CMPE487 Lecture 6: Flow Control

Dr. Eren Soyak

erensoyak@gmail.com
Boğaziçi Üniversitesi, Bilgisayar Mühendisliği Bölümü

Spring Semester, 2023


2+2=5

INGSOC

WAR
IS PEACE

FREEDOM
IS SLAVERY

IGNORANCE
IS STRENGTH

1929
1930

1931
1932

АРИФМЕТИКА ВСТРЕЧНОГО ПРОМФИНПЛАНА

2+2=5

ПЛЮС
ЭНТУЗИАЗМ
РАБОЧИХ

Я. Гуминер


2
3


Believe in something. Even if it means sacrificing everything.
Just do it.

Pitches

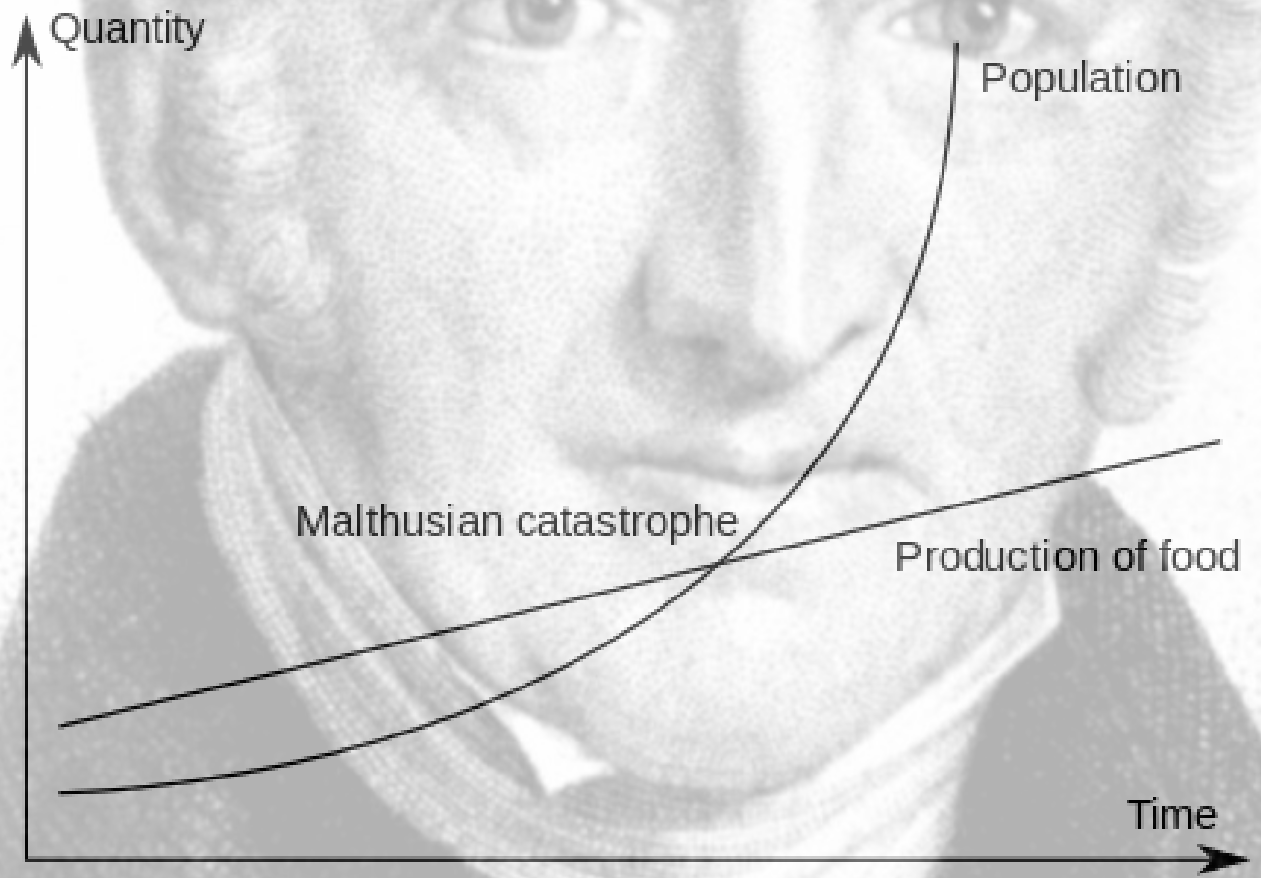
- Each person has 1 minute to pitch.
- Pitches to be given pre-recorded – record, upload to Youtube or other video service, share private link
- The pitch will be judged by everyone based on the following:
  - Was it well presented?
  - Would you like to work on it?
- Everyone gets 10 votes to give out for their highest ranking pitches

