Introduction to Proxy Lab

Richard Ha (rkh)

Today

- **Administrivia**
- Introduction to Proxy Lab

Administrivia

- malloc() lab is now past due (Thursday)
- Proxy lab is out!
  - Final lab of the semester
  - Due August 5, 2015 @ 11:59pm EST
- Shell lab style grades will be sent out soon (by the weekend?)
- Might skip malloc() for style grading
- Proxy lab will definitely have style grading

Topics

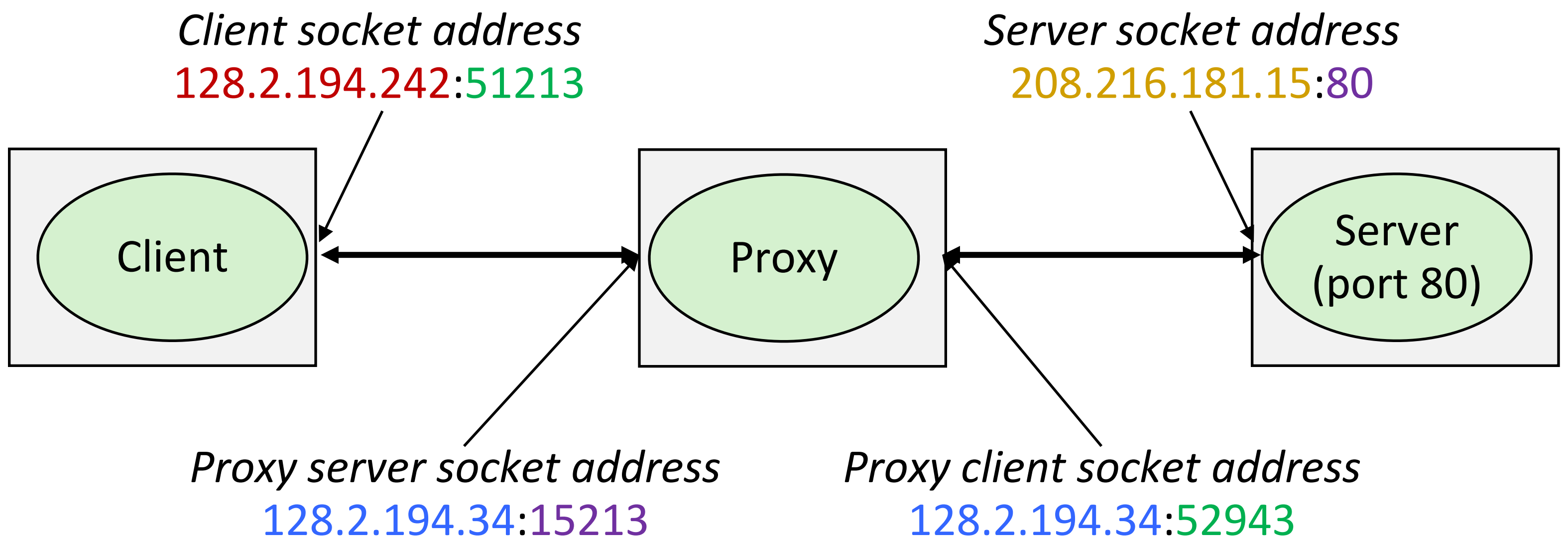
- Administrivia
- Introduction to Proxy Lab

What is a Proxy?

- So far you have only seen the traditional client-server model for networks
  - Everything is framed as either a client or a server.
  - The client sends a request to the server.
  - The server sends a response to the client.
- Real world applications are much more complex than that.

