ELEC / COMP 177 – Fall 2013

Computer Networking
→ Network Tester (Project 3 and 4)

Some slides from Kurose and Ross, *Computer Networking*, 5th Edition

Upcoming Schedule

- **Presentation 2** – Privacy/Security
  - Discuss requirements…
  - Topic Approval – **Tuesday, Nov 5th**
  - Presentations – **Nov 12th and Nov 14th**
    - Upload slides to Sakai by midnight on Nov 11th

Upcoming Schedule

- **Project 3 – Network Tester, Part One**
  - Work day: Tuesday Oct 29th
  - **Due Thursday, October 31st by 11:55pm**

Lab Practical Dates

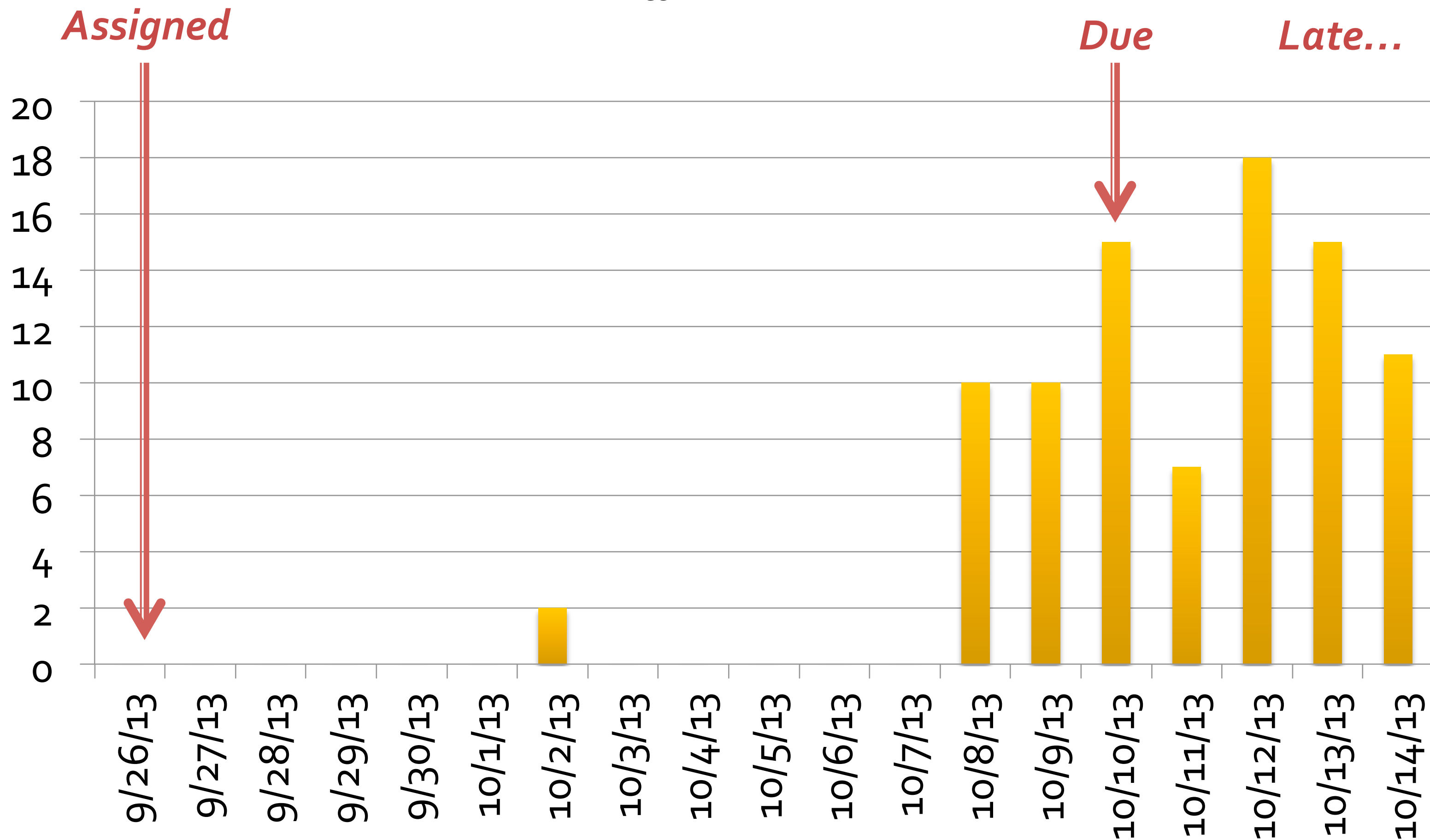
- Class schedule:
  - http://ecs-network.serv.pacific.edu/ecpe-177/schedule
- Problem:
  - Lab practical is day before Thanksgiving break
  - Subsequent week is senior project presentations during our lab time
- Proposal:
  - Cancel Lab 10 and replace with Lab Practical
  - Start at 1pm (early) and run until finished (6pm?)
  - Advantage: All done with Lab on Nov 12th

