UNIVERSITY OF
MICHIGAN

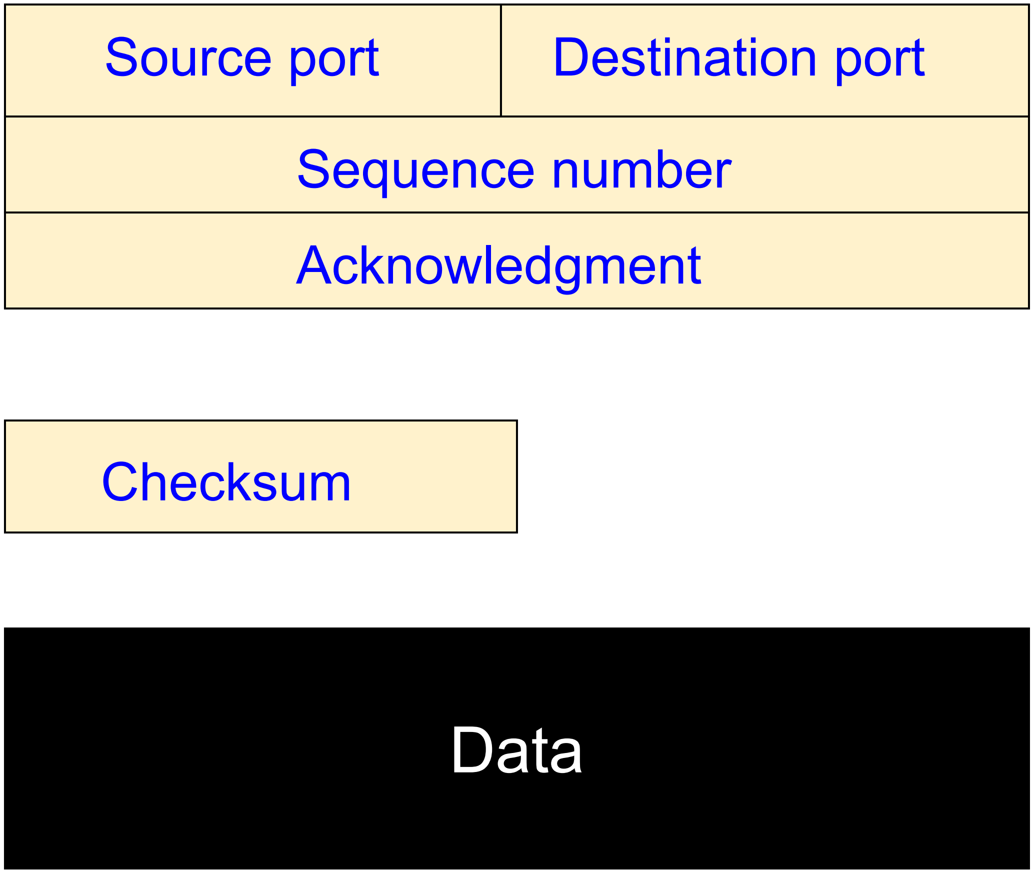
EECS 489

Computer Networks

Transport Control Protocol - TCP

TCP: Transmission Control Protocol

Build the TCP header



Source port

Destination port

Sequence number

Acknowledgment

Checksum

Data

What does TCP do?

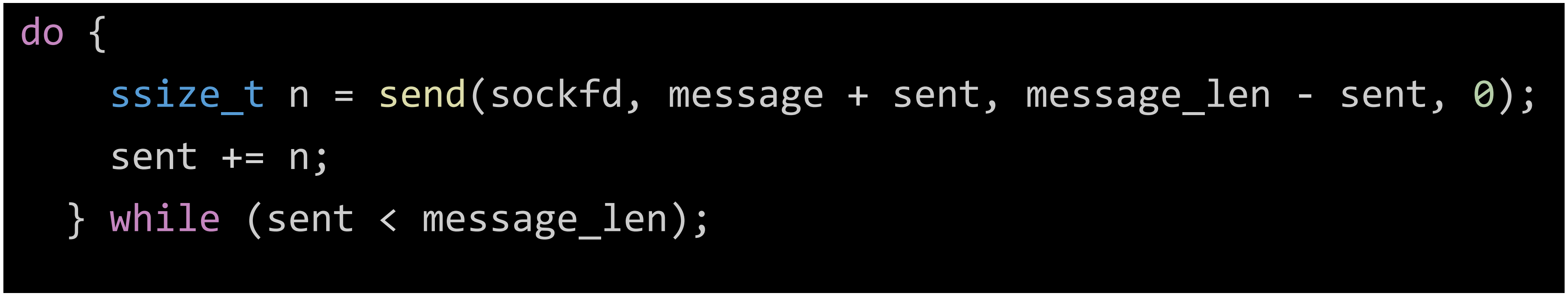
- Most of what we’ve seen
  - Checksum
  - Sequence numbers are byte offsets
  - Receiver sends cumulative acknowledgements (like GBN)
  - Receivers can buffer out-of-sequence packets (like SR)

What does TCP introduce?

- Most of what we’ve seen
  - Checksum
  - Sequence numbers are byte offsets
  - Receiver sends cumulative acknowledgements (like GBN)
  - Receivers buffer out-of-sequence packets (like SR)
- Introduces fast retransmit: duplicate ACKs trigger early retransmission
- Sender maintains a single retransmission timer (like GBN) and retransmits on timeout

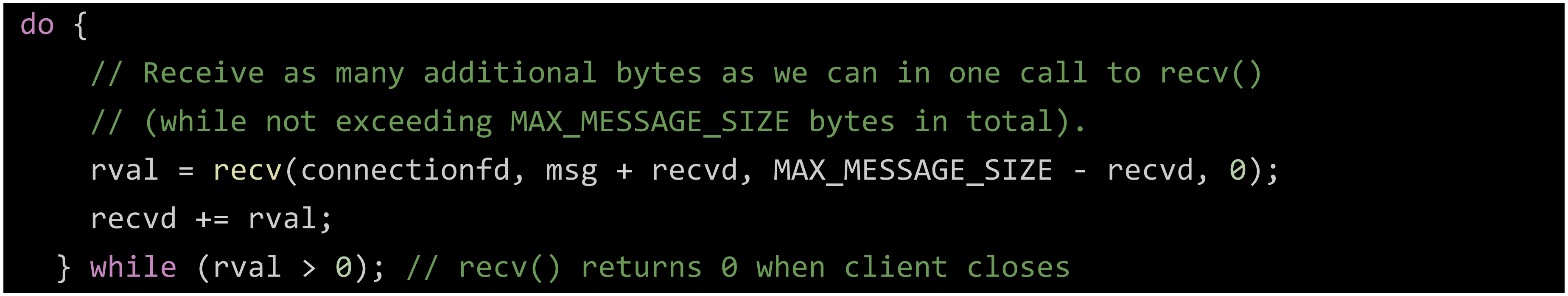
Review from Discussion 1:

- Send message



```
do {
    ssize_t n = send(sockfd, message + sent, message_len - sent, 0);
    sent += n;
  } while (sent < message_len);
```

- Receive message



```
do {
    // Receive as many additional bytes as we can in one call to recv()
    // (while not exceeding MAX_MESSAGE_SIZE bytes in total).
    rval = recv(connectionfd, msg + recvd, MAX_MESSAGE_SIZE - recvd, 0);
    recvd += rval;
  } while (rval > 0); // recv() returns 0 when client closes
```

HTTP exchange example

- Client connects to a webserver and generates a GET request

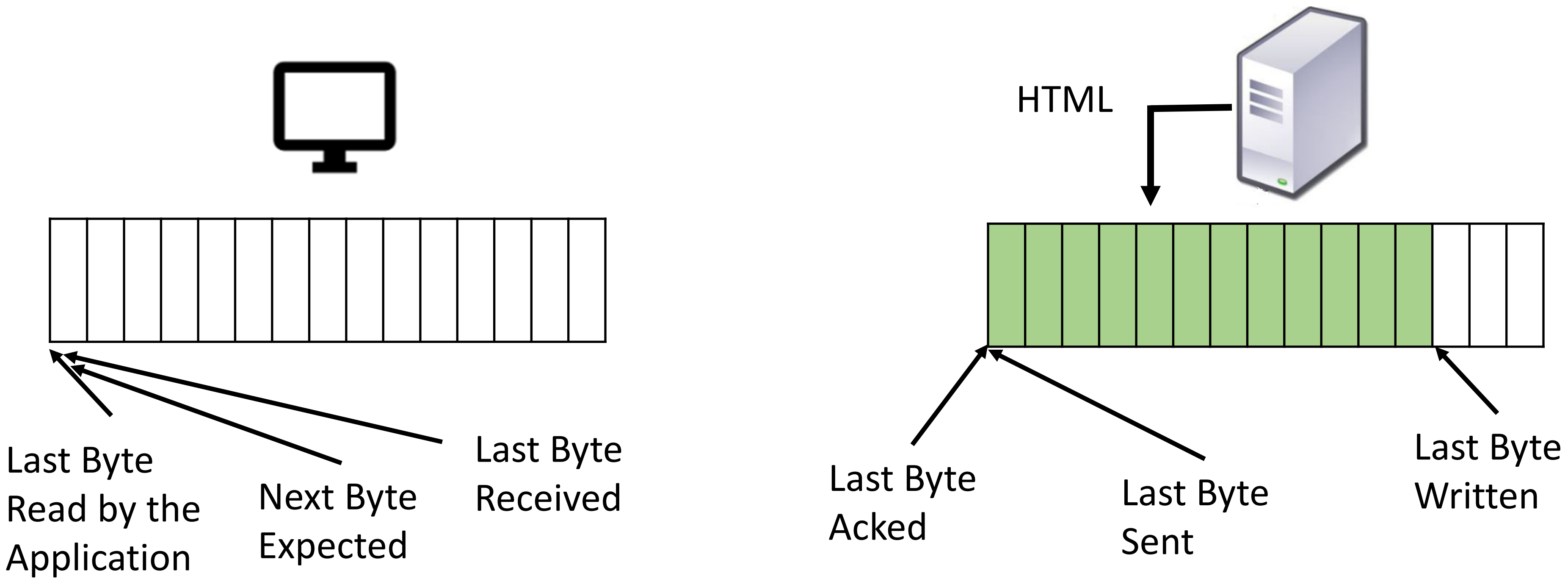
```
GET / HTTP/1.1
Accept: text/html,application/xhtml+xml,application/xml;q=0.9,image/avif,image/webp,image/apng,*/*;q=0.8,application/signed-exchange;v=b3;q=0.7
Accept-Encoding: gzip, deflate
Accept-Language: en-US,en;q=0.9
Cache-Control: no-cache
Connection: keep-alive
Host: www.eecs489.org
Pragma: no-cache
Upgrade-Insecure-Requests: 1
User-Agent: Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/116.0.0.0 Safari/537.36
```

HTTP exchange example

- Server sends a response

```
HTTP/1.1 200 OK
Connection: keep-alive
Content-Length: 5000
Server: GitHub.com
Content-Type: text/html; charset=utf-8
Last-Modified: Wed, 20 Sep 2024 17:31:52 GMT
… (Total of 5,000 bytes)
```