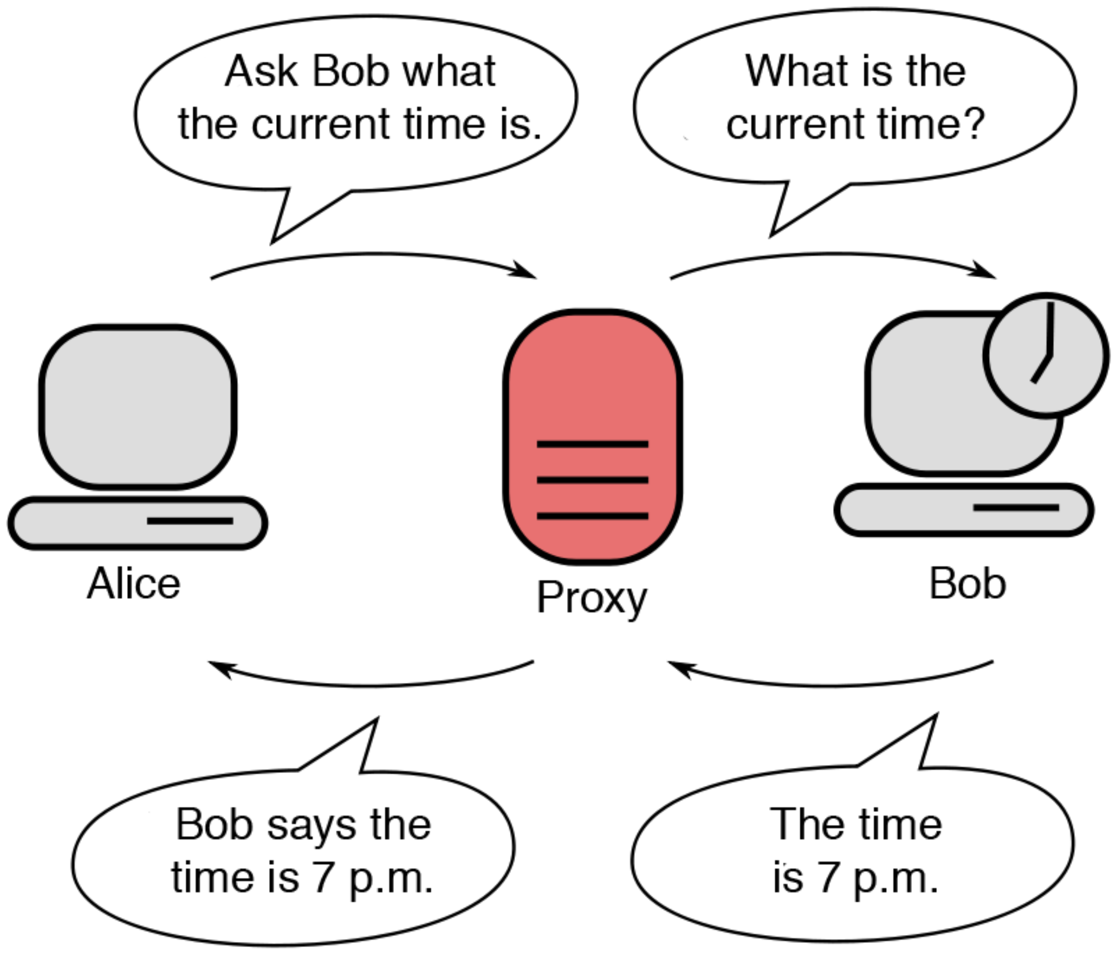
What is a Proxy?

- A proxy is a go-between a client and a server



- It is both a server AND a client!

What is a Proxy?


Ask Bob what the current time is.
What is the current time?
Alice
Proxy
Bob
Bob says the time is 7 p.m.
The time is 7 p.m.

https://en.wikipedia.org/wiki/Proxy_server#/media/File:Proxy_concept_en.svg

Pros and Cons

- Pros
  - Can be used to bypass network access restrictions
  - Anonymity
  - Caching
  - Monitoring and filtering content (e.g. unwanted content/malware)
- Cons
  - Monitoring and filtering content (i.e. censorship, eavesdropping)
  - Security concerns (Man-in-the-middle attacks)

Implementation Specifics

- Listens on a port for client requests
- Make request to target server for client
- Pass on responses from server to the client
- Step 1: Make a sequential proxy
- Step 2: Make a concurrent proxy
- Step 3: Implement caching

More Implementation Specifics

- Your proxy should be able to handle HTTP/1.0 GET requests
  - Your browser will make requests as HTTP/1.1 – Your proxy should make its requests as HTTP/1.0 anyway.
- Other types of HTTP requests (HEAD, PUT, POST, etc.) are not required. Strictly optional.
  - Site features such as logging in will most likely not work without POST though
- HTTPS is not required, and will not work.
  - Many popular websites nowadays will mandate site-wide HTTPS for security concerns, so they might not work for real-life testing/debugging
  - Our tests will use the websites listed in the handout for grading