Thoughts on Project 2…

Thoughts on Project 2

- Project 1 went very easily…
  - *Much easier than last year, when we did socket programming in C*
- **Lulled you into complacency?**

Complacency
(Measured in Questions / Day)



I’m not going to spoon-feed you code snippets for every problem in advance of the project deadline

You need time to rewrite or re-factor your solution in order to fix bugs

Timeouts

- **Will this work for a 30-second socket timeout?**
  - *Imagine it's inside your thread/process*
  - *time.time() is measured in seconds since "start of epoch"*

```
start = time.time()
while (time.time() - start) < 30:
   # Main HTTP loop
   # Call recv() to get request(s)
   # Pull off a single request / save extra for next loop
   # etc…
```

A nice idea, but NO…

The program will be blocked inside of `recv()`, waiting in vain for more data. You'll never get back to the while loop to check on `time.time()` again.

Timeouts / Exception Handling

- `my_socket.settimeout(30)`
- Generates a `socket.timeout` exception
  - I can be blocked on recv() waiting for client data
  - At some point, let's give up and consider this socket "dead" (close it and move on)

- Pitfall / confusion:
  - `socket.timeout` is a subset (specific example) of `socket.error`

Exception Handling

```
s.settimeout(30)

try:
   raw_data = s.recv()

except socket.timeout:
   print("Timeout on recv()")
   # Do something

except socket.error:
   print("General error on recv()")
   # Do something
```

Check for more specific exception before general exception…

Mysteries of recv()

- Consider the following line:
  - `raw_data = my_socket.recv(4096)`
- **Which of the following choices are valid outcomes?**

1. `raw_data` is exactly 4096 bytes?
2. `raw_data` is 0 bytes?
3. `raw_data` is between 0 and 4096 bytes?
4. `raw_data` is greater than 4096 bytes?

Mysteries of recv()

```
raw_data = my_socket.recv(4096)
```

- POSSIBLE - Result of **4096 bytes**
  - OS had "plenty" of data (perhaps more) and gave you the max amount you requested. Extra data is saved until next recv() call
- POSSIBLE - Result **between 0 and 4096** bytes
  - OS had "some" data, and gave you all it had
- POSSIBLE - Result of **0 bytes**
  - Other endpoint closed socket – no more data!
- NOT POSSIBLE - Result > 4096 bytes

Mysteries of recv()

```
raw_data = my_socket.recv(4096)
```

- **Will this function call in Project 2 give me**
  1. Exactly 1 HTTP request?
  2. Less than 1 HTTP request?
  3. More than 1 HTTP request?

You have NO GUARANTEE
Any of these events could happen!

Mysteries of recv()

```
raw_data = my_socket.recv(4096)
```

- You got **lucky** in Project 1
  - Web browser only sends 1 request at a time
  - That request was usually small enough to fix in 4096 bytes
  - You got the full 4096 bytes (or the complete client request) 99.9% of the time
- Things are **harder** in Project 2
  - The server is busier with multiple sockets (might get less data than a full request)
  - With pipelining, the client can send several requests at once (i.e. 4096 bytes can hold **several** requests)

