S

Now you are an undergraduate student at SUSTech, briefly share your experience, no more than 100 words.

As an undergraduate student at SUSTech, I have had the opportunity to gain a strong foundation in computer science and engineering. I have taken a range of courses in subjects such as algorithms, data structures, and computer systems, and I have also had the opportunity to gain hands-on experience through internships and research projects. I have found the program at SUSTech to be challenging and rewarding, and I am grateful for the support and guidance of my professors and peers.

S

Do you think CS305: Computer Networking is a good course at SUSTech? Share your opinion within 100 words.

As an undergraduate student at SUSTech, I have taken the course CS305: Computer Networking and I believe it is an excellent course. The course provides a comprehensive introduction to the field of computer networking, covering topics such as networking protocols, network architecture, and network security. The professors are knowledgeable and engaging, and the course includes hands-on activities and projects that provide students with practical experience. Overall, I have found the CS305: Computer Networking course to be a valuable and enriching experience.

Welcome

CS305 Project Tutorial Session

Dec.11, 2022

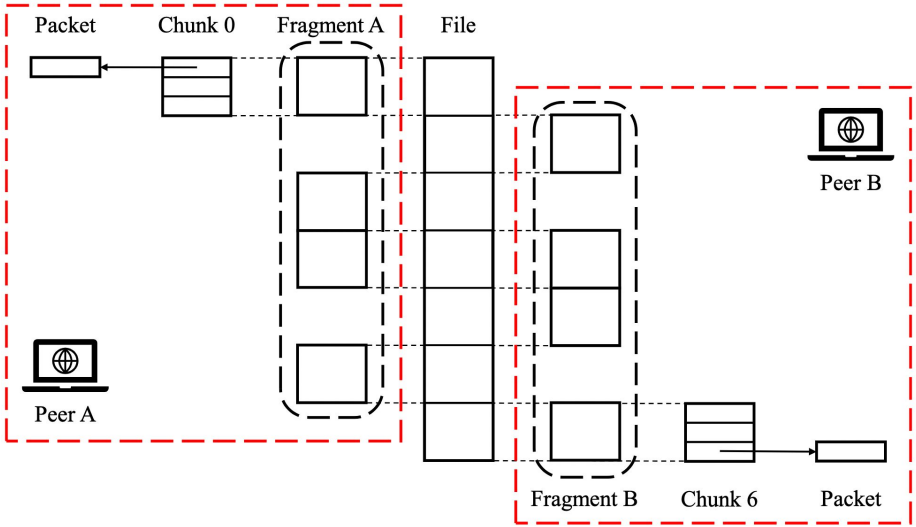
Sixu Tan

Content

1. Overview
    a. P2P
    b. Congestion Control
    c. Concurrency
2. Provided Code
    a. Skeleton
    b. Network Simulator
    c. Testing script
3. Grading
4. Bonus
5. FAQ
6. Q&A

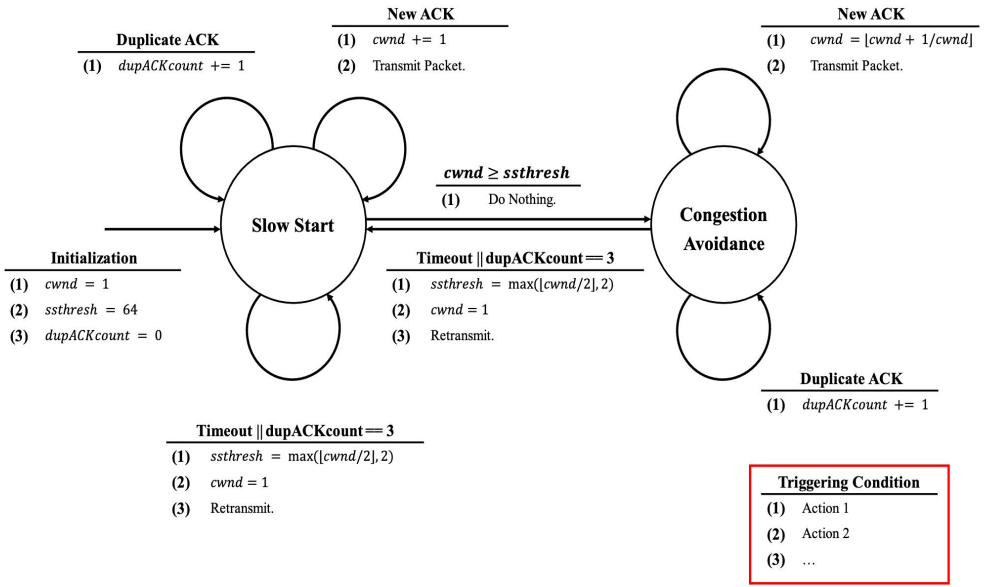
Content

Overview - P2P



- File is splitted into Chunks
- Each chunk is 512KB
- Each chunk is uniquely identified by its hash
- Each peer initially owns some chunks, named fragment
- A peer can download a chunk via a provided chunkhash

Overview - Congestion Control

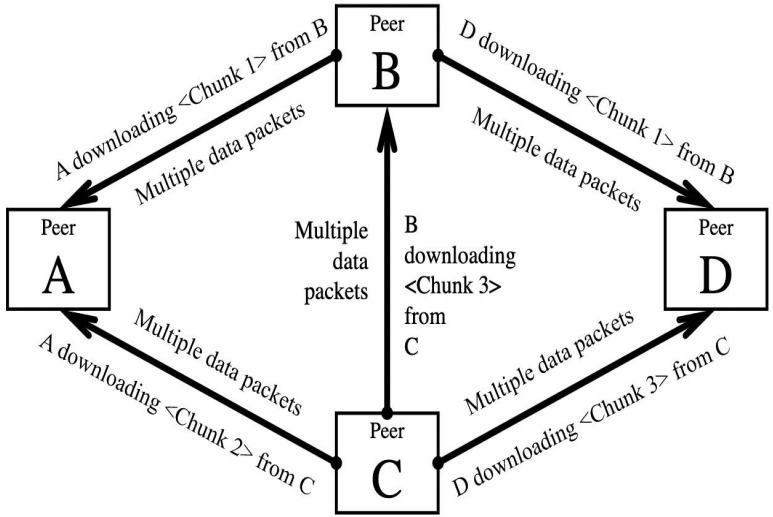


- window size is based on packet, not bytes.
- Ack the received Seq, not the wanted one.
- Only slow start, congestion avoidance, and fast retransmit. No fast recovery.
- Fast retransmit will be triggered at most **once** for a certain packet.