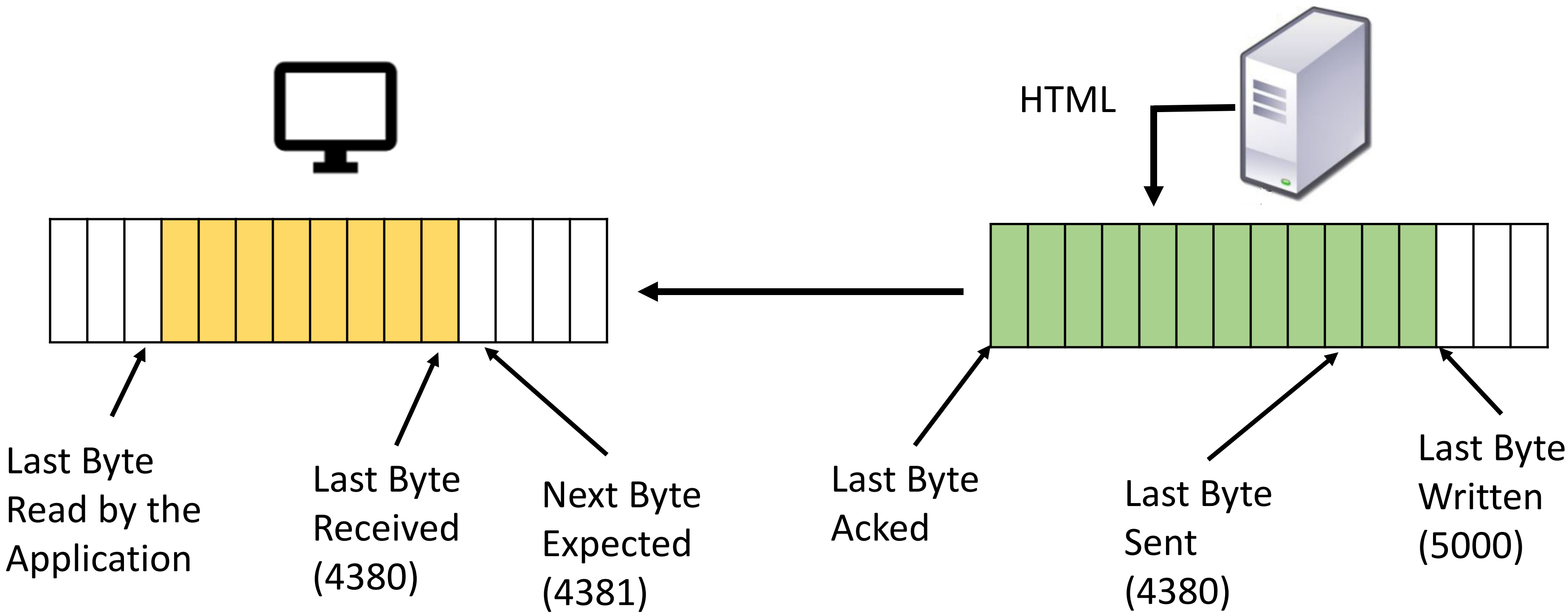
Now the server is going to send the webpage



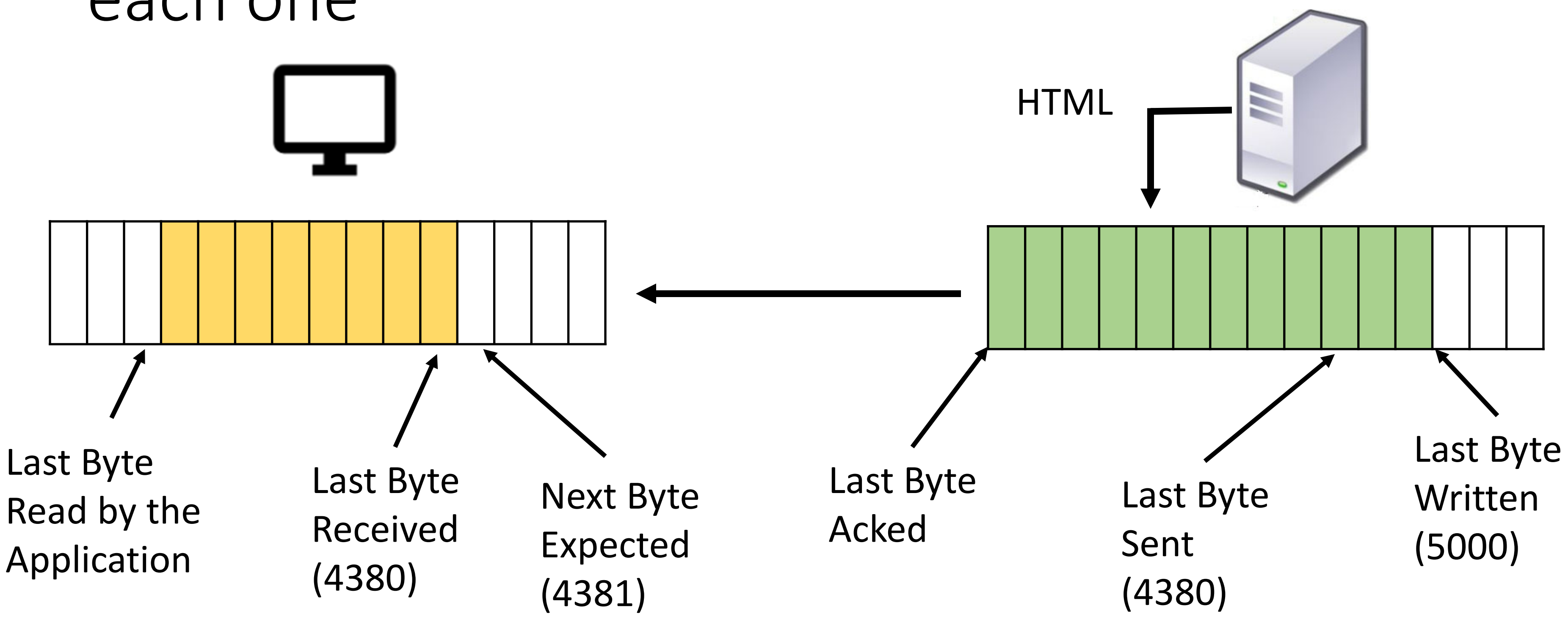
How much data can the server send

- Let’s say for each MTU (Maximum Transmission Unit) is 1500 Bytes
- IP Header and TCP headers are 20 bytes each.
- Each TCP Packet can hold 1500 - 20 - 20 = 1460 Bytes
- So TCP can transmit 1460 bytes of data in each packet
- Server’s Sliding window is 5000 Bytes

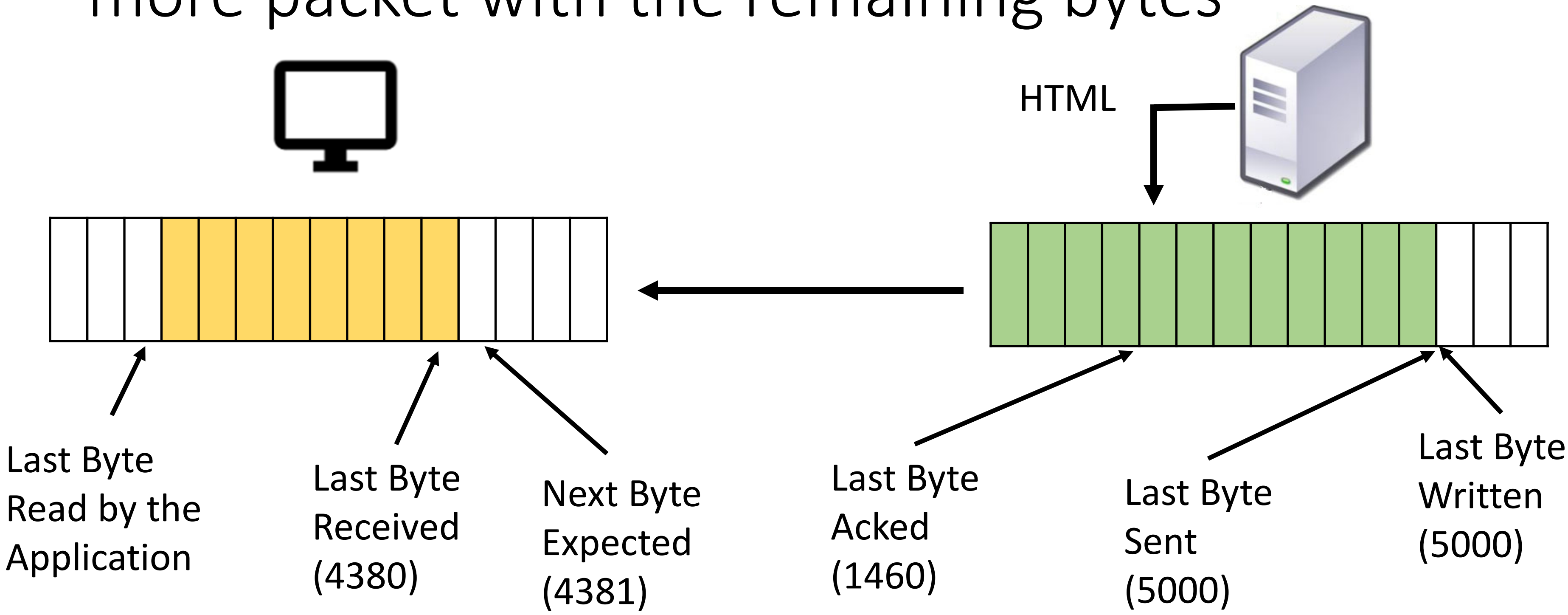
Server sends 3 packets, each with 1460 bytes


HTML
Last Byte Read by the Application
Last Byte Received (4380)
Next Byte Expected (4381)
Last Byte Acked
Last Byte Sent (4380)
Last Byte Written (5000)

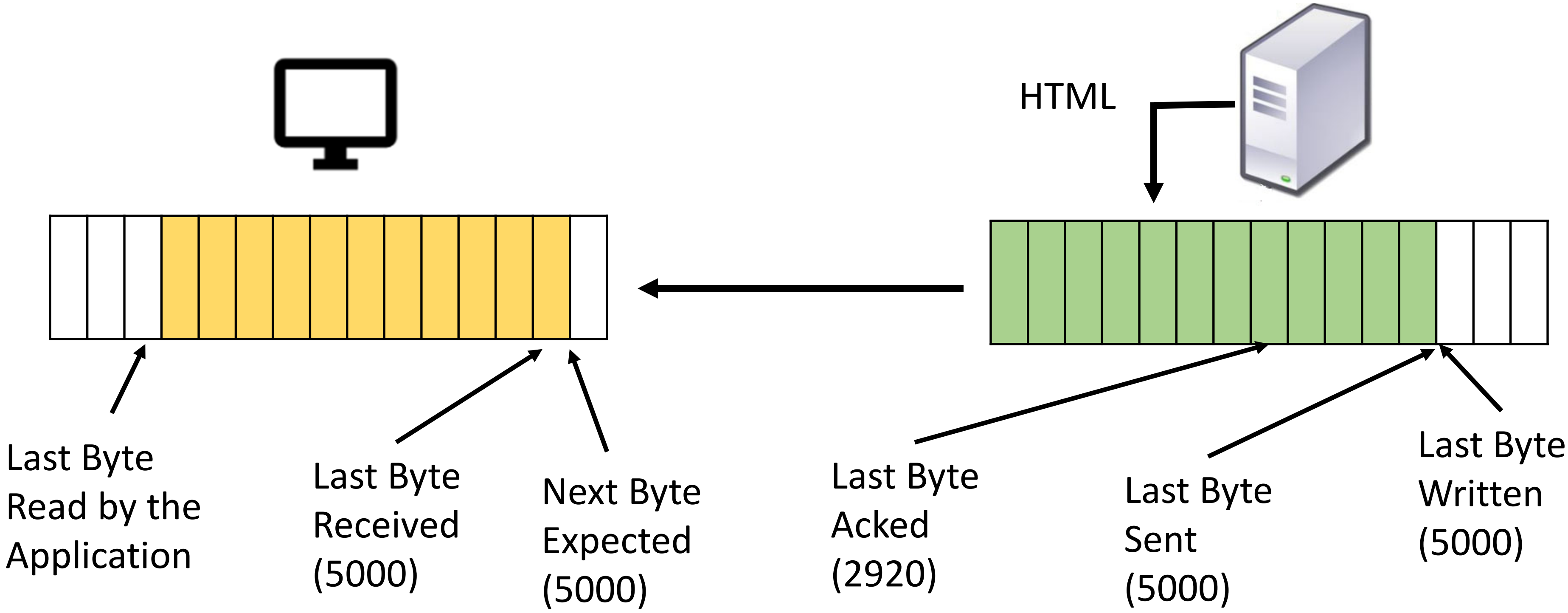
Host receives the three packets and ACKS each one