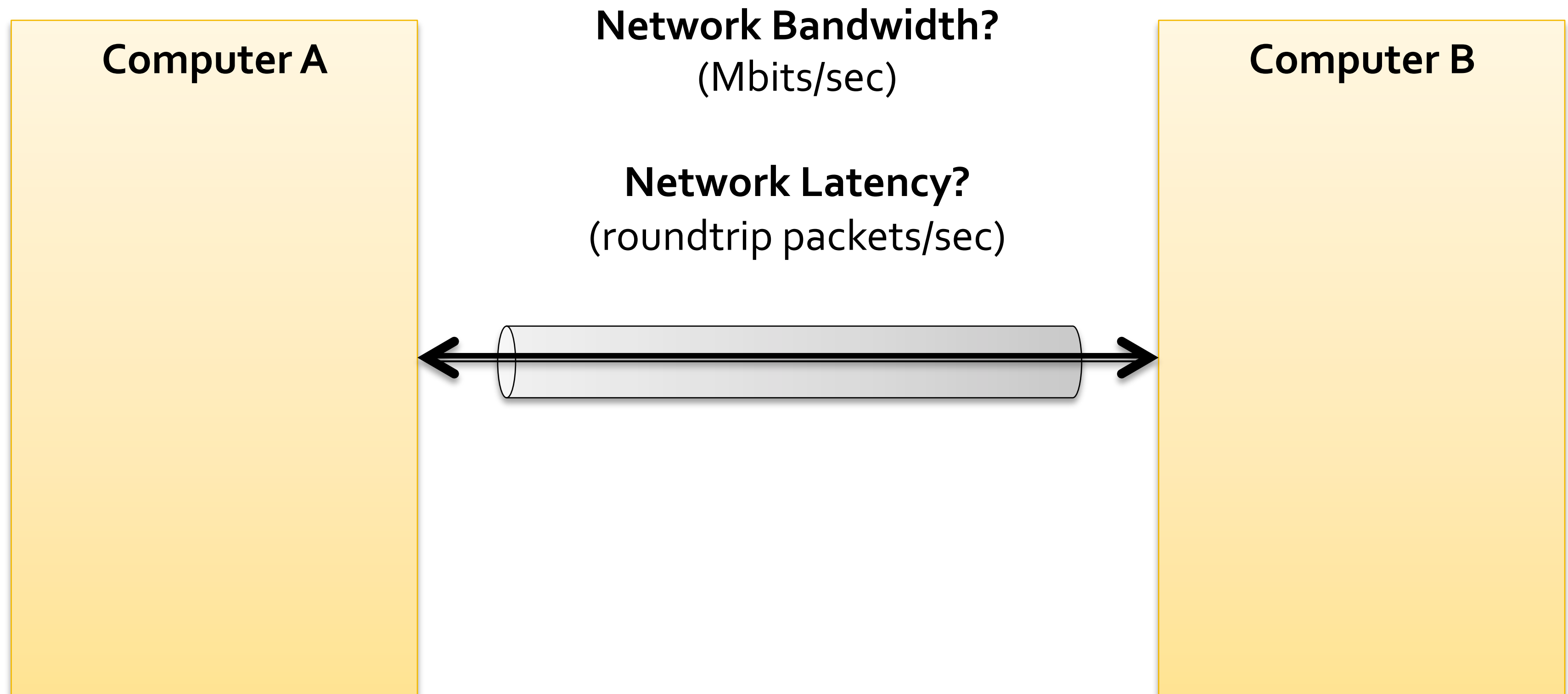
Mysteries of recv()

- So how do I get a single HTTP request then? *(and not less than 1 request, or more than 1 request?)*