Overview - Concurrency

- **Concurrent**: You are watching a movie. Your boss urges you to work. You pause the movie, and go to work. After you finish the work, you resume the movie. This is concurrency.
- **Parralel**: You do the work while watching a movie.

- In this project, you need to implement concurrency.
- You need to concurrently download multiple chunks from multiple peers.
- You need to concurrently send chunks to multiple different peers.



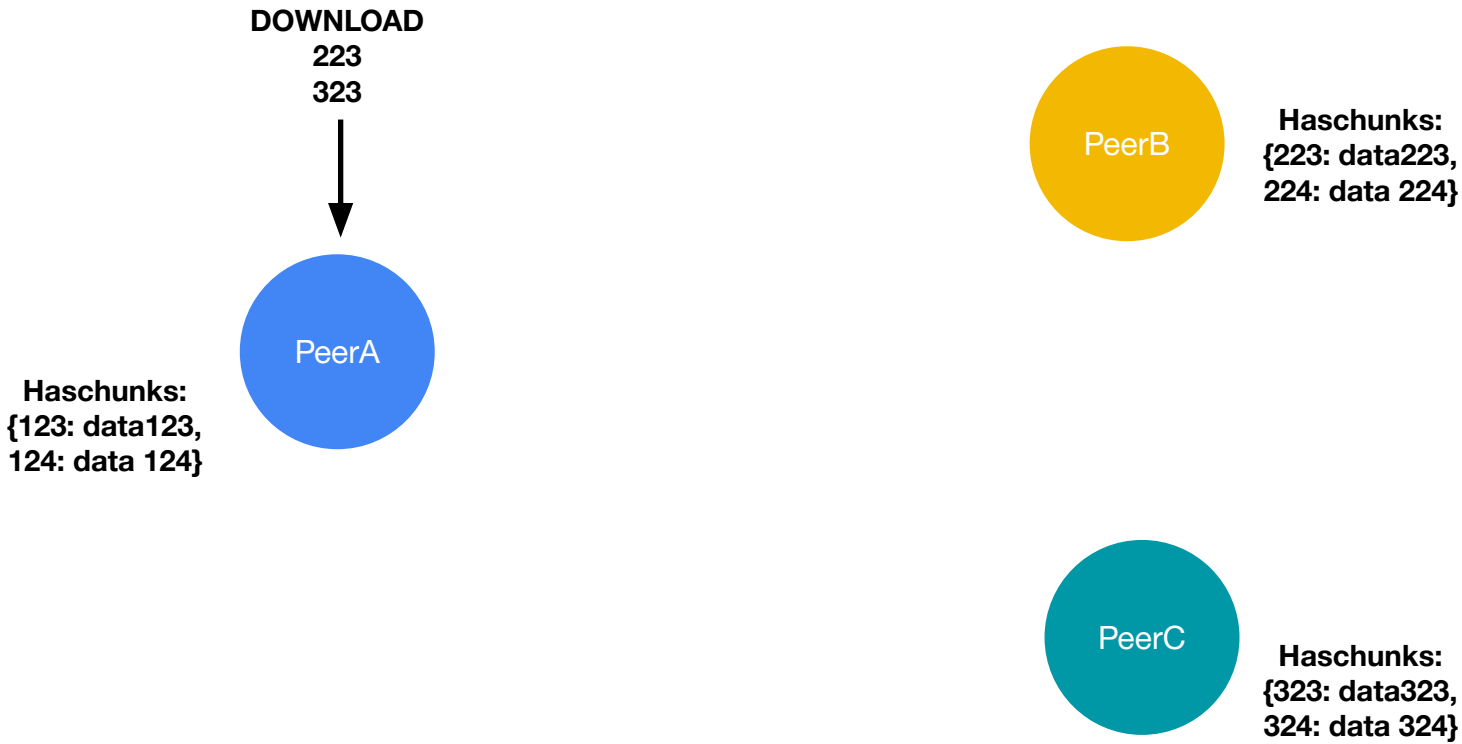
Overview - example

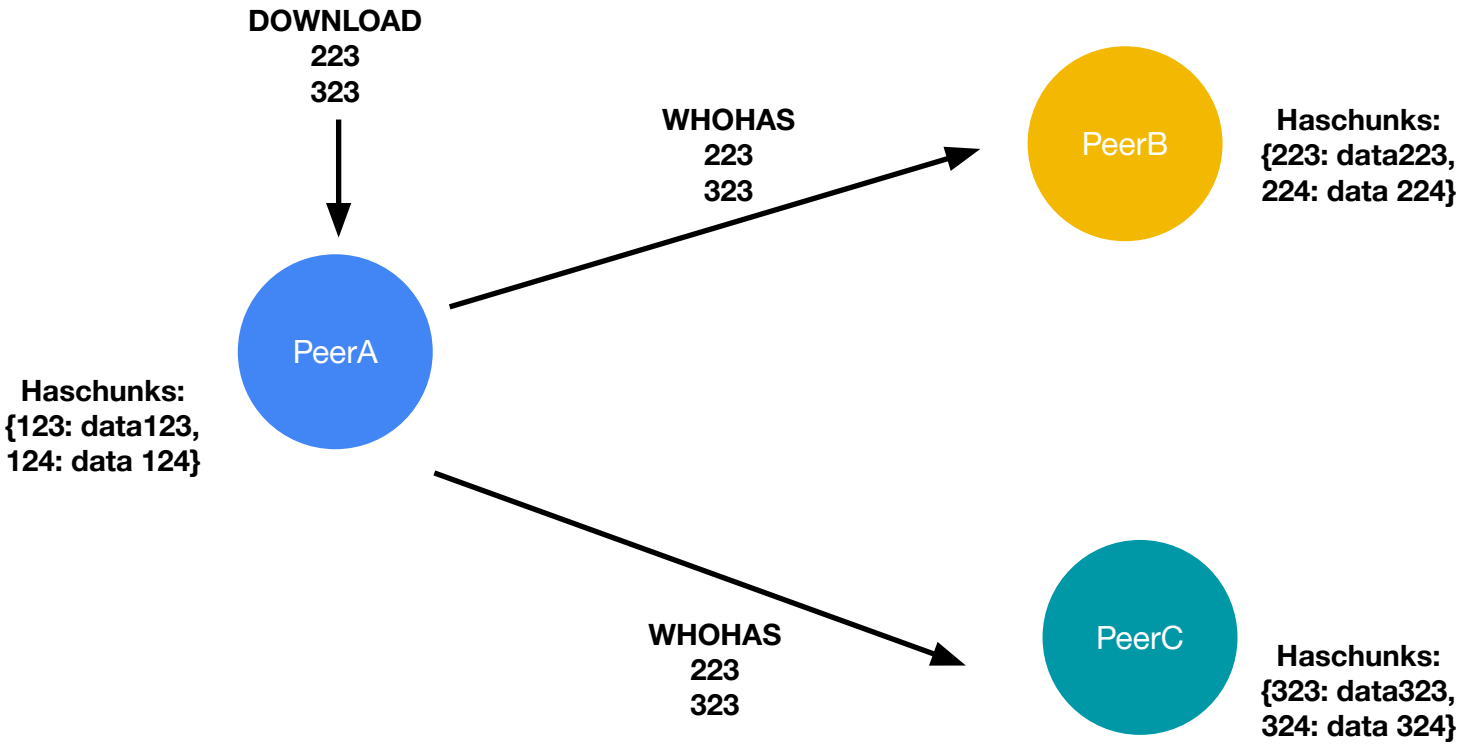
Haschunks:
{123: data123,
124: data 124}

Haschunks:
{223: data223,
224: data 224}

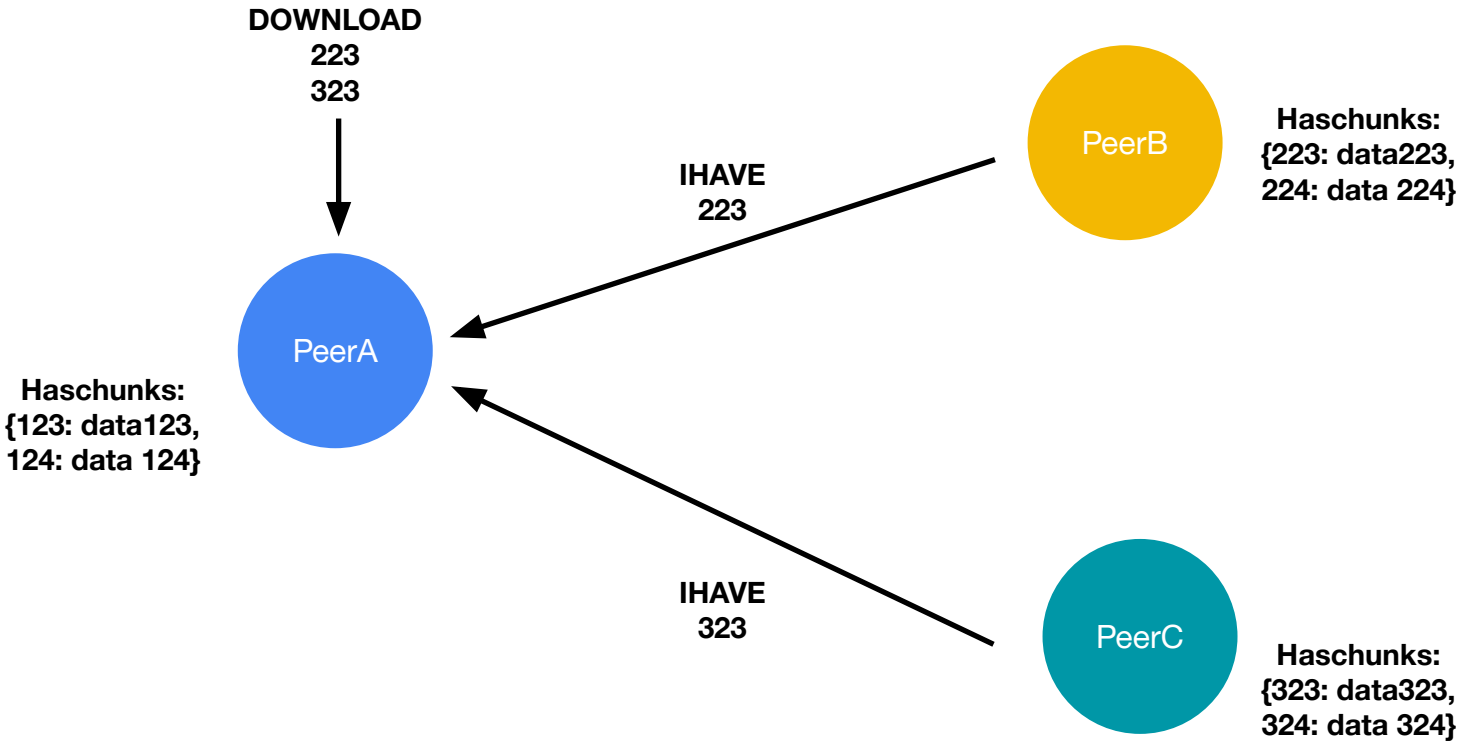
Haschunks:
{323: data323,
324: data 324}