HTML
Last Byte Read by the Application
Last Byte Received (4380)
Next Byte Expected (4381)
Last Byte Acked
Last Byte Sent (4380)
Last Byte Written (5000)

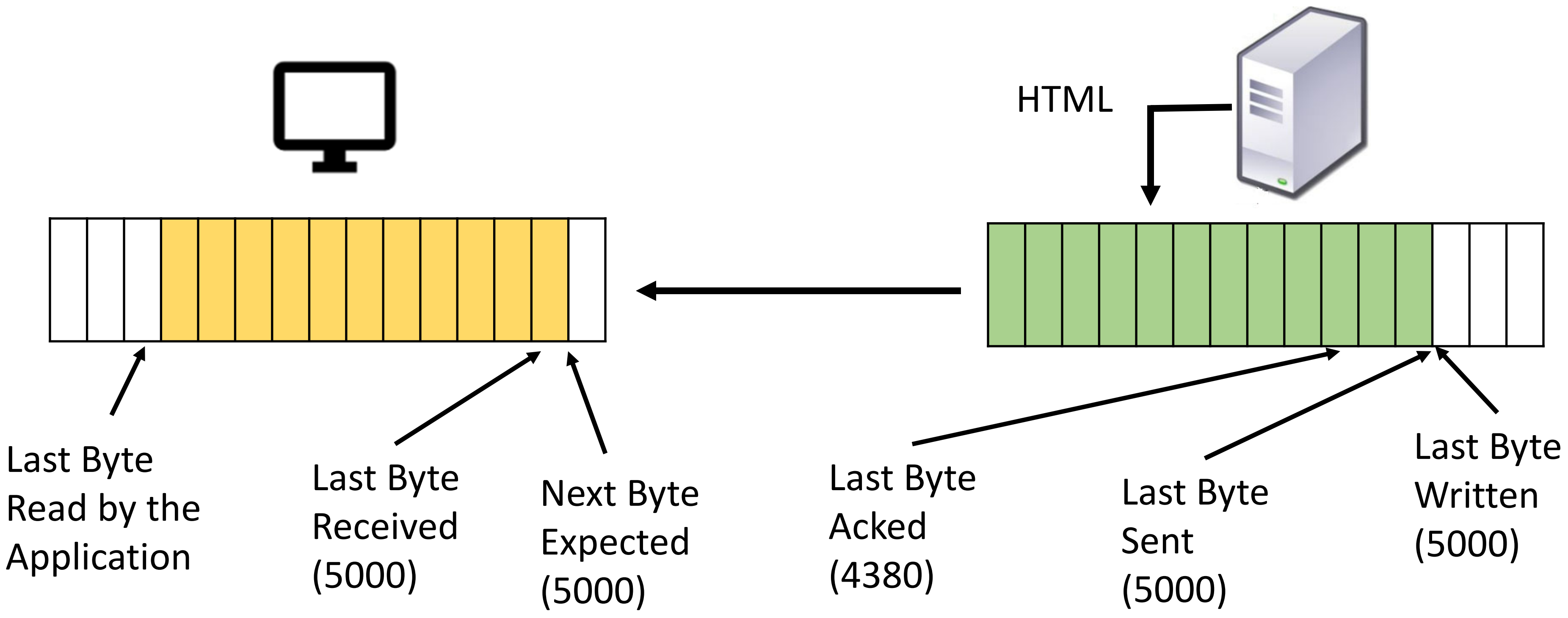
Server after receiving first ACK – send one more packet with the remaining bytes


HTML
Last Byte Read by the Application
Last Byte Received (4380)
Next Byte Expected (4381)
Last Byte Acked (1460)
Last Byte Sent (5000)
Last Byte Written (5000)

Server after receiving second ACK


HTML
Last Byte Read by the Application
Last Byte Received (5000)
Next Byte Expected (5000)
Last Byte Acked (2920)
Last Byte Sent (5000)
Last Byte Written (5000)

Server after receiving third ACK



HTML

Last Byte Read by the Application

Last Byte Received (5000)

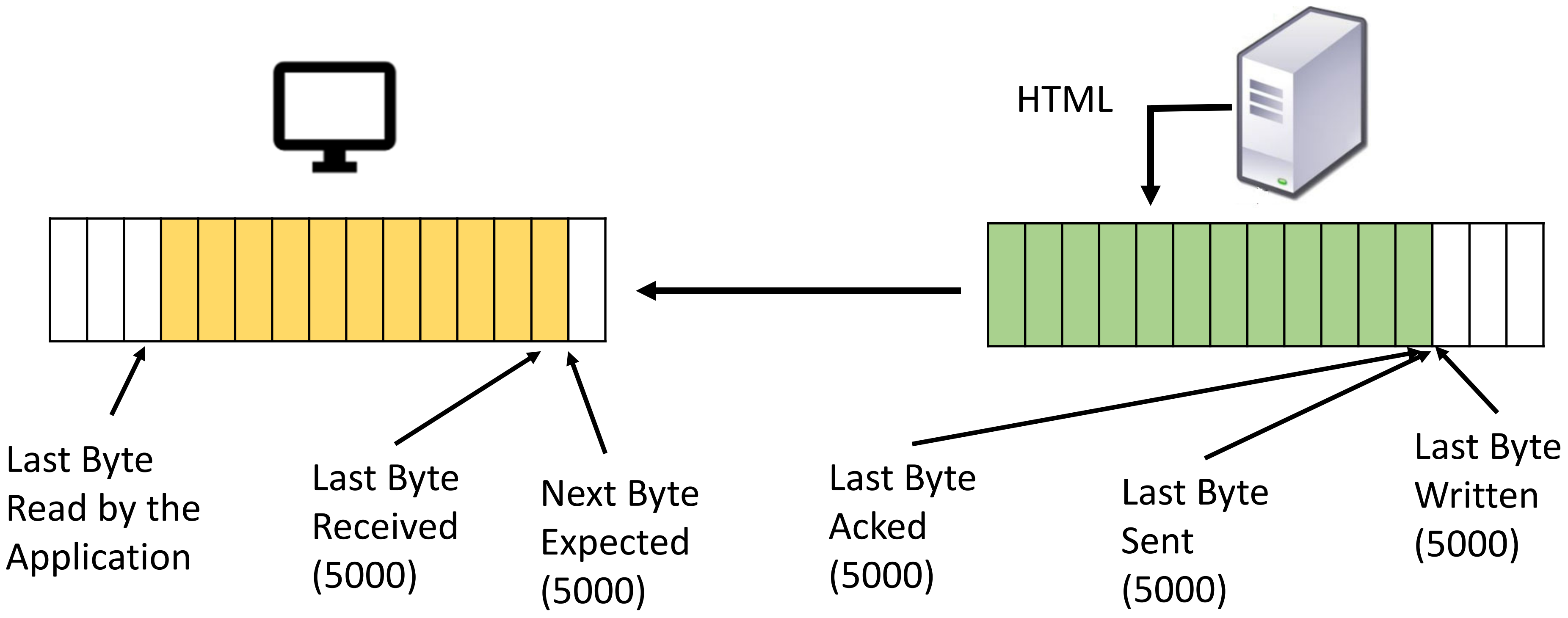
Next Byte Expected (5000)

Last Byte Acked (4380)

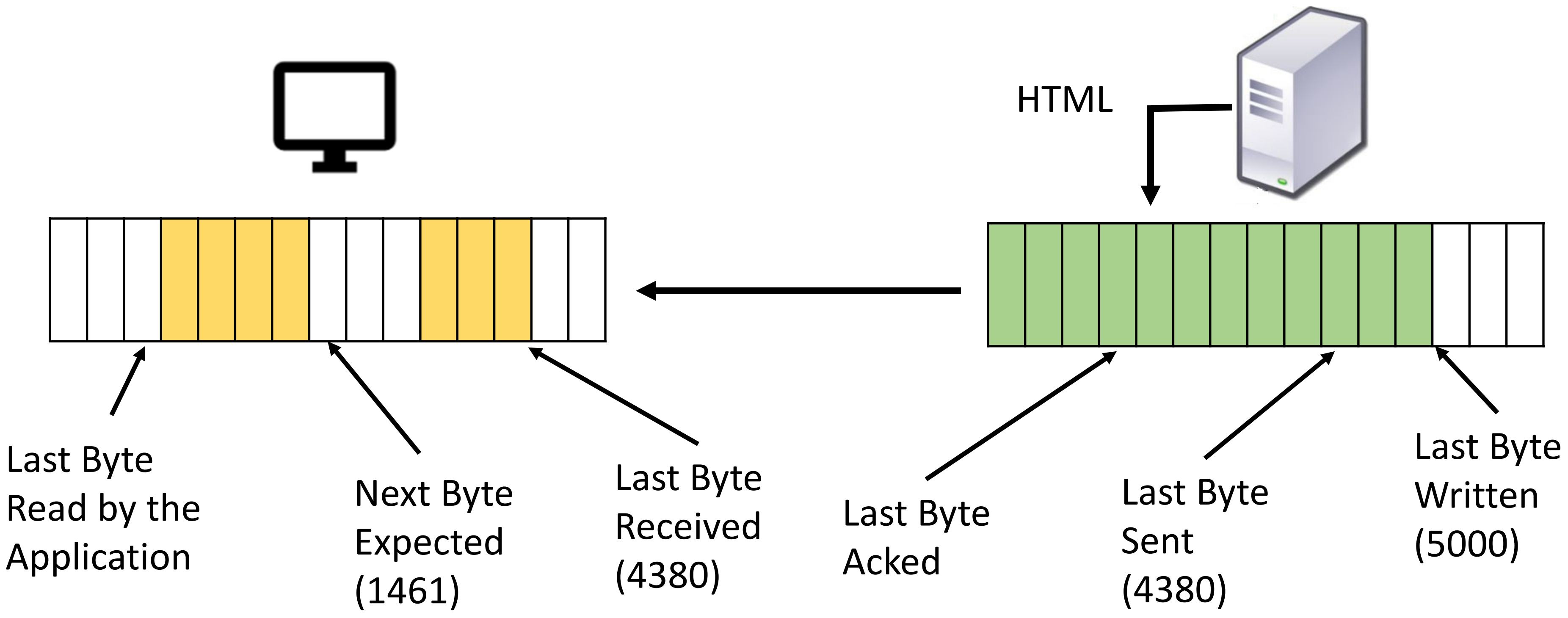
Last Byte Sent (5000)