Network Tester

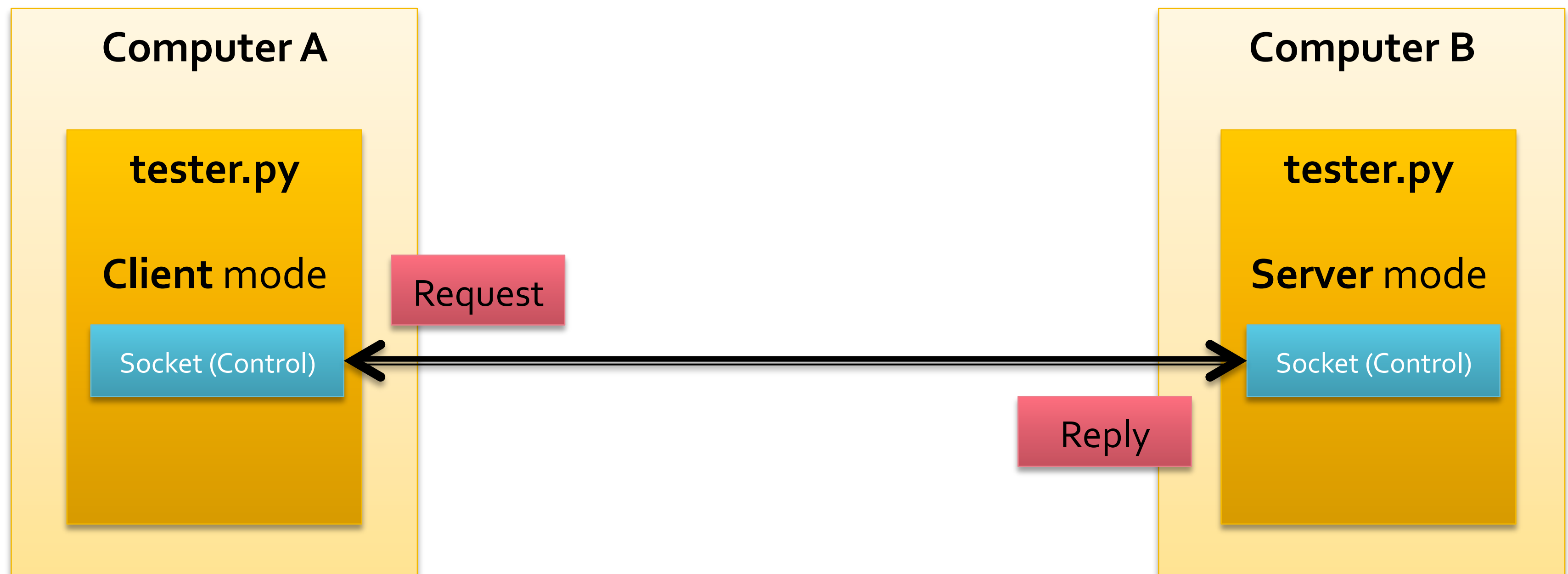
Project 3 and 4

Network



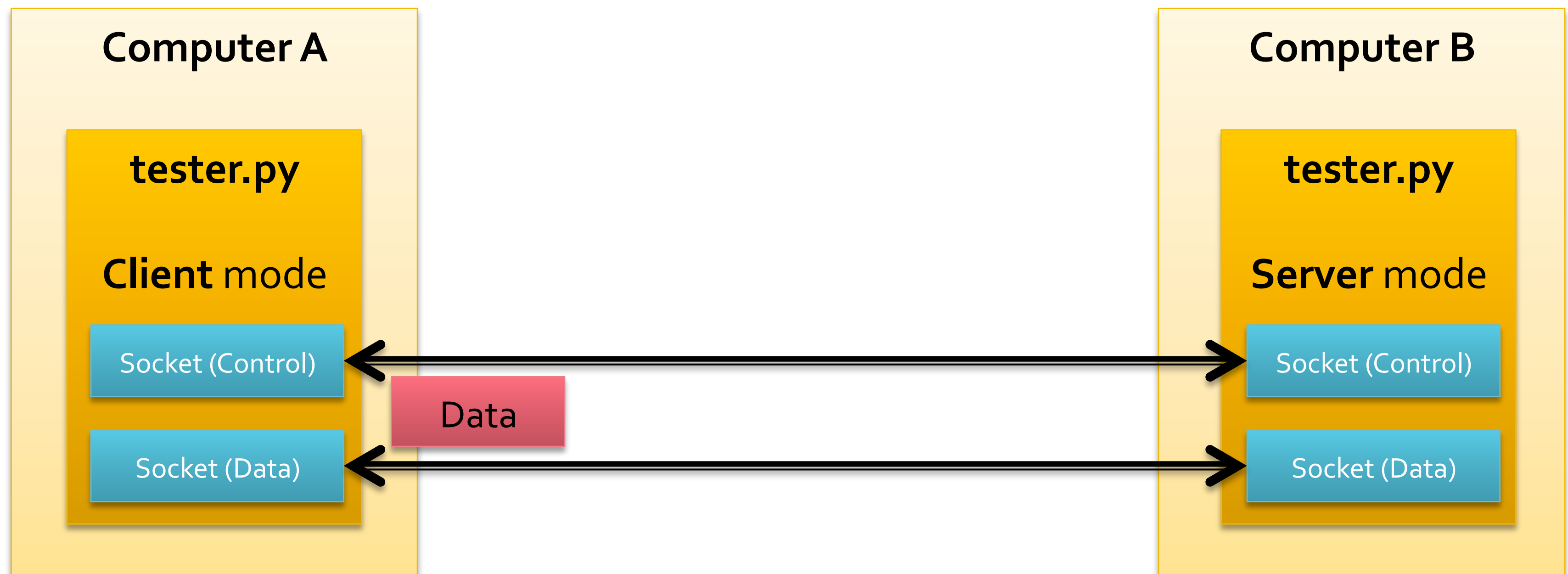
tester.py Benchmark Program

Showing TCP STREAM test…



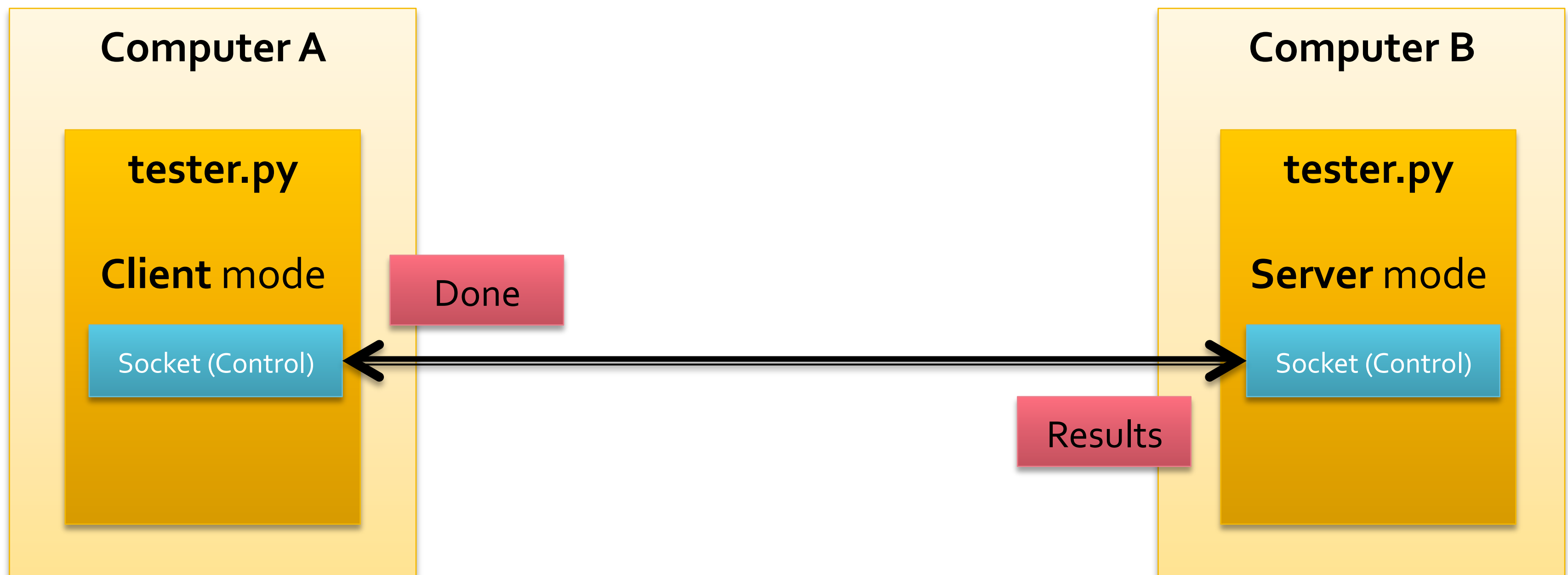
tester.py Benchmark Program

Showing TCP STREAM test…



tester.py Benchmark Program

Showing TCP STREAM test…



Demo

- TCP_STREAM demo
- TCP_RR demo
- Note how server stays running after a test finishes, waiting for more tests…
- Compare *localhost* test versus network test

Test: TCP_STREAM

- Client continuously sends TCP data to server
- Message size: 64kB
- Measurements:
  - Bandwidth: Megabits/sec

- Goal: Result should be limited by network bandwidth, not by crappy Python coding!
  - *I was able to get ~900Mbits/sec on gigabit Ethernet, which should normally see ~945Mbits/sec*

Test: UDP_STREAM

- Client continuously sends UDP data to server
- Message size: 1472 bytes
- Measurements:
  - Bandwidth: Megabits/sec
  - Packet loss (percentage of total packets sent)

Test: TCP_RR

- Client sends 1-byte TCP message to server
- Server sends 1-byte TCP message to client
- Repeat…
  - RR = “Round Robin”
- Measurements:
  - Roundtrip packets/sec

Test: UDP_RR

- Client sends 1-byte UDP message to server
- Server sends 1-byte UDP message to client
- Repeat…

- Measurements:
  - Roundtrip packets/sec
  - Packet loss
    - **What will happen if we lose a packet on this test?**

Parallel Programming

- Must use threads for this project for both client and server
  - Main (original) thread – control tasks
  - Child (new) thread – data tasks

- Why do we need it?
  - 1 problematic case: UDP_STREAM / UDP_RR test
  - How does the server know when the client has finished?

Parallel Programming

- **Does the server need to allow multiple clients to connect simultaneously and run tests?**

- Ans: **No**, that would distort the results of our bandwidth measurement, making the tool useless. Subsequent clients should wait for the first one to finish (serial, not parallel)