

Recitation 12: ProxyLab Part 1

Instructor: TA(s)

Outline

- **Feedback**
- **Proxies**
- **Networking**
- **PXYDRIVE Demo**

Reminder:

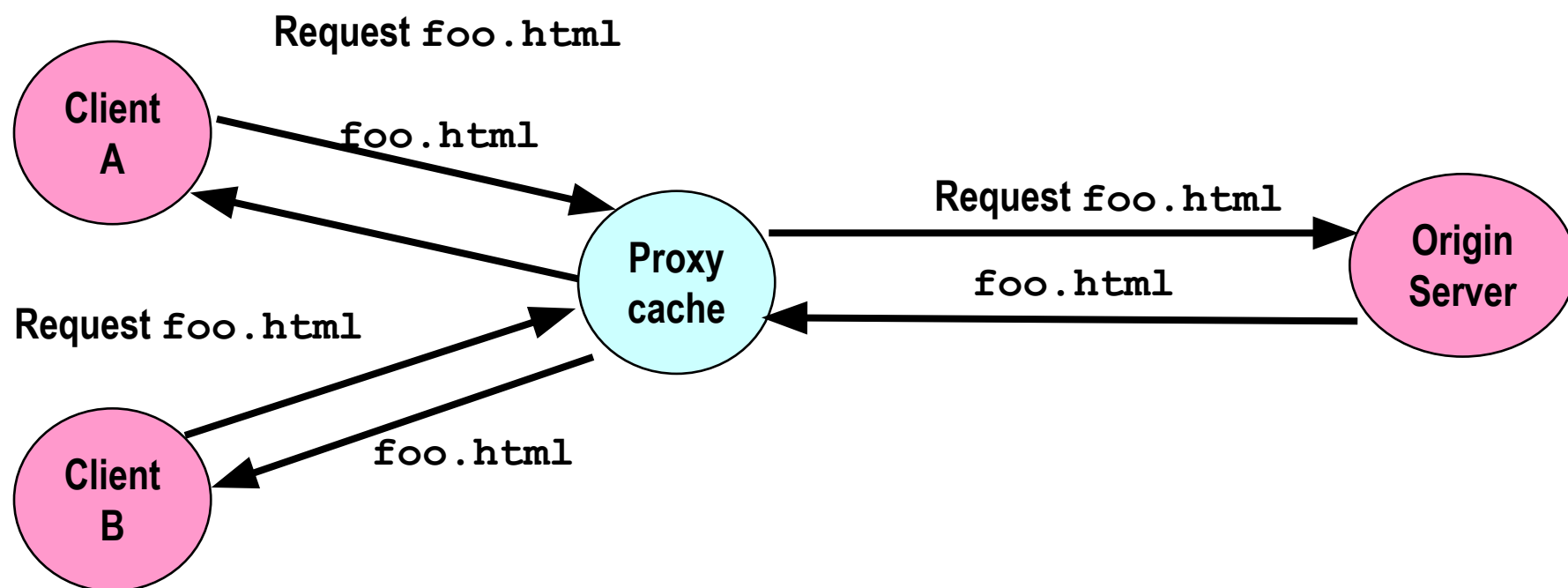
- Please sign up for code reviews for Shell Lab by Thursday