Last Byte Written (5000)

Server after receiving fourth ACK


HTML
Last Byte Read by the Application
Last Byte Received (5000)
Next Byte Expected (5000)
Last Byte Acked (5000)
Last Byte Sent (5000)
Last Byte Written (5000)

Out of Order Delivery/Missing packet

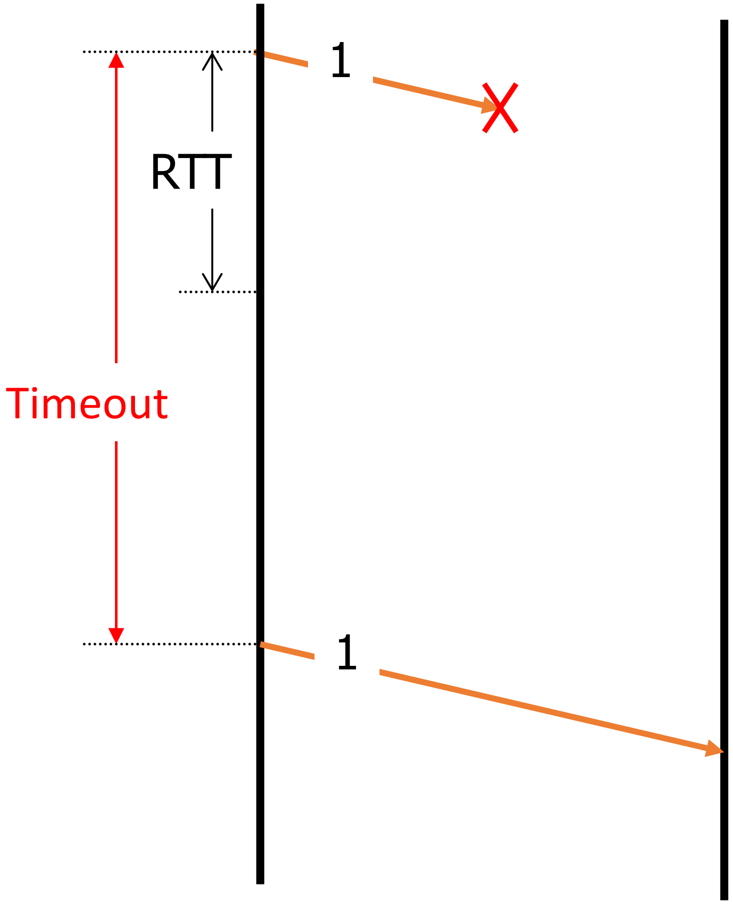

HTML
Last Byte Read by the Application
Next Byte Expected (1461)
Last Byte Received (4380)
Last Byte Acked
Last Byte Sent (4380)
Last Byte Written (5000)

- Host Acks with 1461 on Receiving Bytes 2920 – 4380
- Server retransmits Bytes 1461 – 2919 on receiving the above ACK or on a timeout

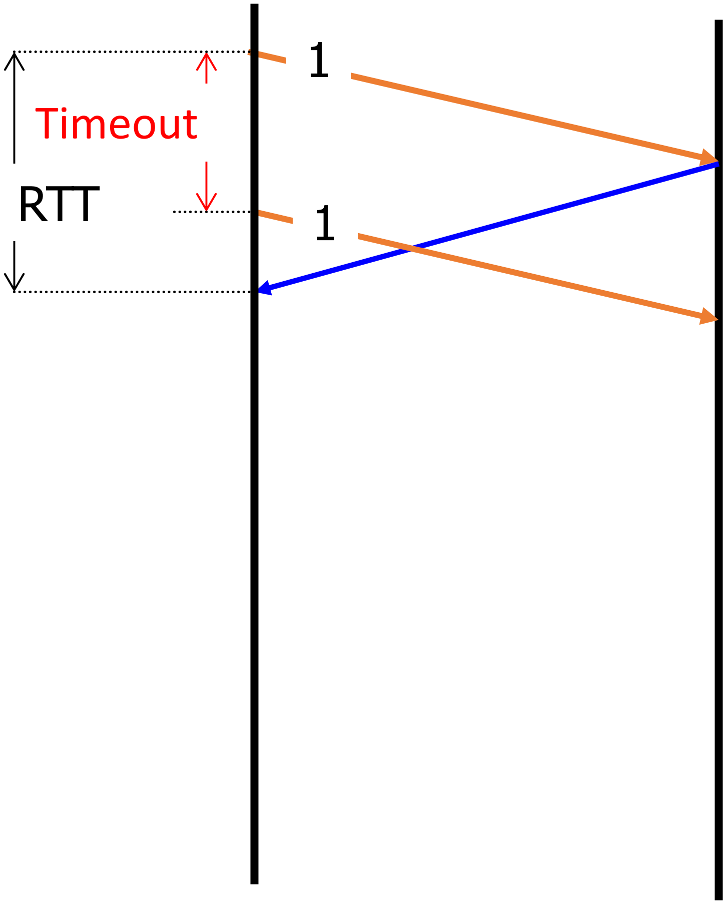
Retransmission timeout

- If the sender hasn’t received an ACK by timeout, retransmit the first packet in the window
- How do we pick a timeout value?

Timing illustration



Timeout too long → inefficient



Timeout too short → duplicate packets

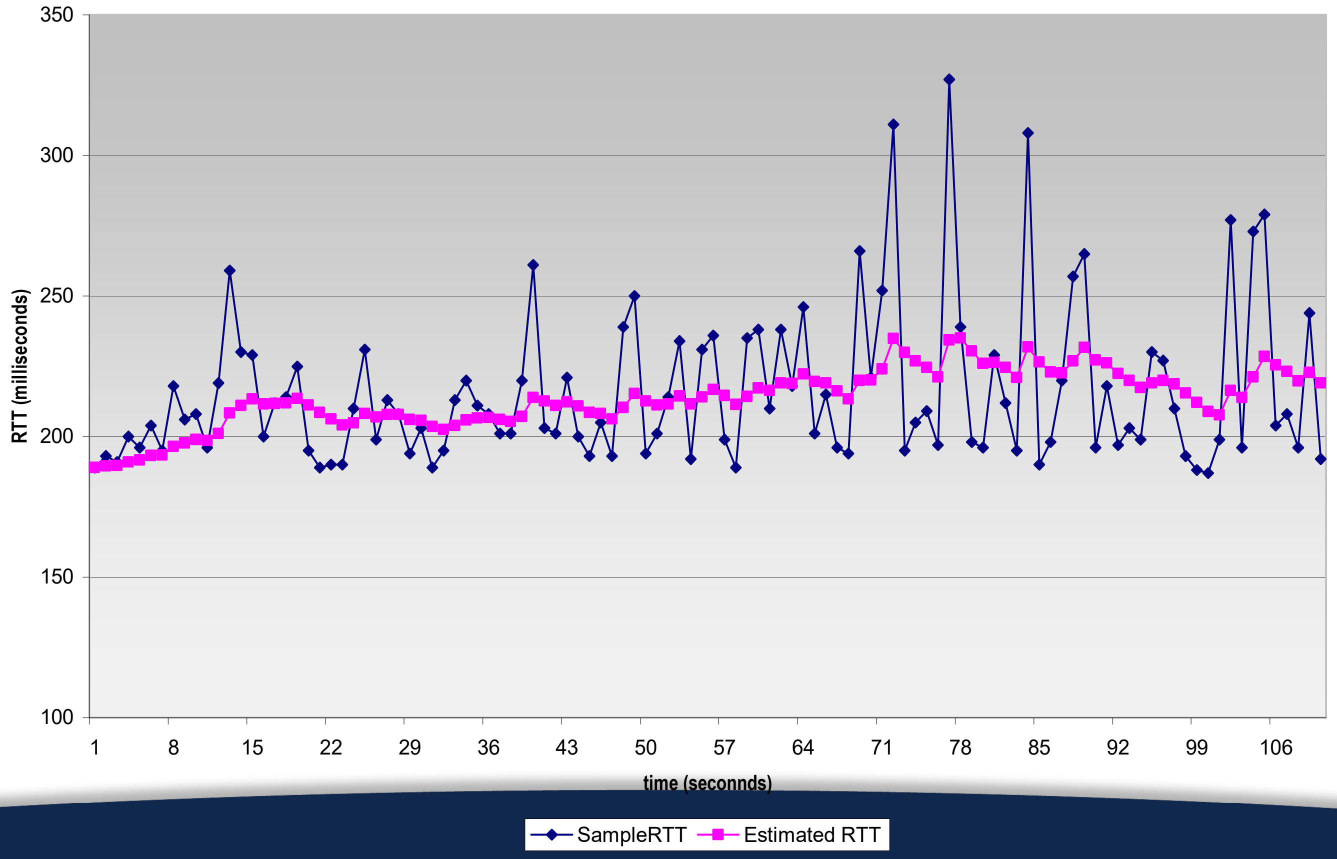
Retransmission timeout

- If the sender hasn’t received an ACK by timeout, retransmit the first packet in the window
- How to set timeout?
  - Too long: connection has low throughput
  - Too short: retransmit packet that was just delayed
- Solution: make timeout proportional to RTT
  - But how do we measure RTT?

RTT estimation

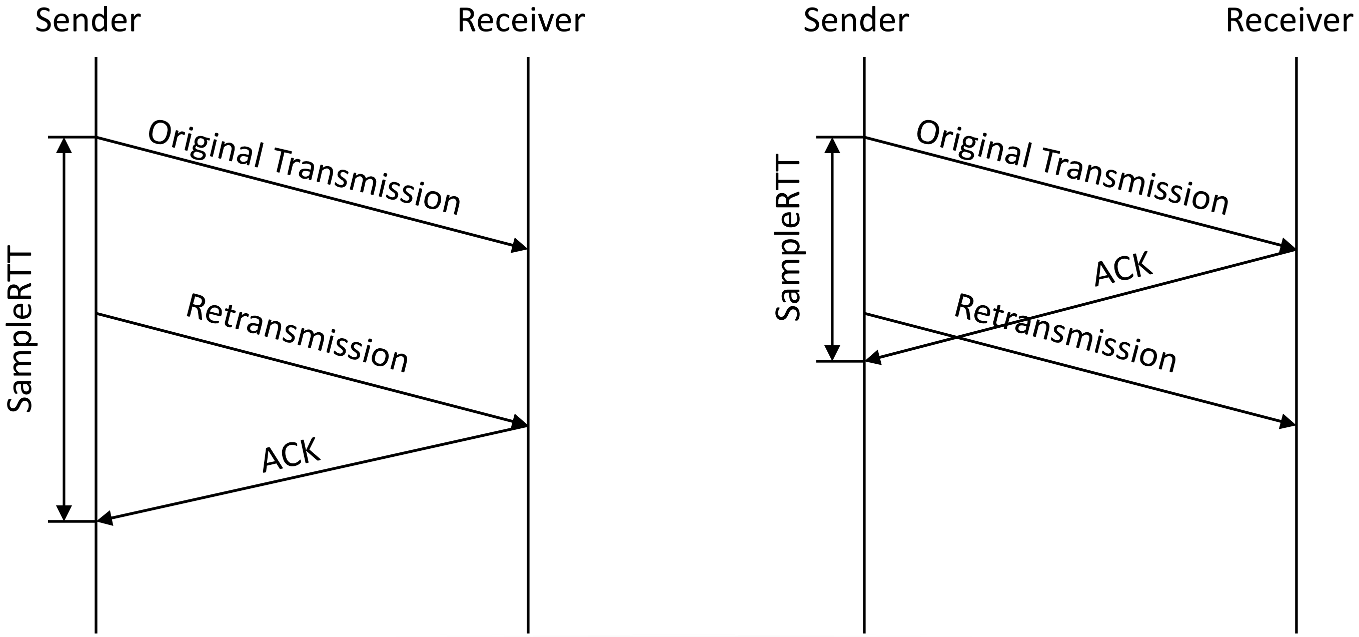
- Exponential weighted average of RTT samples

EstimatedRTT = (1- α)*EstimatedRTT + α*SampleRTT



Problem: Ambiguous measurements

- How do we differentiate between the real ACK, and ACK of the retransmitted packet?



