.edu:3092 http://www.cmu.edu/
```

Live Telnet/cURL Demo

Threads

- Threads have similarities to processes and signal handlers
  - like both, they are a separate flow of execution
  - like processes, all threads continue executing in parallel
  - like signal handlers, threads share an address space
- Threads have a life-cycle very similar to processes
  - create, execute, terminate, join

Thread Life Cycle



Creating a Thread

-
  ```
  int pthread_create(pthread_t *tID, const pthread_attr_t *attr,
                               void *(*thread_func)(void*), void *arg);
  ```
- create a new thread that will execute `thread_func`, passing it `arg`.
  - to pass the function multiple arguments, put them in a struct and pass a pointer to the struct
  - `attr` is NULL for this course
- returns 0 if successful, an error number otherwise
  - updates `tID` with the thread ID of the new thread if successful

Terminating a Thread

- A thread can terminate in multiple ways
  - call `pthread_exit` with an exit status
  - return from the `thread_func`; equivalent to calling `pthread_exit` with the returned value
  - another thread can kill it with `pthread_cancel`
  - call `exit()` -- this kills all threads in the process

Joining Threads

- `int pthread_join(pthread_t thread, void **retval);`
- wait for the specified thread to terminate, then retrieve its return value and update the void* pointed at by the second argument (if not NULL)
  - returns immediately if the thread has already terminated
  - if the thread was canceled, its return value is PTHREAD_CANCELED
- using a void* as the return/status value of a thread allows for arbitrary data to be returned

Gotchas with Threads

- Threads share everything except local variables
  - if multiple threads want to access a single global object (variable, file, etc.), access needs to be synchronized
    - we'll cover synchronization next week
- **Hard** to write correct multi-threaded code that actually runs faster than single-threaded code
  - look for tasks that don't depend on other things
  - split up tasks into chunks that can be processed independently

Hints

- **Start small**
  - grab the echo server and echo client from the textbook
  - review the tiny.c basic web server code to see how to handle HTTP headers
    - tiny.c ignores them, you must not
- Cache individual web objects, not the whole page
- **Test liberally**
  - the web is full of special cases that want to break your proxy
  - generate a port for yourself with `./port-for-user.pl {andrewid}`
  - generate more ports for web servers, etc. with `./free-port.sh`
  - consider using your Andrew space to host test files
    - visit https://www.andrew.cmu.edu/server/publish.html to make ~/www public

APPENDIX

HTTP

- Every HTTP session starts with a request
  - command followed by any additional header fields
  - blank line ends header
  - specified number of additional bytes (or until EOF) for *content*
- Server responds with a status, followed by optional header fields, a blank line, and then the content
- Headers use \r\n instead of just \n to terminate lines!

HTTP Commands

version may be 1.0 or 1.1

- GET {uri} HTTP/{version}
  - retrieve the object at the specified URI
- HEAD {uri} HTTP/{version}
  - like GET, but don't return the actual object, just the headers
- PUT {uri} HTTP/{version}
  - store the following content at the specified URI
- POST {uri} HTTP/{version}
  - used to send forms or other data
- DELETE {uri} HTTP/{version}
  - remove the object at the specified URI

HTTP Response Codes

- First line of the server's response is the status of the request
  - HTTP/{version} {codenumber} {description}
- Status codes are three-digit numbers; the first gives the category
  - 1xx informational
  - 2xx success
    - 200 OK, 201 Created, ...
  - 3xx redirection
    - 301 Moved Permanently, 302 Moved Temporarily, 304 Not Modified, ...
  - 4xx client error
    - 400 Bad Request, 404 Not Found, ...
