

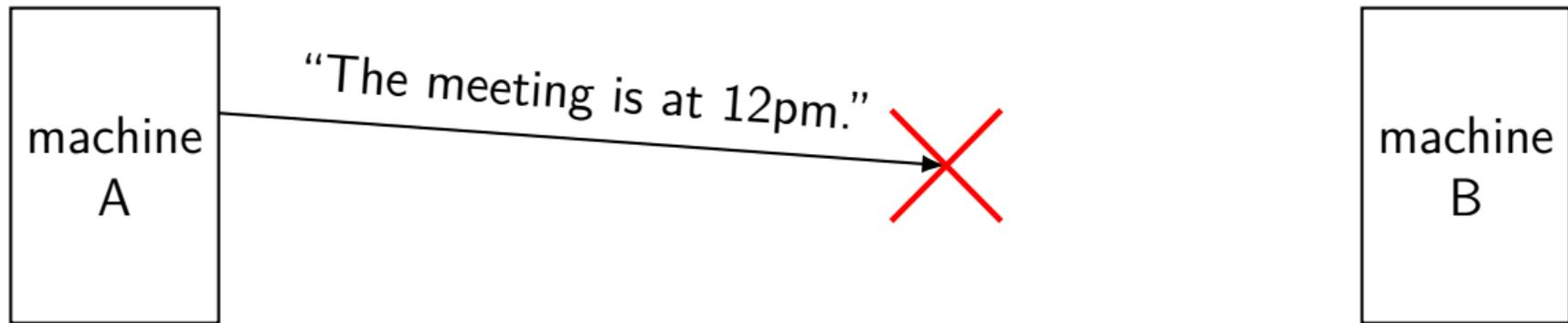
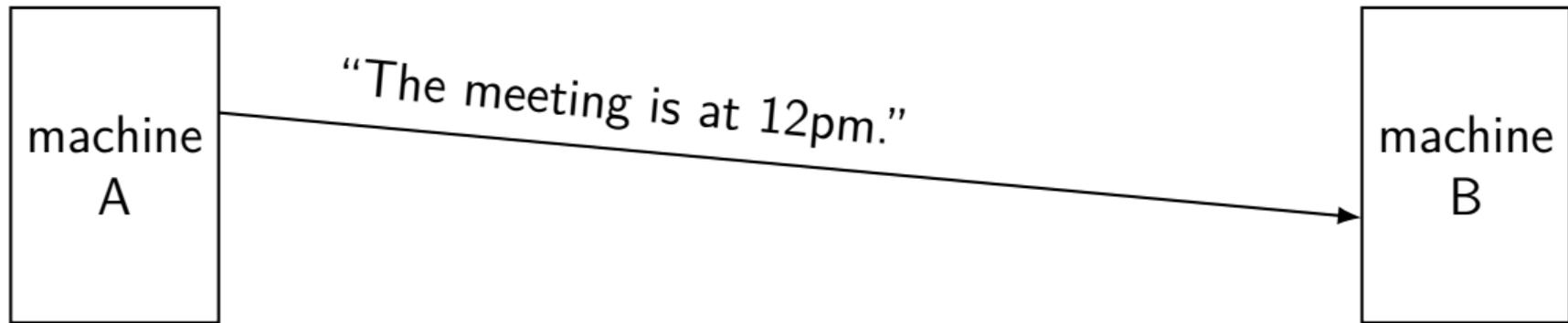
changelog

6 Sep 2024: add explicit note re: ACK up to X being inclusive first time it appears

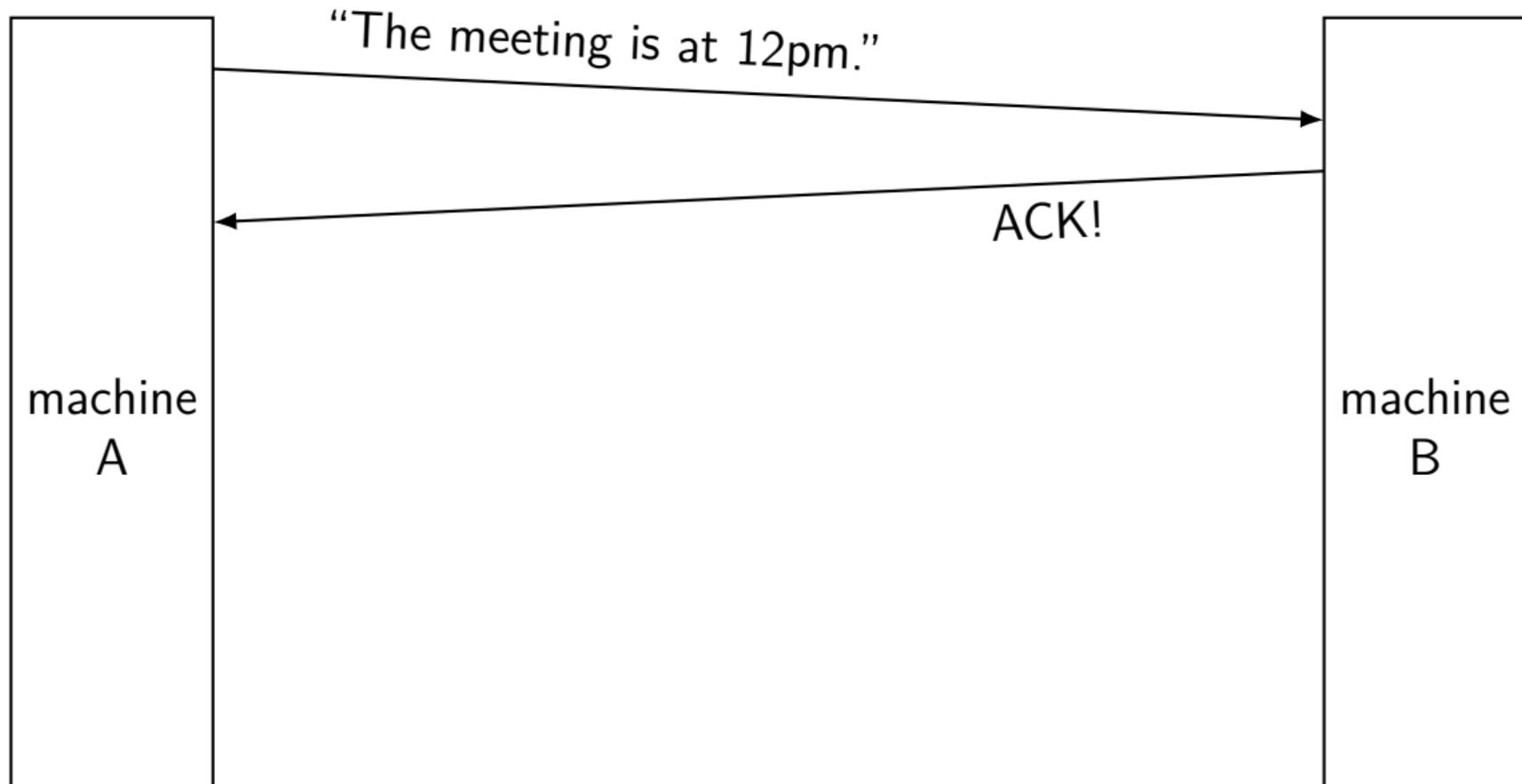
10 Sep 2024: be more clear that ACK number is 1+last byte sequence number

10 Sep 2024: correct discussion of window size variation to discuss burstiness, which means that the maximum possible latency may not be where the throughput collapse happens

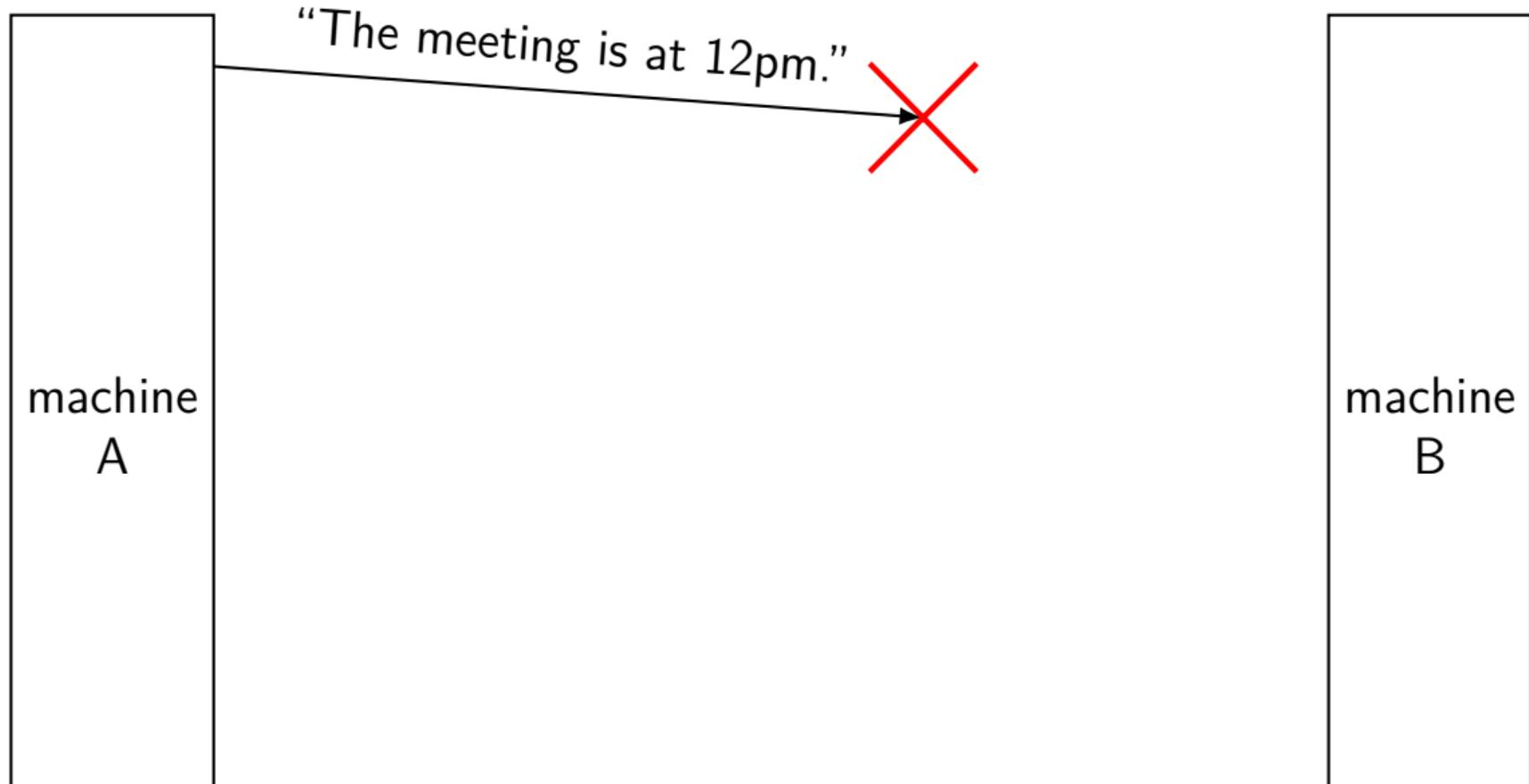
dealing with network message lost



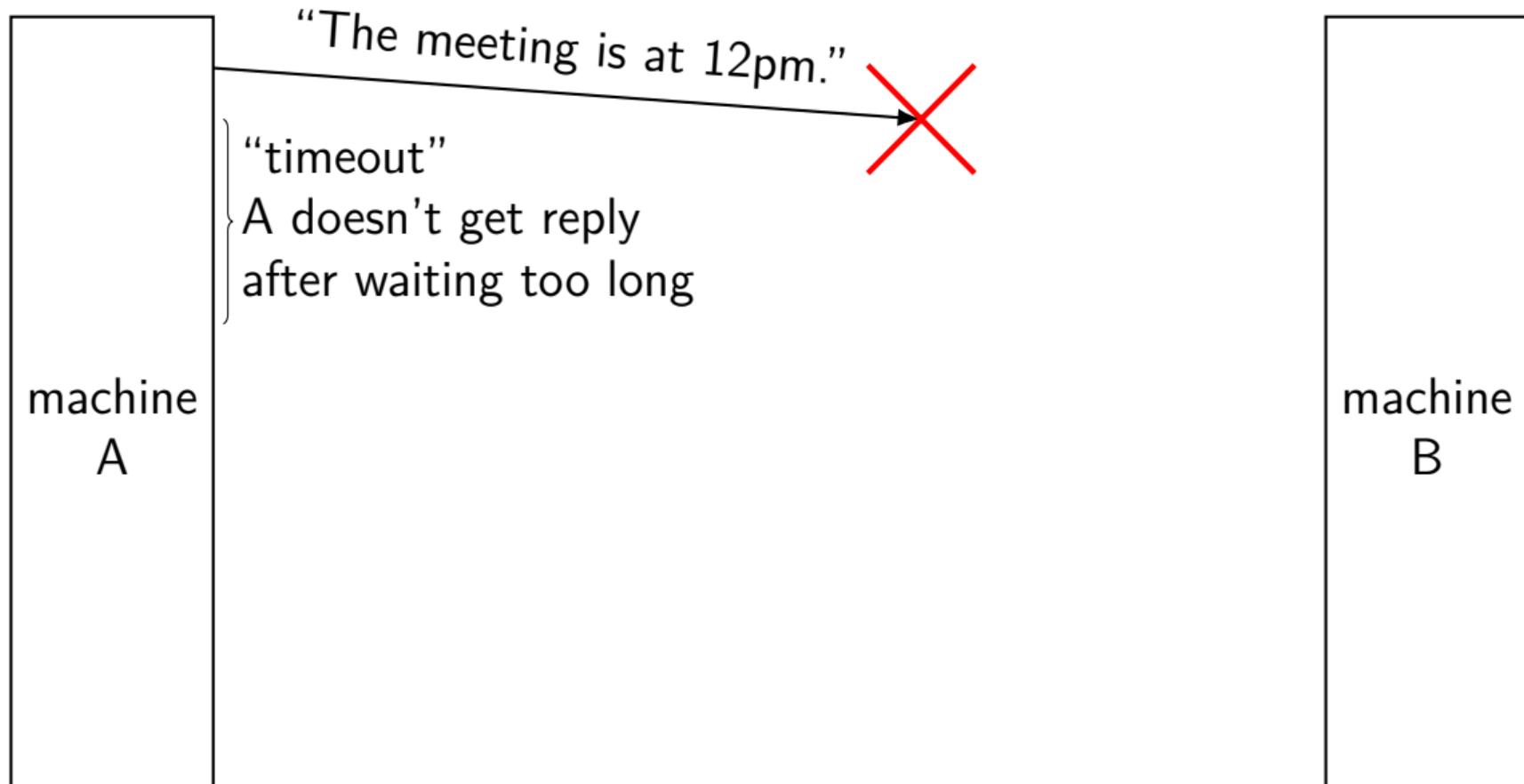
handling lost message: acknowledgements



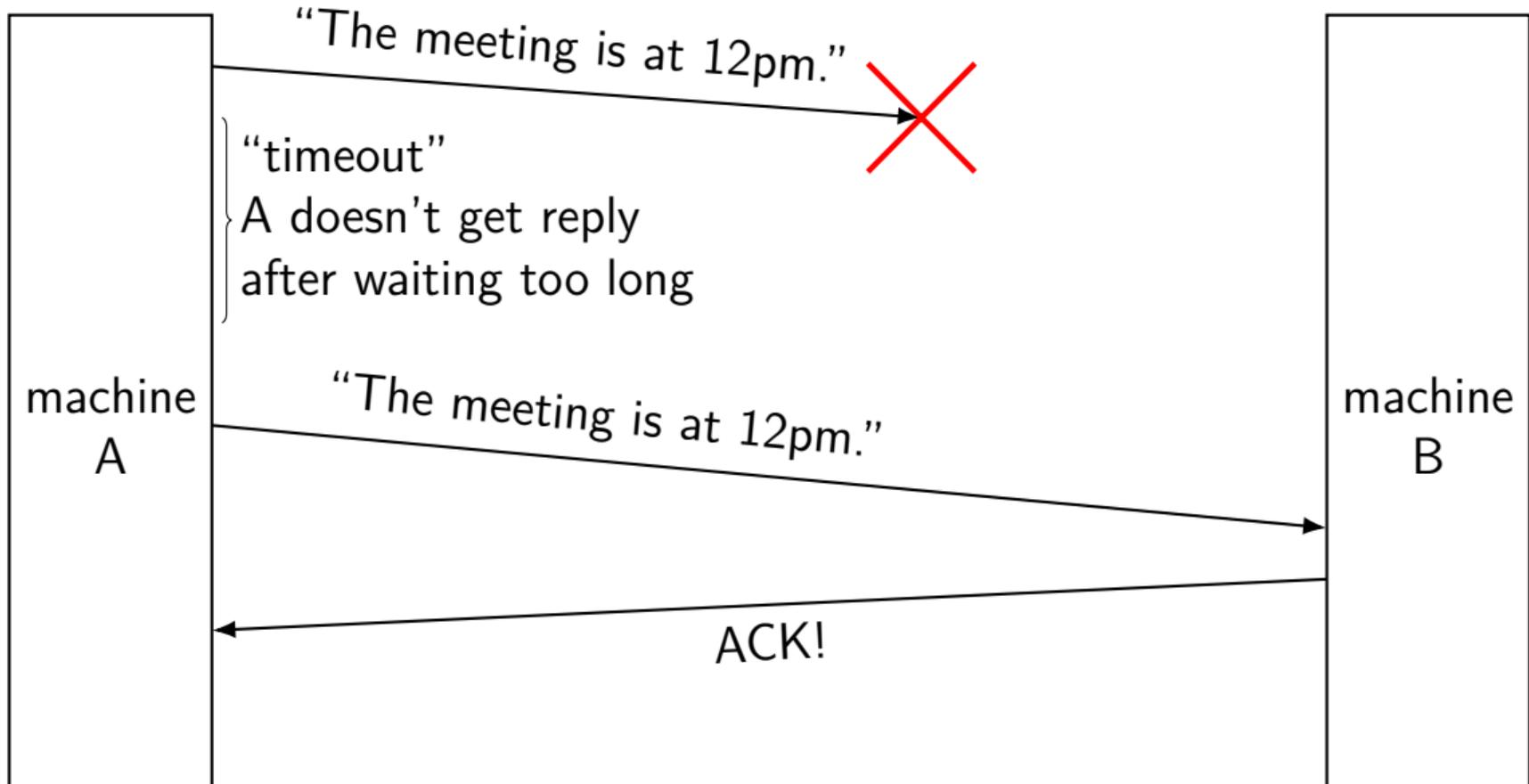
handling lost message



handling lost message



handling lost message



protocol so far

on sender: until ACK received:

- (re)send frame of data

- wait fixed amount of time for ACK

on receiver: continuously:

- wait for frame of data

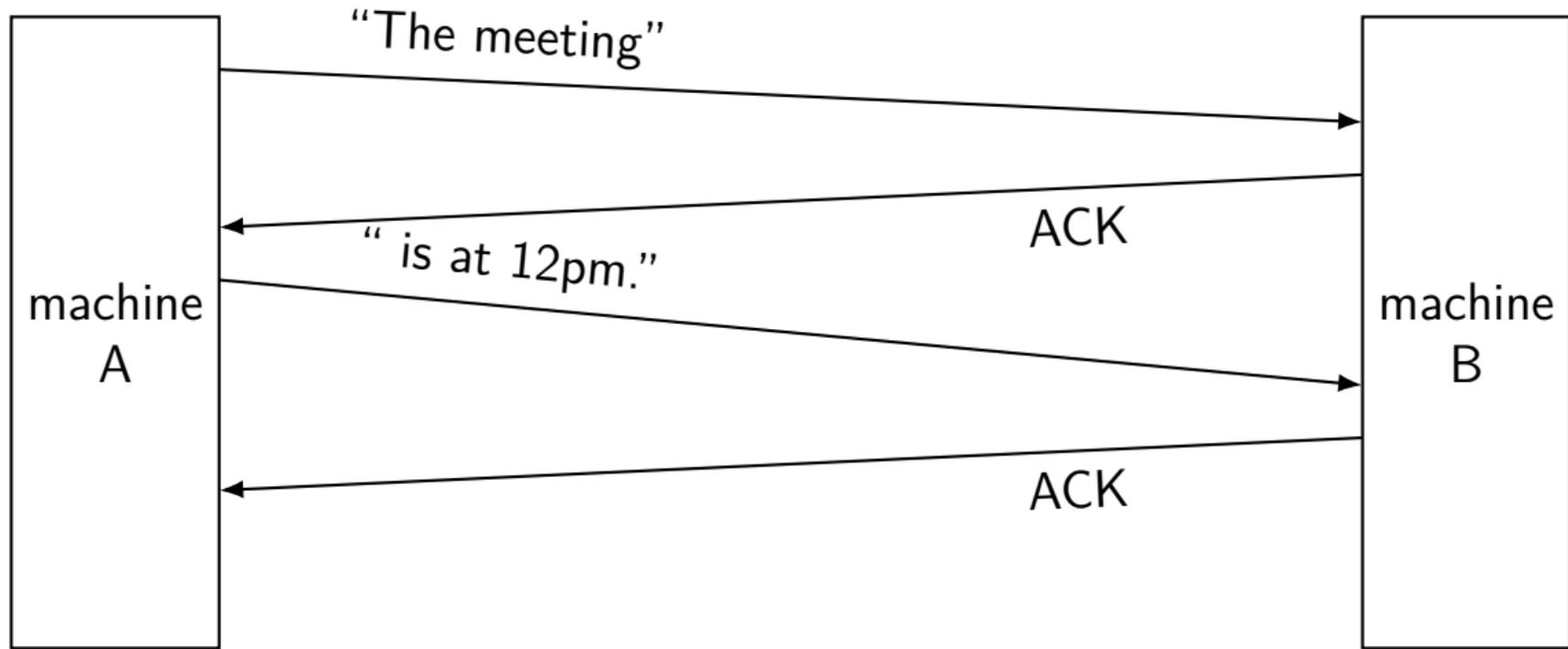
- send ACK back

problem

really want to send multiple frames

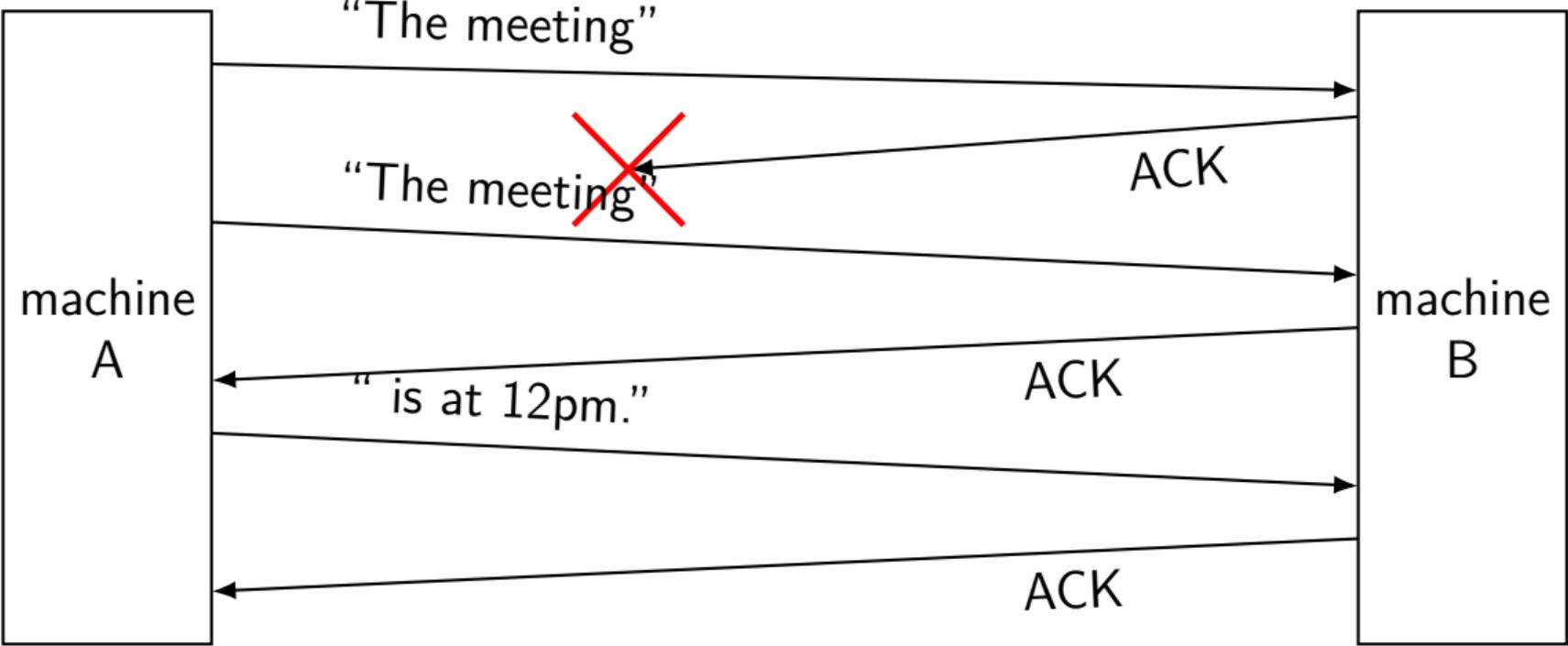
example: data split in multiple pieces

splitting messages: try 1

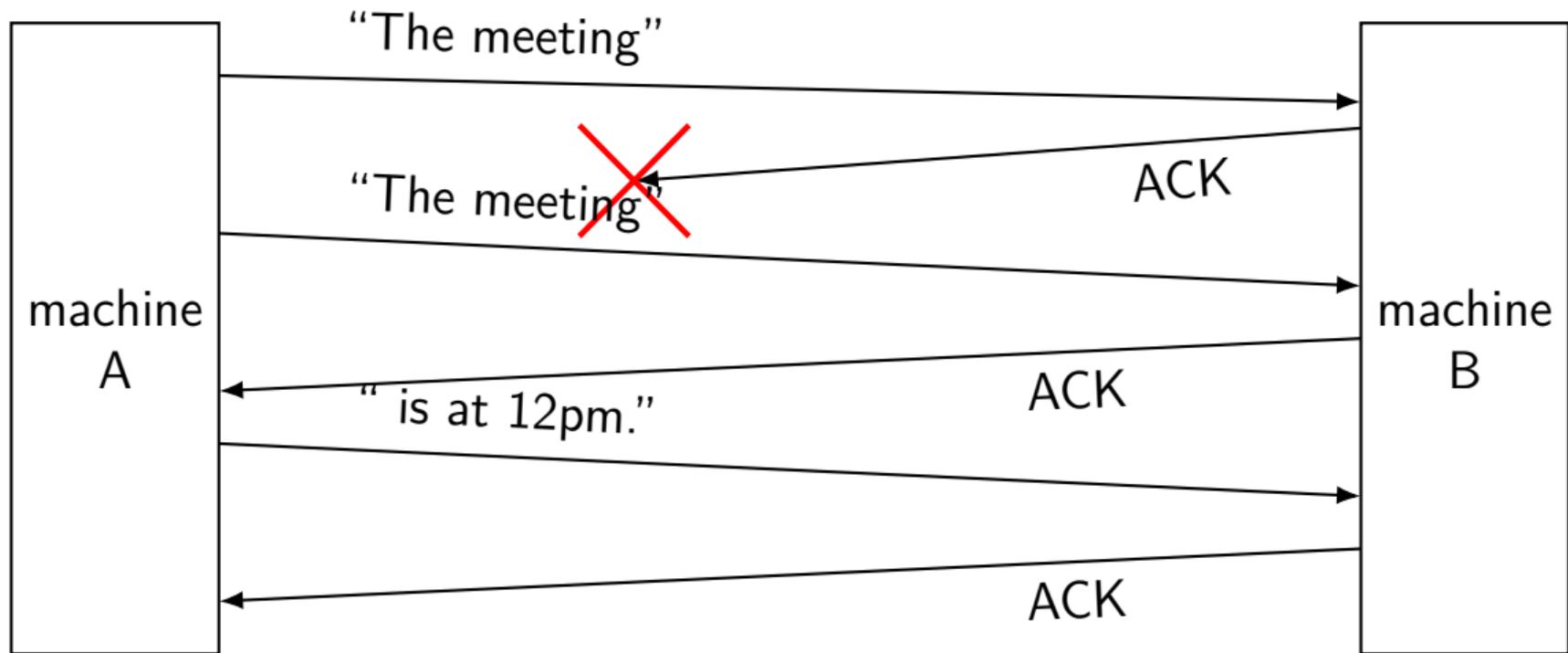


reconstructed message:
The meeting is at 12pm.

splitting messages: try 1 — problem 1



splitting messages: try 1 — problem 1



reconstructed message:

The meetingThe meeting is at 12pm.

exercise: other problems?

sending 'The meeting', 'is at 12pm'

what would be received for each of these scenarios?

1. message (instead of acknowledgment) is lost
2. first message from machine A is delayed a long time by network
3. acknowledgment of second message lost instead of first

aside: message delays

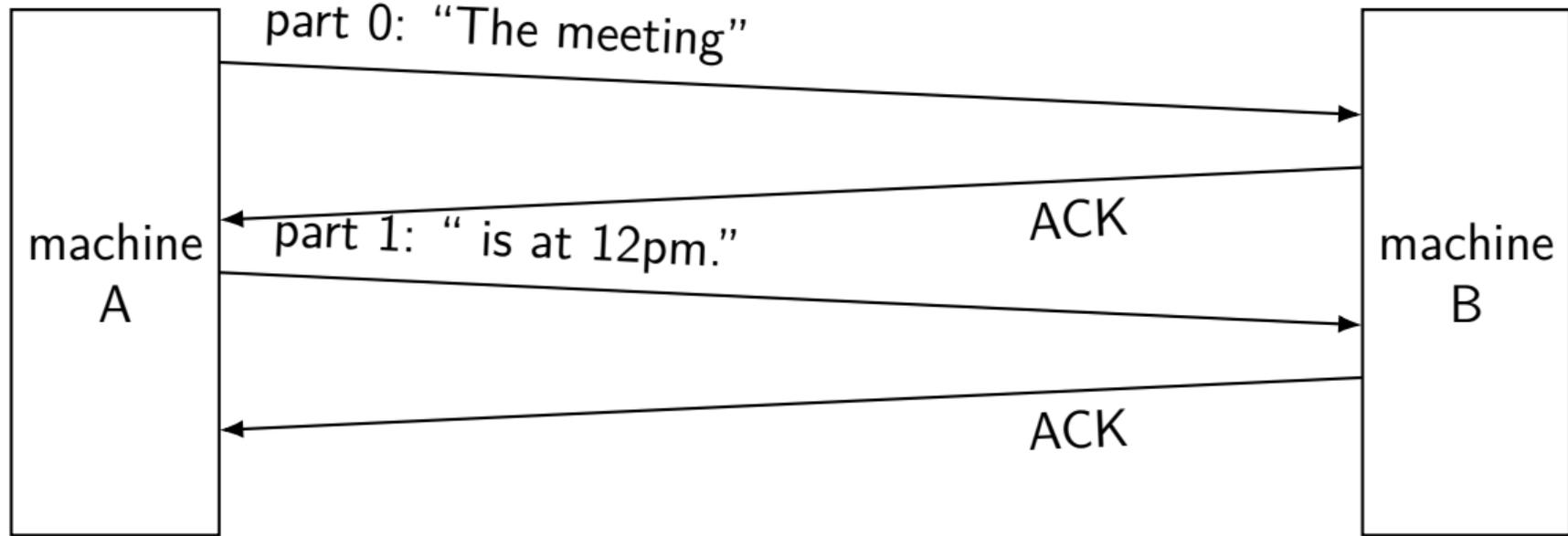
long message delays not possible with direct link

but are possible with:

- multiple paths from A to B

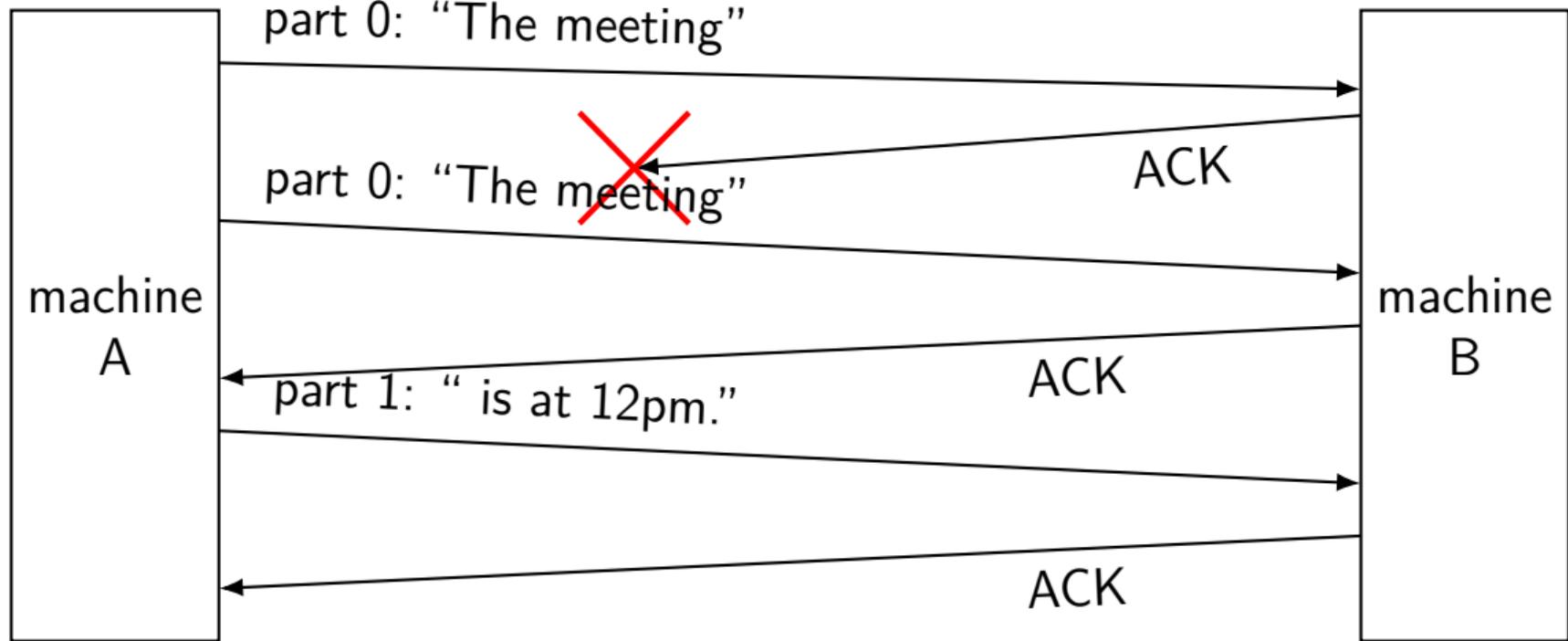
- doing this kind of acknowledgment + resending hop-by-hop

splitting messages: try 2



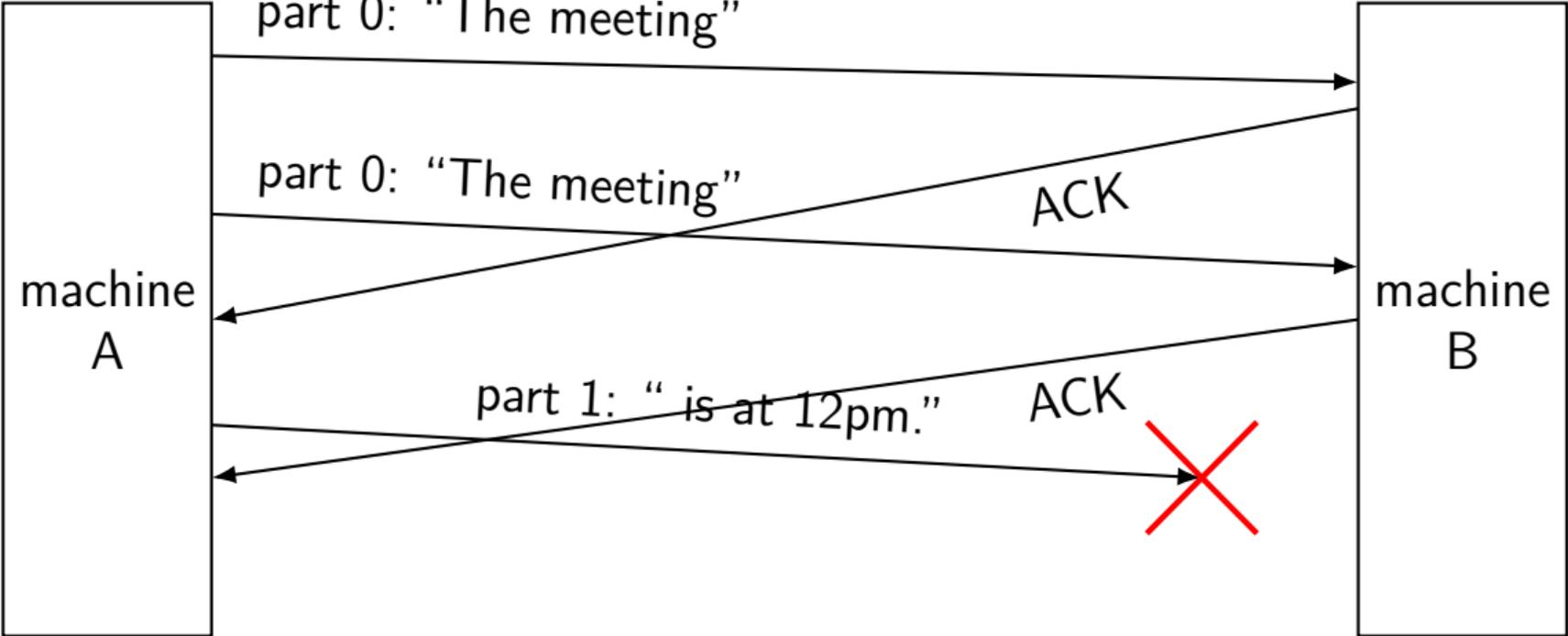
reconstructed message:
The meeting is at 12pm.

splitting messages: try 2 — missed ack



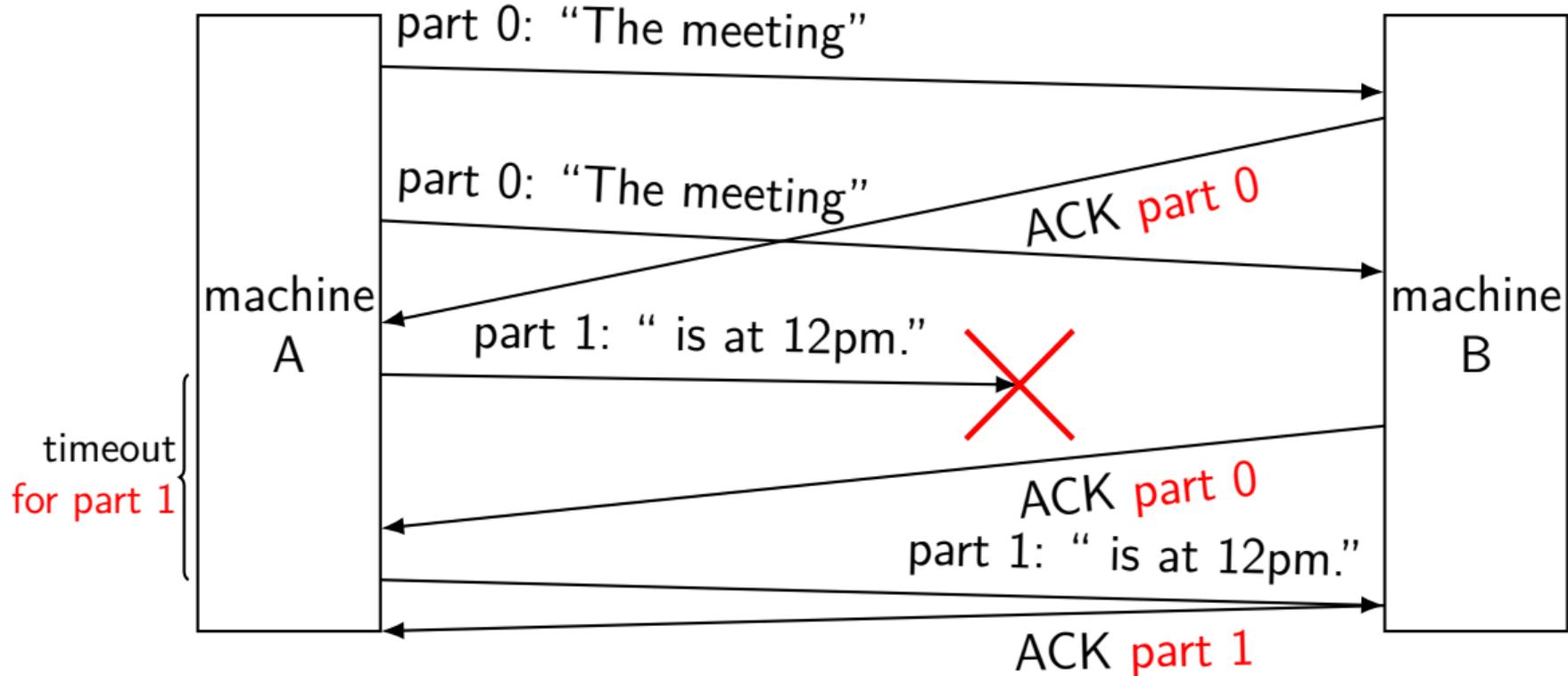
reconstructed message:
The meeting is at 12pm.

splitting messages: try 2 — problem



A thinks: part 0 + part 1 acknowledged!

splitting messages: version 3



sequence numbers

call the 'part' label *sequence number*

for now: sequence number = message (or *segment*) number

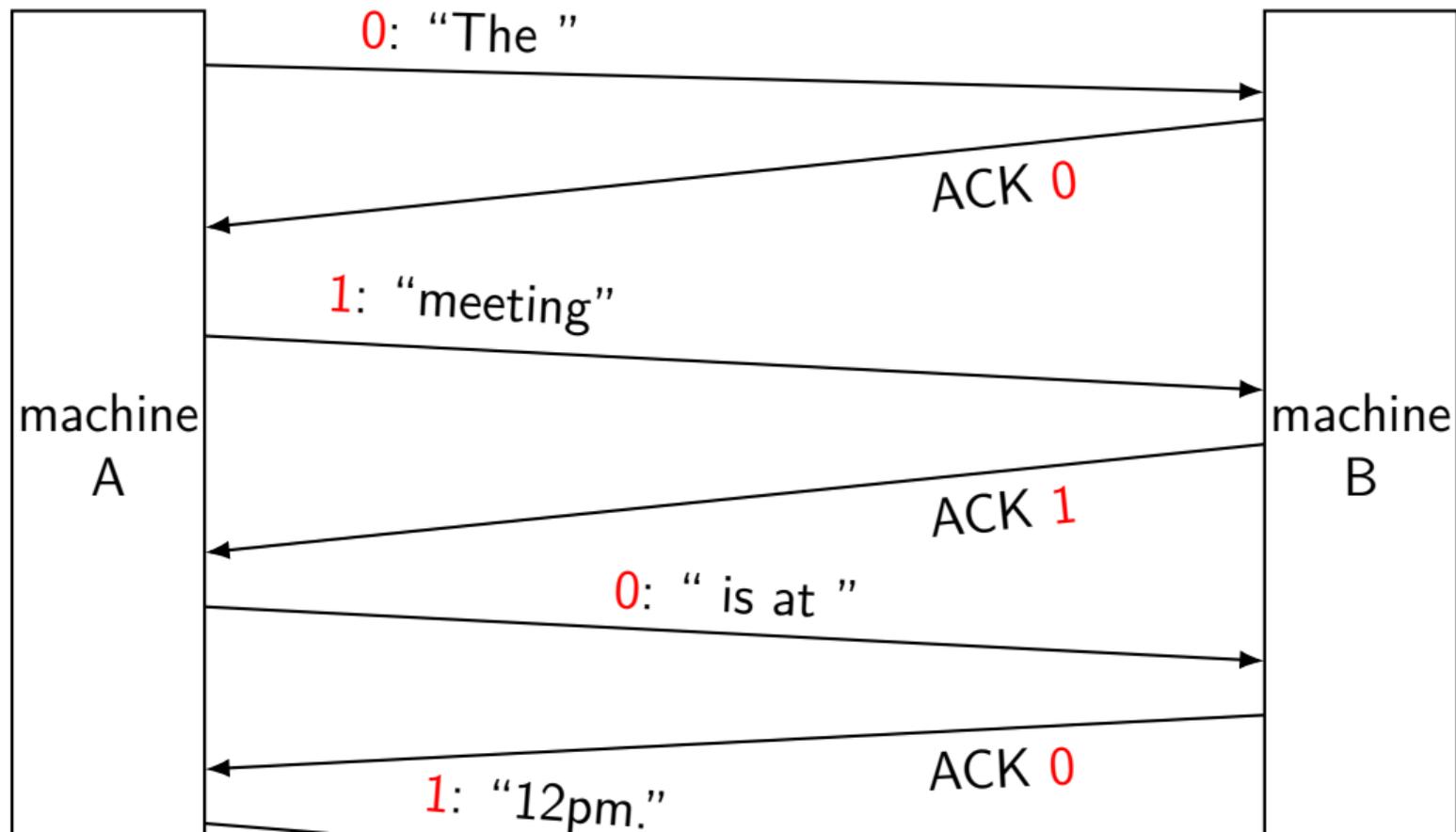
in TCP: sequence number = byte number

important question: how large can they get?

if we never reuse them — infinite!

so *really* want to reuse them

1-bit sequence number



'stop and wait'

machine A is only sending **one thing at a time**

never start sending next thing until after sending previous thing

stop-and-wait exercise (receive, 1)

machine B receives 0 : X

machine B sends ACK 0

machine B receives 0 : X

what should machine B do now?