More Implementation Specifics

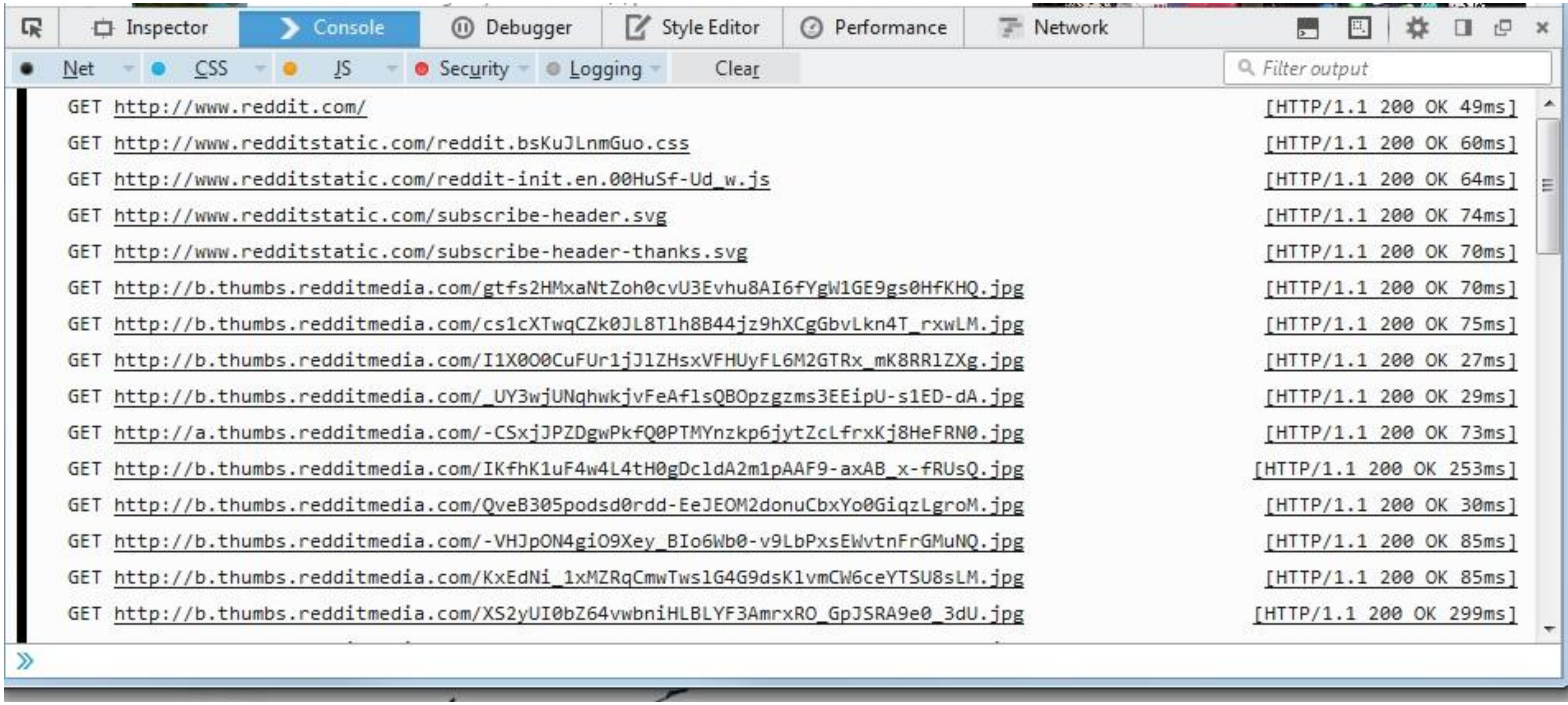
- Your proxy must be robust.
  - It must be able to handle bad requests/responses and other errors without quitting / crashing / terminating.
  - CS:APP code is provided to you in the lab handout. You are welcome to use them however you lik. But you might want to consider writing your own wrappers that do not immediately quit on error.

Step 1: Sequential Proxy

- Get a sequential proxy working first.
- Proxy listen()s in a loop for clients
- Handles exactly one client at a time
- Once it finishes the client’s request, it will handle the next client’s request.
- This is slow in real-life today