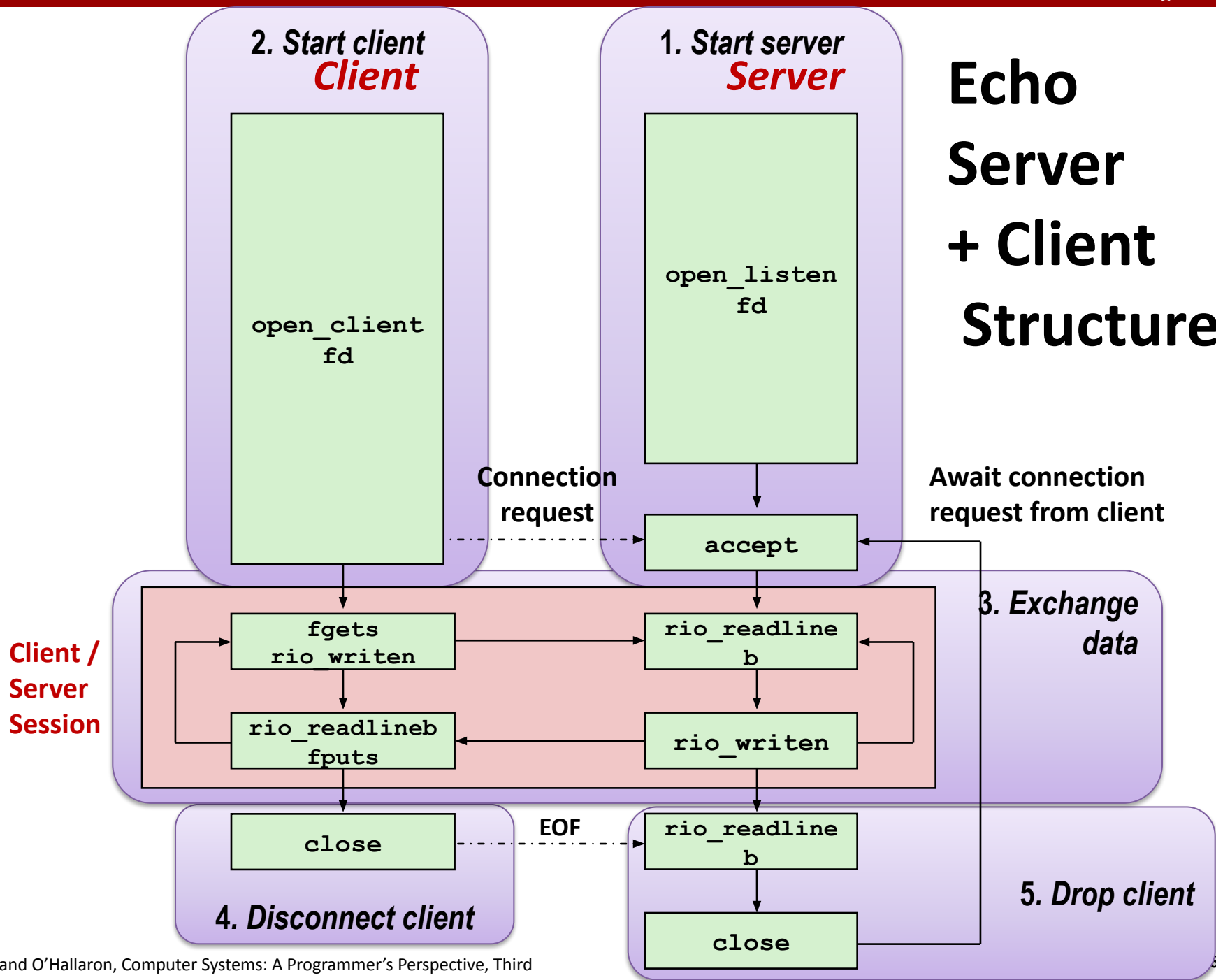
Proxy Lab

- Checkpoint is worth 2%, due Tuesday, November 23rd
- Final is worth 6%, due Thursday, December 2nd
- Current situation w/ grace / late days (subject to change):
 - 1 grace / late day allowed for both checkpoint and final
- You are submitting an entire project
 - Modify the makefile
 - Split source file into separate pieces
- Submit regularly to verify proxy builds on Autolab
- Your proxy is a server, it should not crash!

Why Proxies?

- Proxies are both clients and servers
- Can perform useful functions as requests and responses pass by
 - Examples: Caching, logging, anonymization, filtering, transcoding