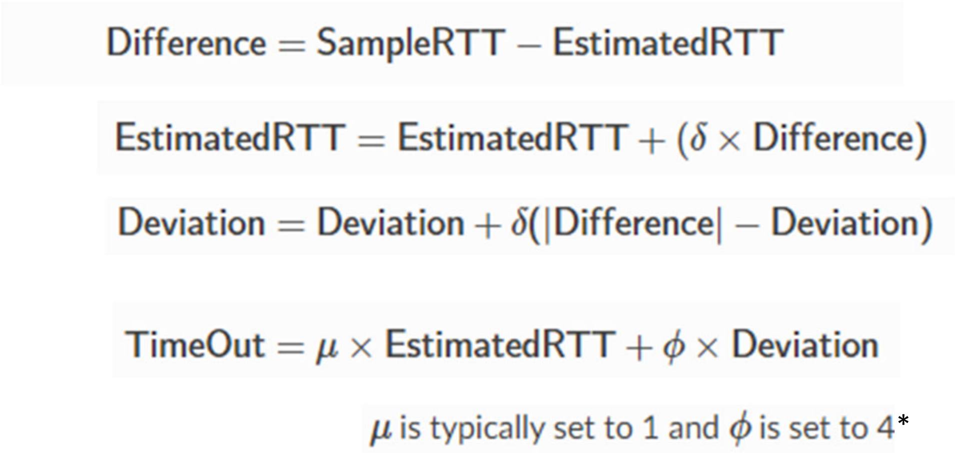
Karn/Partridge algorithm

- Don’t use SampleRTT from retransmissions
  - Once retransmitted, ignore that segment in the future
- Computes EstimatedRTT using α = 0.125
- Timeout value (RTO) = 2 × EstimatedRTT
  - Employs exponential backoff
    - Every time RTO timer expires, set RTO ← 2·RTO

      (Up to maximum ≥ 60 sec)
    - Every time new measurement comes in (= successful original transmission), collapse RTO back to 2 × EstimatedRTT
- Sensitive to RTT variations

Jacobson/Karels algorithm

- Problem: need to better capture variability in RTT
  - Directly measure deviation



$$\text{Difference} = \text{SampleRTT} - \text{EstimatedRTT}$$

$$\text{EstimatedRTT} = \text{EstimatedRTT} + (\delta \times \text{Difference})$$

$$\text{Deviation} = \text{Deviation} + \delta(|\text{Difference}| - \text{Deviation})$$

$$\text{TimeOut} = \mu \times \text{EstimatedRTT} + \phi \times \text{Deviation}$$

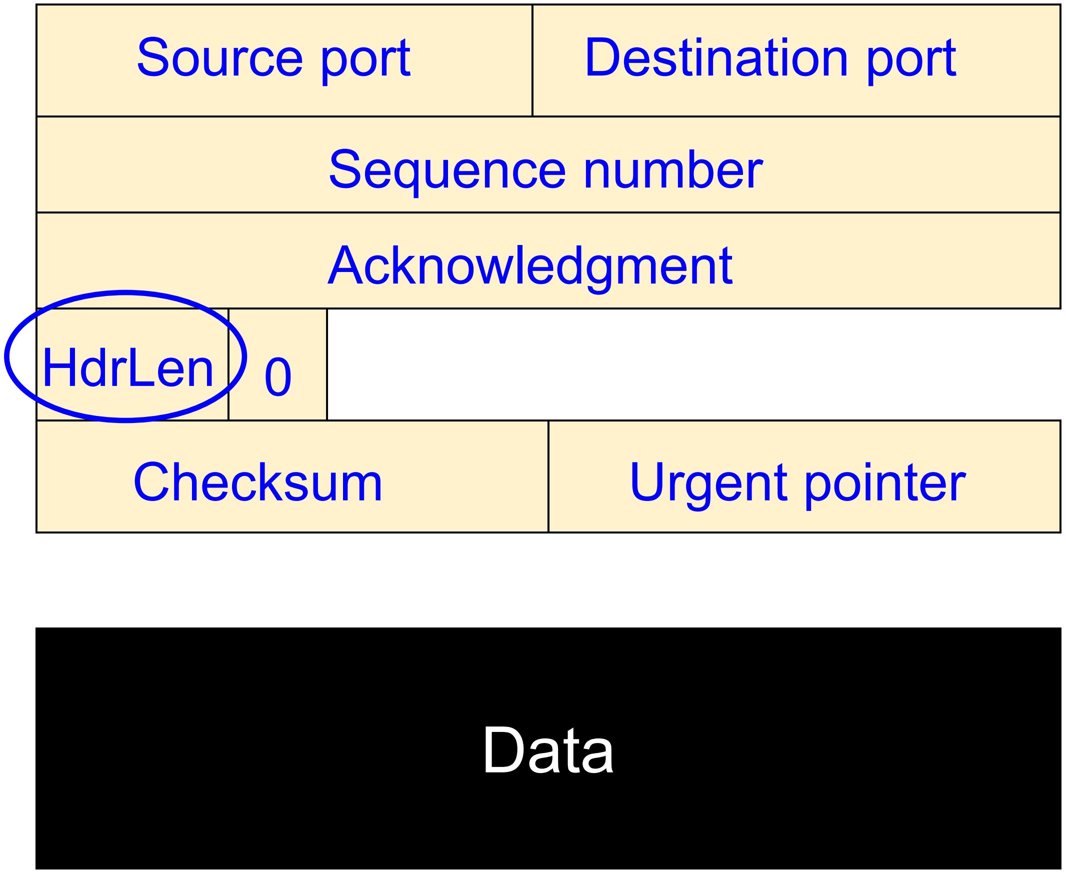
μ is typically set to 1 and ϕ is set to 4*

- RTO = EstimatedRTT + 4 x DeviationRTT

*Peterson and Davie

Build the TCP header

Number of 4-byte words in the header; 5: No options



Source port

Destination port

Sequence number

Acknowledgment

HdrLen

0

Checksum

Urgent pointer

Data

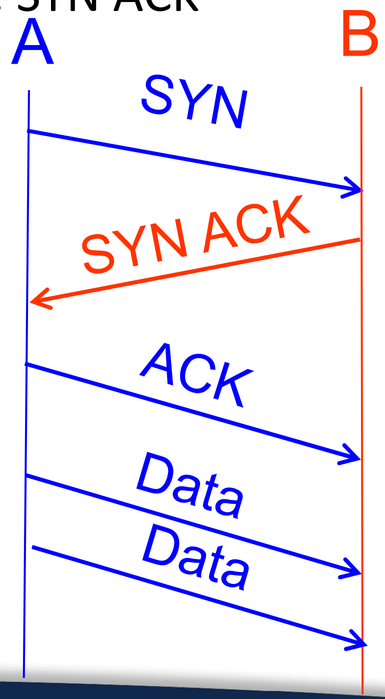
TCP Connection Establishment

Initial Sequence Number (ISN)

- Sequence number for the very first byte
- Why not just use ISN = 0?
    - Practical issue
        - IP addresses and port #s uniquely identify a connection
        - Eventually, though, these port #s do get used again; small chance an old packet is still in flight
        - Also, others might try to spoof your connection
    - Why does using ISN help?
- Hosts exchange ISNs when establishing connection

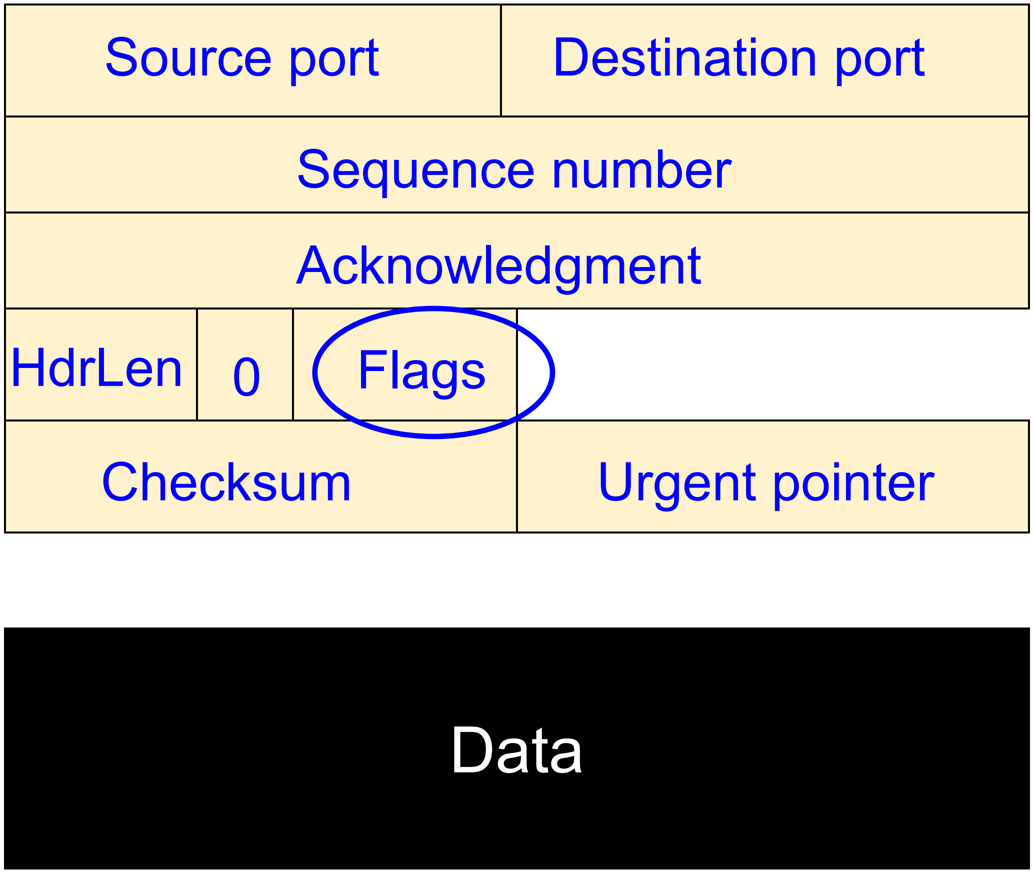
Establishing a TCP connection

- Three-way handshake to establish connection
  - Host A sends a SYN (open; “synchronize sequence numbers”) to host B
  - Host B returns a SYN acknowledgment (SYN ACK)
  - Host A sends an ACK to acknowledge the SYN ACK



