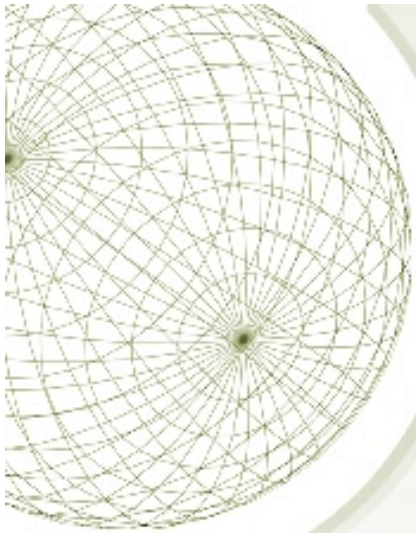
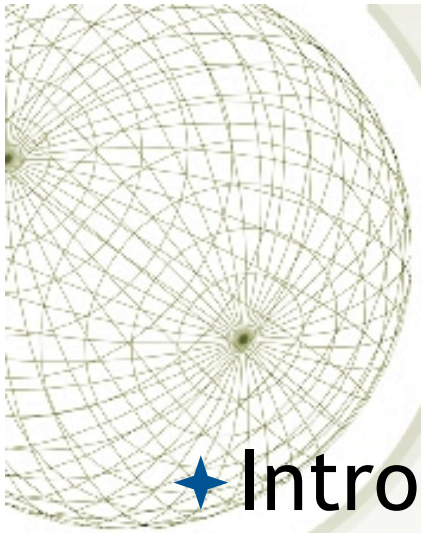


15-213 Recitation: Proxy Lab

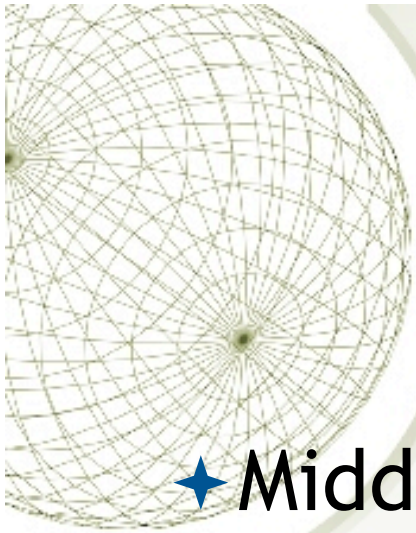
Elie Krevat

(with wisdom from past terms)



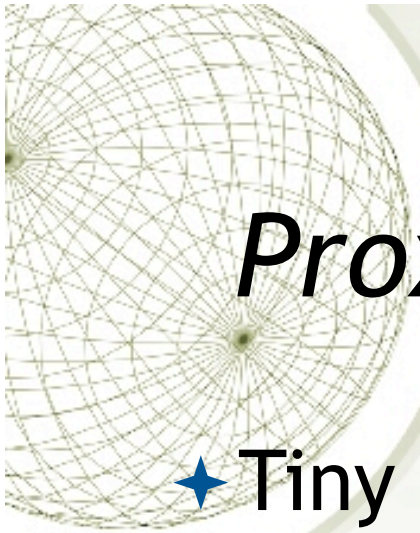
Outline

- Intro to Proxy Lab
- This week: Sequential Proxy
  - HTTP over TCP/IP
  - What to parse from HTTP headers
  - What headers to suppress
  - Handling broken pipes and using RIO
  - Testing and Debugging
- Next week: Concurrency and caching



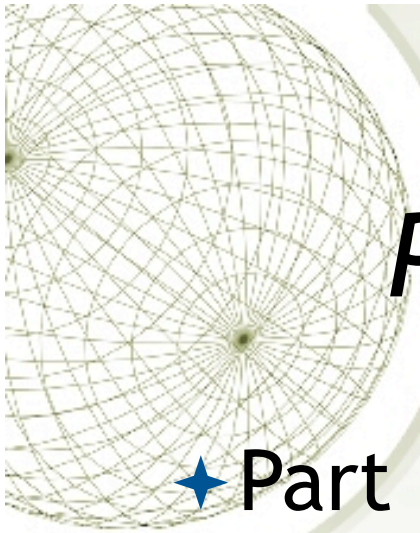
What is a proxy

- Middle-man between browser (client) and web server
  - Acts as client to web server
  - Also acts as server to browser
- Useful as firewall, logger, **cache**