Project

- Teams of 2
- Must build something that works, and can be demonstrated in class

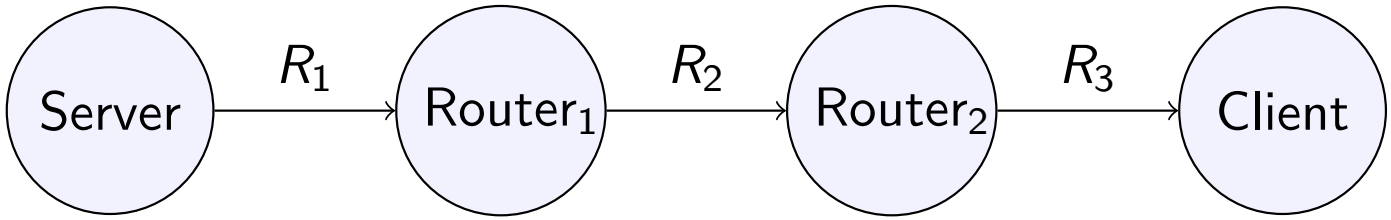
Malthus and You

- The **Malthusian spectre** will break down a society when population growth outpaces agricultural production (1779)
- TCP congestion control aims to prevent a **congestive collapse**



Congestion Control vs Flow Control

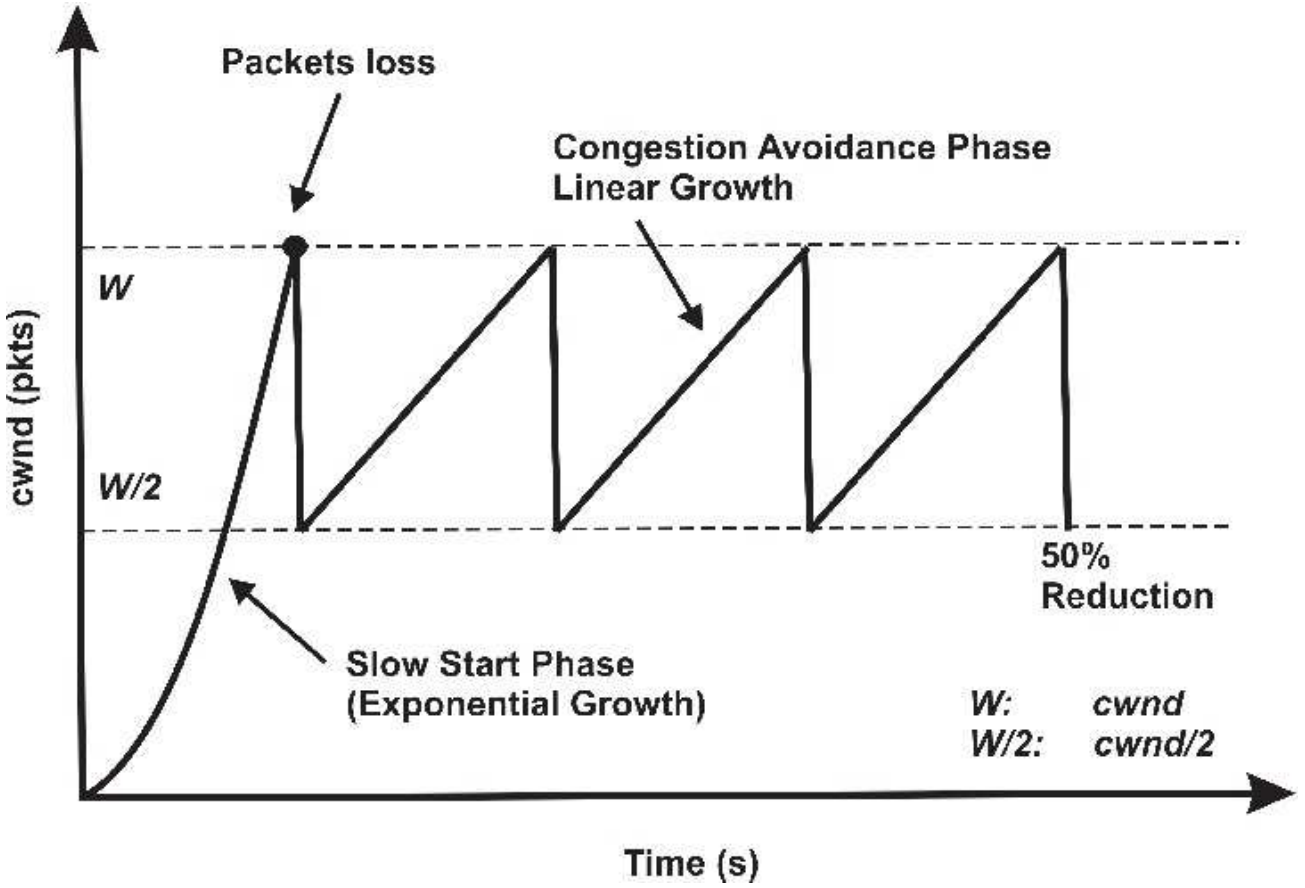
- The Internet was not built for Netflix
- **Flow Control** uses back pressure via a sliding window to prevent a TCP server from overwhelming a client
- **Congestion Control** uses an end-to-end congestion window to prevent routers from being overwhelmed