Build the TCP header

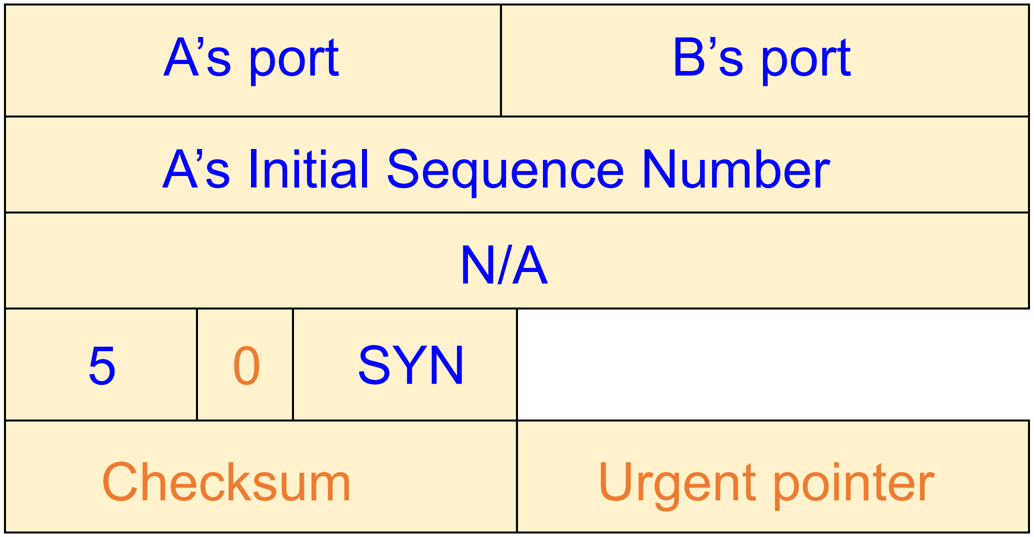
Flags:
SYN
ACK
FIN
RST
PSH
URG



Source port
Destination port
Sequence number
Acknowledgment
HdrLen
0
Flags
Checksum
Urgent pointer
Data

Step 1: A’s initial SYN packet

A tells B to open a connection



A’s port

B’s port

A’s Initial Sequence Number

N/A

5

0

SYN

Checksum

Urgent pointer

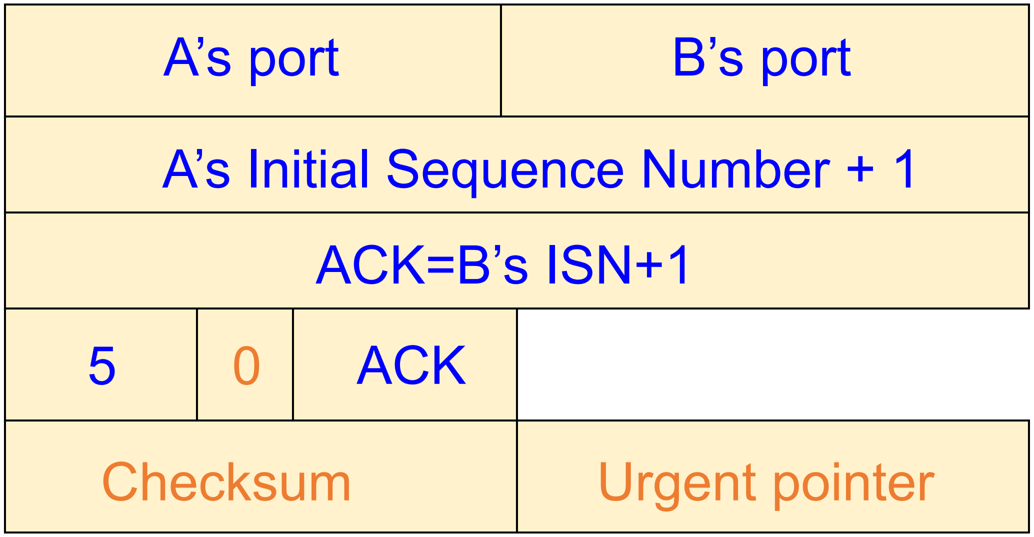
Step 1: B’s SYN-ACK packet

B tells it accepts and is ready to accept next packet

<table>
<tr><td colspan="3">B’s port</td><td>A’s port</td></tr>
<tr><td colspan="4">B’s Initial Sequence Number</td></tr>
<tr><td colspan="4">ACK=A’s ISN+1</td></tr>
<tr><td>5</td><td>0</td><td>SYN|ACK</td><td></td></tr>
<tr><td colspan="3">Checksum</td><td>Urgent pointer</td></tr>
</table>

Step 1: A’s ACK to SYN-ACK

A tells B to open a connection



A’s port

B’s port

A’s Initial Sequence Number + 1

ACK=B’s ISN+1

5

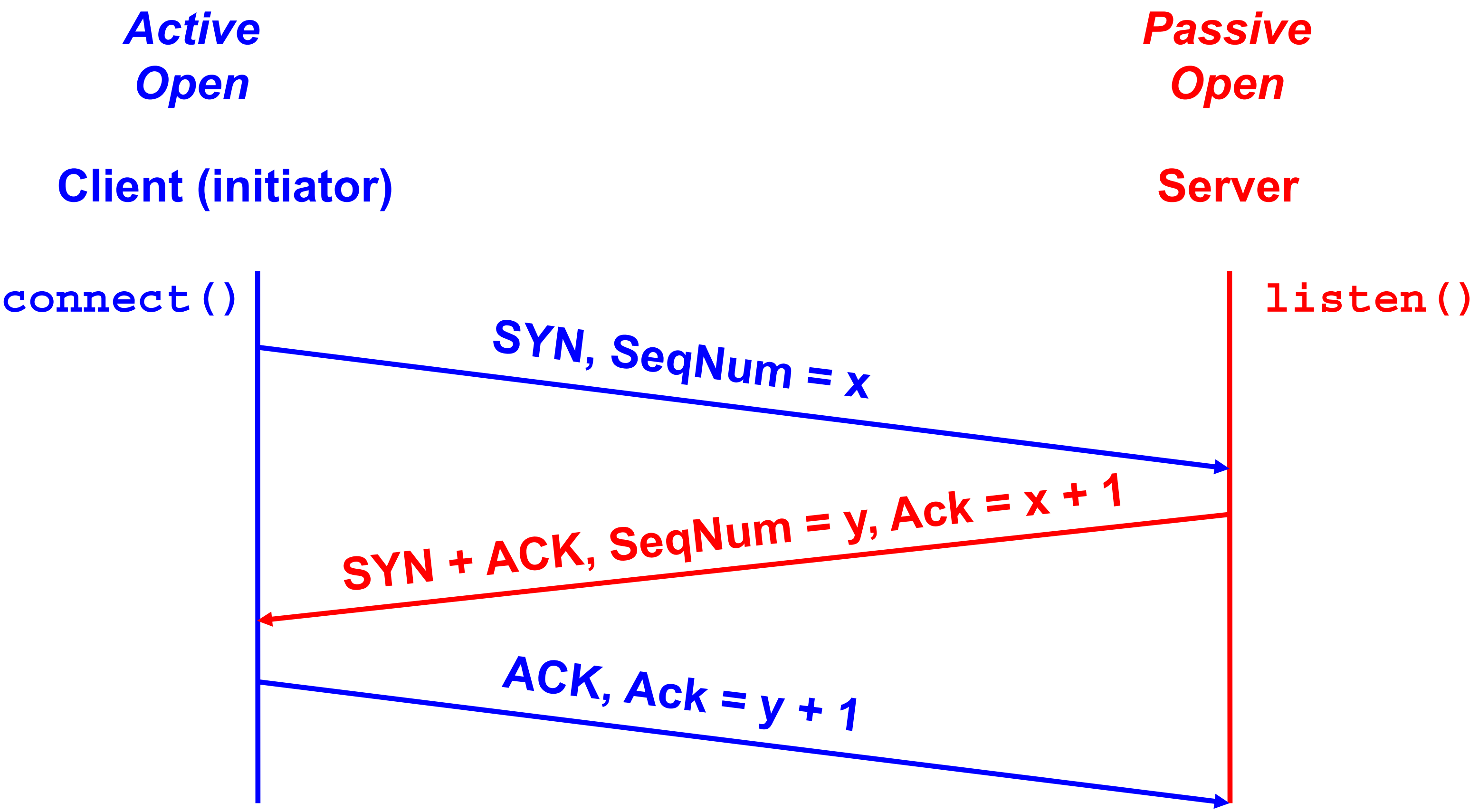
0

ACK

Checksum

Urgent pointer

TCP’s 3-Way handshaking



What if the SYN Packet Gets Lost?

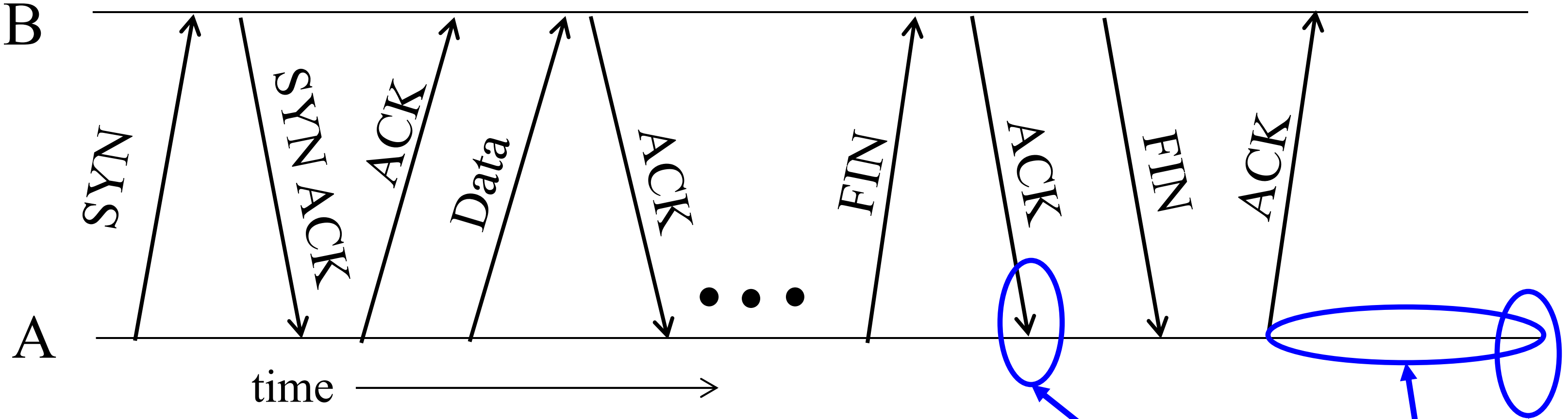
- Suppose the SYN packet gets lost
  - Packet dropped by the network or server is busy
- Eventually, no SYN-ACK arrives
  - Sender retransmits the SYN on timeout
- How should the TCP sender set the timer?
  - Sender has no idea how far away the receiver is
  - Hard to guess a reasonable length of time to wait
  - SHOULD (RFCs 1122 & 2988) use default of 3 seconds
    - Some implementations instead use 6 seconds

SYN loss and web downloads

- User clicks on a hypertext link
  - Browser creates a socket and does a “connect”
  - The “connect” triggers the OS to transmit a SYN
- If the SYN is lost…
  - 3-6 seconds of delay: can be very long
  - User may become impatient and can retry
- User triggers an “abort” of the “connect”
  - Browser creates a new socket and another “connect”
  - Can be effective in some cases