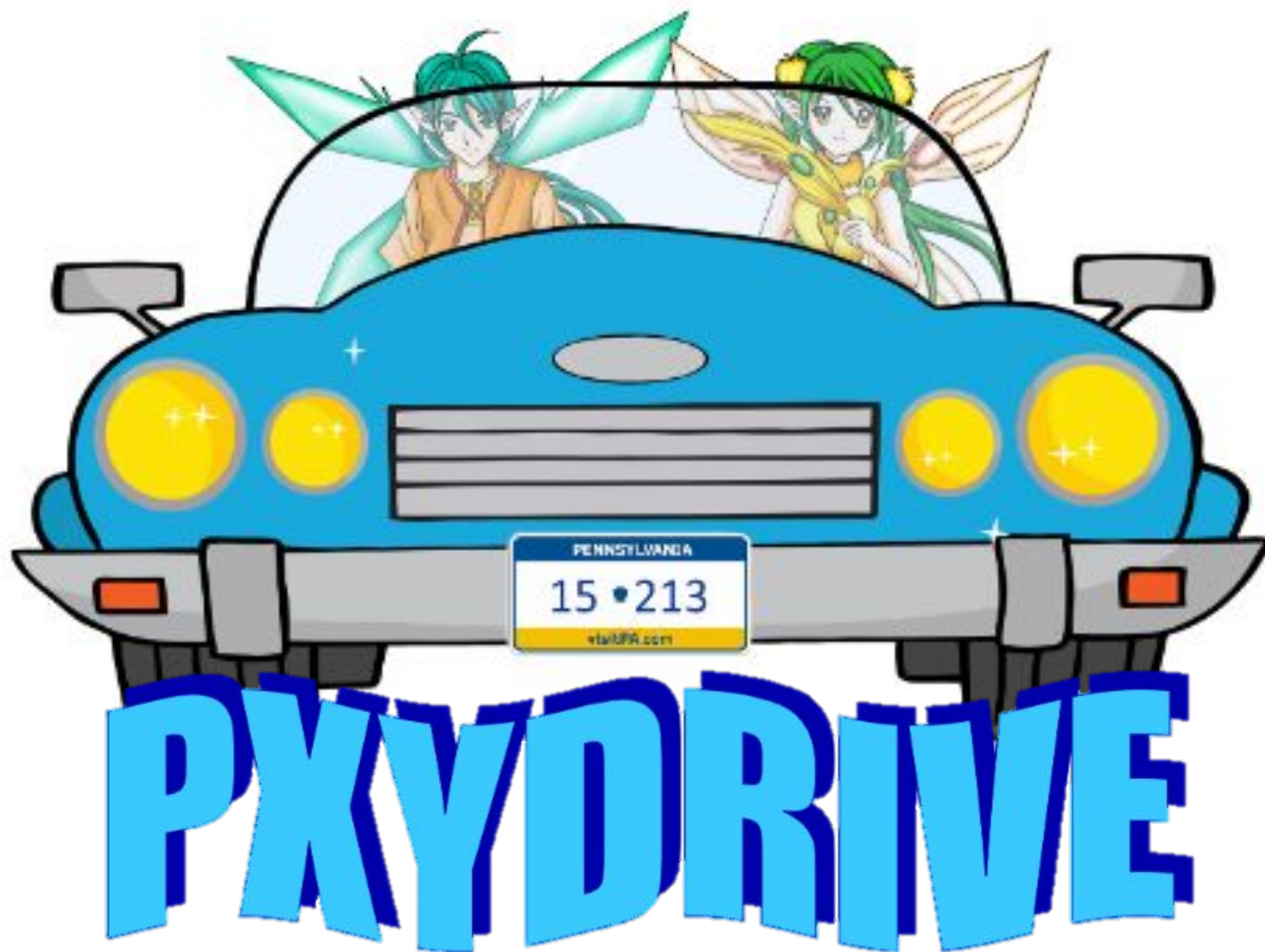


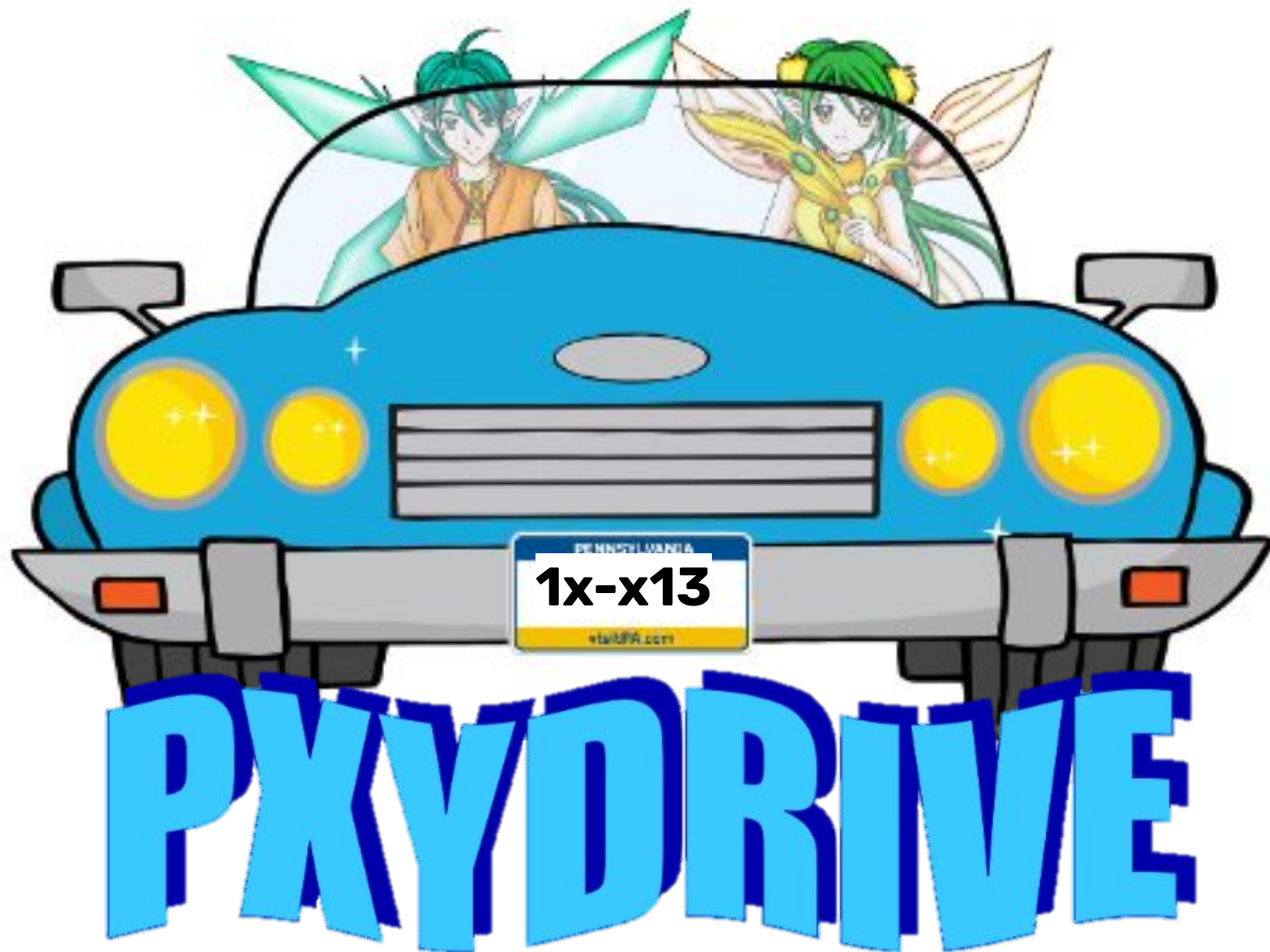


Transferring HTTP Data

If something requests a file from a web server, how does it know that the transfer is complete?

- A) It reads a NULL byte.**
- B) The connection closes.**
- C) It reads a blank line.**
- D) The HTTP header specifies the number of bytes to receive.**
- E) The reading function receives EOF.**





Introducing PXYDRIVE¹

- **A REPL for testing your proxy implementation**