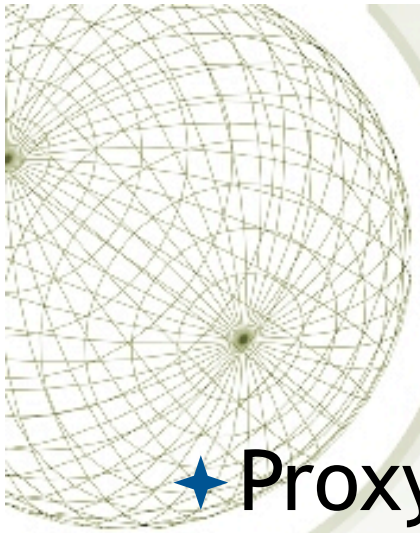
Proxy Lab: What we give you

- Tiny Web Server
  - Example of web server code
  - Debug: Change code to control behavior
- csapp.c/h
  - RIO package, wrapper/helper functions
- port_for_user.pl
  - Script to generate port # for your proxy
- proxy.c – Empty!



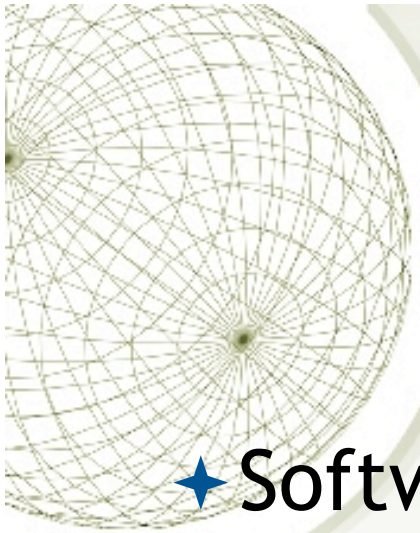
Proxy Lab: What you'll do

- Part 1: Sequential Web Proxy
  - Accept conn, read req, parse it, forward req to server, get reply, forward to client
- Part 2: Thread-based Concurrent Proxy
  - Spawn threads for each request in parallel
- Part 3: Adding a Cache
  - Apply LRU eviction policy
  - Make cache *efficient* and *thread-safe*!



Start early (please!)

- Proxy Lab is less intricate than Malloc
- BUT you'll be writing a full proxy, with code basically from scratch…
- …and you'll still need to understand some conceptual hurdles…
- …AND IT TAKES ABOUT AS LONG, IF NOT LONGER, THAN MALLOC LAB!!
- **So please start early!**



What Proxy Lab covers

- Software engineering skills
  - Writing projects from scratch, in groups
  - Reading formal specifications
  - Testing and extending functionality
- Unix socket programming
- Internet communication
- Threading and concurrency
- Caching and data structure design

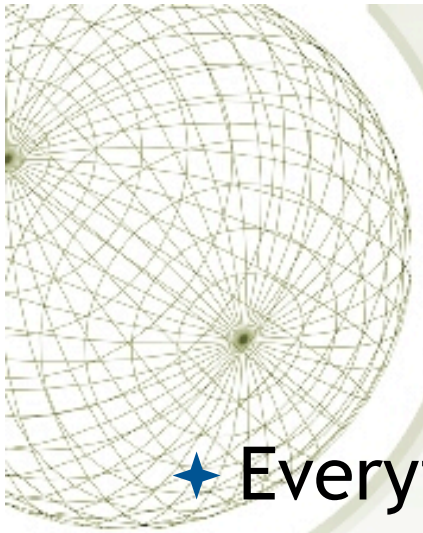
Socket programming (briefly)

- Socket is a file descriptor, special init
  - Identifies endpoint of communication
- Imp. functions: *connect*, *bind*, *accept*
- Sockets opened with *sockaddr*

```
struct sockaddr {
  unsigned short  sa_family;    /* protocol family */
  char            sa_data[14];  /* address data.   */
};
```

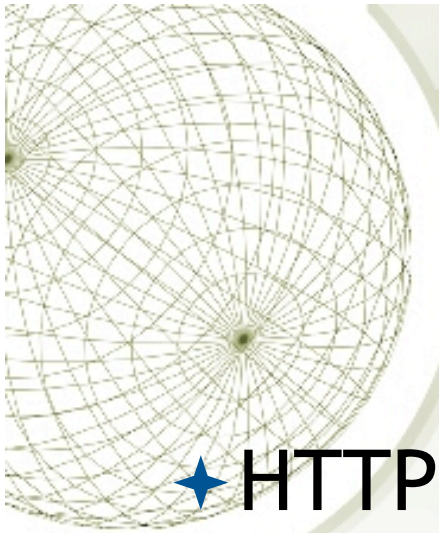
- For Internet, use *sockaddr_in*

```
struct sockaddr_in  {
  unsigned short  sin_family;  /* address family (always AF_INET) */
  unsigned short  sin_port;    /* port num in network byte order */
  struct in_addr  sin_addr;    /* IP addr in network byte order */
  unsigned char   sin_zero[8]; /* pad to sizeof(struct sockaddr) */
};
```