- A. send ACK 0 B. send ACK 1 C. send nothing

stop-and-wait exercise (receive, 2)

machine B receives 0: X

machine B sends ACK 0

machine B receives 1: X

what should machine B do now?

- A. send ACK 0 B. send ACK 1 C. send nothing

stop-and-wait exercise (receive, 3)

machine B receives 0: X

machine B sends ACK 0

machine B receives 1: Y

machine B sends ACK 1

machine B receives 0: X

what should machine B do now?

- A. send ACK 0 B. send ACK 1 C. send nothing

stop-and-wait exercise (send, 1)

A trying to send 'X', then 'Y', then 'Z'

machine A sends 0: X

machine A sends 0: X

machine A receives ACK 0

machine A sends 1: Y

machine A receives ACK 0

what should machine A do now?

- A. send 0: X again
- B. send 1: Y again
- C. send 0: Z
- D. something else

stop-and-wait exercise (send, 2)

A trying to send 'X', then 'Y', then 'Z'

machine A sends 0: X

machine A sends 0: X

machine A receives ACK 0

machine A sends 1: Y

machine A receives ACK 1

what should machine A do now?

- A. send 0: X again
- B. send 1: Y again
- C. send 0: Z
- D. something else

stop-and-wait issues

two issues with stop-and-wait:

doesn't use close to full capacity of network

not clear how to set timeouts

looking at metrics

several important metrics we'll care about

(both for this and future topics)

looking at metrics

several important metrics we'll care about

(both for this and future topics)

throughput and *bandwidth* (\sim how much capacity used/available)

latency and *round-trip time* (RTT) (\sim what timeouts needed)

jitter (\sim safety margin for timeouts)

bandwidth / throughput

bandwidth / data rate: maximum rate we can send per unit time
most commonly measuring the speed of a link

1 gigabit/second = transmit 1 bit / nanosecond

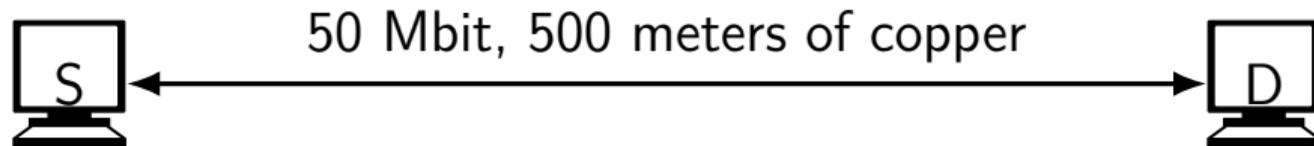
throughput: achieved rate per unit time

often lower than total bandwidth because of losses
(we'll give several examples throughout the semester)

latency (1)

latency: time for message: SOURCE \rightarrow DEST

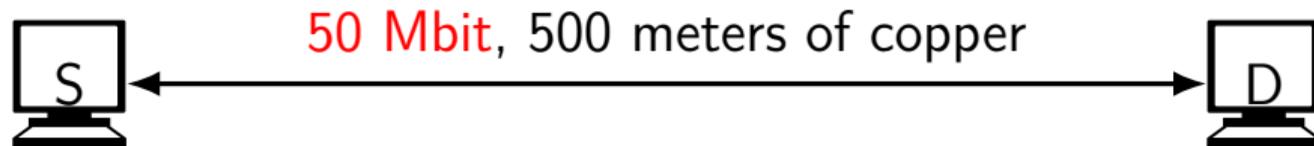
example: 1000 bit message from S to D:



latency (1)

latency: time for message: SOURCE \rightarrow DEST

example: 1000 bit message from S to D:



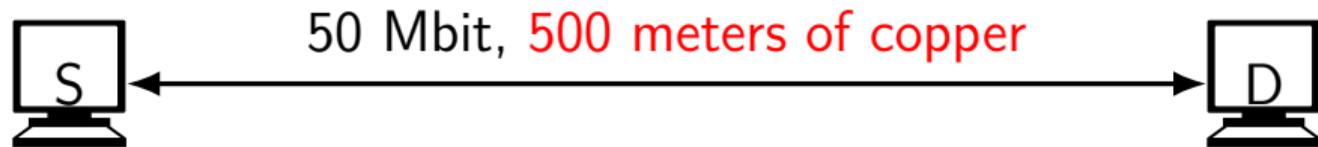
one bit sent each $1/50\text{M}$ second = $0.02 \mu\text{s}$

1000 bits take $0.02 \times 1000 = 20 \mu\text{s}$ to sent
“transmission delay”

latency (1)

latency: time for message: SOURCE \rightarrow DEST

example: 1000 bit message from S to D:



one bit sent each $1/50\text{M}$ second = $0.02 \mu\text{s}$

1000 bits take $0.02 \times 1000 = 20 \mu\text{s}$ to sent

“transmission delay”

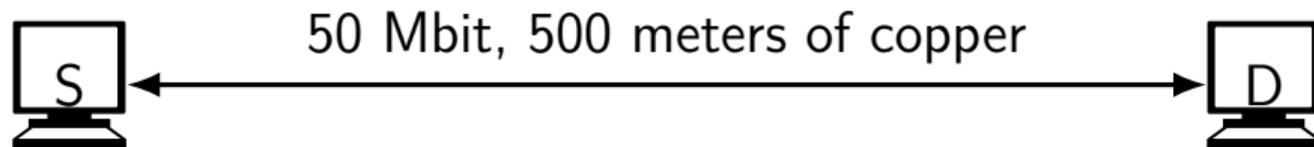
+ 2.2 microseconds for bit to go down cable ($2.3 \times 10^8 \text{ m/s}$)

“propagation delay”

latency (1)

latency: time for message: SOURCE \rightarrow DEST

example: 1000 bit message from S to D:



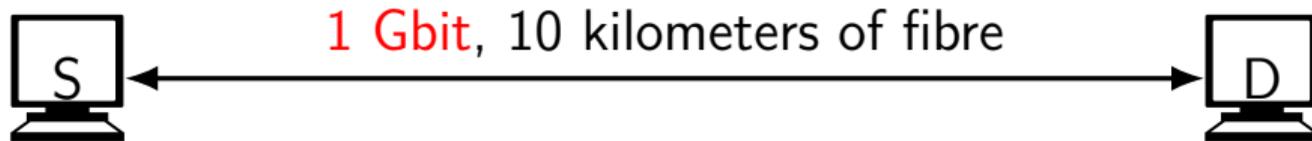
one bit sent each $1/50\text{M}$ second = $0.02 \mu\text{s}$

1000 bits take $0.02 \times 1000 = 20 \mu\text{s}$ to sent
"transmission delay"

+ 2.2 microseconds for bit to go down cable ($2.3 \times 10^8 \text{ m/s}$)
"propagation delay"

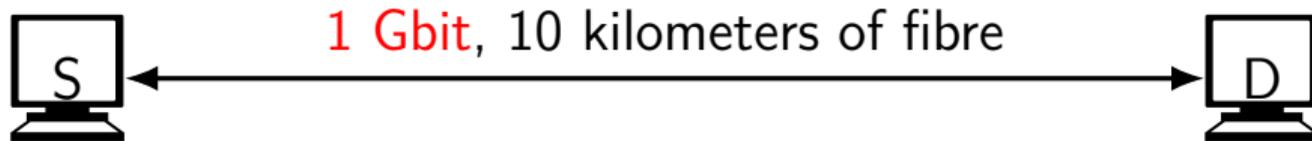
total latency of about $22.2 \mu\text{s}$

latency (1, ex)



exercise: latency for 20000 bit message from S to D
assume speed of signal through fiber of 2.0×10^8 m/s

latency (1, ex)



exercise: latency for 20000 bit message from S to D
assume speed of signal through fiber of 2.0×10^8 m/s

latency (1, ex)



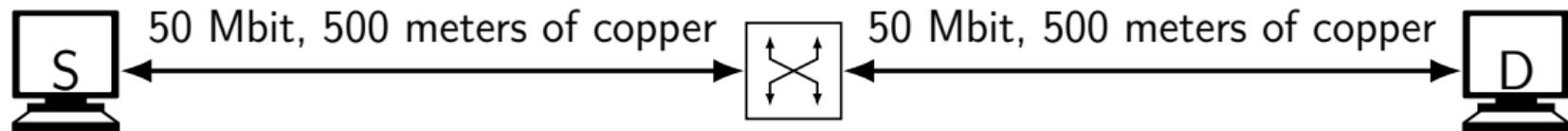
exercise: latency for 20000 bit message from S to D
assume speed of signal through fiber of 2.0×10^8 m/s

latency (2)

example: 1000 bit packet from S to D

assume when message is received:

5 other 1000-bit packets in queue; no extra bits between packets
no other switch processing time



S to switch, switch to D: $22.2 \mu s$ (transmit+propagate delay)

within switch: wait $20 \times 5 = 100 \mu s$ for 5 other packets ($20 \mu s = 1$ packet transmit delay)

“queueing delay”

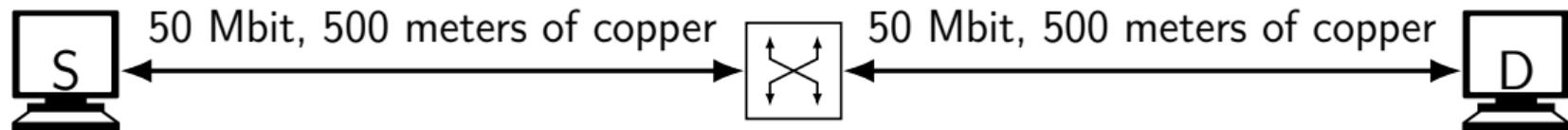
total latency: $22.2 + 100 + 22.2 = 144.4$ microseconds

latency (2)

example: 1000 bit packet from S to D

assume when message is received:

5 other 1000-bit packets in queue; no extra bits between packets
no other switch processing time



S to switch, switch to D: $22.2 \mu s$ (transmit+propagate delay)

within switch: wait $20 \times 5 = 100 \mu s$ for 5 other packets ($20 \mu s = 1$ packet transmit delay)
“queueing delay”

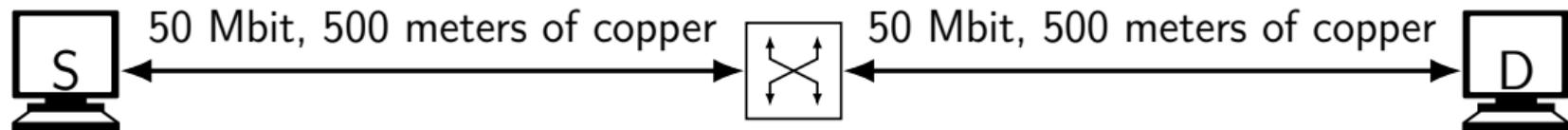
total latency: $22.2 + 100 + 22.2 = 144.4$ microseconds

latency (2)

example: 1000 bit packet from S to D

assume when message is received:

5 other 1000-bit packets in queue; no extra bits between packets
no other switch processing time



S to switch, switch to D: $22.2 \mu s$ (transmit+propagate delay)

within switch: wait $20 \times 5 = 100 \mu s$ for 5 other packets ($20 \mu s = 1$ packet transmit delay)

“queueing delay”

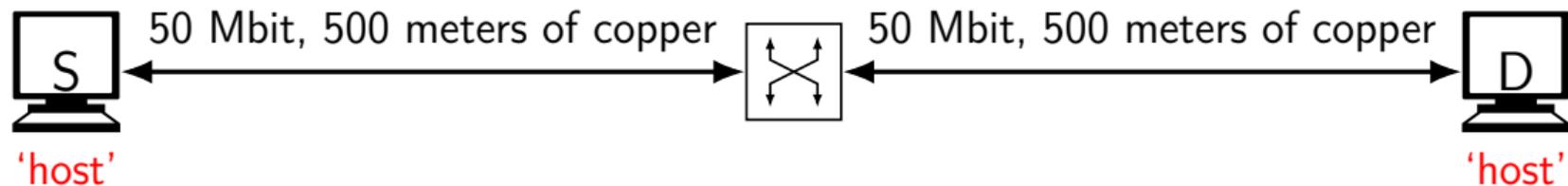
total latency: $22.2 + 100 + 22.2 = 144.4$ microseconds

latency (2)

example: 1000 bit packet from S to D

assume when message is received:

5 other 1000-bit packets in queue; no extra bits between packets
no other switch processing time



S to switch, switch to D: $22.2 \mu s$ (transmit+propogate delay)

within switch: wait $20 \times 5 = 100 \mu s$ for 5 other packets ($20 \mu s = 1$ packet transmit delay)

“queueing delay”

round trip time

round-trip-time (RTT): time for message:
SOURCE → DEST → SOURCE

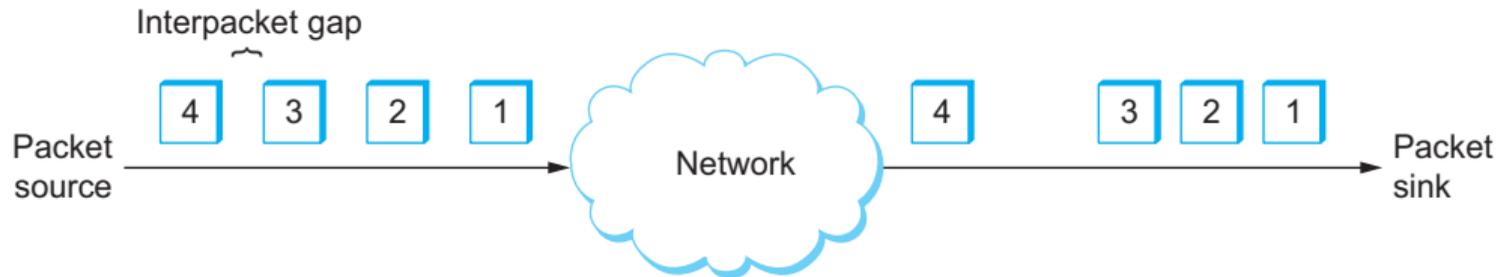
much easier to measure than one-way latency

typically how we'll set latency

jitter

variation in latency

most commonly from changing queuing delays



measuring round-trip time (1a)

```
charles@reisst14$ ping 1.1.1.1
PING 1.1.1.1 (1.1.1.1) 56(84) bytes of data.
64 bytes from 1.1.1.1: icmp_seq=1 ttl=52 time=13.8 ms
64 bytes from 1.1.1.1: icmp_seq=2 ttl=52 time=15.0 ms
64 bytes from 1.1.1.1: icmp_seq=3 ttl=52 time=12.5 ms
64 bytes from 1.1.1.1: icmp_seq=4 ttl=52 time=12.3 ms
64 bytes from 1.1.1.1: icmp_seq=5 ttl=52 time=13.5 ms
64 bytes from 1.1.1.1: icmp_seq=6 ttl=52 time=12.5 ms
64 bytes from 1.1.1.1: icmp_seq=7 ttl=52 time=13.3 ms
64 bytes from 1.1.1.1: icmp_seq=8 ttl=52 time=13.2 ms
64 bytes from 1.1.1.1: icmp_seq=9 ttl=52 time=13.3 ms
64 bytes from 1.1.1.1: icmp_seq=10 ttl=52 time=14.1 ms
^C
--- 1.1.1.1 ping statistics ---
10 packets transmitted, 10 received, 0% packet loss, time 9014ms
rtt min/avg/max/mdev = 12.273/13.343/15.024/0.786 ms
```

measuring round-trip-time (1b)

No.	Time	Source	Destination	Protocol	Length	Info
17	2.766137597	172.25.188.87	1.1.1.1	ICMP	98	Echo (ping) request id=0x0002, seq=1/256, ttl=
18	2.779916793	1.1.1.1	172.25.188.87	ICMP	98	Echo (ping) reply id=0x0002, seq=1/256, ttl=
28	3.768363948	172.25.188.87	1.1.1.1	ICMP	98	Echo (ping) request id=0x0002, seq=2/512, ttl=
29	3.783346606	1.1.1.1	172.25.188.87	ICMP	98	Echo (ping) reply id=0x0002, seq=2/512, ttl=
59	4.769791044	172.25.188.87	1.1.1.1	ICMP	98	Echo (ping) request id=0x0002, seq=3/768, ttl=
60	4.782213735	1.1.1.1	172.25.188.87	ICMP	98	Echo (ping) reply id=0x0002, seq=3/768, ttl=
73	5.771827936	172.25.188.87	1.1.1.1	ICMP	98	Echo (ping) request id=0x0002, seq=4/1024, ttl=
74	5.784055865	1.1.1.1	172.25.188.87	ICMP	98	Echo (ping) reply id=0x0002, seq=4/1024, ttl=
79	6.773358205	172.25.188.87	1.1.1.1	ICMP	98	Echo (ping) request id=0x0002, seq=5/1280, ttl=
80	6.786831460	1.1.1.1	172.25.188.87	ICMP	98	Echo (ping) reply id=0x0002, seq=5/1280, ttl=
81	7.775177274	172.25.188.87	1.1.1.1	ICMP	98	Echo (ping) request id=0x0002, seq=6/1536, ttl=
82	7.787654160	1.1.1.1	172.25.188.87	ICMP	98	Echo (ping) reply id=0x0002, seq=6/1536, ttl=
85	8.776273952	172.25.188.87	1.1.1.1	ICMP	98	Echo (ping) request id=0x0002, seq=7/1792, ttl=
86	8.789562086	1.1.1.1	172.25.188.87	ICMP	98	Echo (ping) reply id=0x0002, seq=7/1792, ttl=
93	9.777262659	172.25.188.87	1.1.1.1	ICMP	98	Echo (ping) request id=0x0002, seq=8/2048, ttl=
94	9.790425188	1.1.1.1	172.25.188.87	ICMP	98	Echo (ping) reply id=0x0002, seq=8/2048, ttl=
110	10.778251280	172.25.188.87	1.1.1.1	ICMP	98	Echo (ping) request id=0x0002, seq=9/2304, ttl=
119	10.791477471	1.1.1.1	172.25.188.87	ICMP	98	Echo (ping) reply id=0x0002, seq=9/2304, ttl=
120	11.779834642	172.25.188.87	1.1.1.1	ICMP	98	Echo (ping) request id=0x0002, seq=10/2560, tt
121	11.793858285	1.1.1.1	172.25.188.87	ICMP	98	Echo (ping) reply id=0x0002, seq=10/2560, tt

measuring round-trip-time (1c)

- ▶ Frame 17: 98 bytes on wire (784 bits), 98 bytes captured (784
- ▶ Ethernet II, Src: f4:6d:3f:d3:64:59 (f4:6d:3f:d3:64:59), Dst:
- ▶ Internet Protocol Version 4, Src: 172.25.188.87, Dst: 1.1.1.1
- ▼ Internet Control Message Protocol
 - Type: 8 (Echo (ping) request)
 - Code: 0
 - Checksum: 0xfa68 [correct]
 - [Checksum Status: Good]
 - Identifier (BE): 2 (0x0002)
 - Identifier (LE): 512 (0x0200)
 - Sequence Number (BE): 1 (0x0001)
 - Sequence Number (LE): 256 (0x0100)
 - [\[Response frame: 18\]](#)
 - Timestamp from icmp data: Sep 2, 2024 12:59:20.000000000 E
 - [Timestamp from icmp data (relative): 0.093073545 seconds]
- ▼ Data (48 bytes)
 - Data: 7f6b01000000000000101112131415161718191a1b1c1d1e1f20
 - [Length: 48]

non-ICMP pings (1)

```
HPING www (enp0s31f6 128.143.67.8): NO FLAGS are set, 40 headers + 0 data
len=46 ip=128.143.67.8 ttl=63 DF id=0 sport=0 flags=RA seq=0 win=0 rtt=3.5
len=46 ip=128.143.67.8 ttl=63 DF id=0 sport=0 flags=RA seq=1 win=0 rtt=3.2
len=46 ip=128.143.67.8 ttl=63 DF id=0 sport=0 flags=RA seq=2 win=0 rtt=7.1
len=46 ip=128.143.67.8 ttl=63 DF id=0 sport=0 flags=RA seq=3 win=0 rtt=6.8
len=46 ip=128.143.67.8 ttl=63 DF id=0 sport=0 flags=RA seq=4 win=0 rtt=6.5
len=46 ip=128.143.67.8 ttl=63 DF id=0 sport=0 flags=RA seq=5 win=0 rtt=6.2
len=46 ip=128.143.67.8 ttl=63 DF id=0 sport=0 flags=RA seq=6 win=0 rtt=5.8
len=46 ip=128.143.67.8 ttl=63 DF id=0 sport=0 flags=RA seq=7 win=0 rtt=5.4
len=46 ip=128.143.67.8 ttl=63 DF id=0 sport=0 flags=RA seq=8 win=0 rtt=5.0
len=46 ip=128.143.67.8 ttl=63 DF id=0 sport=0 flags=RA seq=9 win=0 rtt=4.7
^C
```

```
--- www hping statistic ---
```

```
10 packets transmitted, 10 packets received, 0% packet loss
round-trip min/avg/max = 3.2/5.4/7.1 ms
```

non-ICMP pings (2)