Overview - example

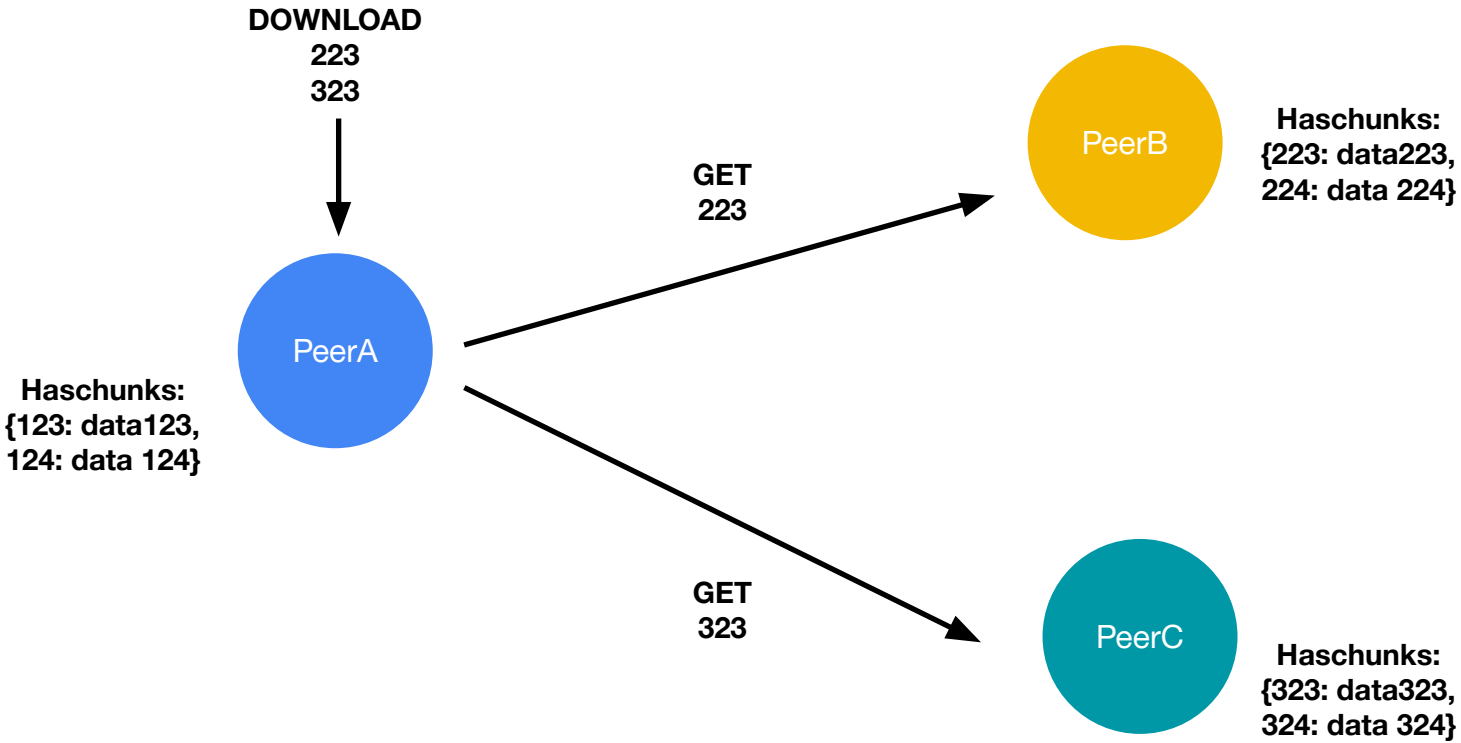



DOWNLOAD
223
323
PeerA
Haschunks:
{123: data123,
124: data 124}
WHOHAS
223
323
PeerB
Haschunks:
{223: data223,
224: data 224}
WHOHAS
223
323
PeerC
Haschunks:
{323: data323,
324: data 324}