HTTP over TCP/IP

- Everything is handled in layers
- IP handles **addressing, unreliable comm.**
  - Which computer, basic message passing
- TCP over IP handles **multiplexing** and **reliable comm.**
  - Which process, moving bytes in-order through congestion and packet loss
- HTTP over TCP handles **semantics**
  - Bytes become ordered text and pictures on page



HTTP Request

- HTTP defines protocol between web servers and clients
- Headers hold meta-data of connection
  - **Type** of request (GET, PUT, POST…)
    - Proxy Lab only covers GET requests!
  - **URL** destination (http://www.cmu.edu)
- Body holds actual data

HTTP Request (cont.)

Request Type Host Path Version

```
GET http://csapp.cs.cmu.edu/simple.html HTTP/1.1
Host: csapp.cs.cmu.edu
User-Agent: Mozilla/5.0 ...
Accept: text/xml,application/xml ...
Accept-Language: en-us,en;q=0.5 ...
Accept-Encoding: gzip,deflate ...
```

An empty line (“\r\n”) terminates a request.

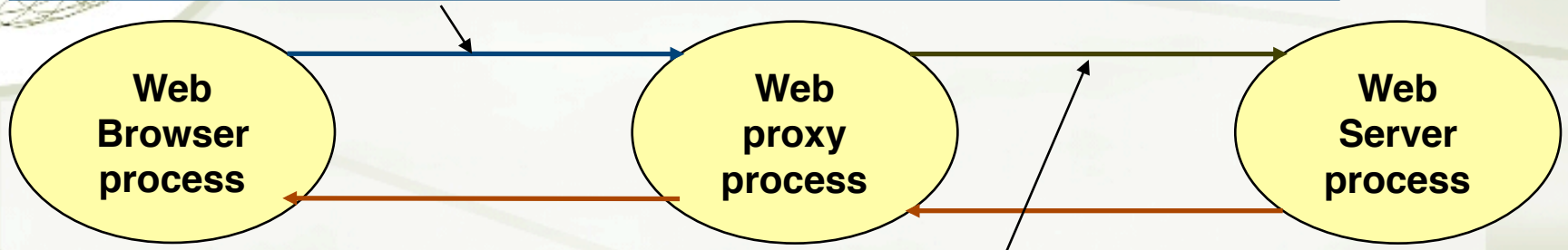
Parsing the headers

```
GET http://www.cmu.edu:80/index.html HTTP/1.1
<other information>
```

- Complete URL
  - Extract the path URI for HTTP request
- Version
  - Change to HTTP 1.0 for server request
- Hostname
  - Needed for the Host: field in server request
- Port
  - Proxy needs dest. port of server (default 80)

Forwarding requests

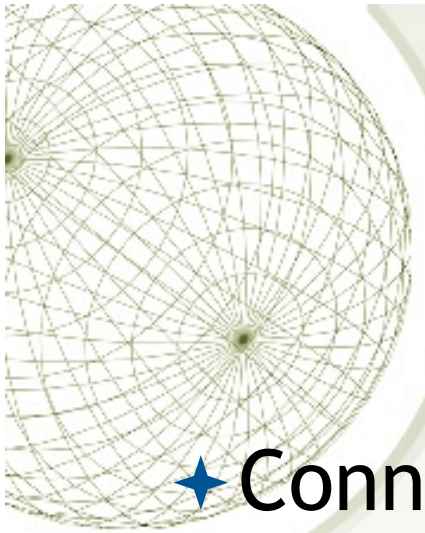
```
GET http://www.cmu.edu:80/index.html HTTP/1.1
<other information>
```



- Connects to target web server, sends request:

```
GET /index.html HTTP/1.0
<other information in the original request>
```