What happens is $R_1 > R_2$ or $R_1 > R_3$?

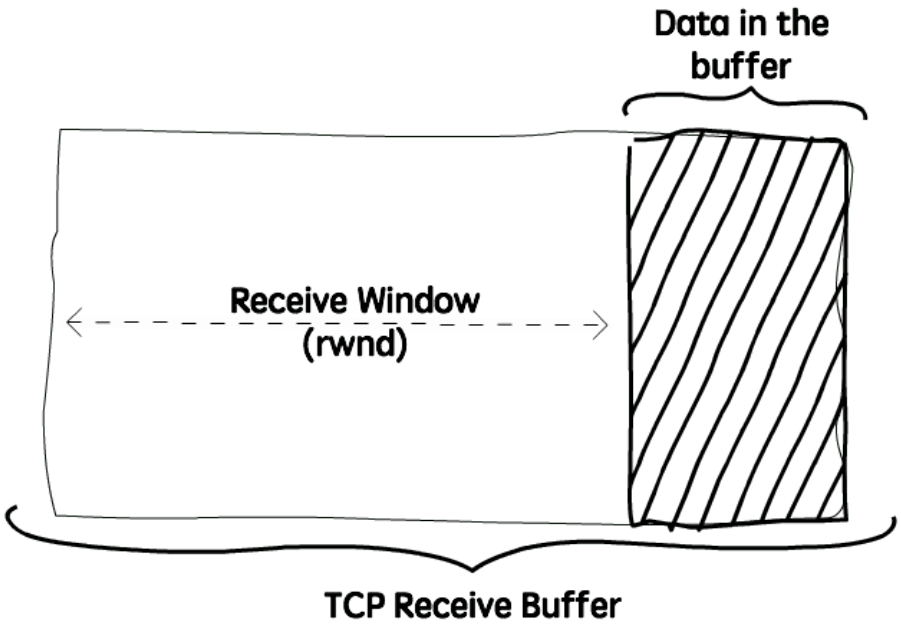
Congestion Control

- Additive Increase, Multiplicative Decrease and Slow Start are enforced by `cwnd`



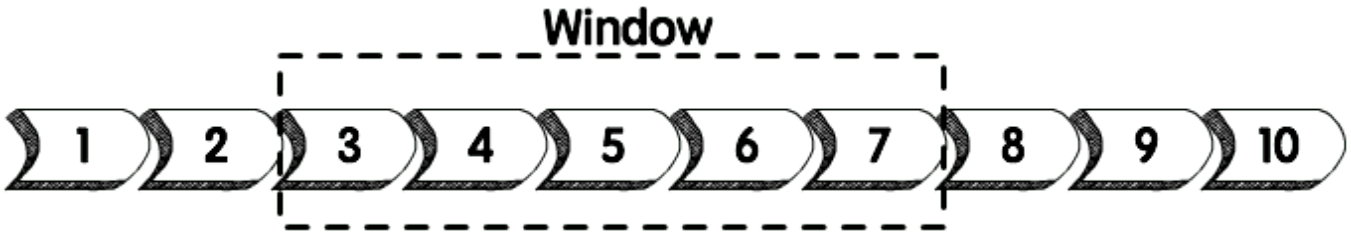
Flow Control: rwnd

- `https://www.brianstorti.com/tcp-flow-control/` has real nice discussion
- Flow control works by exerting back pressure via the `rwnd` (Receive Window) it advertises with each ACK