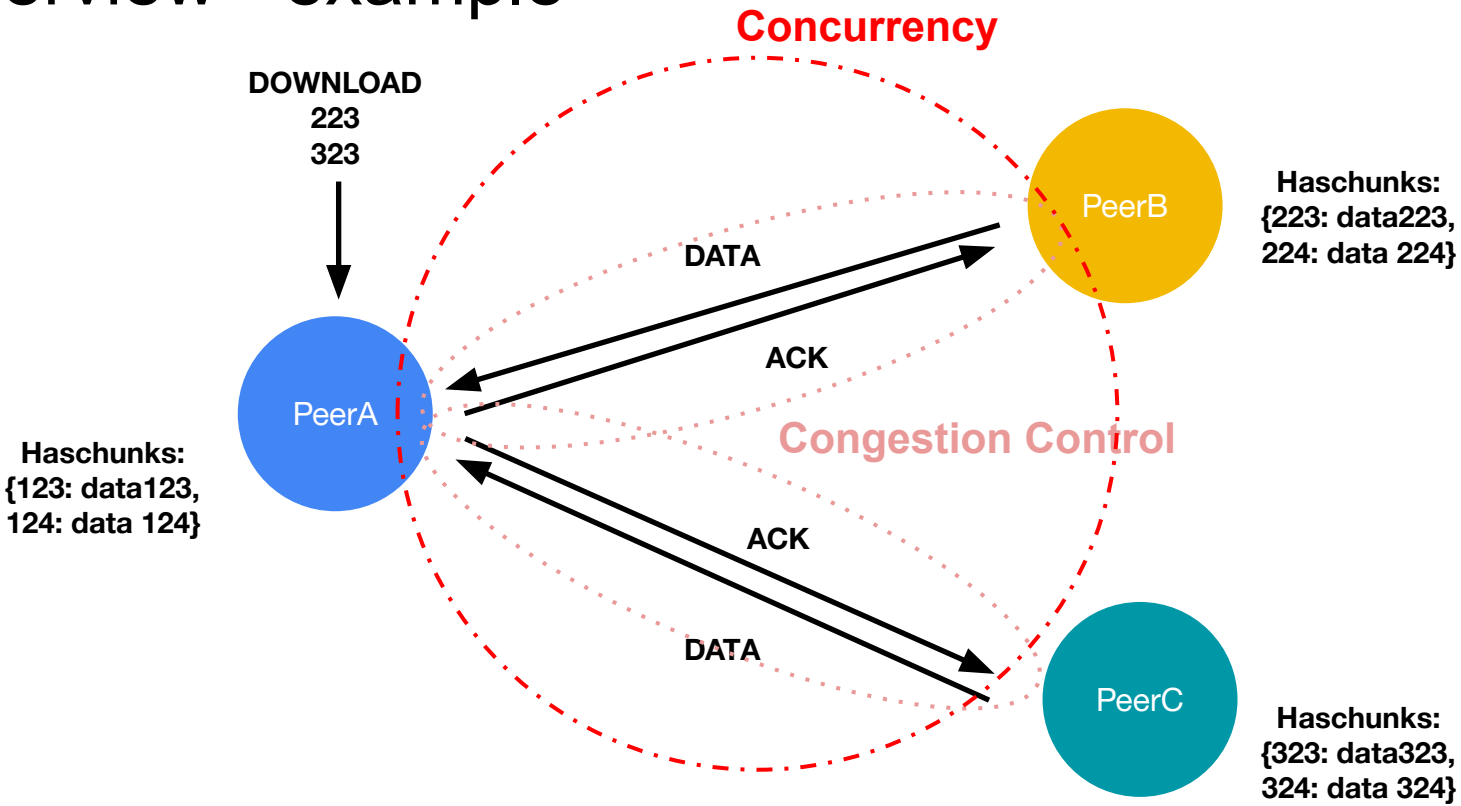
Overview - example



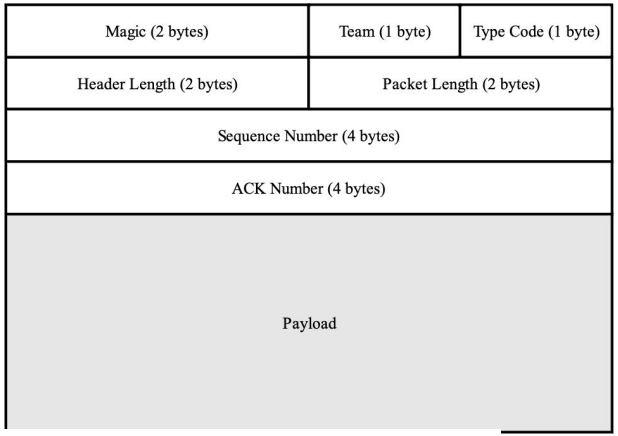
Overview - example



Overview - example



Overview - Packet Header



Packet Type	Type Code
WHOHAS	0
IHAVE	1
GET	2
DATA	3
ACK	4
DENIED	5

Fig. 1.3. Type codes

- Magic: Any (was for error detection)
- Team: You team number
- Extension of header is allowed, however, you cannot modify the existing fields.

Overview - Tasks

- P2P-like handshaking and RDT
- Congestion Control
- Concurrency and Robustness
- (Optional) Optimization

Content

Provided Code - File format

- chunkdata: 512KB of data
- chunkhash: 20 bytes sha-1 hash value of a certain chunkdata
- *.fragment: Serialized dictionary {chunkhash: chunkdata}
- *.chunkhash: Files that only contain chunkhashes: ex. master.chunkhash, download.chunkhash

```
class BtConfig:
    output_file: str                  # Not used
    peer_list_file: str               # Filename of peer list (including this peer)
    has_chunk_file: str               # Filename of serialized chunk dictionary
    max_conn: int                     # Maximum concurrently connections
    identity: int                     # This peer's ID
    peers: list[tuple[str, str, str]] # List of tuple (ID, hostname, port). Peer list
    haschunks: dict[str, bytes]       # Chunks data (hash -> data)
    verbose: int                      # Verbosity
    timeout: int                      # Timeout value, default 0.
                                      # If 0, you should estimate timeout value using estimated RTT.
                                      # If not 0, you should always use the set value.


    ip: str                           # This peer's IP
    port: int                         # This peer's port
```

Provided Code - Skeleton

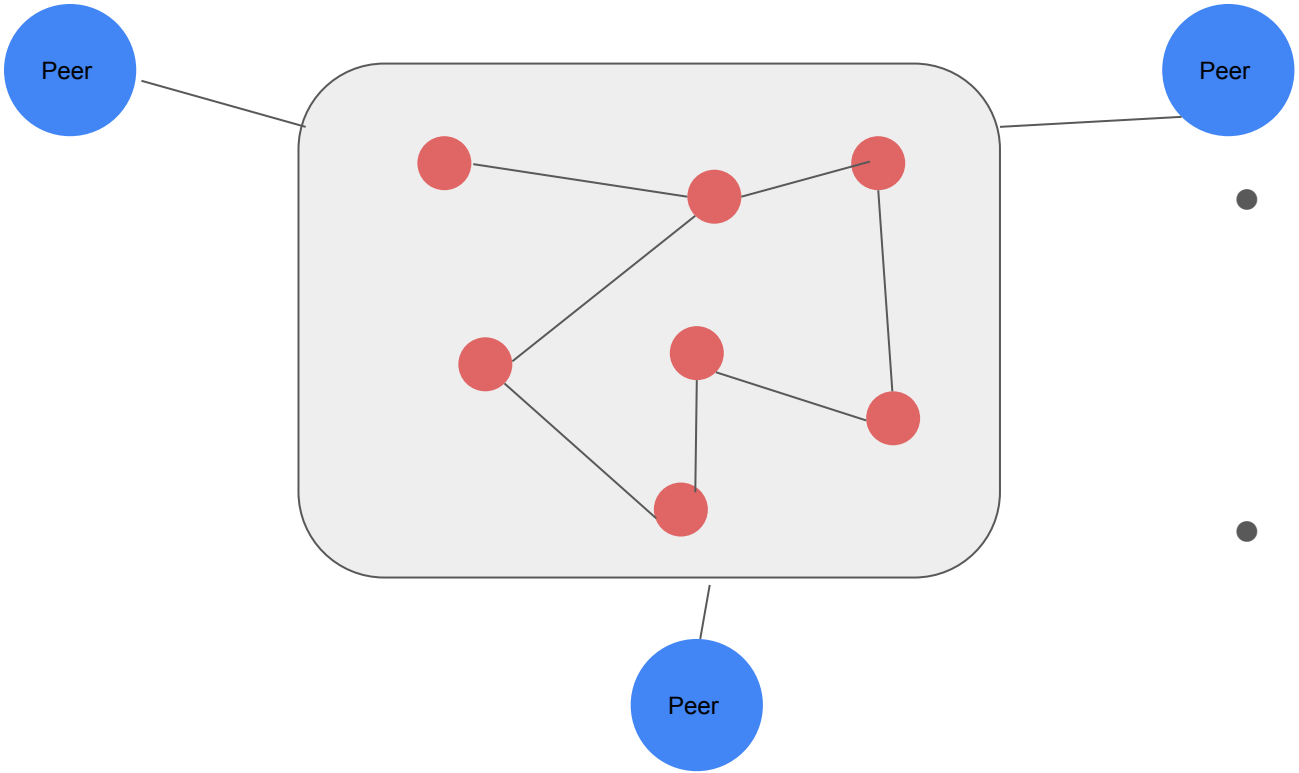
- It’s like a busy-waiting model.
- “select” multiplexes IO channels
- Alway use “simsocket”, it will direct packets to our simulator.

```
def peer_run(config):
    addr = (config.ip, config.port)
    sock = simsocket.SimSocket(config.identity, addr, verbose=config.verbose)

    try:
        while True:
            ready = select.select([sock, sys.stdin],[],[], 0.1)
            read_ready = ready[0]
            if len(read_ready) > 0:
                if sock in read_ready:
                    process_inbound_udp(sock)
                if sys.stdin in read_ready:
                    process_user_input(sock)
            else:
                # No pkt nor input arrives during this period
                pass
    except KeyboardInterrupt:
        pass
    finally:
        sock.close()
```