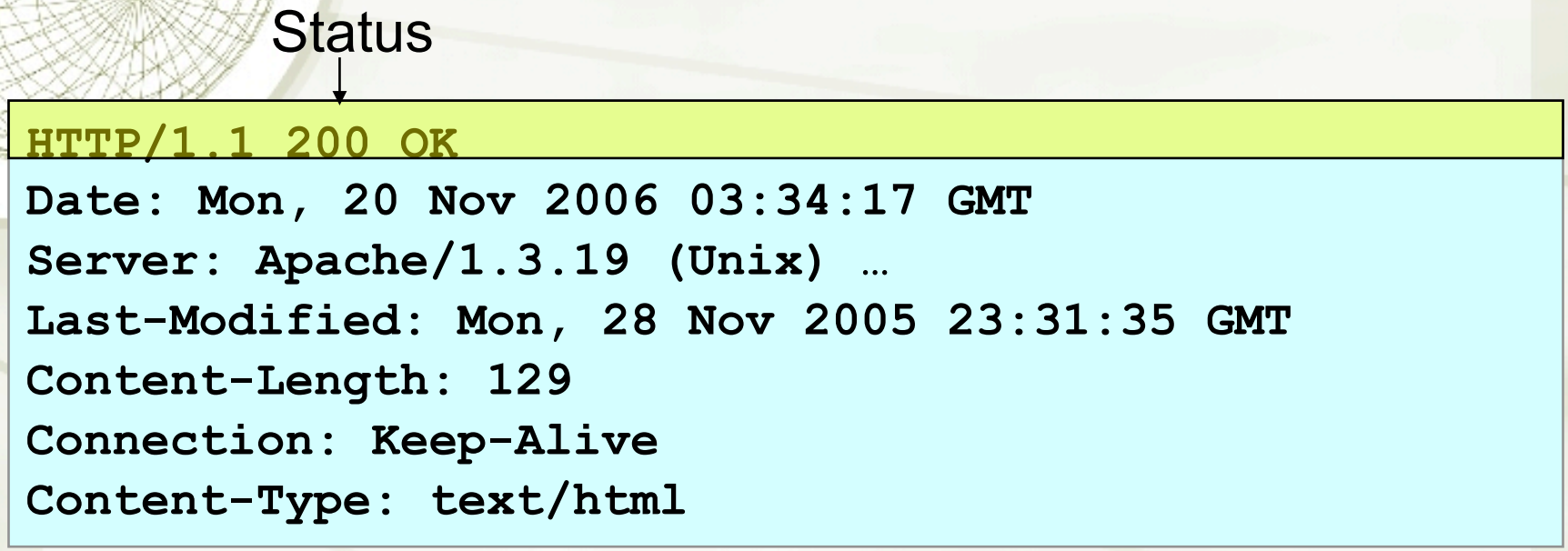
- Proxy parses HTTP request
- Port not always specified (default 80)
- Proxy suppresses/modifies headers for server req



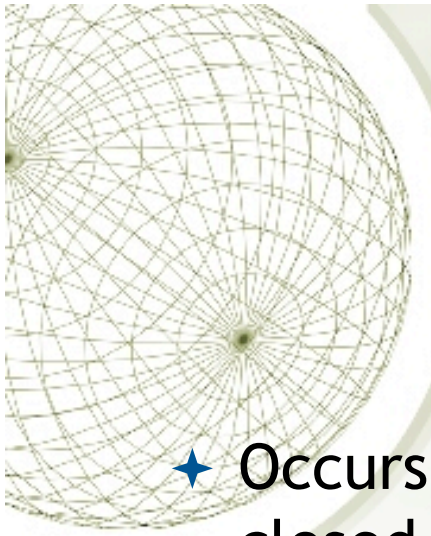
Headers to suppress

- Connection/Proxy-Connection
  - Change the field value to *close*
- Keep-Alive
  - Remove this, don’t want persistent connections with HTTP/1.0
- Keep the rest!

HTTP Response

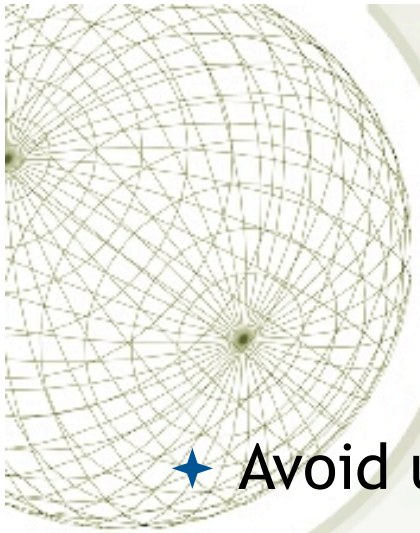


- Status indicates success
- Send complete response back to client



Broken pipe errors

- Occurs when writing to socket and connection closed prematurely at other end
  - E.g., click “stop” on browser
- Kernel returns normally on first write
- But on subsequent writes, kernel sends SIGPIPE
  - Terminates process by default (can be blocked or caught)
  - Returns -1 with errno set to EPIPE
- When reading from socket with closed connection
  - Returns -1 with errno set to ECONNRESET