No.	Time	Source	Destination	Protocol	Length	Info
1	0.000000000	128.143.71.27	128.143.67.8	TCP	54	1385 → 0 [<None>] Seq=1 Win=512 Len=0
2	0.000228953	128.143.67.8	128.143.71.27	TCP	60	0 → 1385 [RST, ACK] Seq=1 Ack=1 Win=0 Len=0
3	1.000254996	128.143.71.27	128.143.67.8	TCP	54	1386 → 0 [<None>] Seq=1 Win=512 Len=0
4	1.000555183	128.143.67.8	128.143.71.27	TCP	60	0 → 1386 [RST, ACK] Seq=1 Ack=1 Win=0 Len=0
5	2.000547949	128.143.71.27	128.143.67.8	TCP	54	1387 → 0 [<None>] Seq=1 Win=512 Len=0
6	2.000861315	128.143.67.8	128.143.71.27	TCP	60	0 → 1387 [RST, ACK] Seq=1 Ack=1 Win=0 Len=0
7	3.000765651	128.143.71.27	128.143.67.8	TCP	54	1388 → 0 [<None>] Seq=1 Win=512 Len=0
8	3.000975736	128.143.67.8	128.143.71.27	TCP	60	0 → 1388 [RST, ACK] Seq=1 Ack=1 Win=0 Len=0
9	4.000919160	128.143.71.27	128.143.67.8	TCP	54	1389 → 0 [<None>] Seq=1 Win=512 Len=0
10	4.001230544	128.143.67.8	128.143.71.27	TCP	60	0 → 1389 [RST, ACK] Seq=1 Ack=1 Win=0 Len=0
11	5.001235561	128.143.71.27	128.143.67.8	TCP	54	1390 → 0 [<None>] Seq=1 Win=512 Len=0
12	5.001548682	128.143.67.8	128.143.71.27	TCP	60	0 → 1390 [RST, ACK] Seq=1 Ack=1 Win=0 Len=0
13	6.001564520	128.143.71.27	128.143.67.8	TCP	54	1391 → 0 [<None>] Seq=1 Win=512 Len=0
14	6.001886342	128.143.67.8	128.143.71.27	TCP	60	0 → 1391 [RST, ACK] Seq=1 Ack=1 Win=0 Len=0
15	7.001812043	128.143.71.27	128.143.67.8	TCP	54	1392 → 0 [<None>] Seq=1 Win=512 Len=0
16	7.002151624	128.143.67.8	128.143.71.27	TCP	60	0 → 1392 [RST, ACK] Seq=1 Ack=1 Win=0 Len=0
17	8.002159545	128.143.71.27	128.143.67.8	TCP	54	1393 → 0 [<None>] Seq=1 Win=512 Len=0
18	8.002492960	128.143.67.8	128.143.71.27	TCP	60	0 → 1393 [RST, ACK] Seq=1 Ack=1 Win=0 Len=0
19	9.002512076	128.143.71.27	128.143.67.8	TCP	54	1394 → 0 [<None>] Seq=1 Win=512 Len=0
20	9.002826074	128.143.67.8	128.143.71.27	TCP	60	0 → 1394 [RST, ACK] Seq=1 Ack=1 Win=0 Len=0
21	10.002865037	128.143.71.27	128.143.67.8	TCP	54	1395 → 0 [<None>] Seq=1 Win=512 Len=0
22	10.003183918	128.143.67.8	128.143.71.27	TCP	60	0 → 1395 [RST, ACK] Seq=1 Ack=1 Win=0 Len=0

measuring throughput?

```
$ scp test.dat portal.cs.virginia.edu:test.dat
test.dat          100%   32MB  23.0MB/s   00:01
$ scp portal.cs.virginia.edu:test.dat .
test.dat          100%   32MB  28.2MB/s   00:01
```

(but might be measuring disk speed instead)

also more specialized tools like `iperf`
require program to run on both ends

measuring throughput

```
$ iperf -s
```

```
-----  
Server listening on TCP port 5001  
TCP window size: 128 KByte (default)  
-----
```

```
[ 1] local 128.143.71.87 port 5001 connected with 128.143.71.27 port 54760  
[ ID] Interval          Transfer      Bandwidth  
[ 1] 0.0000-10.0147 sec  1.09 GBytes   934 Mbits/sec
```

```
—  
$ iperf -c kytos02 | tee iperf.out
```

```
-----  
Client connecting to kytos02, TCP port 5001  
TCP window size: 85.0 KByte (default)  
-----
```

```
[ 1] local 128.143.71.27 port 54760 connected with 128.143.71.87 port 5001  
[ ID] Interval          Transfer      Bandwidth  
[ 1] 0.0000-10.0256 sec  1.09 GBytes   933 Mbits/sec
```

measuring transmission delay?

```
PING www.cs.virginia.edu (128.143.67.8) 1400(1428) bytes of data.  
--- www.cs.virginia.edu ping statistics ---  
1000 packets transmitted, 1000 received, 0% packet loss, time 50638ms  
rtt min/avg/max/mdev = 0.319/0.461/1.222/0.039 ms  
$ ping -s 16 www -i 0.05 -c 1000 -q  
PING www.cs.virginia.edu (128.143.67.8) 16(44) bytes of data.  
--- www.cs.virginia.edu ping statistics ---  
1000 packets transmitted, 1000 received, 0% packet loss, time 50995ms  
rtt min/avg/max/mdev = 0.156/0.345/1.539/0.068 ms
```

approx. $0.461 - 0.345 = 0.116$ ms delay for 1400 – 16 extra bytes

with two links in each direction = approx $\frac{0.116}{4} = 0.029$ ms/link

$\frac{1400 - 16\text{byte}}{0.029\text{ms}} \approx 50$ Mbit/sec (does not match Gigabit ethernet)

probably other processing time besides sending on links, though

stop-and-wait performance

stop-and-wait protocol

assuming no packets lost/corrupted

about **one packet per round-trip time**

example: local ethernet

my home wired network: 0.6 ms round trip time

typical packet has about 1400 bytes = 11200 bits of data

throughput with stop-and-wait:

$$11200\text{b}/0.6\text{ms} \approx 19000\text{b}/\text{ms} = 19\,000\,000\text{b}/\text{s} = 19\text{Mbit}/\text{s}$$

available bandwidth is about 1 Gbit/s

example: local ethernet

my home wired network: 0.6 ms round trip time

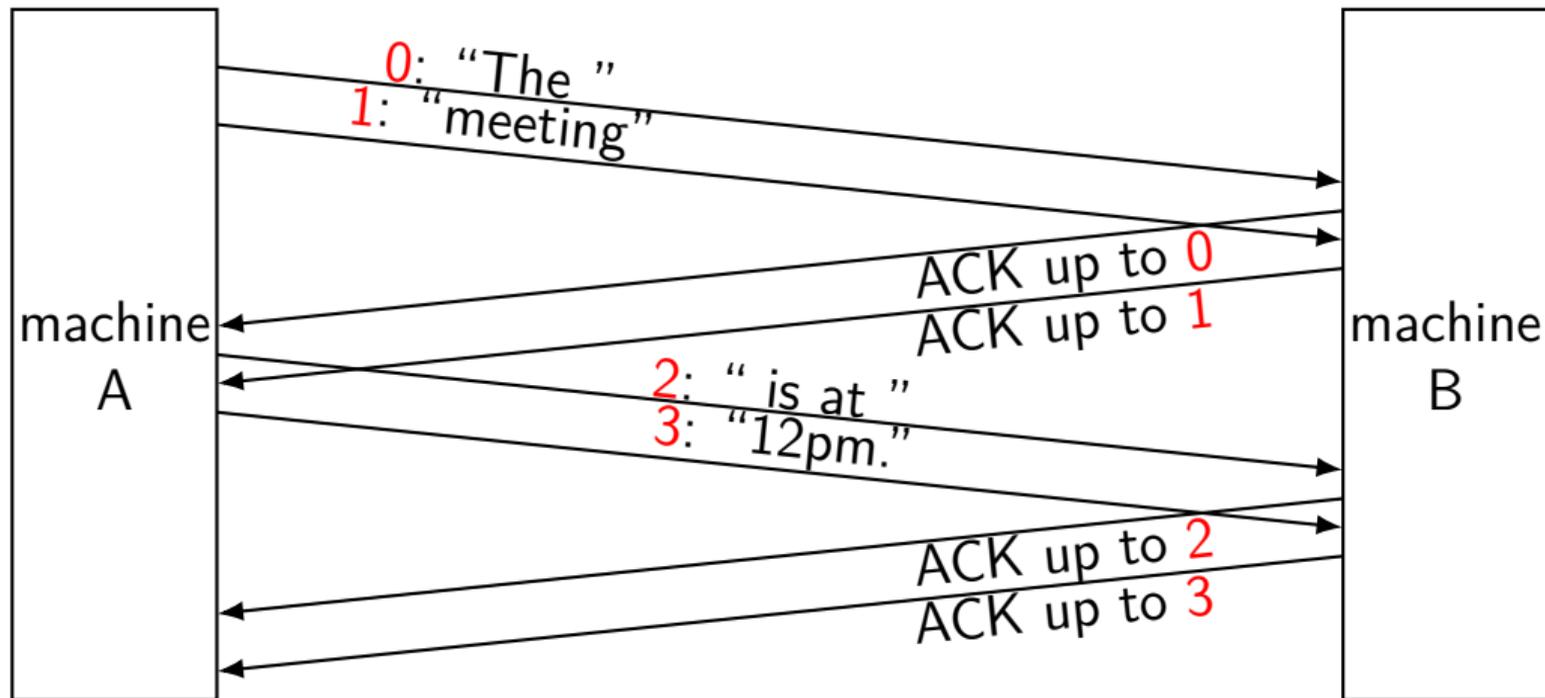
typical packet has about 1400 bytes = 11200 bits of data

throughput with stop-and-wait:

$$11200\text{b}/0.6\text{ms} \approx 19000\text{b}/\text{ms} = 19\,000\,000\text{b}/\text{s} = 19\text{Mbit}/\text{s}$$

available bandwidth is about 1 Gbit/s

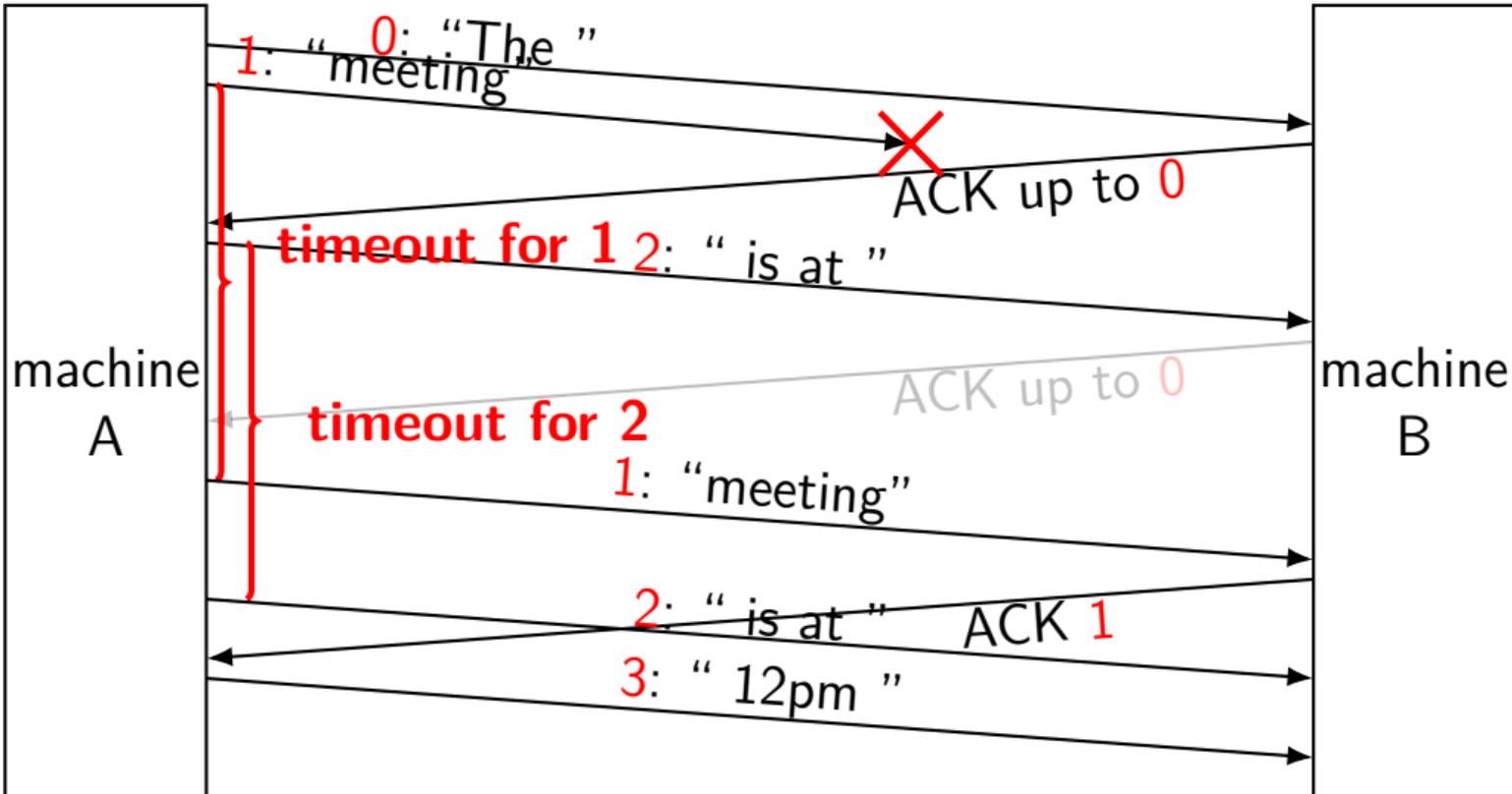
sending two at a time



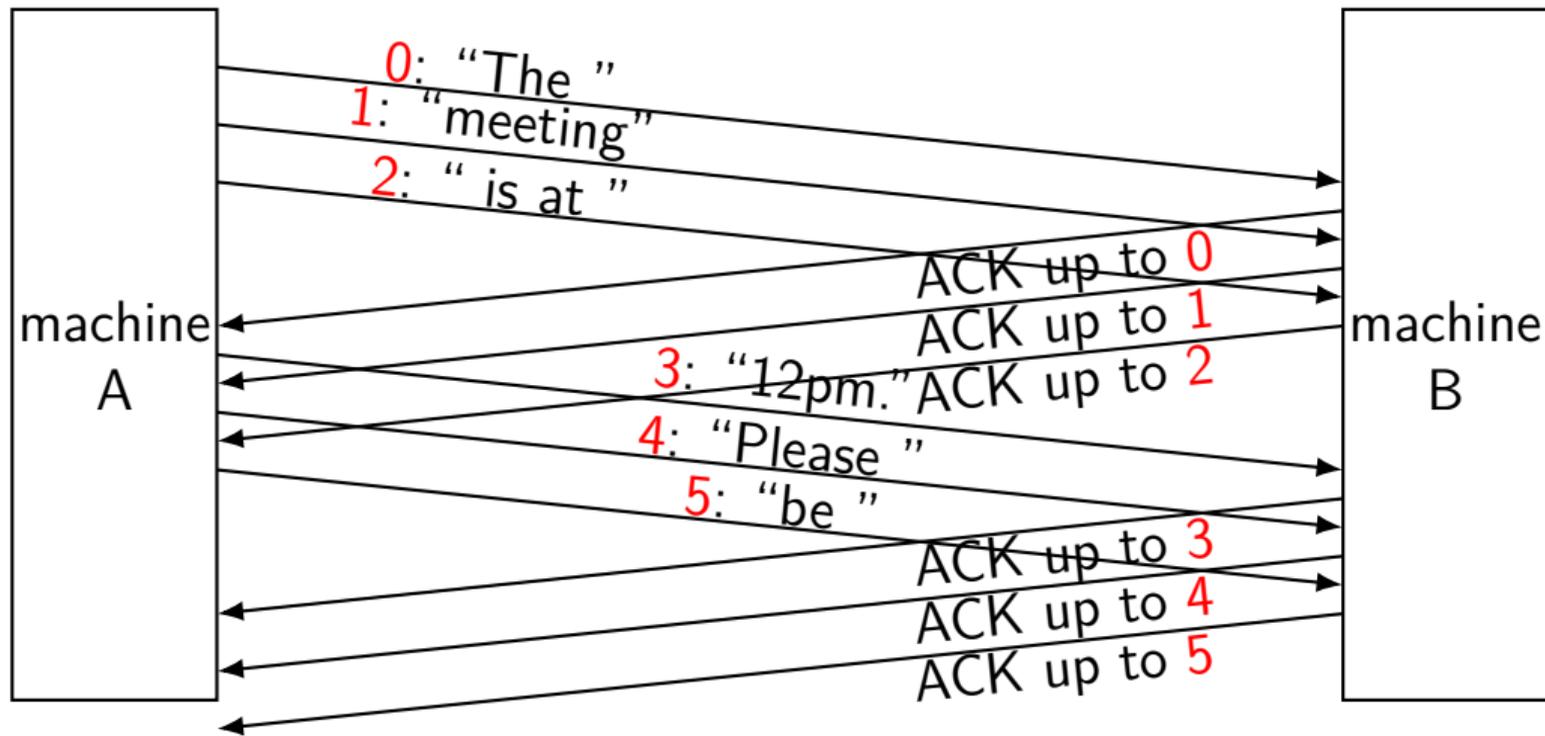
(ACK up to X = ACK X and everything before it)

key idea: always have two in flight

timeouts per message



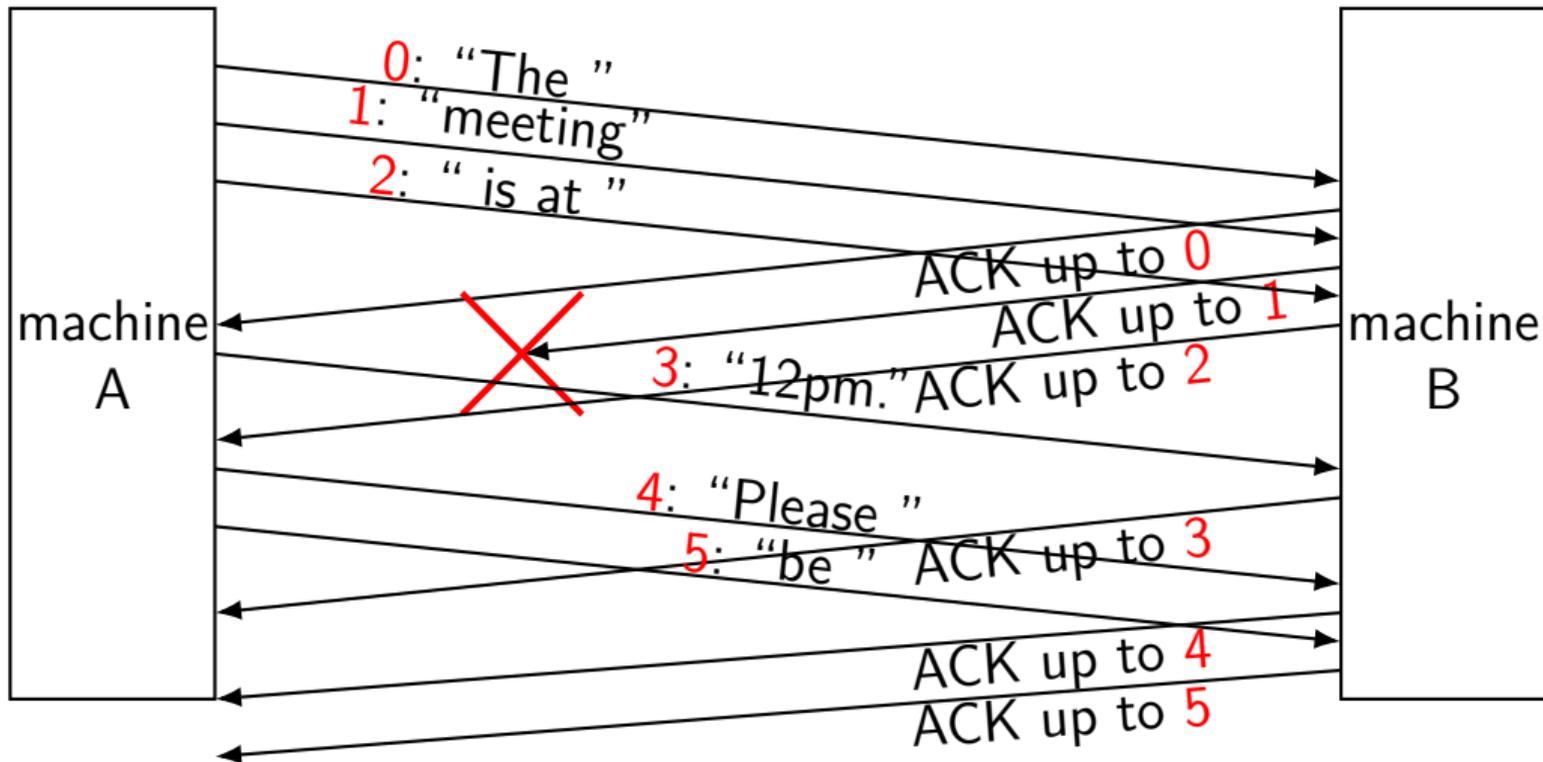
sending three at a time



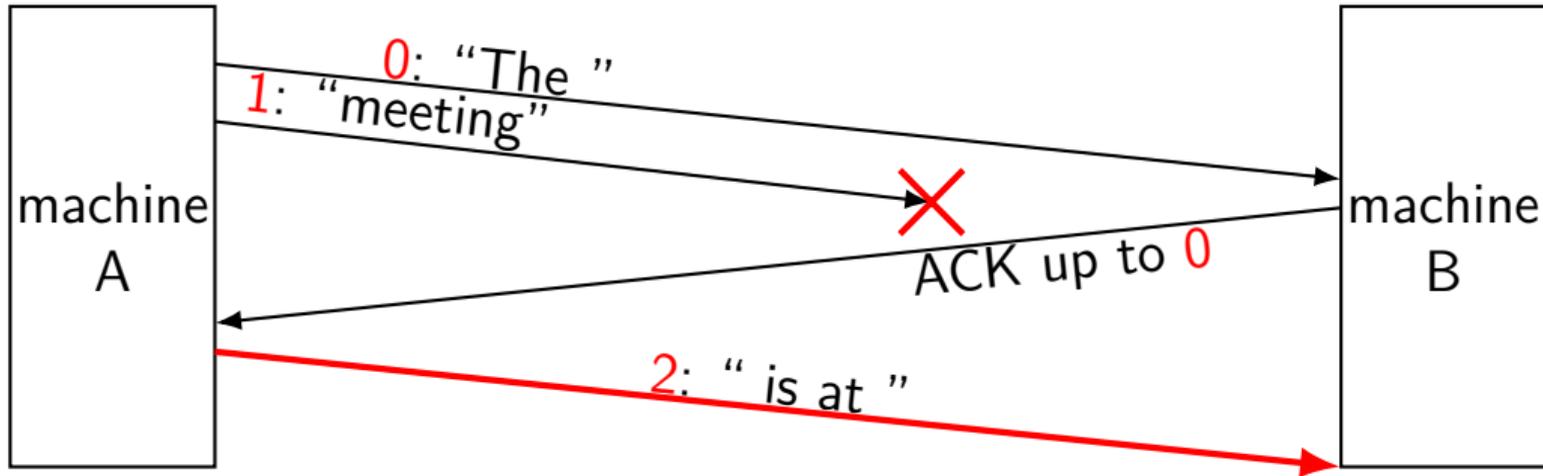
choose "window size" to have in flight

send when previous acknowledged

lost ACKs?