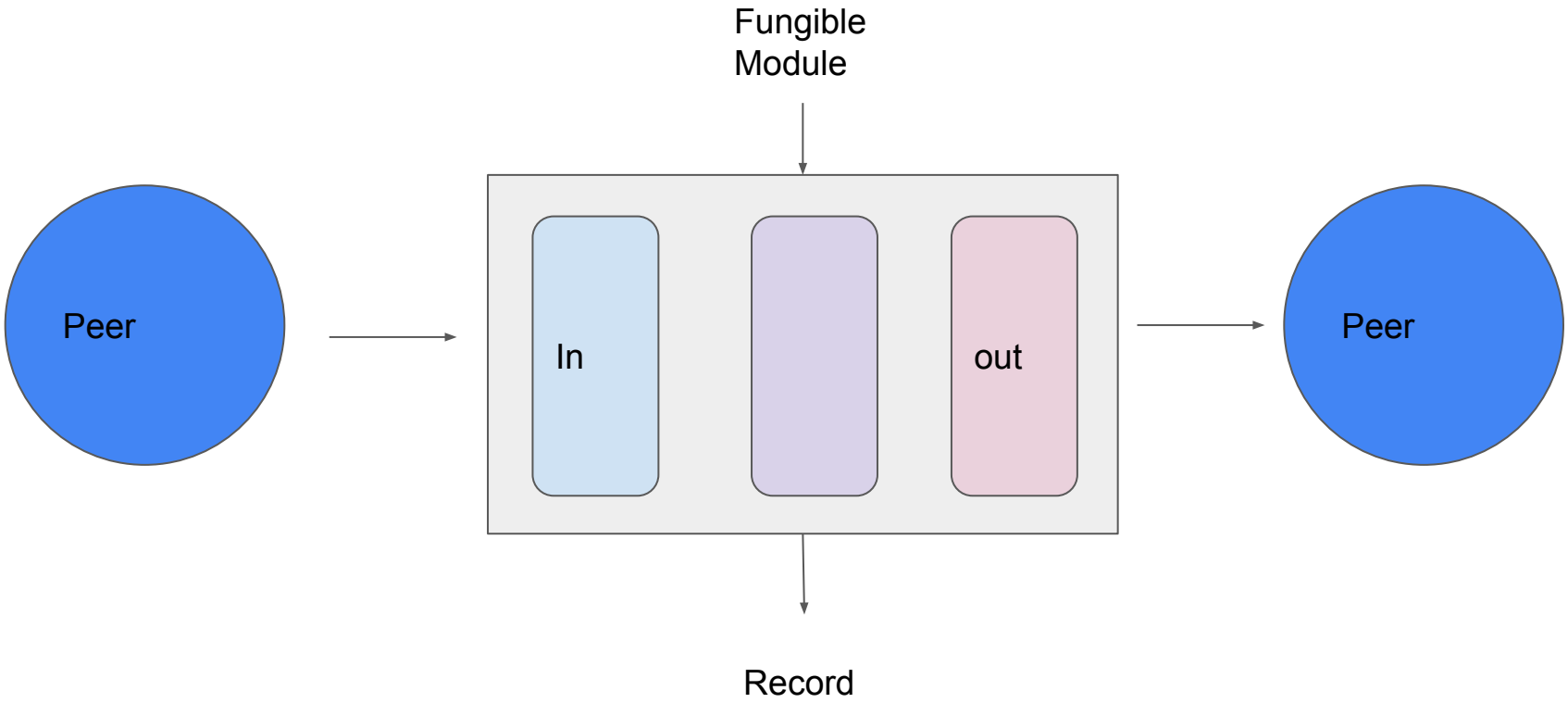
Peer.py

Provided Code - Network Simulator



- The simulator is actually a proxy, the env set in each peer will direct packets to the simulator first.
- Its topology is specified by *nodes.map*

Provided Code - Testing Scripts



Provided Code - Testing Scripts e.x.

```
def normal_handler(recv_queue, send_queue):
    start_time = time.time()
    while True:
        try:
            pkt = recv_queue.get(timeout=0.01)
        except:
            continue
        send_queue.put(pkt)
```