 - We also grade using this
- **Typical pre-f18 proxy debugging experience:**
 - Open up three terminals:
for Tiny server, **gdb proxy** and curl
 - Can make multiple requests, but need more terminals
for multiple instances of the Tiny server
 - If the data is corrupted, need to manually inspect lines
of gibberish binary data to check error
- **Not anymore with PXYDRIVE!**

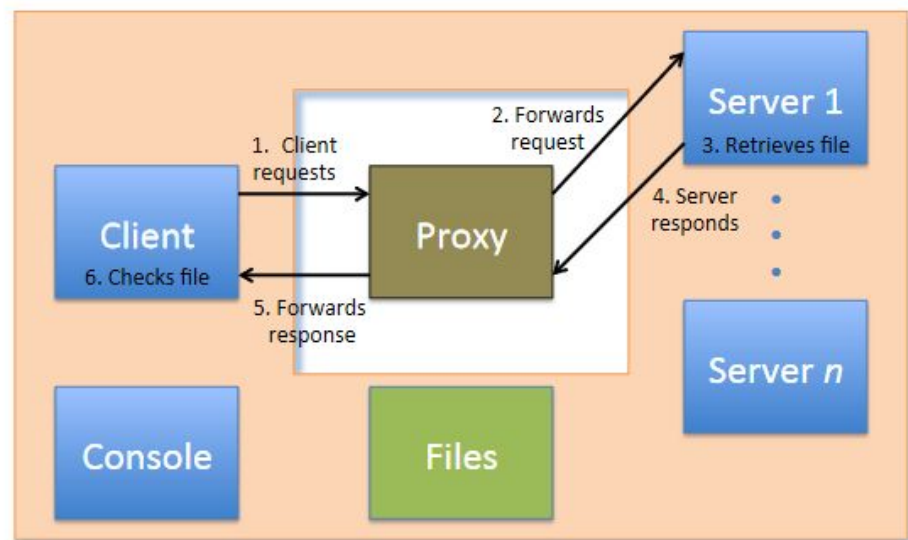
¹ Not typing PXYDRIVE in small-caps is a style violation.