TCP connection teardown

Normal termination, one side at a time



Connection now **half-closed**

Connection now **closed**

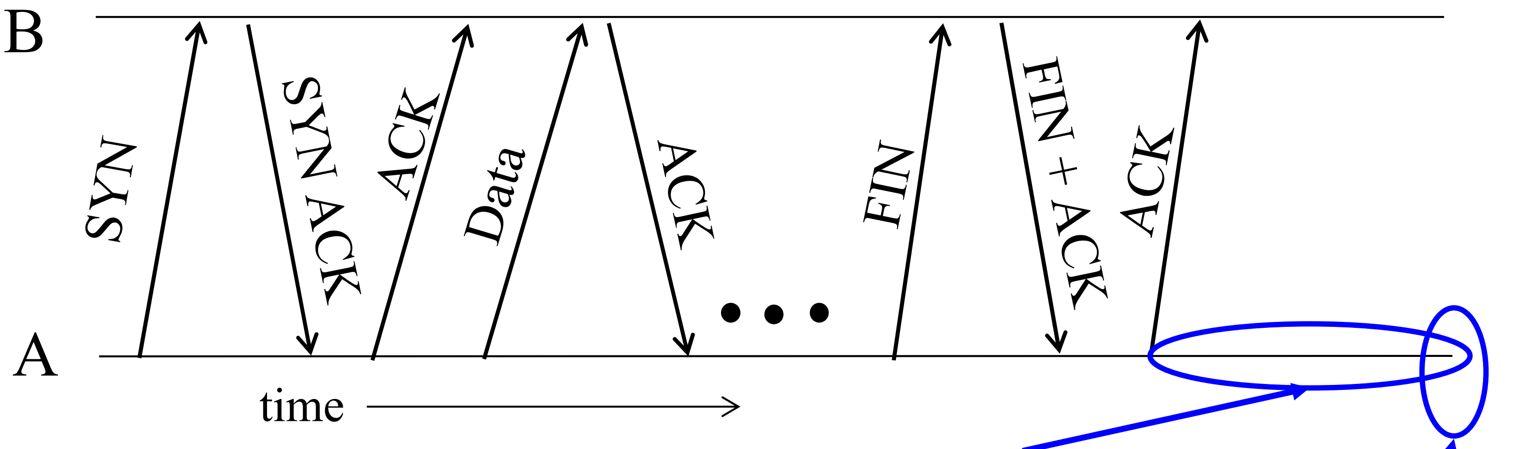
TIME_WAIT:

Avoid reincarnation

B will retransmit FIN if ACK is lost

- Finish (FIN) to close and receive remaining bytes
  - FIN occupies one byte in the sequence space
- Other host acks the byte to confirm
- Closes A’s side of the connection, but not B’s
  - Until B likewise sends a FIN
  - Which A then acks

Normal termination, both together



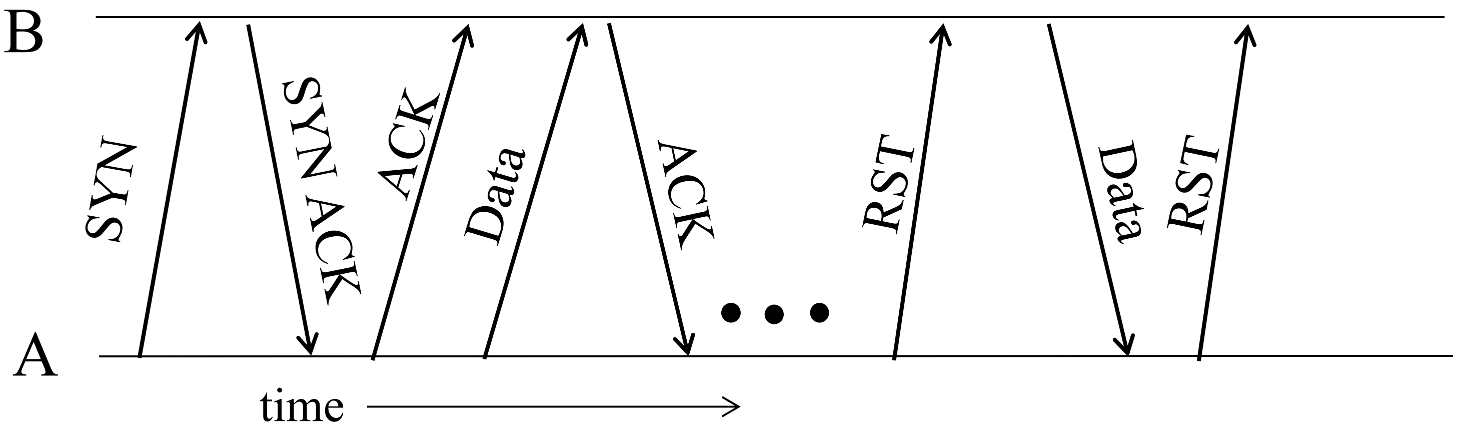
TIME_WAIT:
Avoid reincarnation

Can retransmit
FIN ACK if ACK lost

Connection
now closed

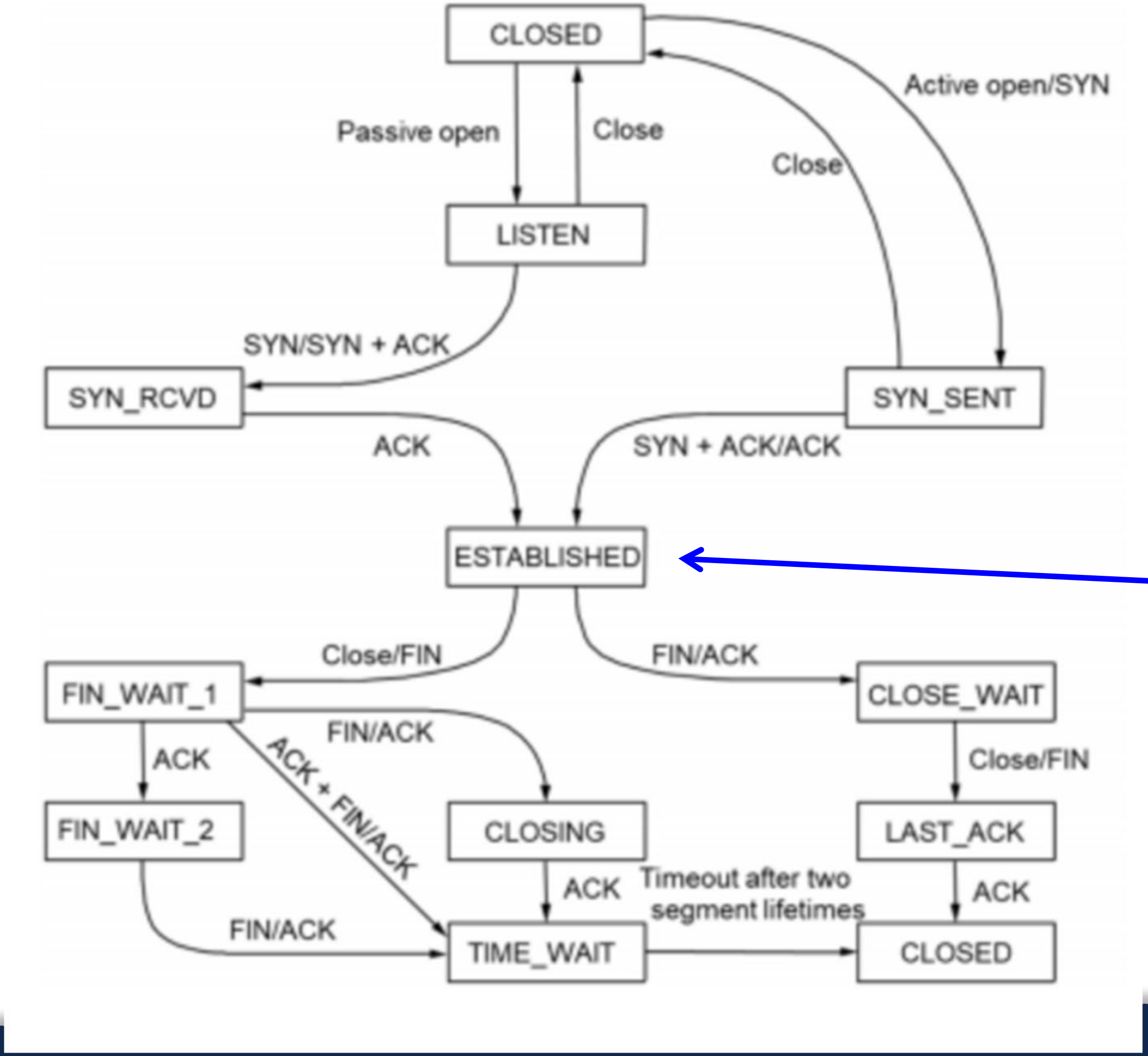
- Same as before, but B sets FIN with their ack of A's FIN

Abrupt termination



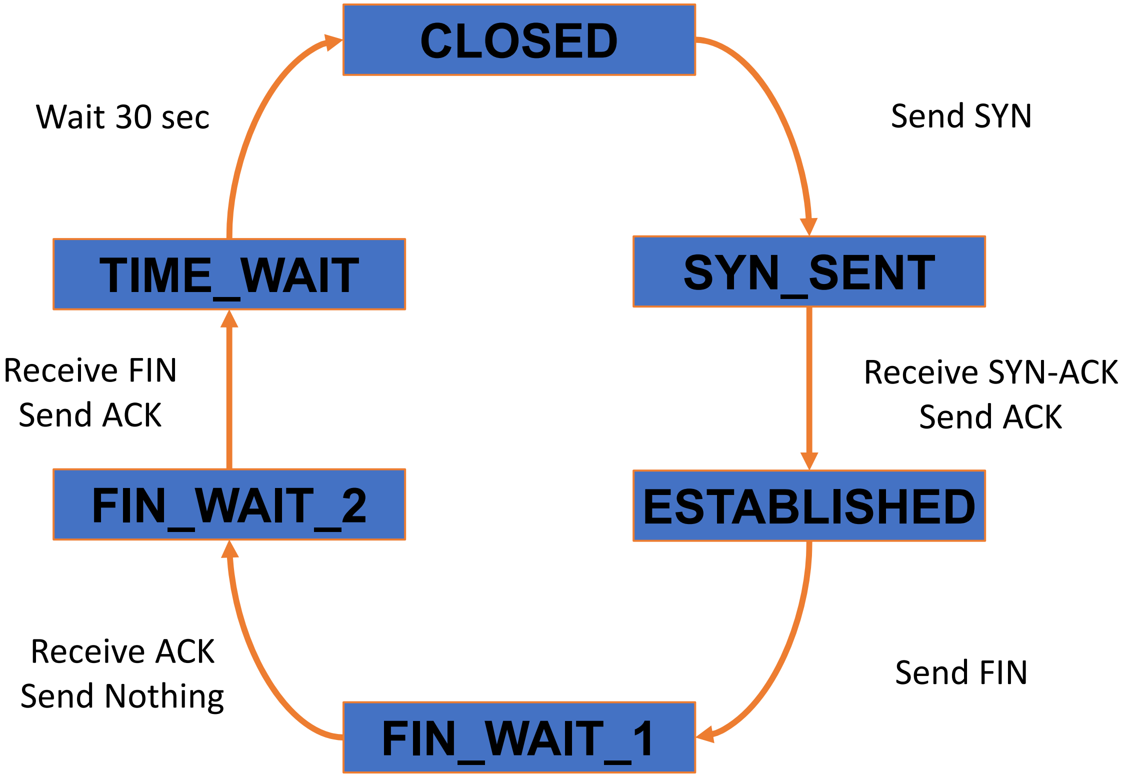
- A sends a RESET (RST) to B
  - E.g., because application process on A crashed
- That’s it
  - B does not ack the RST
  - Thus, RST is not delivered reliably, and any data in flight is lost
  - But: if B sends anything more, will elicit another RST