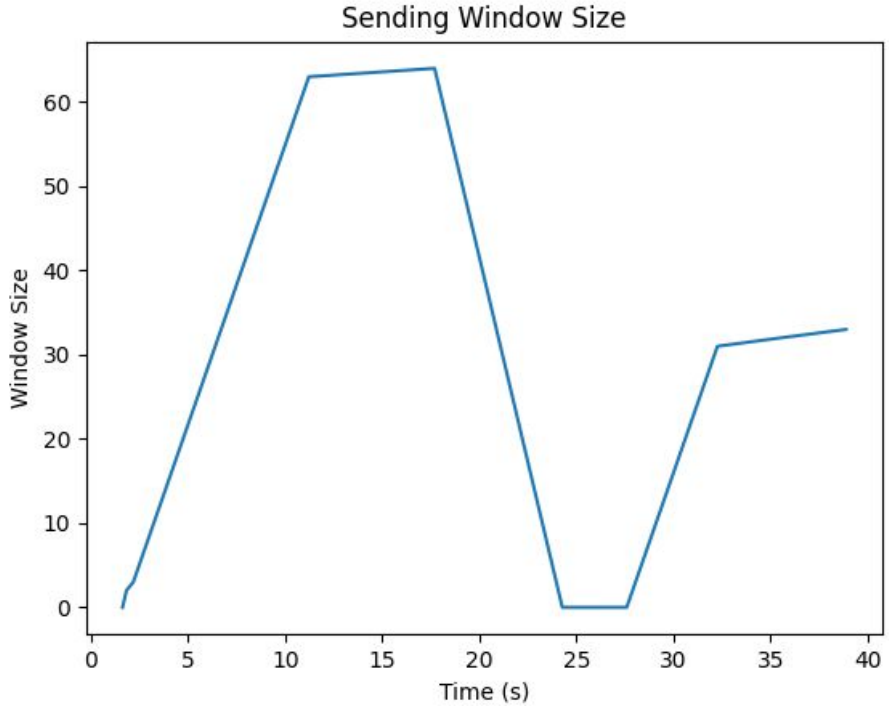
- This means you just let all packet pass.
- But you can also drop some of them in the middle…

Content

Grading

- Basic Test (sanity check) - 70 points
  - Handshaking - 12
  - RDT - 12
  - Congestion Control (Need to draw) - 22
  - Concurrency - 12
  - Robustness - 12
- Comprehensive Test (Run with simulator) - 30 points
  - Simple
  - Med
  - Complex
- Bonus - 20 points (Implement + ranking within participants)
  - Optimization

Grading - Draw



Content

Bonus - Optimization

You may consider the following directions to optimize:

- Higher utilization of bandwitth:
  - Fast recovery
- Less pacekts to retransmit:
  - Delayed ACK
  - Additional error recovery packet like QUIC
  - Hybrid of muktiple approach

Content

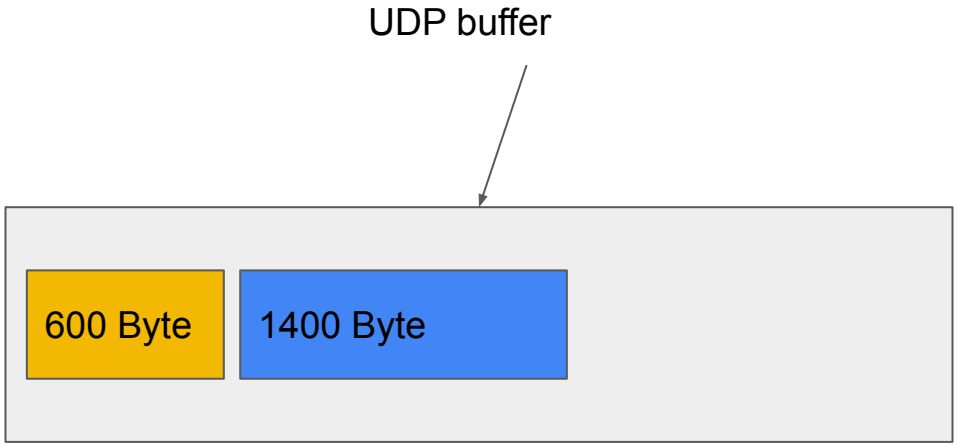
1. Overview
    a. P2P
    b. Congestion Control
    c. Concurrency
2. Provided Code
    a. Skeleton
    b. Network Simulator
    c. Testing script
3. Grading
4. Bonus
5. **FAQ**
6. Q&A

FAQ - How to ask?

Walkthrough: how to post a discussion.

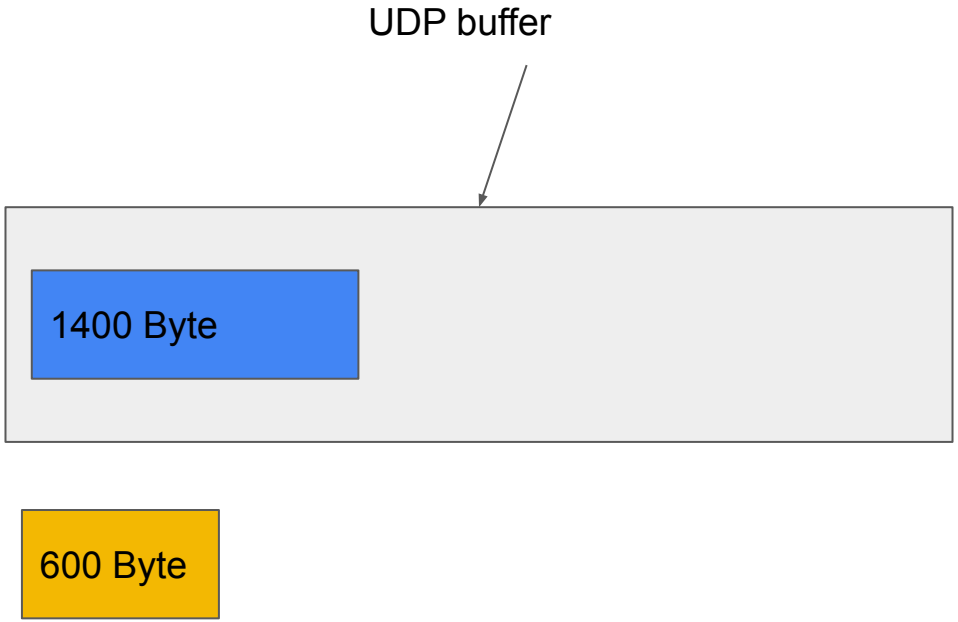
FAQ - udp recvfrom

- What will happen if call recvfrom(700)?