Flow Control: Sliding Window

- The server will make sure it never has more bytes in flight than rwnd



- When the window size drops down to 1, periodic WindowProbe's will be sent to prevent deadlock

Peek Under the Hood

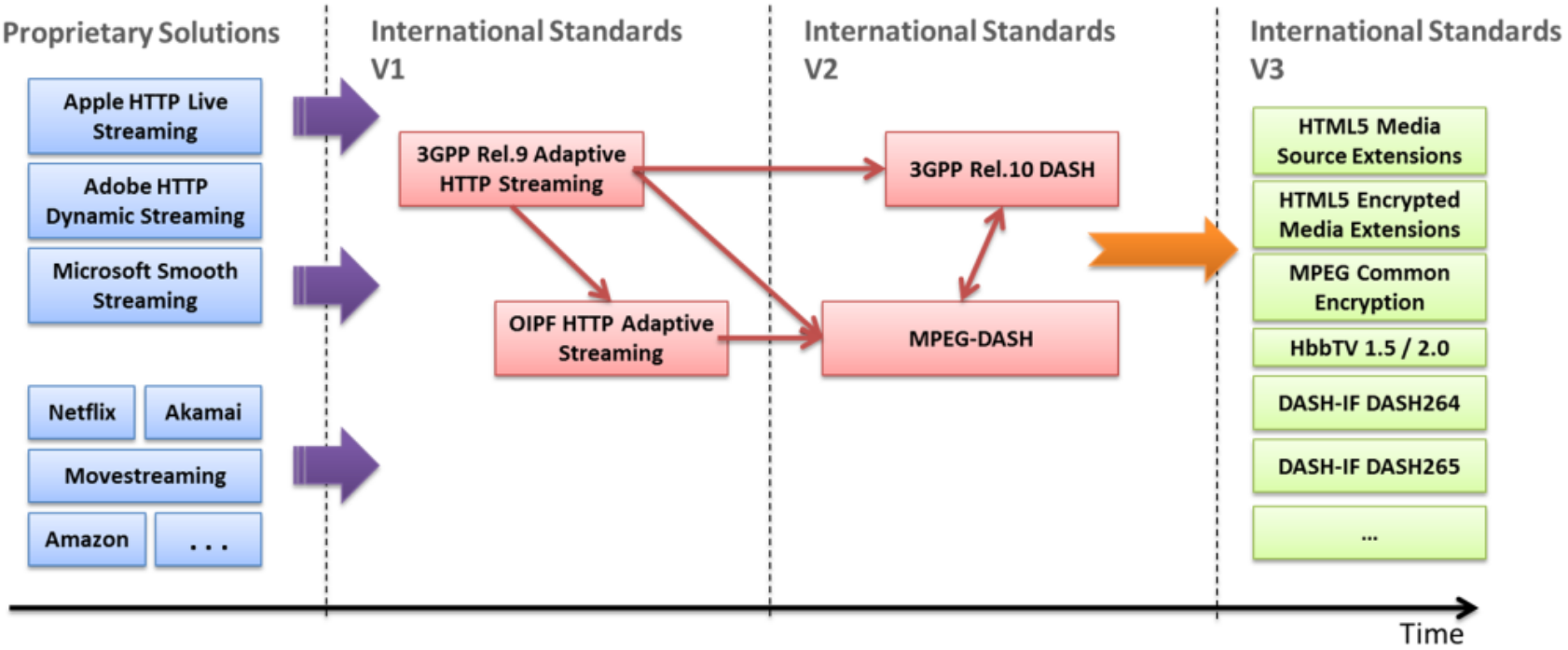
- You can see all this in action when you sniff