TCP State Transition



Data, ACK exchanges are in here

TCP client lifecycle



HTTP exchange example

- Client connects to a webserver and generates a GET request

```
GET / HTTP/1.1
Accept: text/html,application/xhtml+xml,application/xml;q=0.9,image/avif,image/webp,image/apng,*/*;q=0.8,application/signed-exchange;v=b3;q=0.7
Accept-Encoding: gzip, deflate
Accept-Language: en-US,en;q=0.9
Cache-Control: no-cache
Connection: keep-alive
Host: www.eecs489.org
Pragma: no-cache
Upgrade-Insecure-Requests: 1
User-Agent: Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/116.0.0.0 Safari/537.36
```

HTTP exchange example

- Server sends a response

```
HTTP/1.1 200 OK
Connection: keep-alive
Content-Length: 10257
Server: GitHub.com
Content-Type: text/html; charset=utf-8
Last-Modified: Wed, 20 Sep 2023 17:31:52 GMT
… (Total of 53,568 bytes)
```

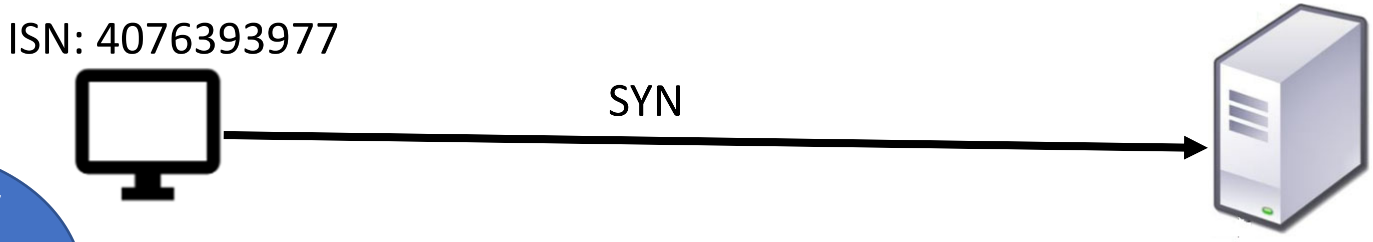
TCP Message Header

U	A	P	R	S	F

<table>
<tr><td colspan="8"><b>Source Port</b></td><td><b>Destination Port</b></td></tr>
<tr><td colspan="9">Sequence Number</td></tr>
<tr><td colspan="9">Acknowledgement</td></tr>
<tr><td>Length</td><td>Reserved</td><td>U</td><td>A</td><td>P</td><td>R</td><td>S</td><td>F</td><td></td></tr>
<tr><td colspan="8">Checksum</td><td>Urgent Pointer</td></tr>
<tr><td colspan="9">Options (0-40 Bytes)</td></tr>
<tr><td colspan="9">Data (Optional)</td></tr>
</table>

Application says connect to server

U	A	P	R	S	F



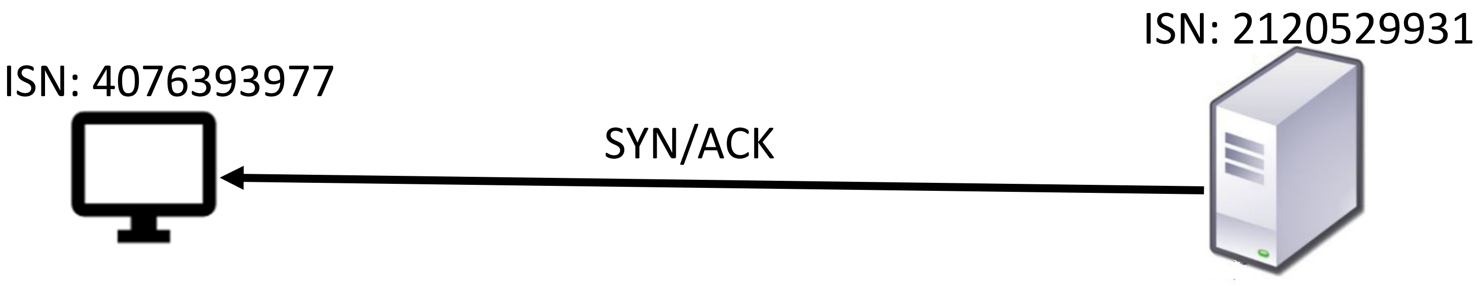
<table>
<tr><td colspan="8">63220</td><td>80</td></tr>
<tr><td colspan="9">4076393977</td></tr>
<tr><td colspan="9">0</td></tr>
<tr><td>5</td><td>Reserved</td><td>0</td><td>0</td><td>0</td><td>0</td><td>1</td><td>0</td><td></td></tr>
<tr><td colspan="8">Checksum</td><td>Urgent Pointer</td></tr>
<tr><td colspan="9">Options (0-40 Bytes)</td></tr>
<tr><td colspan="9">Data (Optional)</td></tr>
</table>

Randomly Picked by the OS for Host

Randomly Picked by TCP

U	A	P	R	S	F

Application says connect to server



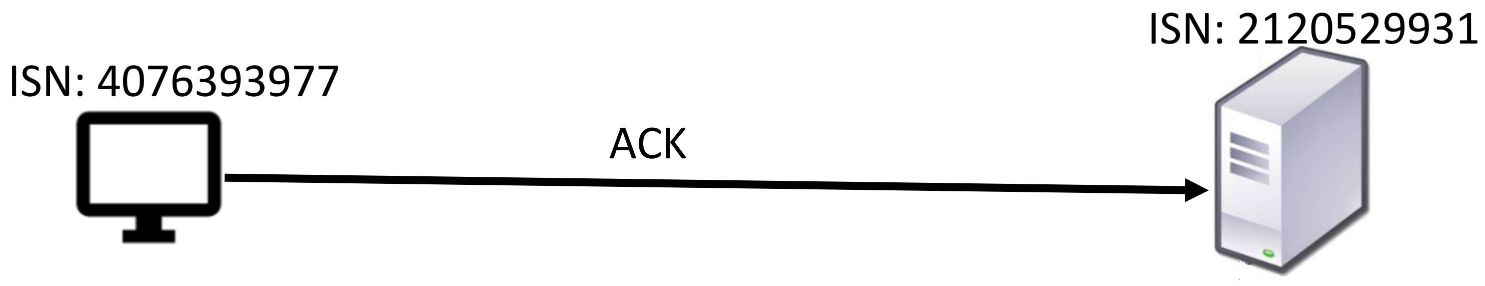
80							63220
2120529931							
4076393978							
ed	0	1	0	0	1	0	
ecksum							Urgent Pointer
Options (0-40 Bytes)							
Data (Optional)							

I have received all packets before this Seq Num

TCP on the server picks an ISN

U	A	P	R	S	F

Application says connect to server



ISN: 4076393977

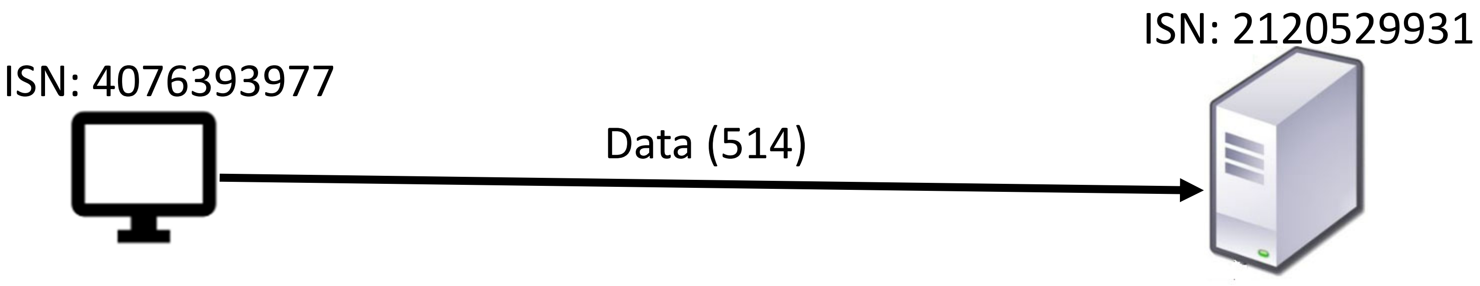
ACK

ISN: 2120529931

<table>
<tr><td colspan="8"><b>63220</b></td><td><b>80</b></td></tr>
<tr><td colspan="9">4076393978</td></tr>
<tr><td colspan="9">2120529932</td></tr>
<tr><td>5</td><td>Reserved</td><td>0</td><td>1</td><td>0</td><td>0</td><td>0</td><td>0</td><td></td></tr>
<tr><td colspan="8">Checksum</td><td>Urgent Pointer</td></tr>
<tr><td colspan="9">Options (0-40 Bytes)</td></tr>
<tr><td colspan="9">Data (Optional)</td></tr>
</table>

Application Sends data (451 Bytes)

U	A	P	R	S	F


ISN: 2120529931

ISN: 4076393977

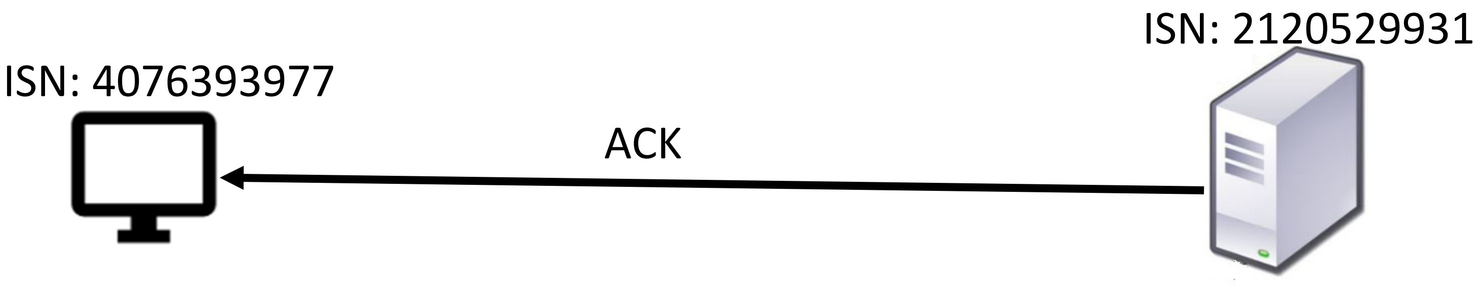
Data (514)

62330								**80**
4076393979								
2120529932								
5	Reserved	0	0	0	0	0	0	
Checksum								Urgent Pointer
Options (0-40 Bytes)								

G	E	T		i	n	d	e	x	.	h	t	m	l		H	T	T	P	/	1	.	1		A	C	C	E	P	T	.	.

U A P R S F

Application says connect to server



80 | 6230

2120529932

4076394493

5 | Reserved | 0 1 0 0 0 0

Checksum | Urgent Pointer

Options (0-40 Bytes)

Data (Optional)

4076393979
+ 514

Bonus Quiz 7

- https://forms.gle/iroh9cEyjUxi1L238