FAQ - udp recvfrom

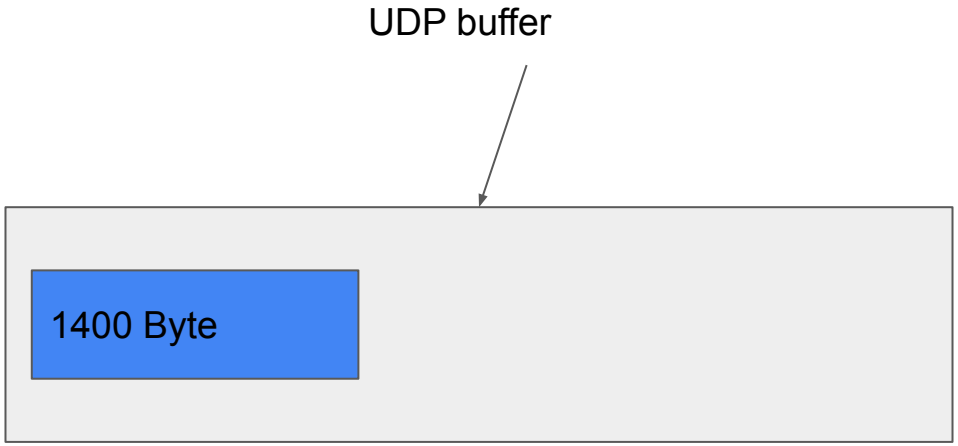
- What will happen if call recvfrom(700)?
  - Only get 600.



“Man, this packet only has 600 bytes! You can read at most **one** packet in a call!”

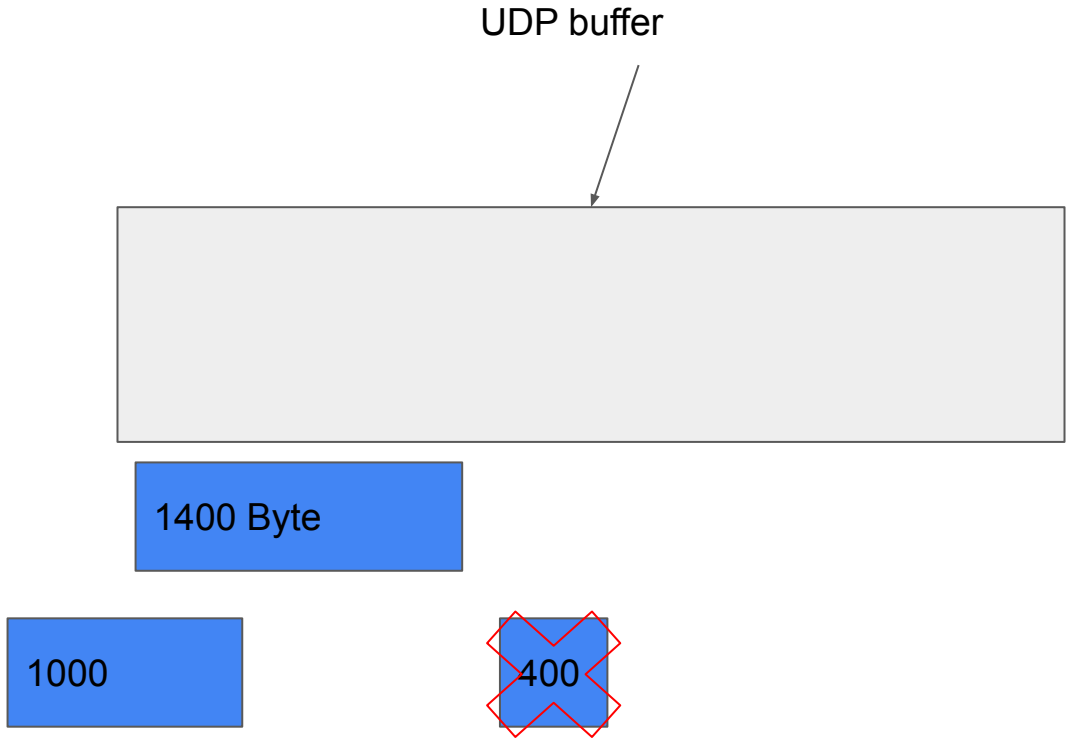
FAQ - udp recvfrom

- What will happen if call recvfrom(700)?
  - Only get 600.
- What will happen if call recvfrom(1000)?



FAQ - udp recvfrom

- What will happen if call recvfrom(700)?
  - Only get 600.
- What will happen if call recvfrom(1000)?
  - Read all 1400, return 1000, 400 discarded.



“OK, I am willing to cut myself and give you the first 1000K bytes.”

FAQ - Concurrency

- Maintain sessions!

Content

1. Overview
    a. P2P
    b. Congestion Control
    c. Concurrency
2. Provided Code
    a. Skeleton
    b. Network Simulator
    c. Testing script
3. Grading
4. Bonus
5. FAQ
6. **Q&A**

Q & A

Ask anything!

Checkpoints

- Checkpoint0: Nov.29th, release handshaking tests.
- Checkpoint1: Dec.12th, release reliable data transfer and congestion control tests.
- Checkpoint2: Dec.17th, release concurrency tests and comprehensive tests examples.
- Checkpoint3: Dec.22th, release robustness tests and optimization tests examples.

In-person Q&A service

- Dec.17 (SAT) 15:00-17:00, Haoming Zhang
- Dec.21 (WED) TBD, Sixu Tan
- Dec.22 (THU) TBD, Sixu Tan
- TBD, Zhiren Deng
- TBD

Thanks