Using Robust I/O (RIO)

- Avoid upper-case wrapper functions (terminate all)
- Instead, close the offending connection
  - Optionally, print error message
- Handle client request:
  - Use rio_readlineb to read client req
    - "\r\n" signals end of the request
  - rio_writen to send request to server
- Handle server response:
  - Use rio_readnb to read server response
    - Binary data, so difference is memcpy vs. strcpy
  - rio_writen to send response to client/browser

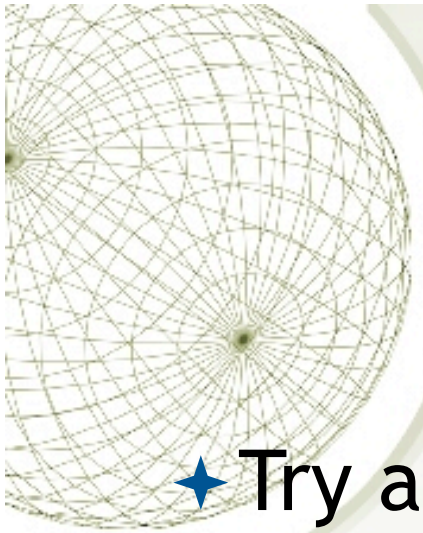
Debug: Is this my problem?

- Web server issued HTTP redirect to the client!
- DNS lookup for requested hostname failed!
- Hostname ok but rest of URL bogus, server ret. 404!
- Web server crashed while it was replying!
- Server sent me mp3 but firefox won't play it!
- Client crashed while I was sending server's reply!
- Webpage contains images that I haven't requested!
- Server sent me something too big for me to cache!
- Client is sending lots of indecipherable headers!

Debug: Is this my problem?

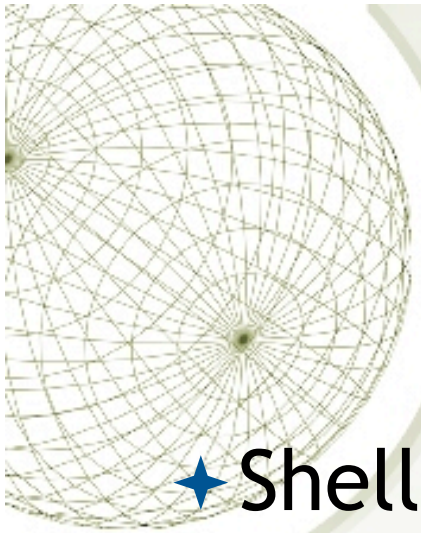
- Web server issued HTTP redirect to the client!
- DNS lookup for requested hostname failed!
- Hostname ok but rest of URL bogus, server ret. 404!
- Web server crashed while it was replying!
- Server sent me mp3 but firefox won't play it!
- Client crashed while I was sending server's reply!
- Webpage contains images that I haven't requested!
- Server sent me something too big for me to cache!
- Client is sending lots of indecipherable headers!

Many things can go wrong, but most are out of scope!



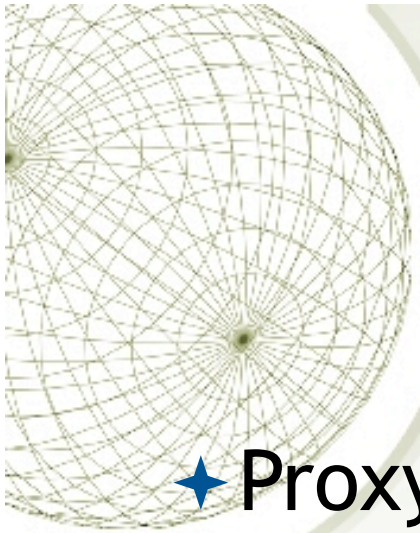
Testing Your Proxy

- Try a variety of web pages
- Test for both static & dynamic content
- Test binary files (e.g., images)
- See proxylab writeup for more tips



Next Week: Concurrency

- Shell lab handled **asynchronous signals**
- Proxy lab enables **concurrent threads**
- Similar ideas:
  - Both handle race conditions when running code at the same time
  - But threads are constantly switching and allow more memory sharing



Summary

- Proxy Lab covers **many concepts** with **lots of code** to write from scratch
- Proxy parses and forwards HTTP reqs
- Also clean error handling, broken pipes
- Start early!
- Next week: Multi-threaded goodness