223…	18.410069	217.31.237.197	192.168.1.134	TCP	1474	[TCP segment of a reassembled PDU]
223…	18.410072	217.31.237.197	192.168.1.134	TCP	1474	[TCP segment of a reassembled PDU]
223…	18.410512	217.31.237.197	192.168.1.134	TCP	1474	[TCP segment of a reassembled PDU]
223…	18.411147	217.31.237.197	192.168.1.134	TCP	1474	[TCP segment of a reassembled PDU]
223…	18.411878	217.31.237.197	192.168.1.134	TCP	1474	[TCP segment of a reassembled PDU]
223…	18.411883	217.31.237.197	192.168.1.134	TCP	1474	[TCP segment of a reassembled PDU]
223…	18.412458	217.31.237.197	192.168.1.134	TCP	1474	[TCP segment of a reassembled PDU]
223…	18.412463	217.31.237.197	192.168.1.134	TCP	1474	[TCP segment of a reassembled PDU]
223…	18.412464	217.31.237.197	192.168.1.134	TCP	1474	[TCP segment of a reassembled PDU]
223…	18.413530	217.31.237.197	192.168.1.134	TCP	1474	[TCP segment of a reassembled PDU]
223…	18.413535	217.31.237.197	192.168.1.134	TCP	1474	[TCP segment of a reassembled PDU]
223…	18.413593	192.168.1.134	217.31.237.197	TCP	66	49797 → 8080 [ACK] Seq=852 Ack=2999508 Win=128256 Len=0 TSval=62791…
223…	18.414433	217.31.237.197	192.168.1.134	TCP	1474	[TCP segment of a reassembled PDU]
223…	18.414437	217.31.237.197	192.168.1.134	TCP	1474	[TCP segment of a reassembled PDU]
223…	18.416761	217.31.237.197	192.168.1.134	TCP	1474	[TCP segment of a reassembled PDU]
223…	18.416765	217.31.237.197	192.168.1.134	TCP	1474	[TCP segment of a reassembled PDU]

```
[TCP Segment Len: 0]
Sequence number: 852    (relative sequence number)
Acknowledgment number: 2999508    (relative ack number)
Header Length: 32 bytes
▶ Flags: 0x010 (ACK)
Window size value: 4008
[Calculated window size: 128256]
[Window size scaling factor: 32]
▶ Checksum: 0x07a4 [validation disabled]
Urgent pointer: 0
▶ Options: (12 bytes), No-Operation (NOP), No-Operation (NOP), Timestamps
▼ [SEQ/ACK analysis]
    [This is an ACK to the segment in frame: 22368]
    [The RTT to ACK the segment was: 0.000058000 seconds]
    [iRTT: 0.025931000 seconds]
```