  - 5xx server error

HTTP Header Fields

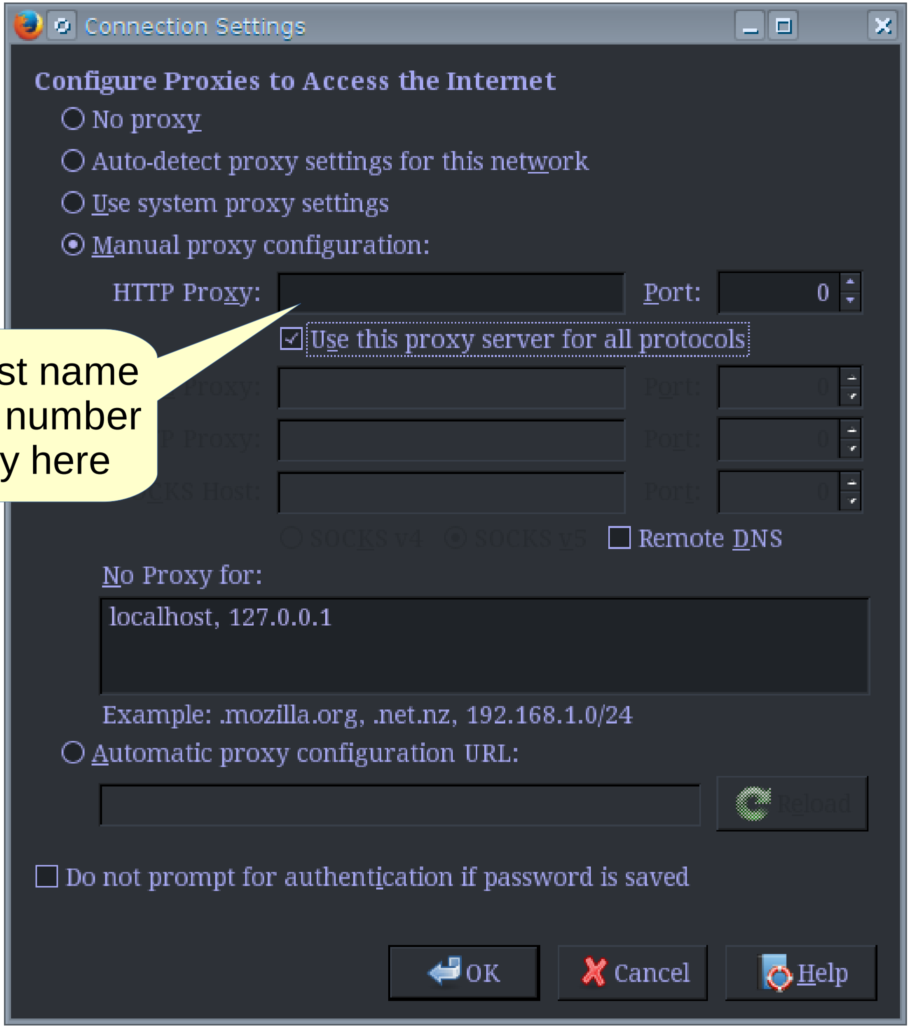
- General format: name followed by a colon, a space, then the value
- Common headers:
    - Host: {hostname}[:{port}] (required for HTTP/1.1)
    - Content-Length: {numbytes}
    - Content-Type: {MIMEtype}
    - Cookie: (client) / Set-Cookie: (server)
    - Connection: close / Connection: keep-alive
    - Cache-Control: no-cache
    - If-Modified-Since: {datestamp} (returns 304 if URI content unchanged)
    - Accept: {MIMEtypes}
    - Location: {URL} (new location for redirection)

Version Control – Using GIT

- git pull
- git add .
- git commit -m “I changed such-and-such” {file}
- git push

Setting Up Firefox

- We'll be grading with Firefox
- Preferences > Advanced > Network > Settings... (under Connection)
- Select “Manual proxy config”
- Check “Use this proxy for all protocols” or your proxy will **seem** to work with HTTPS traffic



Connection Settings

Configure Proxies to Access the Internet

○ No proxy

○ Auto-detect proxy settings for this network

○ Use system proxy settings

● Manual proxy configuration:

HTTP Proxy: Port: 0

☑ Use this proxy server for all protocols

Proxy: Port: 0

P Proxy: Port: 0

KS Host: Port: 0

○ SOCKS v4 ● SOCKS v5 □ Remote DNS

No Proxy for:

localhost, 127.0.0.1

Example: .mozilla.org, .net.nz, 192.168.1.0/24

○ Automatic proxy configuration URL:

□ Do not prompt for authentication if password is saved

enter host name and port number of proxy here