Step 2: Concurrent Proxy

- **Every page that you browse in a website has dozens, if not hundreds, of elements**
  - HTML, CSS, images, scripts, videos, etc.



```
GET http://www.reddit.com/                                                              [HTTP/1.1 200 OK 49ms]
GET http://www.redditstatic.com/reddit.bsKuJLnmGuo.css                                  [HTTP/1.1 200 OK 60ms]
GET http://www.redditstatic.com/reddit-init.en.00HuSf-Ud_w.js                           [HTTP/1.1 200 OK 64ms]
GET http://www.redditstatic.com/subscribe-header.svg                                    [HTTP/1.1 200 OK 74ms]
GET http://www.redditstatic.com/subscribe-header-thanks.svg                             [HTTP/1.1 200 OK 70ms]
GET http://b.thumbs.redditmedia.com/gtfs2HMxaNtZoh0cvU3Evhu8AI6fYgW1GE9gs0HfKHQ.jpg     [HTTP/1.1 200 OK 70ms]
GET http://b.thumbs.redditmedia.com/cs1cXTwqCZk0JL8T1h8B44jz9hXCgGbvLkn4T_rxwLM.jpg     [HTTP/1.1 200 OK 75ms]
GET http://b.thumbs.redditmedia.com/I1X0O0CuFUr1jJ1ZHsxVFHUyFL6M2GTRx_mK8RR1ZXg.jpg     [HTTP/1.1 200 OK 27ms]
GET http://b.thumbs.redditmedia.com/_UY3wjUNqhwkjvFeAflsQBOpzgzms3EEipU-s1ED-dA.jpg     [HTTP/1.1 200 OK 29ms]
GET http://a.thumbs.redditmedia.com/-CSxjJPZDgwPkfQ0PTMYnzkp6jytZcLfrxKj8HeFRN0.jpg     [HTTP/1.1 200 OK 73ms]
GET http://b.thumbs.redditmedia.com/IKfhK1uF4w4L4tH0gDcldA2m1pAAF9-axAB_x-fRUsQ.jpg    [HTTP/1.1 200 OK 253ms]
GET http://b.thumbs.redditmedia.com/QveB305podsd0rdd-EeJEOM2donuCbxYo0GiqzLgroM.jpg     [HTTP/1.1 200 OK 30ms]
GET http://b.thumbs.redditmedia.com/-VHJpON4giO9Xey_BIo6Wb0-v9LbPxsEWvtnFrGMuNQ.jpg     [HTTP/1.1 200 OK 85ms]
GET http://b.thumbs.redditmedia.com/KxEdNi_1xMZRqCmwTws1G4G9dsK1vmCW6ceYTSU8sLM.jpg     [HTTP/1.1 200 OK 85ms]
GET http://b.thumbs.redditmedia.com/XS2yUI0bZ64vwbniHLBLYF3AmrxRO_GpJSRA9e0_3dU.jpg    [HTTP/1.1 200 OK 299ms]
```

Step 2: Concurrent Proxy

- **Every page that you browse in a website has dozens, if not hundreds, of elements**
  - HTML, CSS, images, scripts, videos, etc.
- **Requesting and waiting for all of these elements one at a time takes too much time.**
- **Answer is concurrency!**

Step 2: Concurrent Proxy

- **You had a taste of concurrency during the shell lab**
  - Your shell could run more than on job at once, as many as needed
- **Sort of similar setup in proxy lab except instead of executing several commands at once, you will be handling several clients / requests at once**
- **fork() is not optimal for webserver performance**
  - You clone the proxy server’s whole memory space and process every time
  - It used to be acceptable many years ago, not anymore
- **Now we use threads! (And so will you)**

Step 3: Caching

- **Once your proxy can handle concurrency, you need to add a cache for web objects (not the same kind of cache as cachelab)**
  - Maximum cache size: 1 MiB (1,024,576 bytes)
  - Maximum object size: 100 KiB (102,400 bytes)
- **If object is too big, don't cache it.**
- **If there is not enough space (cache is full), you will need to evict objects from the cache.**
  - Should use a least-recently-used (LRU) eviction policy, or at least something approximately close to it
  - LRU: Evict the object that was last used
  - Reading the object counts as using it
- **Cache has to ensure consistency during concurrent access.**
  - Can't just overwrite / delete an object in one thread while another is copying it to a client.

Things To Watch Out For

- **Race conditions!**
  - Shell lab only ever had two agents touching the job list so it wasn't as complicated to solve
  - Proxy lab can involve an arbitrarily high amount of threads trying to access the cache. Make sure your locking scheme prevents the cache from getting clobbered.
- **SIGPIPE**
- **Memory leaks**
  - Keep close track of your buffers and memory usage amongst threads.
  - 1 thread that leaks 256 bytes doesn't sound bad. 1,000,000 threads that leak 256MB does.
  - Web servers / proxies are expected to run for long periods of time. Memory usage over time is important!
- **Deadlocks**
  - If every thread waits for access to the cache without a way to know when their turn comes up, the proxy will hang forever.

Proxy Lab Tips

- **Start early**
  - It’s the first “real” concurrency lab. You will probably need a lot of time to wrap your head around the implementation specifics.
  - Lab design is also very open-ended. Like malloc(), there is no “perfect” solution – there are multiple ways to approach / solve the problem. Your proxy and cache design is entirely up to you.
- **Plan everything out before writing code.**
  - As some of you probably experienced for malloc(), on-the-fly debugging and fixing take up a lot of time.
- **Make room for debugging infrastructure in your designs**
- **Make use of debugging tools provided to you in the handout and on the Shark machines**
  - tiny server, curl, netcat, telnet, version control, etc.

Questions?

(come to office hours if you need help)