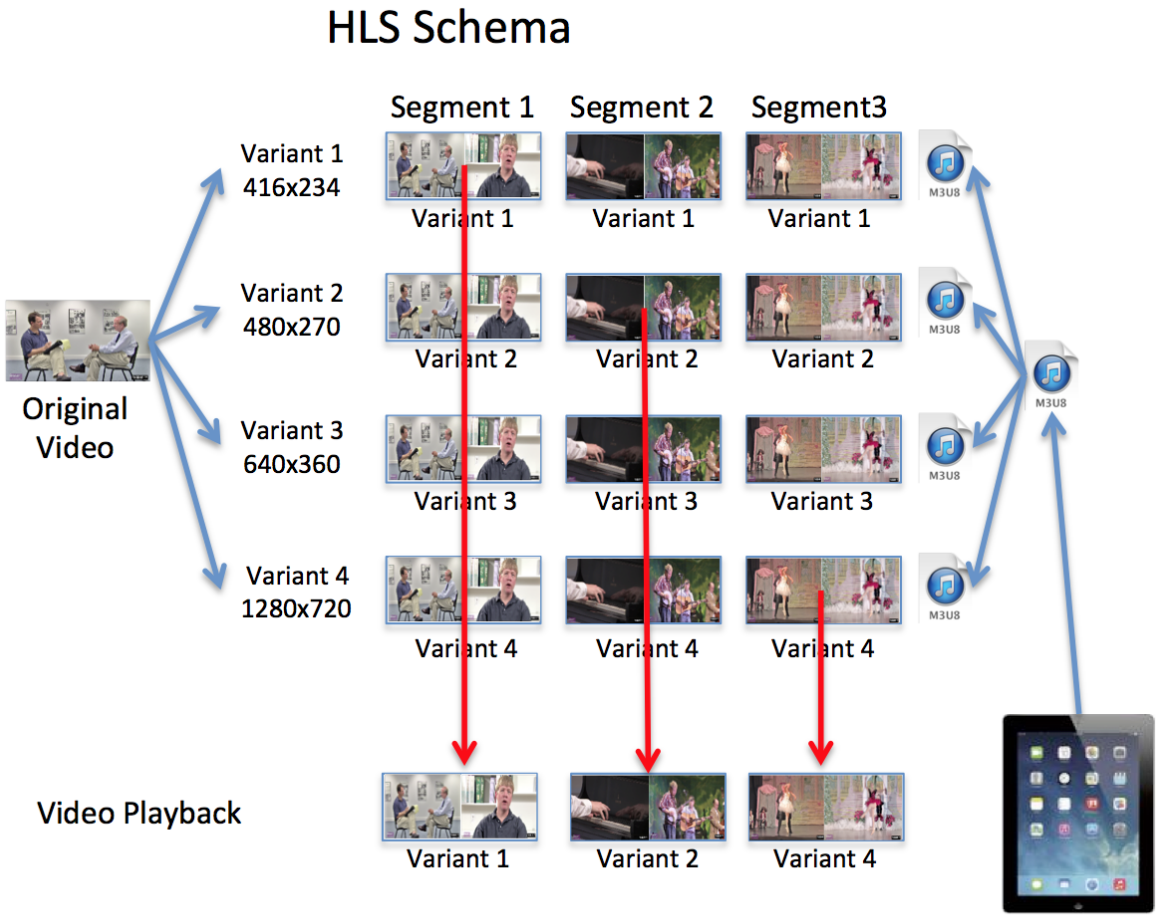
Video Streaming

- Modern video streaming mechanisms are client-side controlled
- The client should demand the highest quality possible which it can sustainably stream
- Many systems exist in this space



Client Direction

- The client can choose among chunks based on how fast it got the previous chunk



Workshop 4: TCP over UDP

UDP as TCP

- We'll modify Python Chat to copy large files while implementing our own flow control
- Bypass TCP flow control by using UDP
  1. Generate packets of 1500 bytes each. Each packet gets a sequence ID "SEQ"
  2. Implement flow control: Send ACK packets for each SEQ, with a "RWND" field so that sender can adjust rate such that no more than `rwnd` bytes are ever in flight
  3. Packets not ACKed for 1 second should be resent
- No need to implement WindowProbe or Congestion Control

New Functions

- We'll add a "send file" option in our application. This will take a file, divide it into 1500 byte packets (last packet can be smaller), and send via the UDP link. File packets will have the following fields:
  1. TYPE=4 (File)
  2. NAME=(whatever the name of the file is)
  3. SEQ=(integer SEQ of file packet starting from 1).
  4. BODY=1500 bytes of payload
- The Listener will append packets of TYPE=4 of the same NAME in the order of SEQ and send an ACK packet.
- A new packet type TYPE=5 (ACK) will have a field SEQ (integer) and RWND (integer). RWND will be in unit of packets. What should RWND be? ☺
- For simplicity, assume the Receiver has a 10 packet buffer per connection (make this buffer size parameterizable for future Workshops!)

Sample Flow

1. **Sender:** Send Packet SEQ=1 (send only once since we don't know receivers RWND)
2. **Receiver:** ACK Packet SEQ=1 with RWND=10
3. **Sender:** Send Packets SEQ=2 to 11
4. **Receiver:** ACK Packet SEQ=2 with RWND=10
5. **Sender:** Send Packet SEQ=12 (still have SEQ 3-11 in flight, so can only send 1 more)
6. **Receiver:** ACK Packet SEQ=3 with RWND=9
7. **Sender:** Don't send another packet yet! (Got SEQ=4-12 in flight and RWND=9)
8. **Receiver:** ACK Packet SEQ=4 with RWND=9
9. **Sender:** Send Packet SEQ=13 (still have SEQ 5-12 in flight, so can only send 1 more)

Note

- We're really simplifying RWND – there is no "consumer" application, so all ACKs will have the same RWND. For example, a video player would not work like this – there would another process depleting the buffer which RWND sizes.
- Buffers would normally be shared – what would happen if we had multiple active simultaneous connections?
- We're totally disregarding Congestion Control – this algorithm really is for the Client ☺

Due Dates

- Deadline for this workshop is 3pm April 11, via Moodle
- Test by sending a file that is at least 2 MB
- Please remember to submit
  1. code
  2. requirements.txt
  3. README (indicating who you tested with)
  4. pcap file showing **only** exchange (filter any irrelevant packets so the pcap you turn in has only the packets for the workshop)