Introducing PXYDRIVE

■ General workflow

- Generate text and binary data to test your proxy with
- Create (multiple) server
- Make **transactions**
- Trace transactions to inspect headers and response data

■ Transaction



Some practice

- Get the tarball
- `$ wget`
<https://www.cs.cmu.edu/~213/activities/proxy-recitation13.tar>
- `$ tar -xvf proxy-recitation13.tar`
- `$ cd pxydrive-tutorial`

Trying out PXYDRIVE

- It's a REPL: the user can run commands
- `$./pxy/pxydrive.py`
 - Just starts PXYDRIVE
 - Try entering commands:
 - `>help`
 - `>help help help help help help...`
 - `>quit`
- `$./pxy/pxydrive.py -p ./proxy-ref`
 - Starts PXYDRIVE and specifies a proxy to run
 - **Proxy set up at <someshark>:30104**
 - Picks the right port and starts the proxy
 - `./proxy-ref` is the reference proxy

PXYDRIVE Tutorial 1

- **Introducing basic procedures:**
generate data, create server, fetch / request file from server,
trace transaction
- **Open `s01-basic-fetch.cmd`**

PXYDRIVE Tutorial 1

- **>generate data1.txt 1K**
 - Generates a 1K text file called *data1.txt*
- **>serve s1**
 - Launches a server called *s1*
- **>fetch f1 data1.txt s1**
 - Fetches *data1.txt* from server *s1*, in a transaction called *f1*
- **>wait ***
 - Waits for all transactions to finish
 - Needed in the trace, not in the command-line
- **>trace f1**
 - Traces the transaction *f1*
- **>check f1**
 - Checks the transaction *f1*

PXYDRIVE Tutorial 1

- Run trace with `-f` option:
- `$./pxy/pxydrive.py -f s01-basic-fetch.cmd -p
./proxy-ref`

Look at the trace of the transaction!

- **Identify:**
 - **GET command**
 - **Host header**
 - **Other headers**
 - **Request from client to proxy**
 - **Request from proxy to server**
 - **Response by server to proxy**
 - **Response by proxy to client**

PXYDRIVE Tutorial 1

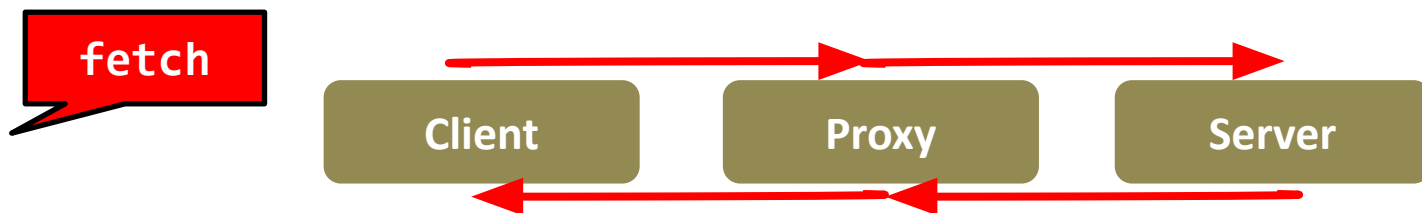
- Run a different trace
- `$./pxy/pxydrive.py -f s02-basic-request.cmd -p ./proxy-ref`
- You should get a different output from the first trace
- Why? Let's look at this trace...