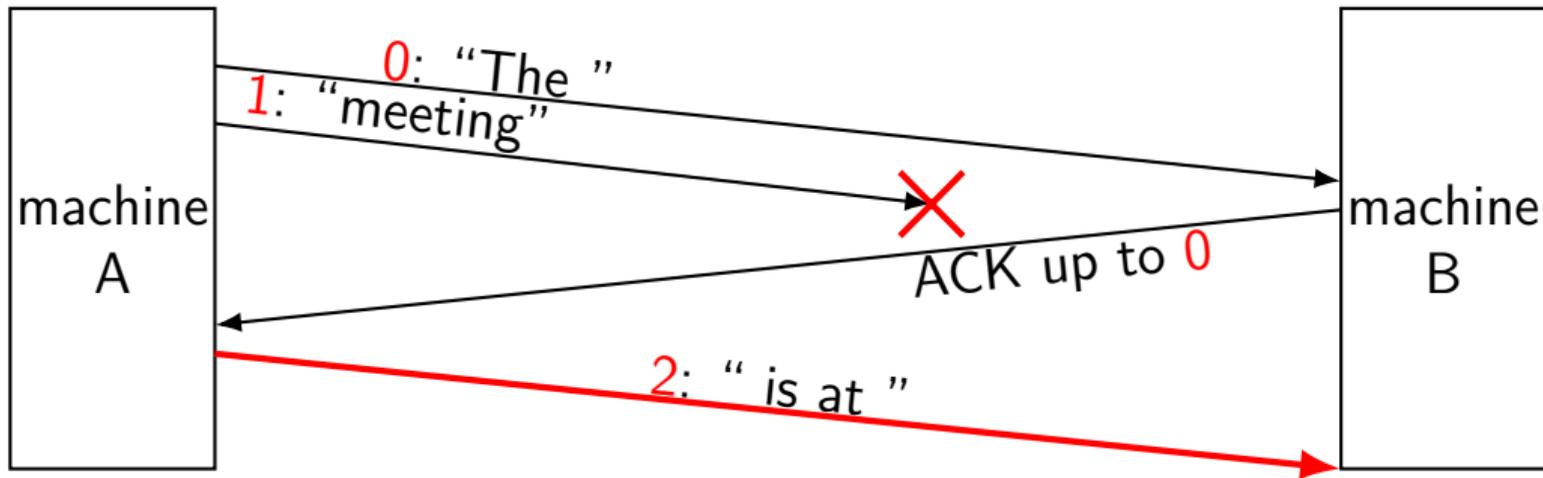


missing messages?



question: what should receiver do with sequence number 2?

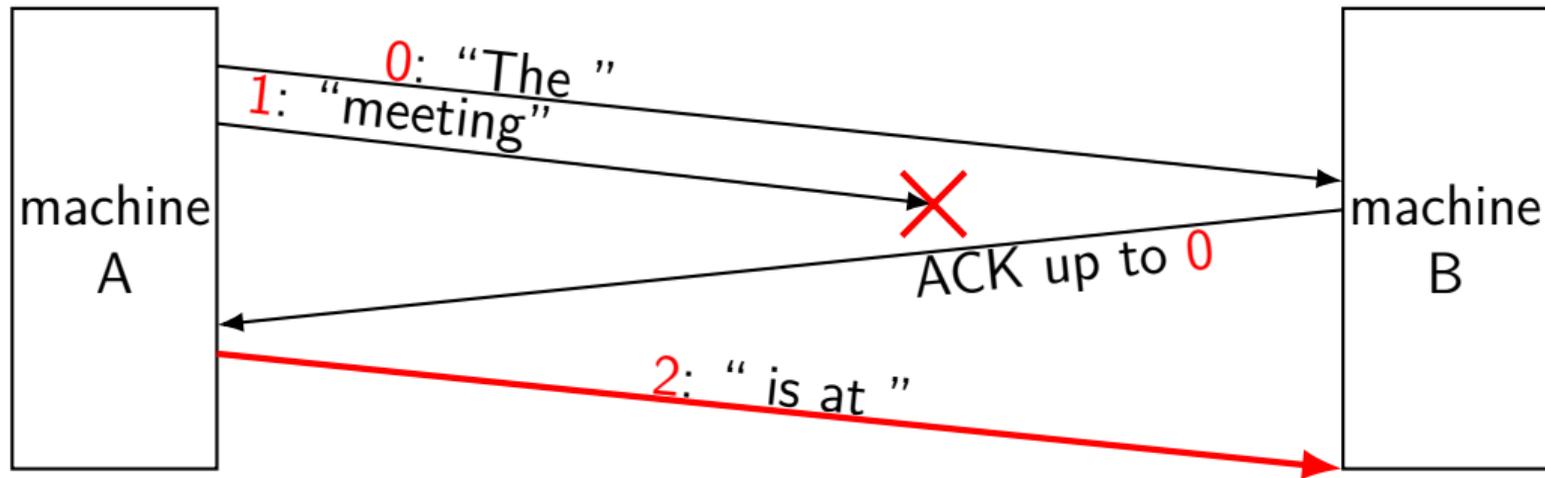
missing messages?



question: what should receiver do with sequence number 2?

one idea: ignore it?

missing messages?

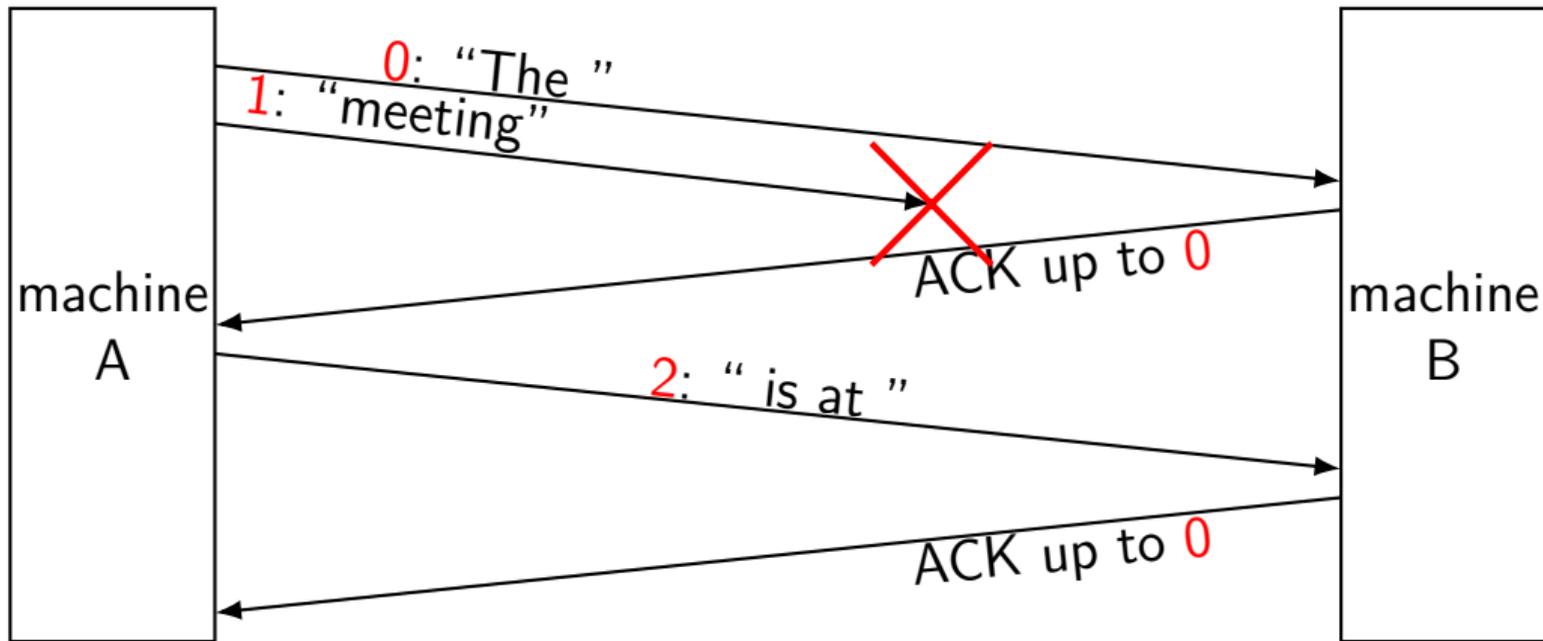


question: what should receiver do with sequence number 2?

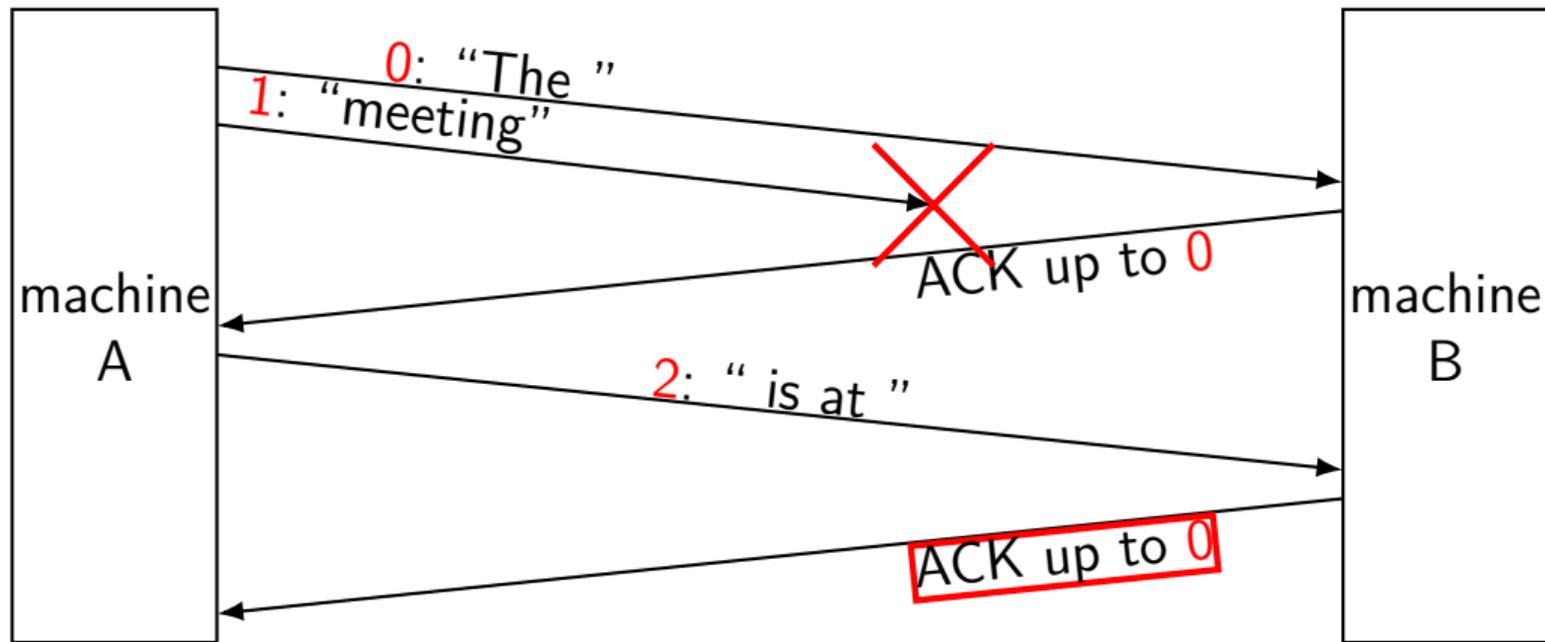
one idea: ignore it?

better idea: send something back to sender

better idea: always ACK



better idea: always ACK



only ACK x if everything up to and including x received

intuition: ACK tells sender where to start sending more

fast retransmit

if large window + data packet 2 is lost, then sender will see

ACK 0, ACK 1, ACK 1, ACK 1, ACK 1, ACK 1

duplicate ACKs indicate missing packet 2

shouldn't wait for timeout

fast retransmit

if large window + data packet 2 is lost, then sender will see

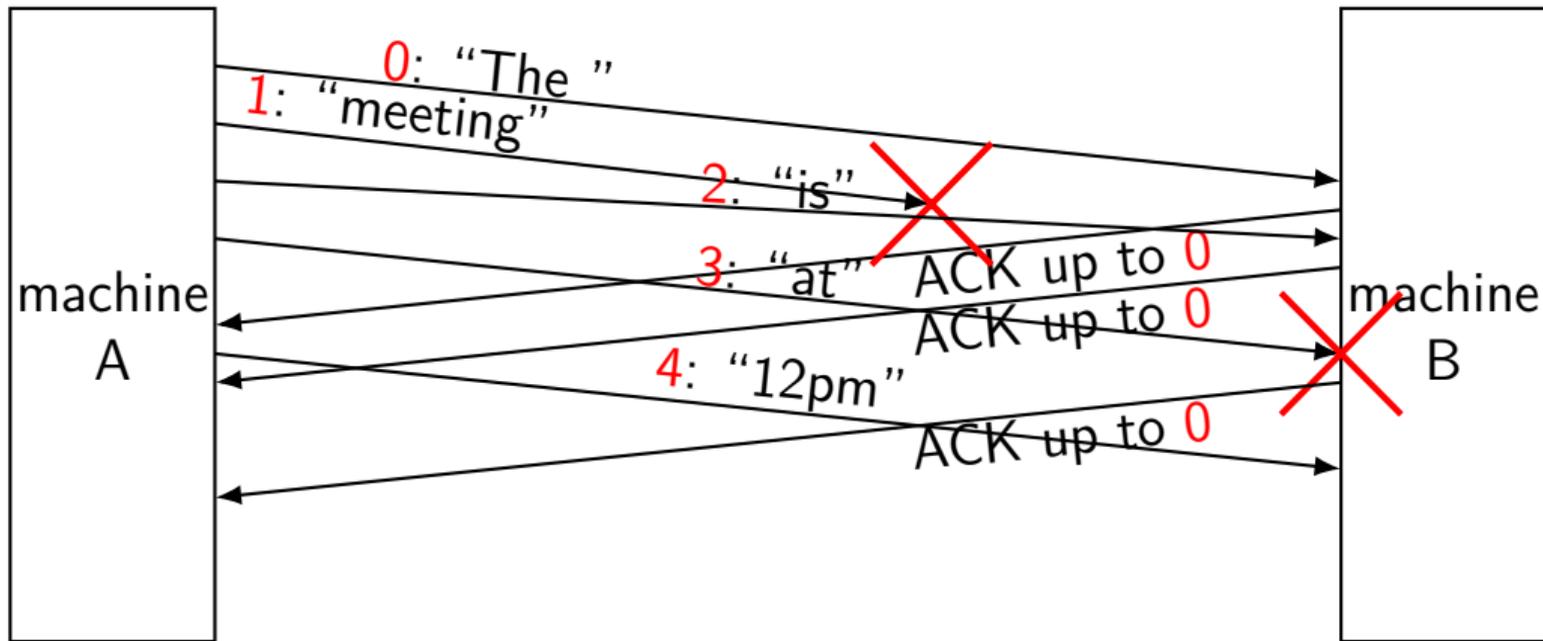
ACK 0, ACK 1, ACK 1, ACK 1, ACK 1, ACK 1

duplicate ACKs indicate missing packet 2

shouldn't wait for timeout

→ TCP heuristic: retransmit immediately after ~ 3 duplicate ACKs
not 1 duplicate ACK to tolerate some reordering
also some other details (we'll talk later)

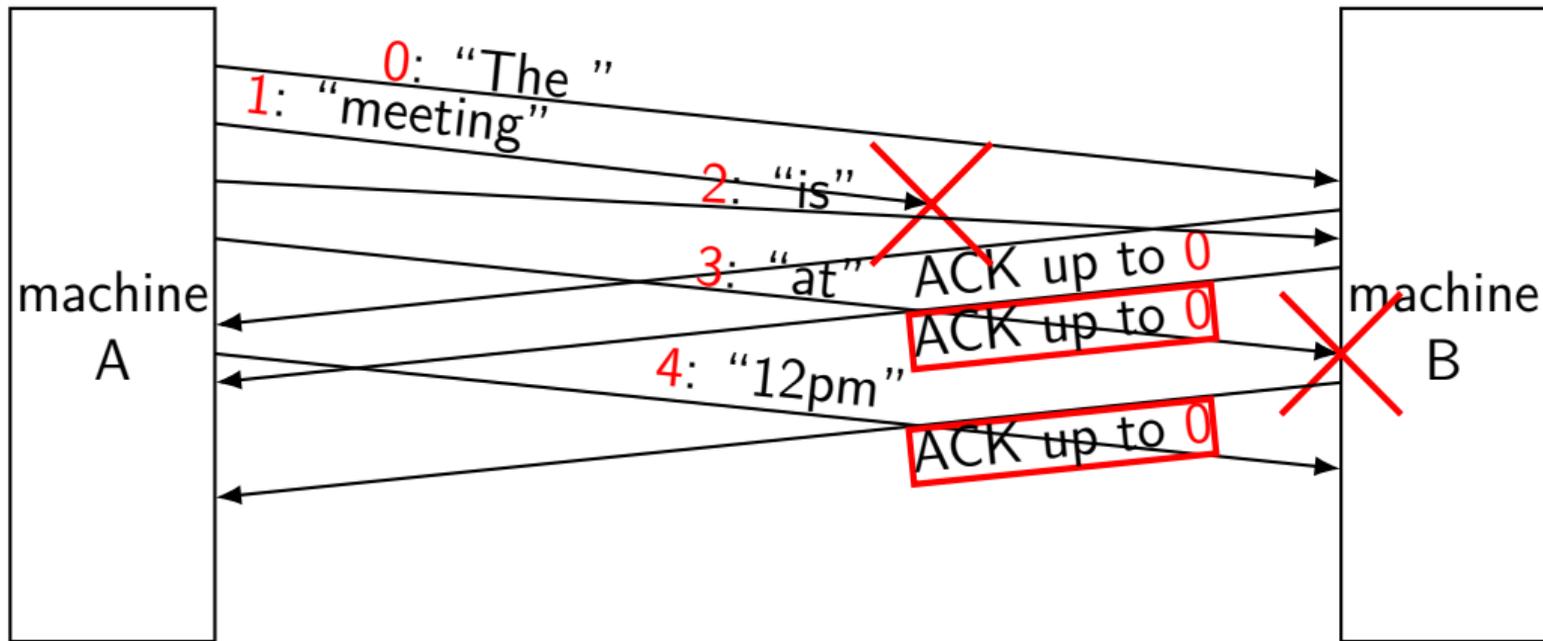
multiple missing



duplicate ACK heuristic will quickly resend 1, but not 3

would like to supply better information

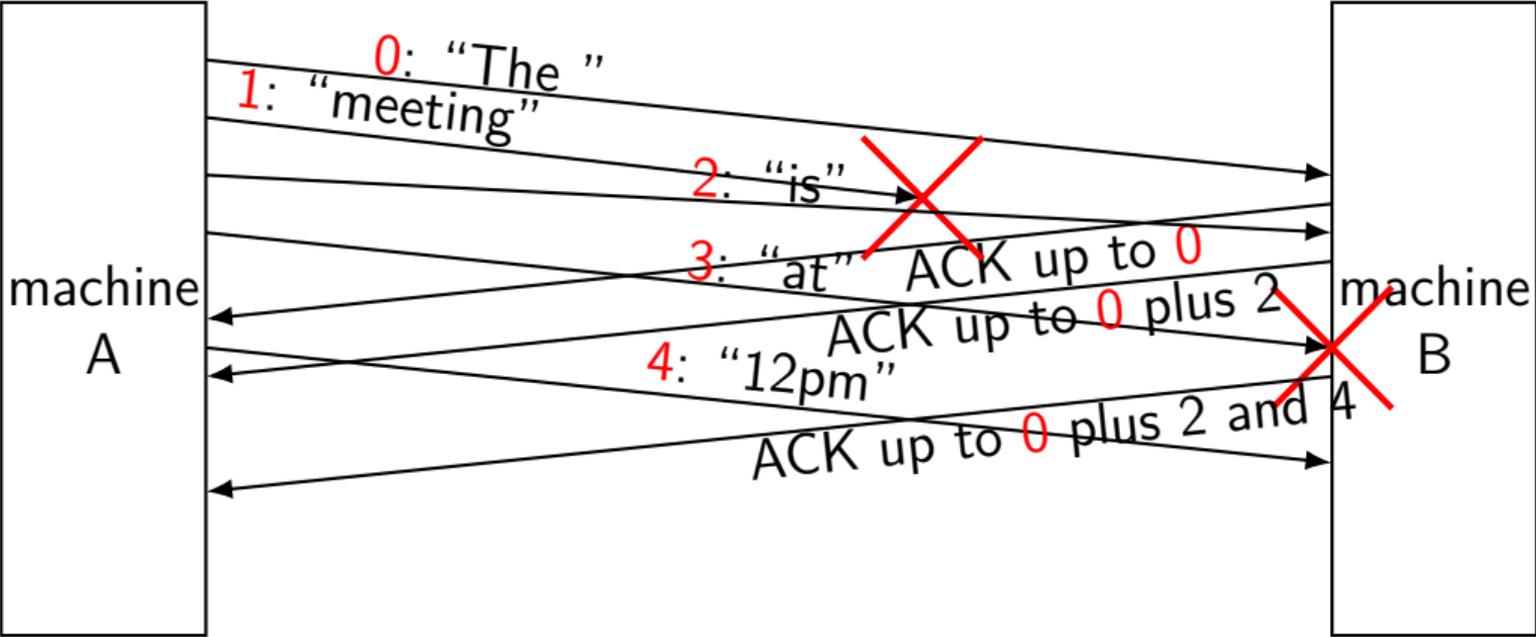
multiple missing



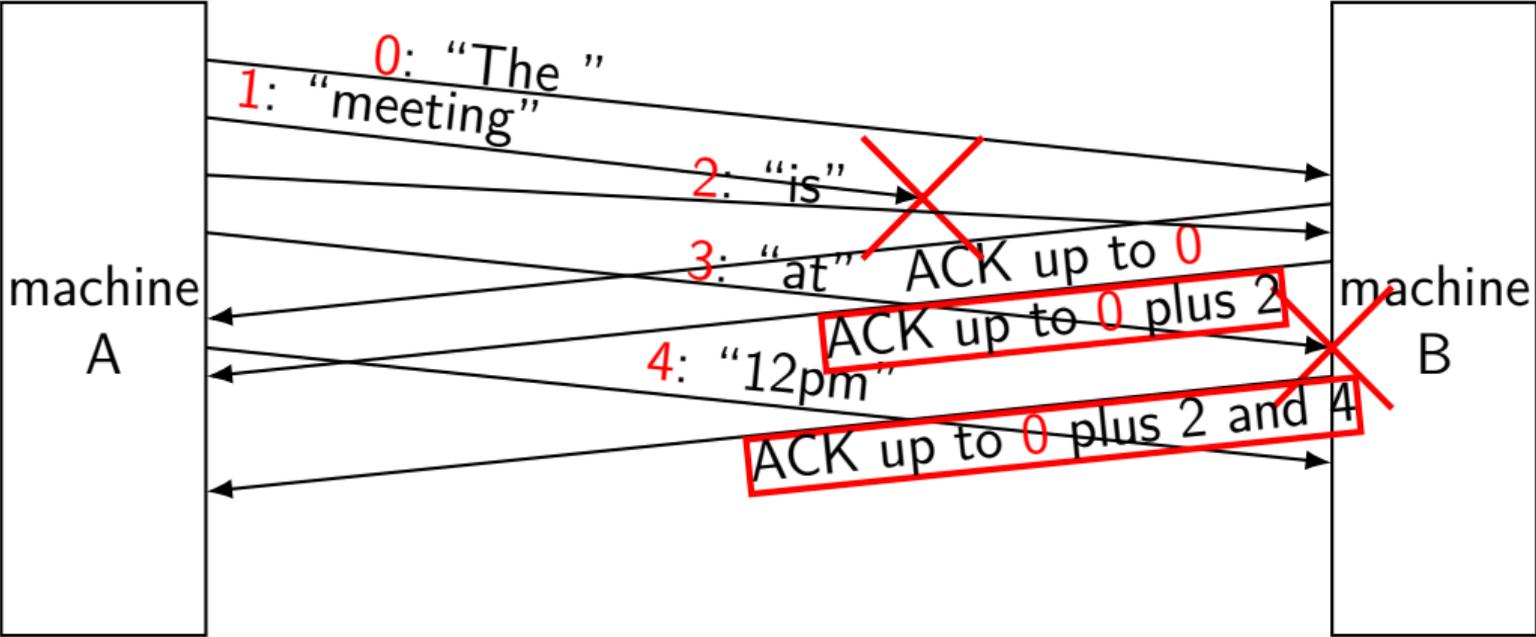
duplicate ACK heuristic will quickly resend 1, but not 3

would like to supply better information

selective acknowledgments



selective acknowledgments



selective acknowledgments in TCP

optional feature (“extension”) described in RFC 2018

send list of ranges received

typically room for 3 ranges

if more than 3 ranges to report, then:

- include range with most recently received frame

- include other ranges until sent three times

sender window tracking

X
□

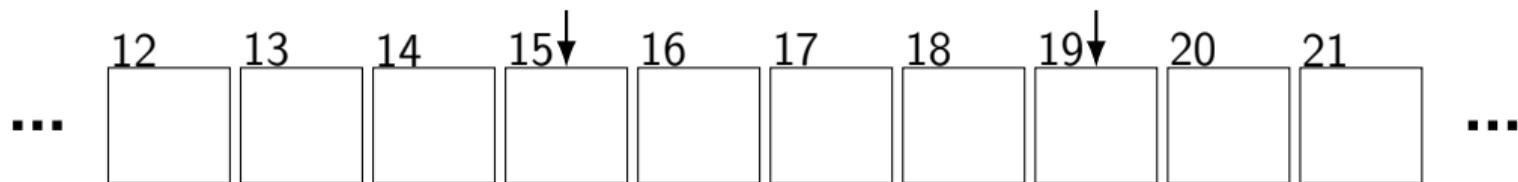
= frame of data with sequence number X

(LAR)

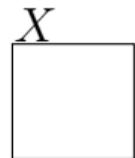
(LFS)

last ACK recv'd

last frame sent



sender window tracking



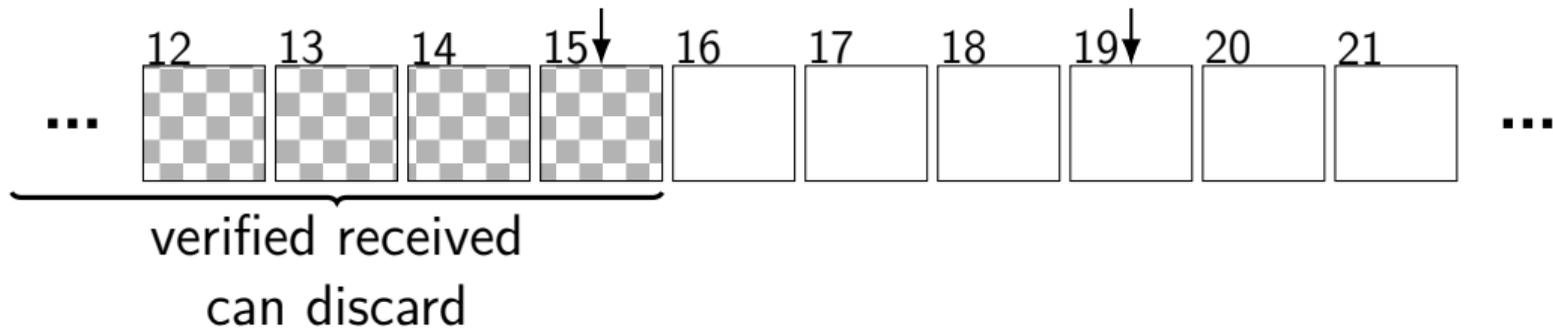
= frame of data with sequence number X

(LAR)

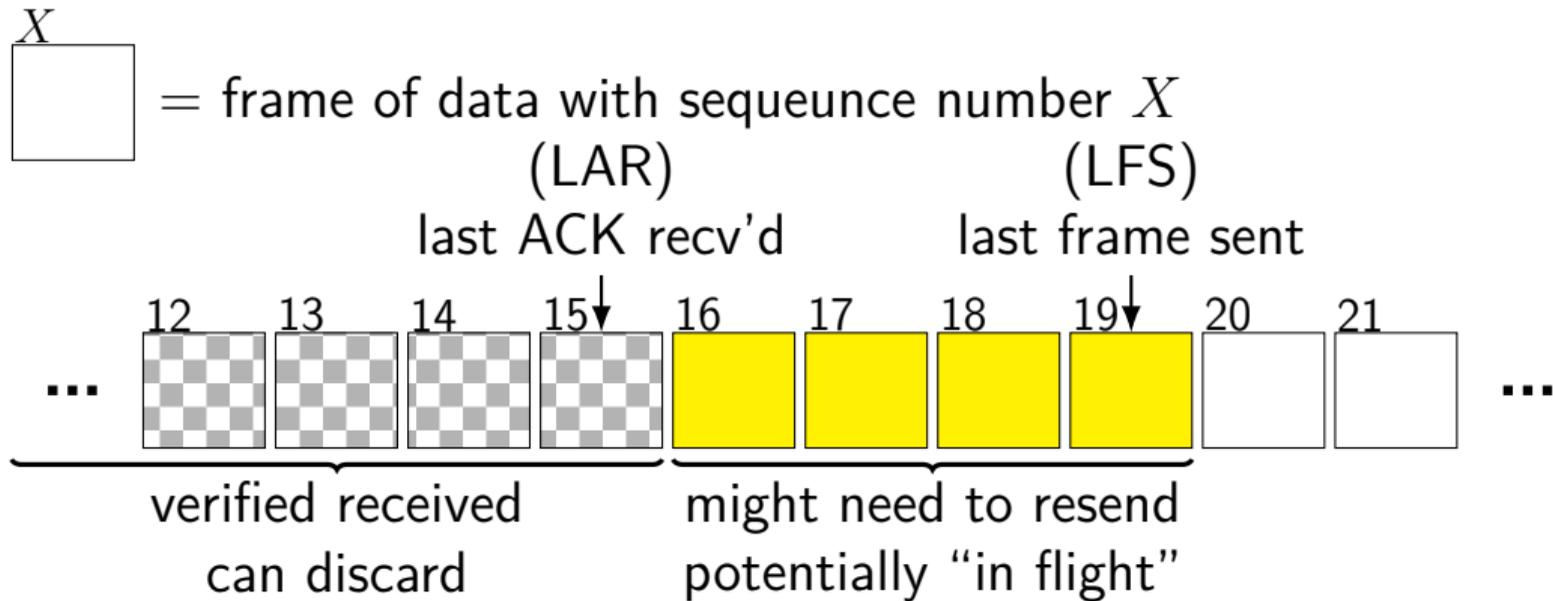
(LFS)

last ACK recv'd

last frame sent



sender window tracking



sender window tracking

X
□

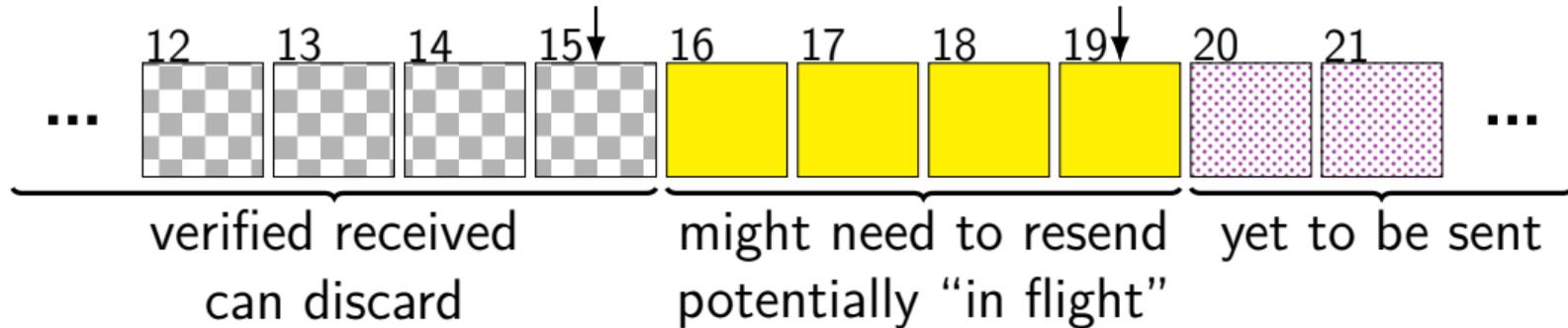
= frame of data with sequence number X

(LAR)

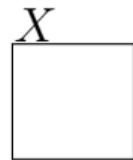
(LFS)

last ACK recv'd

last frame sent



sender window tracking



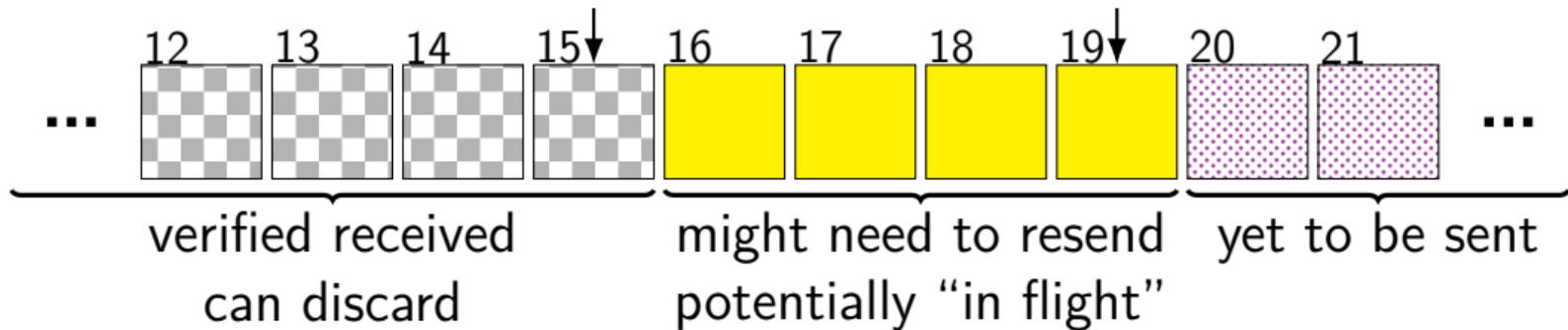
= frame of data with sequence number X

(LAR)

(LFS)

last ACK recv'd

last frame sent



↑
at most the
Send Window Size
(SWS)

exercise 1: out-of-bounds ACK

last ACK recv'd (LAR)		10
last frame sent (LFS)		15
send window size (SWS)		5

what probably happened if we receive an ACK for...

9? 10? 13? 16?

- A. only possible if network reorders frames
- B. only possible from undetected frame corruption
- C. lost ACK for frame ≤ 10
- D. lost ACK for frame > 10
- E. lost frame 11
- F. resent frame from timeout

exercise 2: sender logic

last ACK recv'd (LAR)		10
last frame sent (LFS)		15
send window size (SWS)		5

In this case, there's a timeout that will trigger frame 13 to be resent. If still active, this timeout should be cancelled upon ...

- A. receiving ACK 12
- B. receiving ACK 13
- C. receiving ACK 14
- D. sending frame 16

exercise 3a: new data

last ACK recv'd (LAR)		4
last frame sent (LFS)		6
send window size (SWS)		5

if we compute a new frame of data with sequence number 7 to eventually send, we should

- A. send it now, advancing LFS
- B. wait until we get an ACK for 5 or 6 to send it
- C. wait until we get an ACK for 6 to send it
- D. wait until the frame with sequence number 6 is resent to send it D. som

exercise 3b: new data

last ACK recv'd (LAR)		4
last frame sent (LFS)		8
send window size (SWS)		4

if we compute a new frame of data with sequence number 9 to eventually send, we should

- A. send it now, advancing LFS
- B. wait until we get an ACK for 5 or 6 to send it
- C. wait until we get an ACK for 6 to send it
- D. decline to accept the data because we will never be able to send it
- E. something else

sender logic summarized

track variables:

LFS (last frame sent)

LAR (last ACK recv'd)

SWS (send window size)

when receiving ACK $LAR < X \leq LFS$:

$LAR \leftarrow X$

clear any timers to resend frames $\leq X$

whenever SWS [send window size] $> LFS - LAR$ and data for frame $LFS + 1$ is available:

send frame $LFS + 1$

set timer to resend frame $LFS + 1$

$LFS \leftarrow LFS + 1$

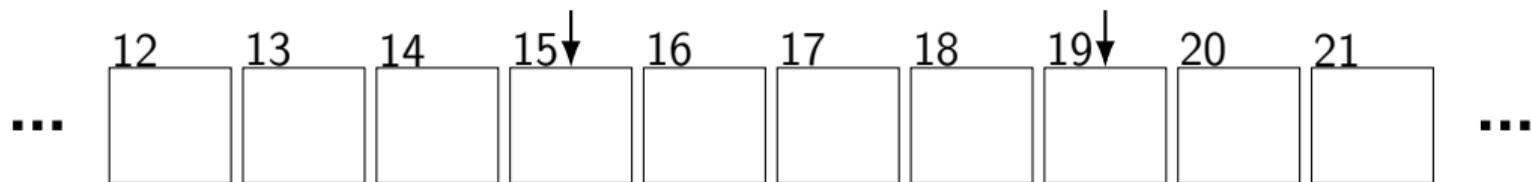
receiver window tracking

X
□

= frame of data with sequence number X
(LFR) (LAF)

last frame recv'd*

last accepted frame



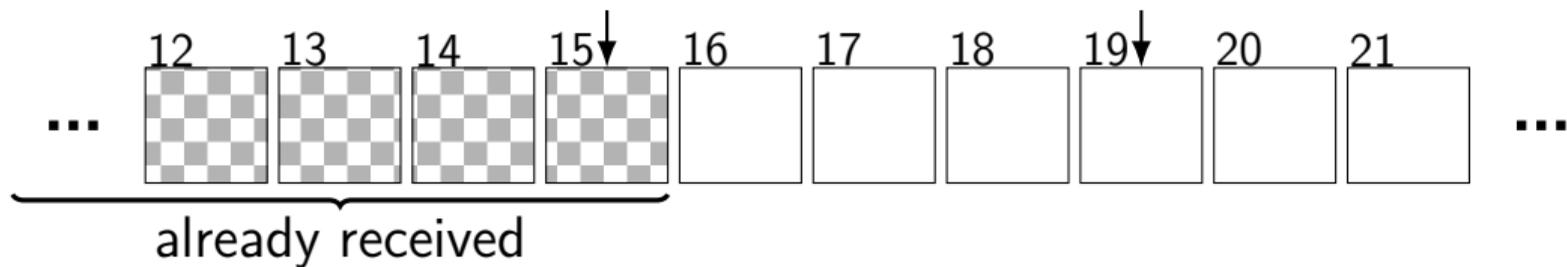
receiver window tracking

X
□

= frame of data with sequence number X
(LFR) (LAF)

last frame recv'd*

last accepted frame



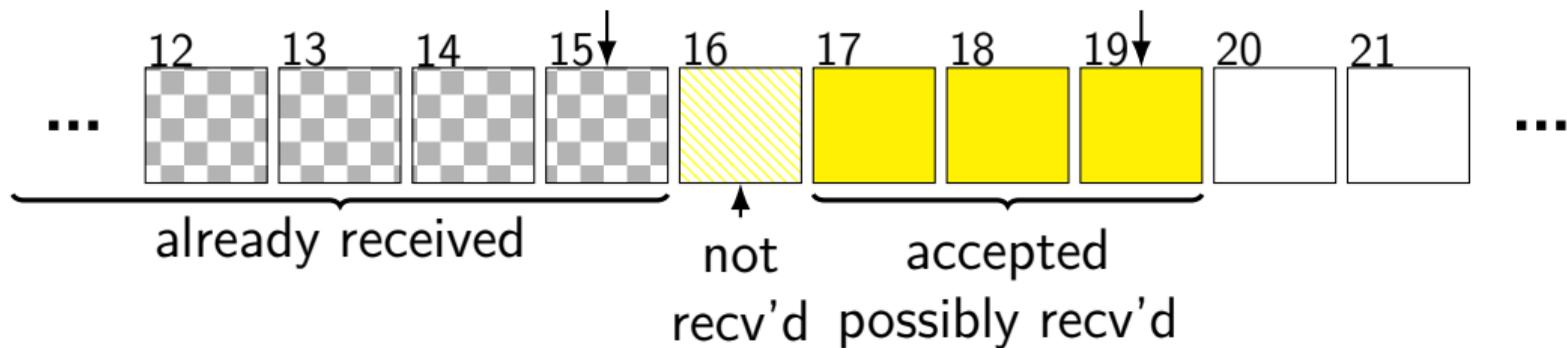
receiver window tracking

X
□

= frame of data with sequence number X
(LFR) (LAF)

last frame recv'd*

last accepted frame



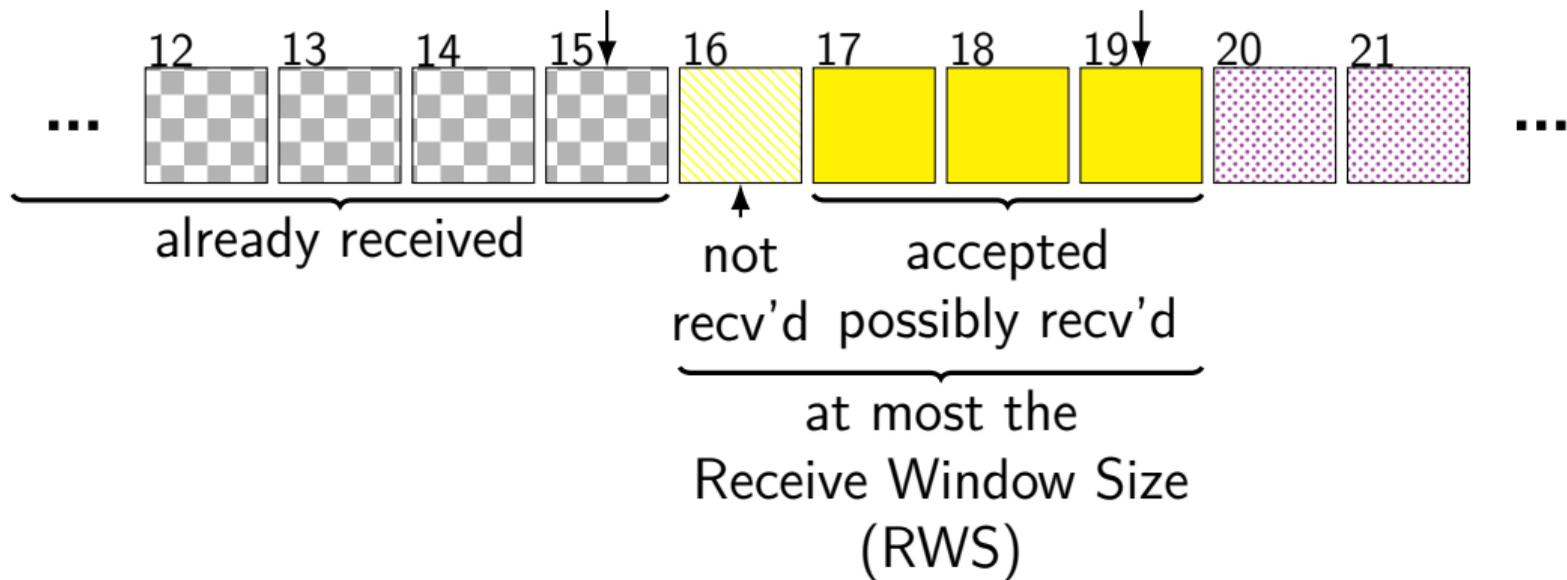
receiver window tracking

X
□

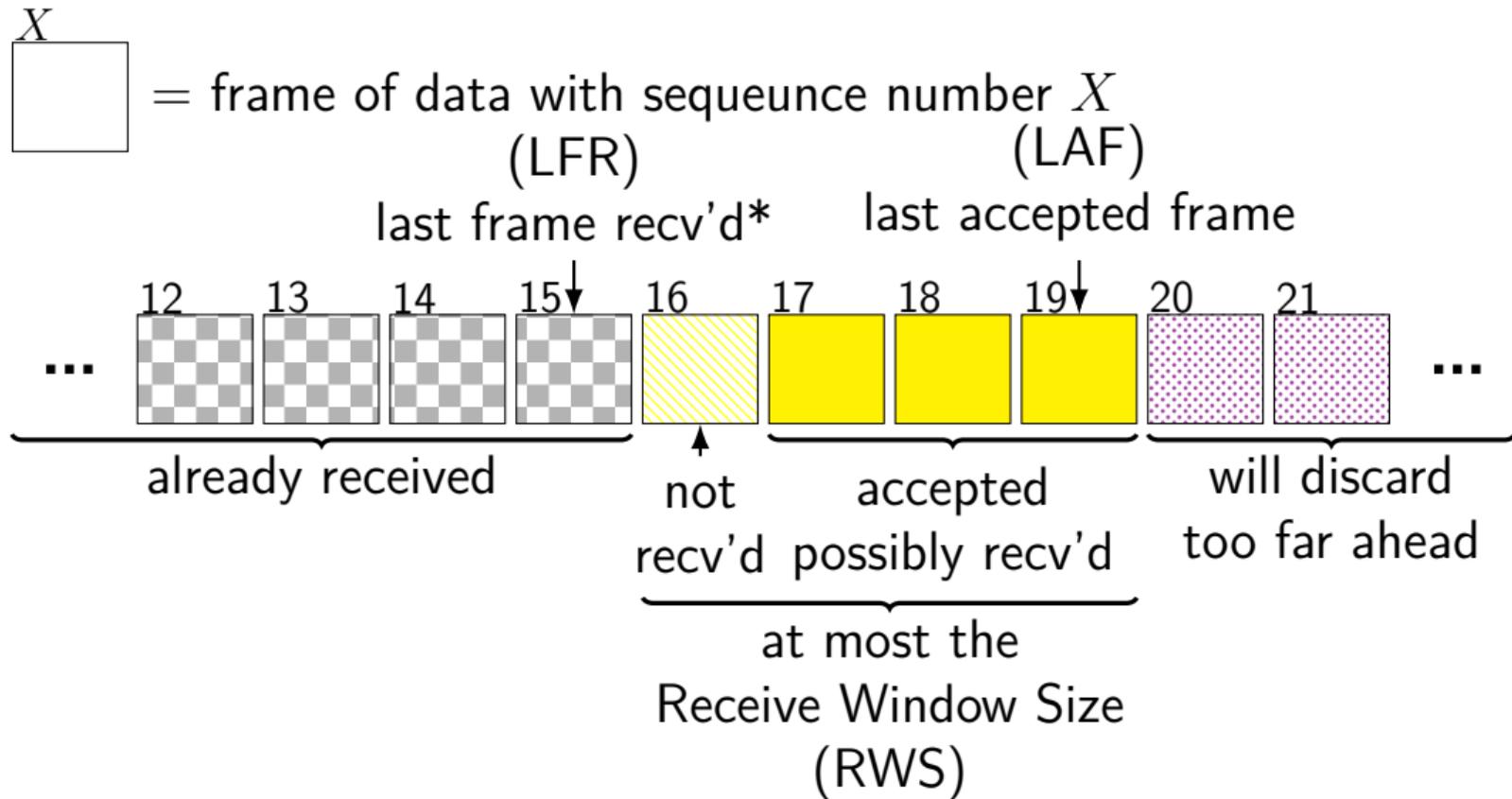
= frame of data with sequence number X
(LFR) (LAF)

last frame recv'd*

last accepted frame



receiver window tracking



receiver logic summarized

track variables:

LFR (last frame recv'd) — excludes frames after a missing frame

LAF (last accepted frame)

RWS (receive window size)

when receiving frame $LFR < X \leq LAF$:

$LFR \leftarrow (\text{first missing frame after LFR}) - 1$

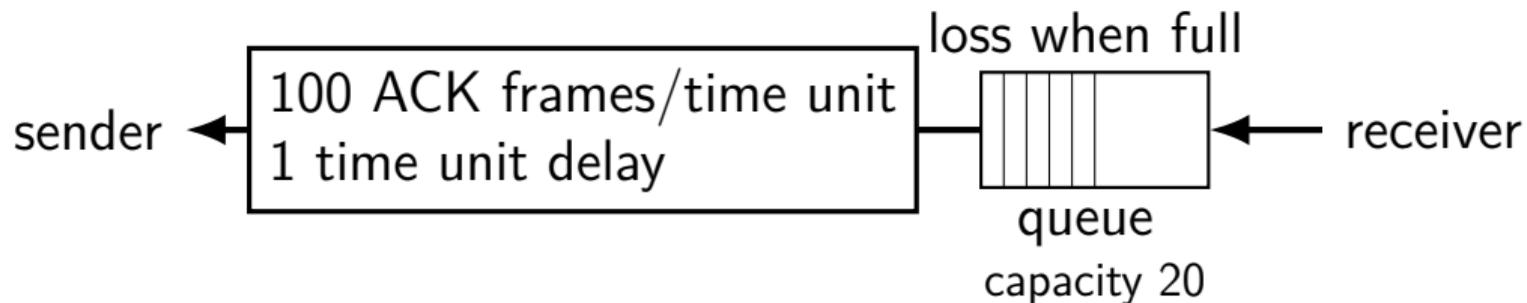
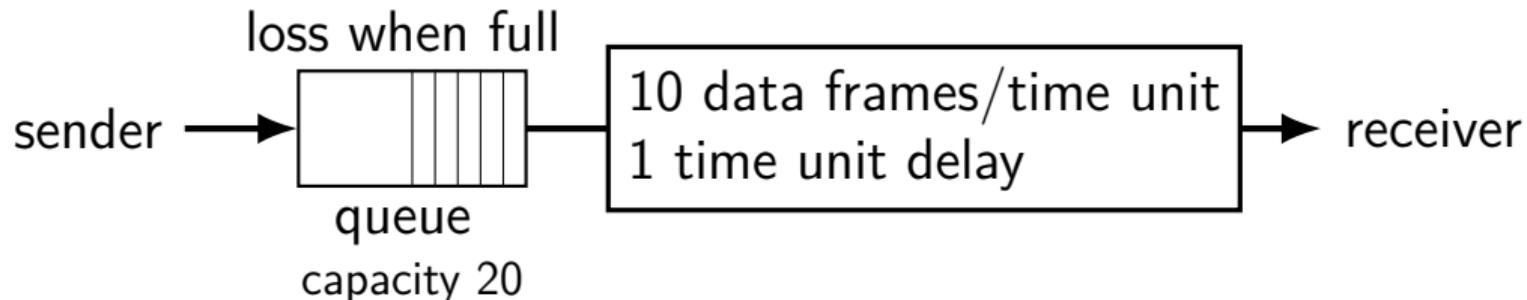
only advances if $X = LFR$

could advance by more than one if frames previously out of order

$LAF \leftarrow LFR + RWS$

only advances if $X = LAF$

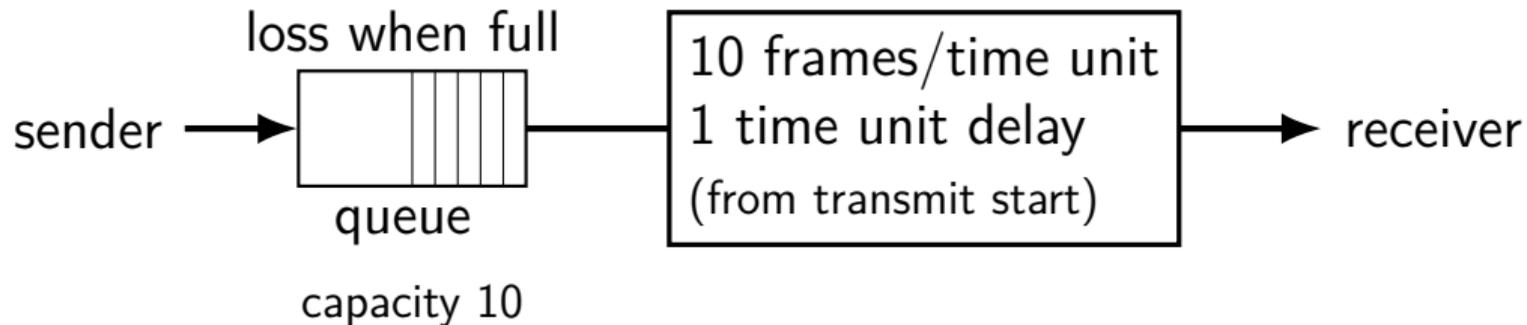
simple network model



simulator from upcoming assignment

```
command line --delay 1 --bandwidth-forward 10  
--bandwidth-backward 100 --buffer 30
```

exercise: forward latency

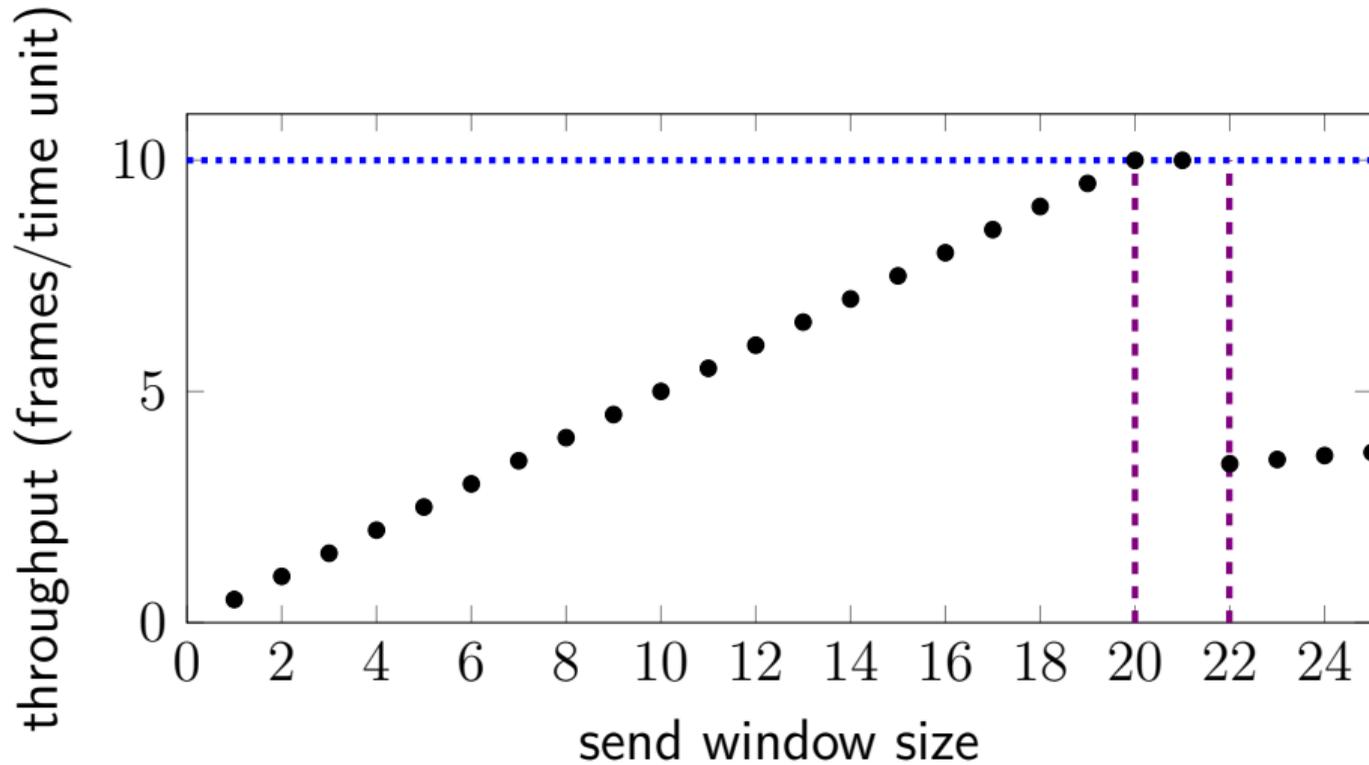


minimum latency = 1 time unit

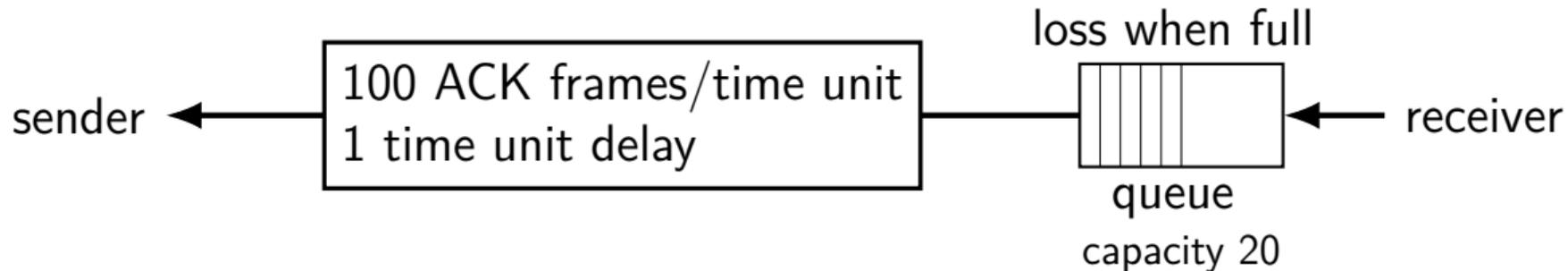
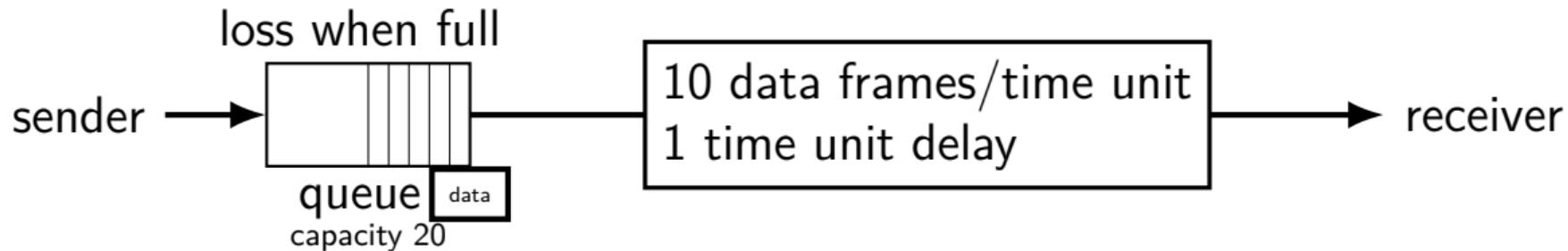
exercise: maximum latency?

- A. 1 time unit
- B. 1.1 time unit
- C. 1.2 time unit
- C. 1.4 time unit
- D. 1.9 time unit
- E. 2.0 time unit
- F. 2.1 time unit
- G. something else

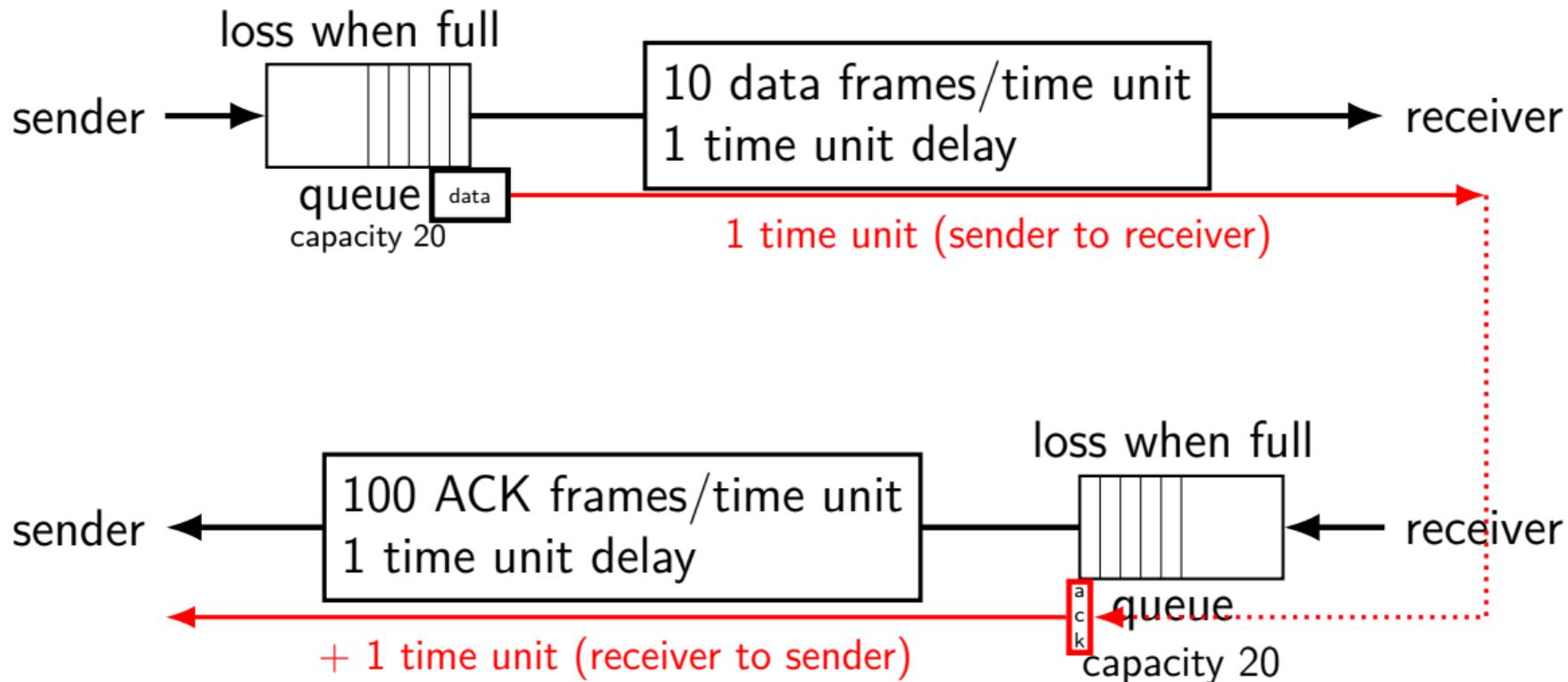
throughput and window size



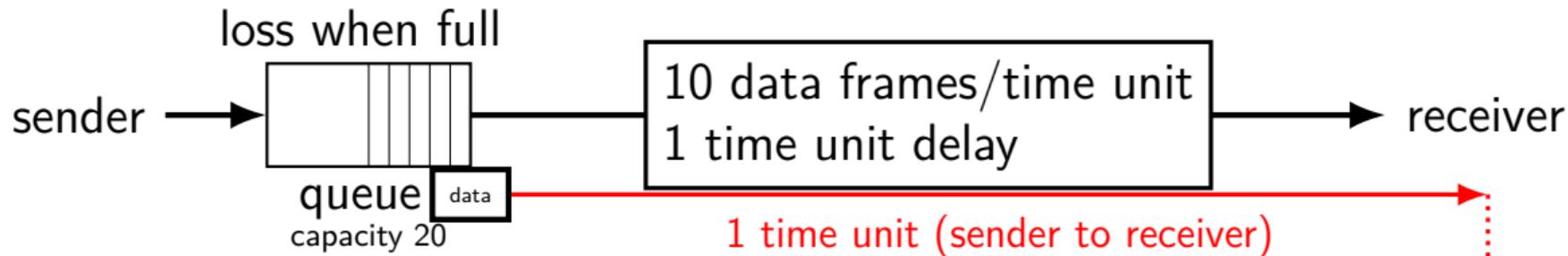
packet transit time



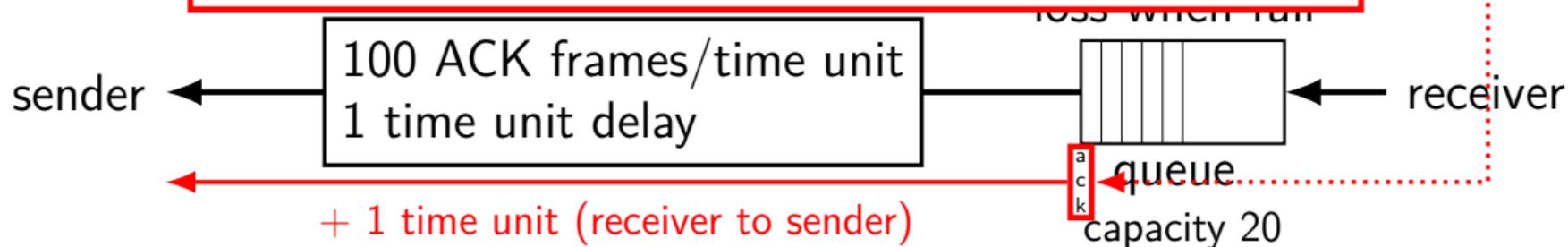
packet transit time



packet transit time



takes $1 + 1$ time units to send message + receive ack
goal: keep sending stuff while waiting



filling the pipe

round-trip time of 2 time units

from send data to receive ACK (assuming no queuing delay)

can send 10 data frames per time unit

= can send 20 data frames while waiting for ACK

filling the pipe

round-trip time of 2 time units

from send data to receive ACK (assuming no queuing delay)

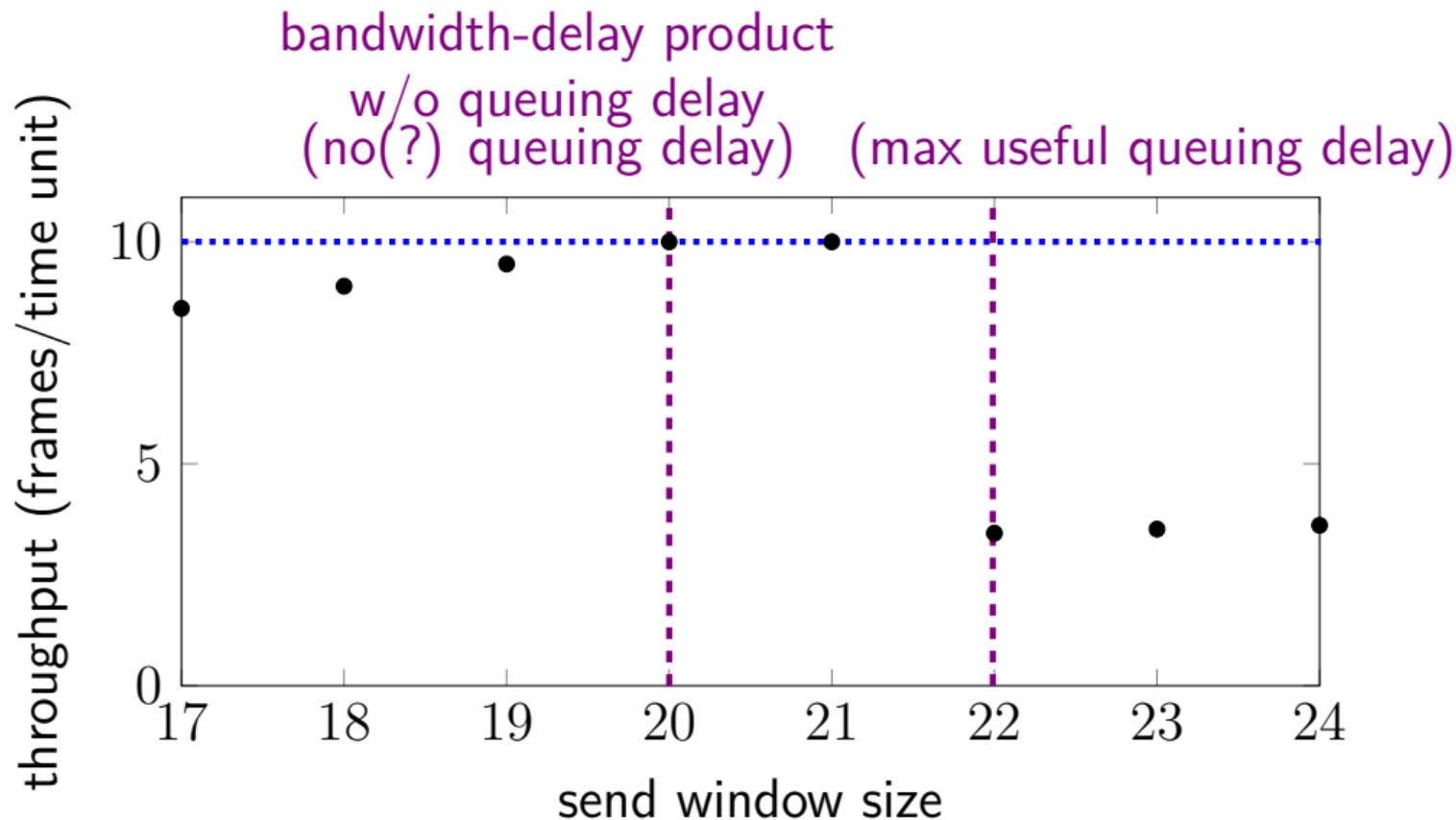
can send 10 data frames per time unit

= can send 20 data frames while waiting for ACK

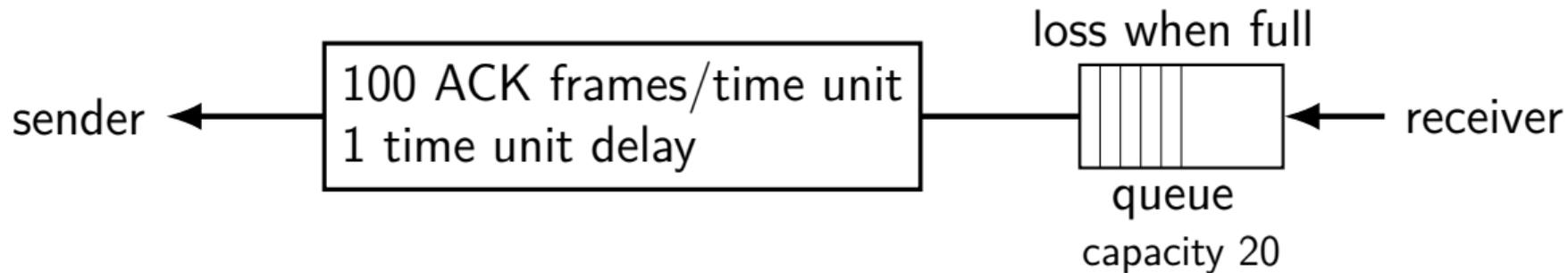
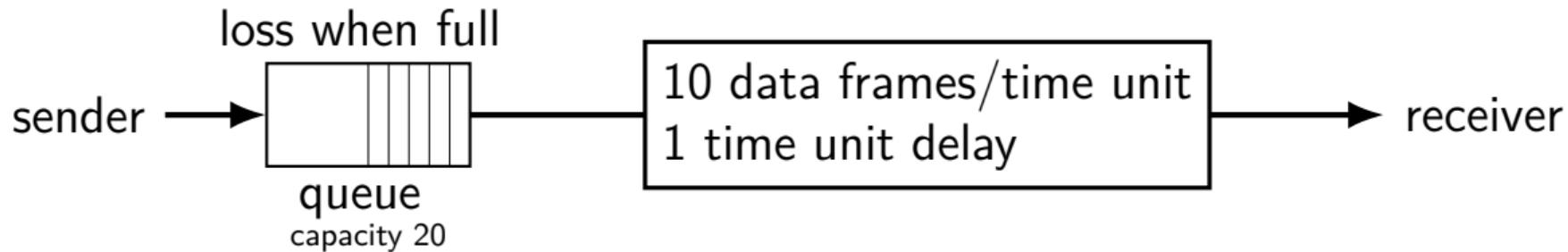
“bandwidth-delay product”