PXYDRIVE Tutorial 1

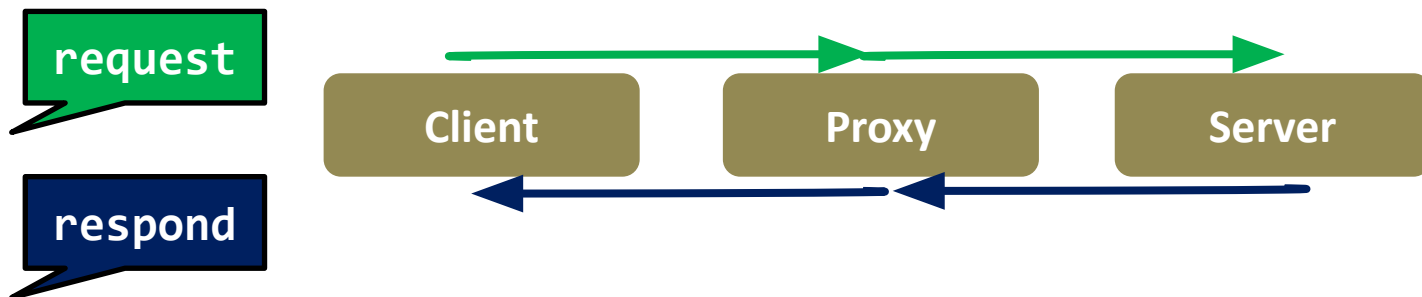
- `>generate data1.txt 1K`
- `>serve s1`
- `>request r1 data1.txt s1`
 - Requests *data1.txt* from server *s1*, in a transaction called *r1*
- `>wait *`
- `>trace r1`
- `>respond r1`
 - Allow server to respond to the transaction *r1*
- `>wait *`
- `>trace r1`
- `>check r1`
 - Checks the transaction *r1*

PXYDRIVE Tutorial 1

- The fetch command makes the server immediately respond to a request.
- All steps of a transaction are complete after a fetch.



- The request command does not complete a transaction.
- A request needs a respond to complete its transaction.



PXYDRIVE Tutorial 2

- Debugging a proxy that clobbers responses
- Run the same trace but with a faulty proxy
- `$./pxy/pxydrive.py -f s01-basic-fetch.cmd
-p ./proxy-corrupt`

What went wrong?

```
Response status: ok
Source file in ./source_files/random/data1.txt
Request status: error (Mismatch between source file ./source_files/random/data1
.txt and response file ./response_files/f1-data1.txt starting at position 447: '
F' (hex 0x46) ≠ 'G' (hex 0x47))
Result file in ./response_files/f1-data1.txt
>#
># Make sure it was retrieved properly
>check f1
ERROR: Request f1 generated status 'error'. Expecting 'ok' (Mismatch between so
urce file ./source_files/random/data1.txt and response file ./response_files/f1-
data1.txt starting at position 447: 'F' (hex 0x46) ≠ 'G' (hex 0x47))
>quit
ERROR COUNT = 1
-bash-4.2$
```

PXYDRIVE Tutorial 3

- Debugging a proxy that clobbers headers
- Run the same trace but with another faulty proxy
- `$./pxy/pxydrive.py -f s01-basic-fetch.cmd
-p ./proxy-strip -S 3`
- -S specifies strictness level

What went wrong?


```
Response status: bad_request (Missing Request-ID header)
Source file in ./source_files/random/data1.txt
Request status: bad_request (Bad request)
Result file in ./response_files/f1-status.html
>#
># Make sure it was retrieved properly
>check f1
ERROR: Request f1 generated status 'bad_request'.  Expecting 'ok' (Bad request)
>quit
ERROR COUNT = 1
-bash-4.2$ _
```


PXYDRIVE Tutorial 4

- Debugging a proxy that crashes
- Run the same trace but with yet another faulty proxy
- `$./pxy/pxydrive.py -f s03-overrun.cmd
-p ./proxy-overrun`
- Is the error message helpful?

PXYDRIVE Tutorial 4

- We resort to multi-window debugging
- Set up another window and run GDB in one:
 - `$ gdb ./proxy-overflow`
 - `(gdb) run <port>`
- In the other window, run PXYDRIVE:
 - `$./pxy/pxydrive.py -P localhost:<port> -f s03-overflow.cmd`
 - `-P` specifies the host and port the proxy is running on



`./port-for-user.pl`
Run this to get your
unique port!

Reminders

- **Read the writeup**
- **One grace / late day for both checkpoint and final**
- **So you really have to start early**
 - Come to office hours this week, before it gets crowded!
- **Work incrementally and take breaks**
- **Simpler tests should be completed in the first week!**

So you wanna TA for 213?



What qualifications are we looking for?

- Decent class performance, but also critical thinking skills
- Like computer systems + want to help others like systems!
- Have a reasonable ability to gauge your schedule + responsibilities
- Leadership potential! Take initiative, we love to see it 😊
- Ability to tell students:
 - “Did you write your heap checker”
 - “Run backtrace for me”
 - rinse and repeat, it’s mouthwash baby

Appendix on echoserver / client

Echo Demo

- See the instructions written in the telnet results to set up the echo server. Get someone nearby to connect using the echo client.
- What does echoserver output? (Sample output:)

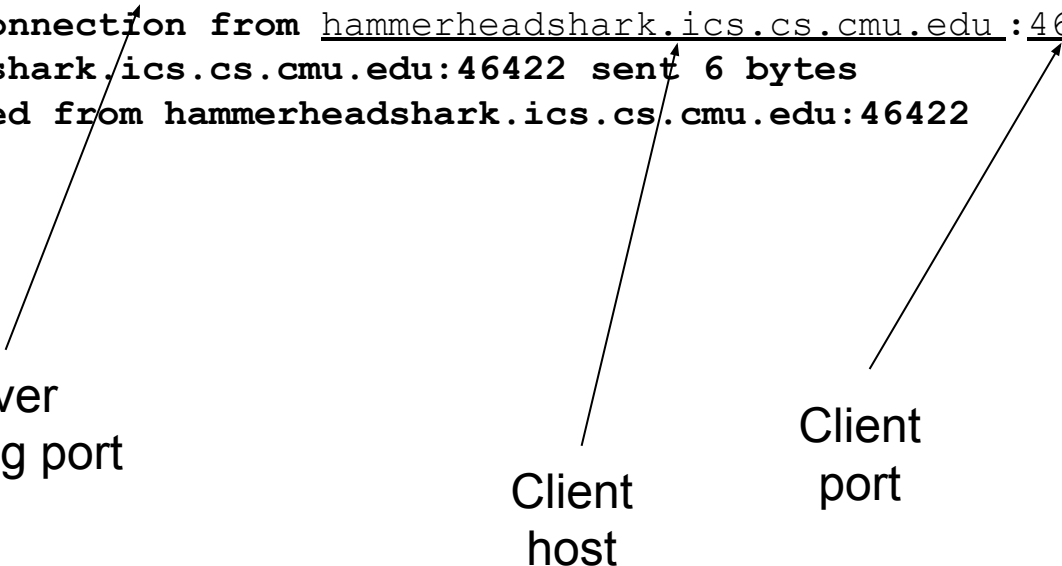
```
$ ./echoserver 10101
```

```
Accepted connection from hammerheadshark.ics.cs.cmu.edu:46422
```

```
hammerheadshark.ics.cs.cmu.edu:46422 sent 6 bytes
```

```
Disconnected from hammerheadshark.ics.cs.cmu.edu:46422
```

Server
listening port



Client
host

Client
port

Echo Demo

■ Look at echoclient.c

- Opens a connection to the server
- Reads/writes from the server

■ Look at echoserver output

- Why is the printed client port different from the server's listening port?
- Server opens **one** “**listening**” port
 - Incoming clients connect to this port
- Once server **accepts** a connection, it talks to client on a **different** “**ephemeral**” port



Echo Demo

- **Try to connect two clients to the same server.**
- **What happens?**
 - Second client has to wait for first client to finish!
 - Server doesn't even accept second client's connection
 - Where/why are we getting stuck?
- **Because we're stuck in `echo()` talking to the first client, echoserver can't handle any more clients**
- **Solution: multi-threading**

Echo Server Multithreaded

- How might we make this server multithreaded?
(Don't look at echoserver_t.c)

```
while (1) {  
    // Allocate space on the stack for client info  
    client_info client_data;  
    client_info *client = &client_data;  
  
    // Initialize the length of the address  
    client->addrlen = sizeof(client->addr);  
  
    // Accept() will block until a client connects to the port  
    client->connfd = Accept(listenfd,  
        (SA *) &client->addr, &client->addrlen);  
  
    // Connection is established; echo to client  
    echo(client);  
}
```

Echo Server Multithreaded

- **echoserver_t.c isn't too different from echoserver.c**
 - To see the changes: ``diff echoserver.c echoserver_t.c``
- **Making your proxy multithreaded will be very similar**
- **However, don't underestimate the difficulty of addressing race conditions between threads!**
 - Definitely the hardest part of proxylab
 - More on this next time...