10/time unit (bandwidth) times 2 time unit (RTT = delay)

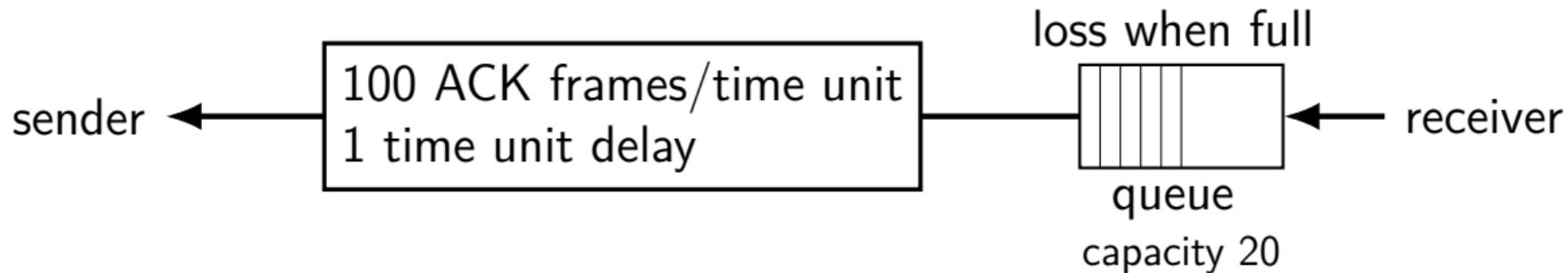
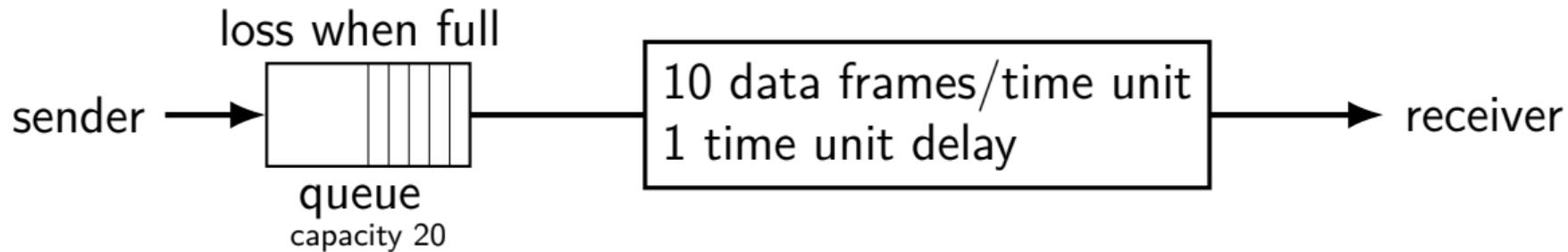
throughput and window size (detail)



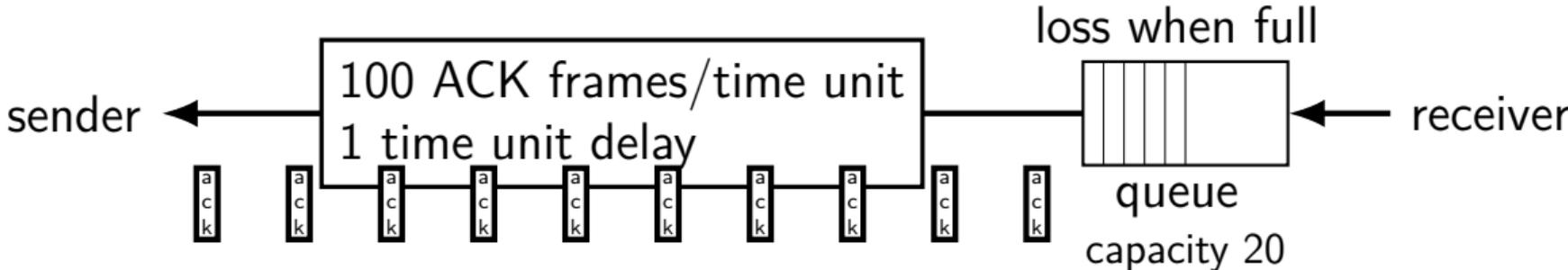
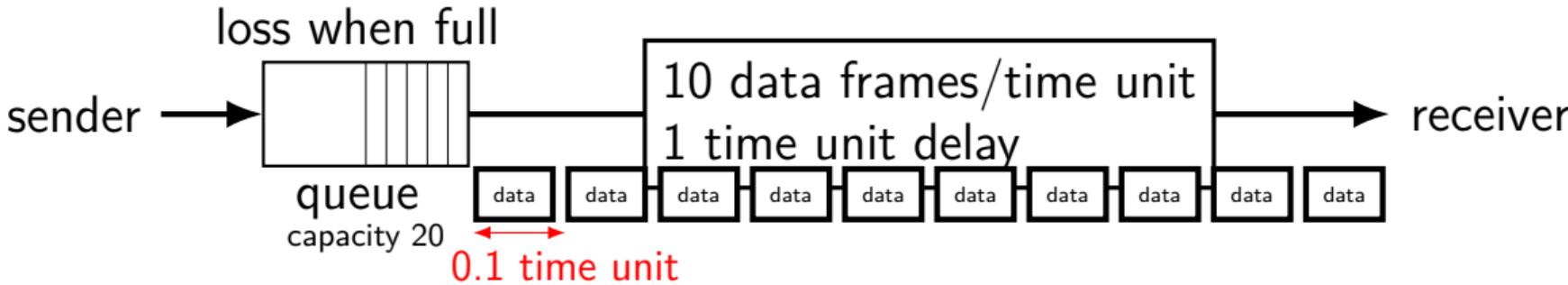
filling the pipe



filling the pipe



filling the pipe



on bursts

max possible queuing delay suggests window size of 30
approx. 3 time units times 10

problem: “bursts” temporarily exceed queue size

achievable average queue size not that high

sender could moderate by “pacing” packets

sliding windows used to solve...

flow control

keep sender from getting too far ahead of receiver

...by having **window sizes set correctly**

how? receiver tells sender what window size is okay

congestion control

keep network from being overloaded

(while making good use of available bandwidth)

...by having **window sizes set correctly**

how? it's complicated — big topic later

sequence number wraparound

protocol so far requires arbitrarily large sequence numbers

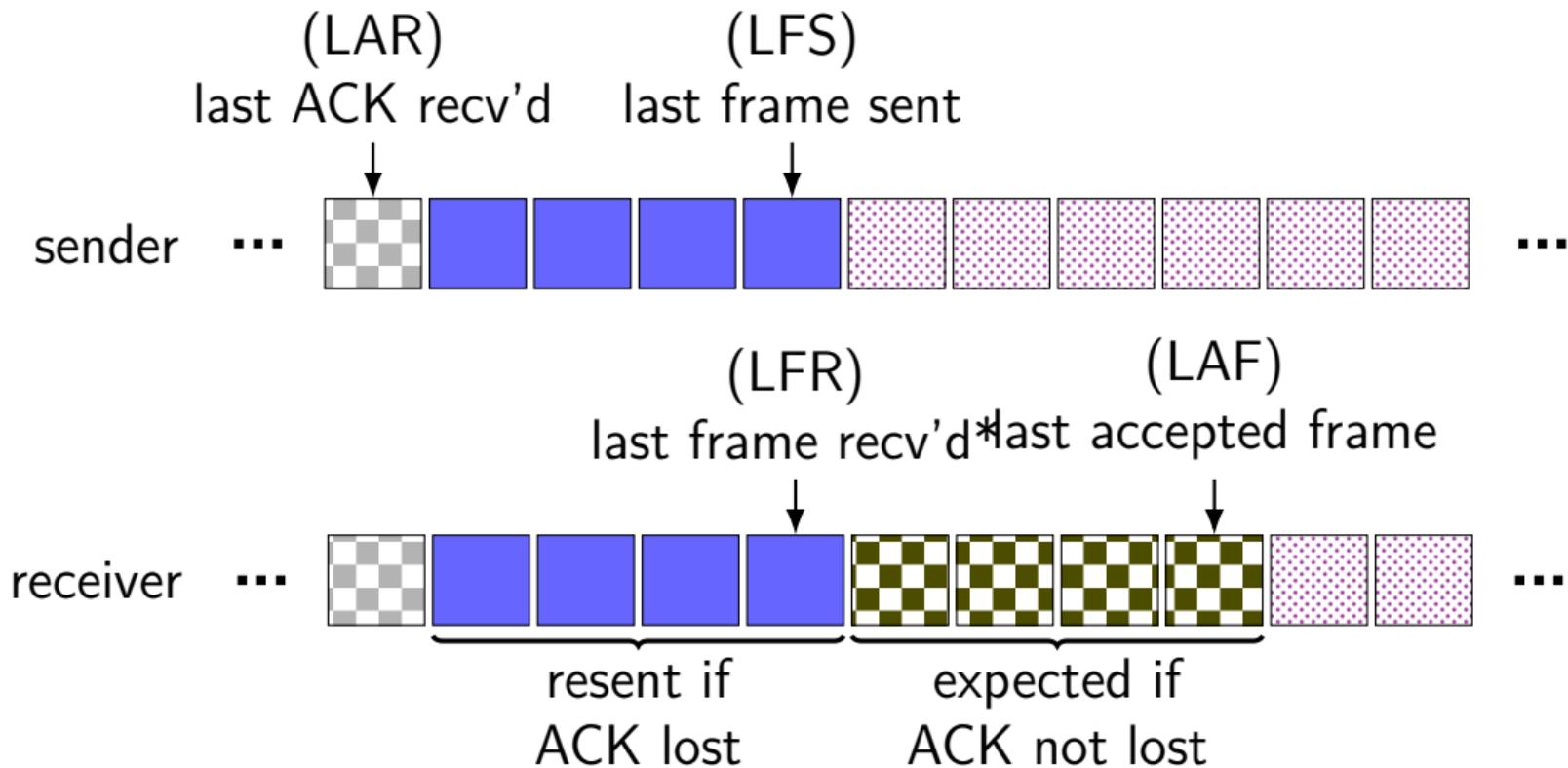
doing $<$ and $>$ checks on sequence number, so they need to increase

would like to use smaller sequence numbers

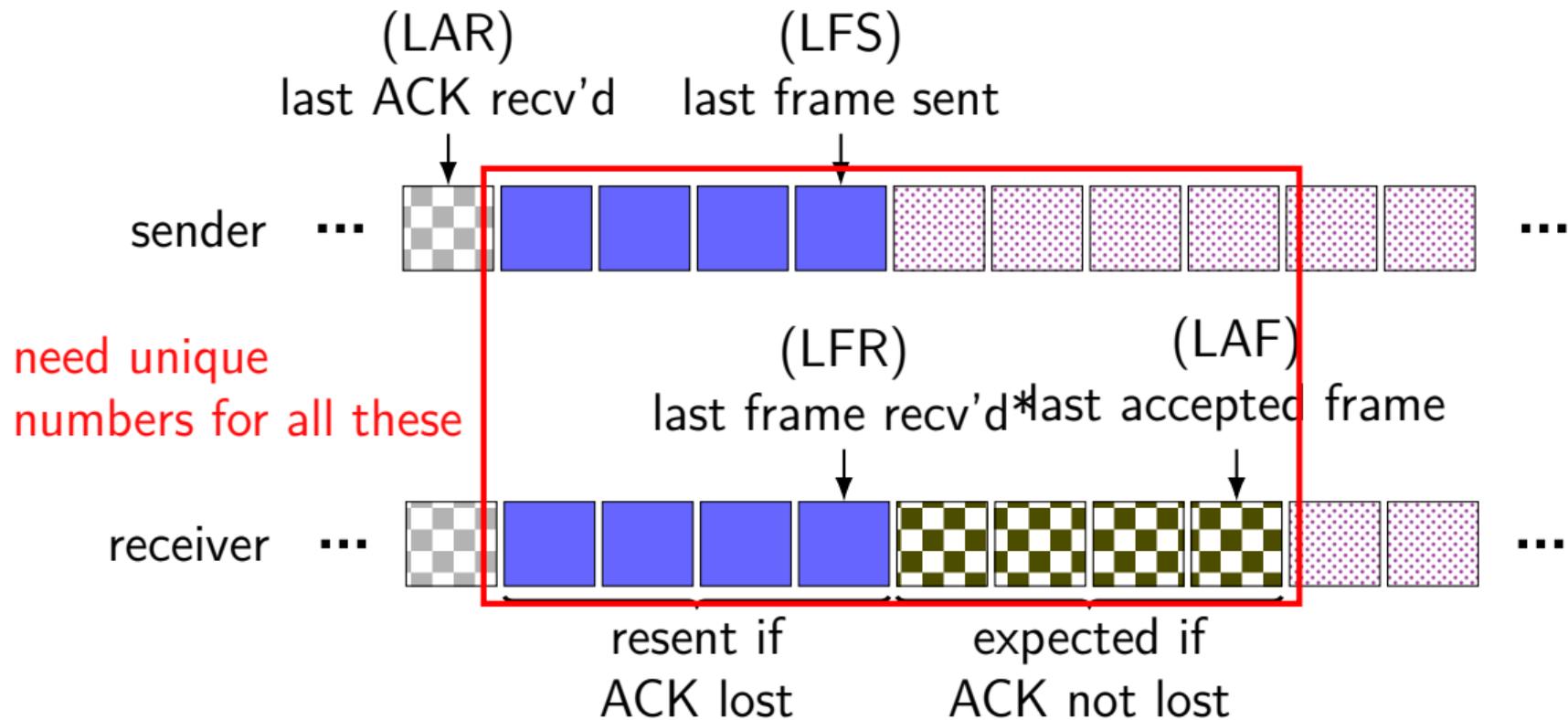
think: transferring multi-gigabyte file

question: what goes wrong when we reuse sequence numbers?

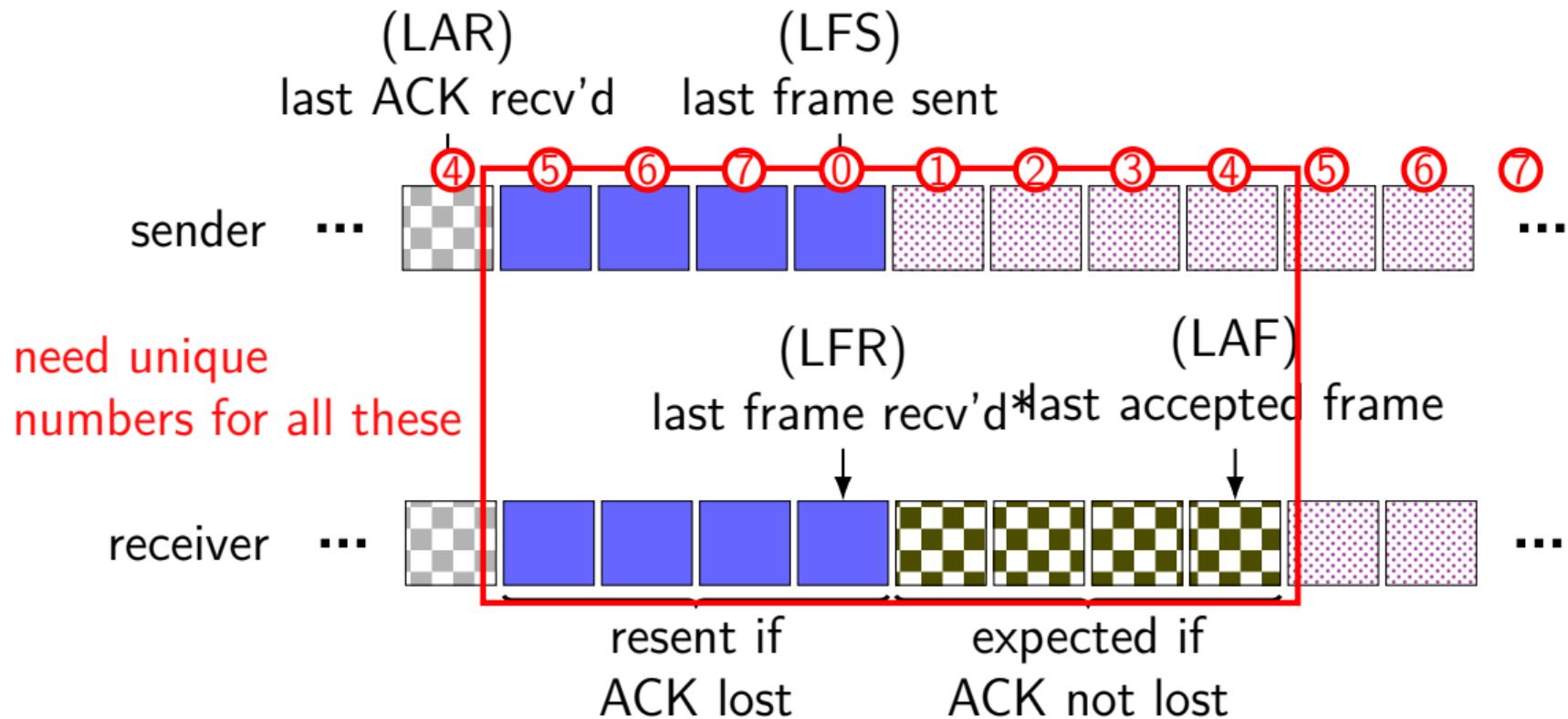
sender/receiver desync: missing ACKs



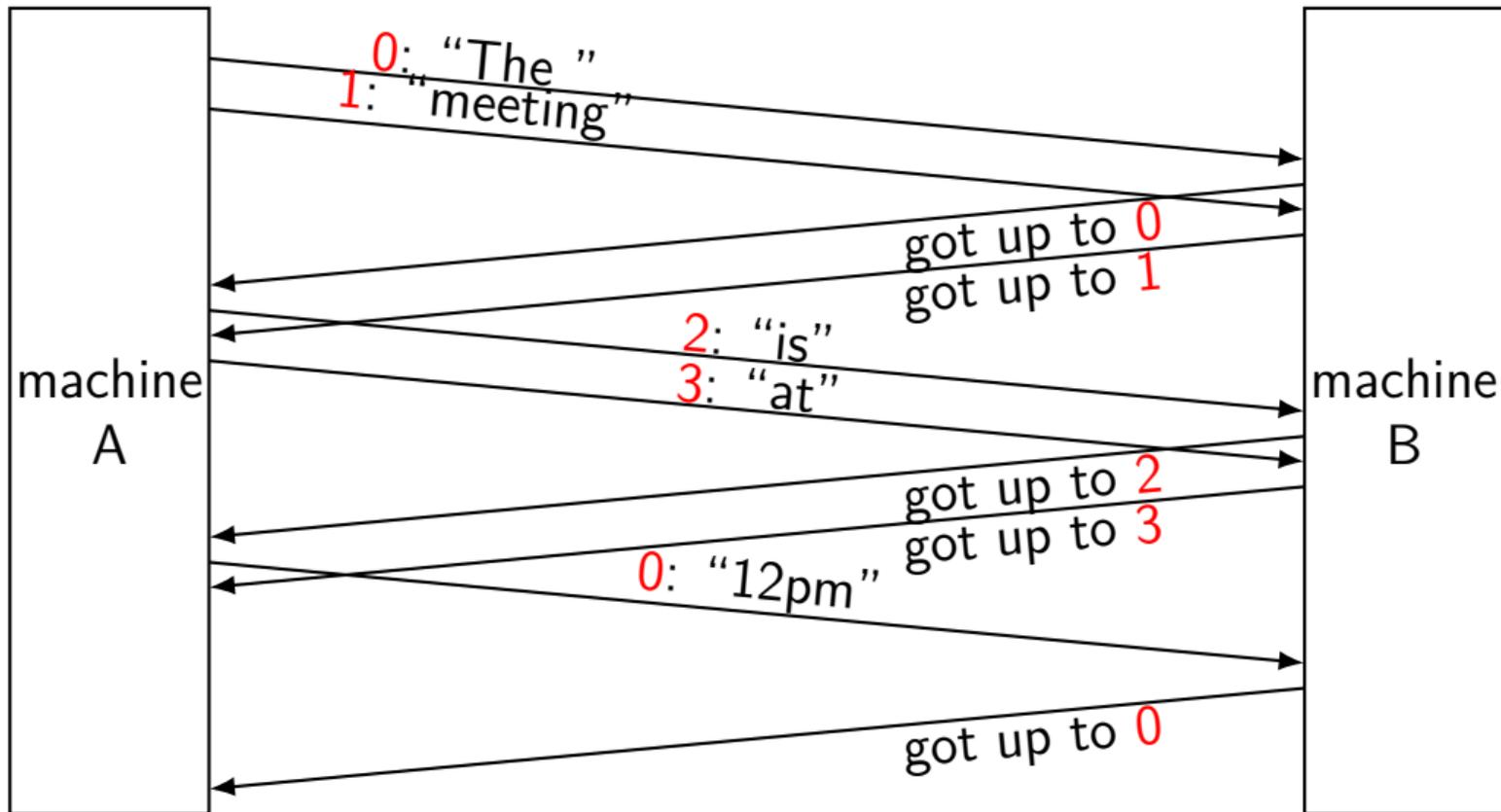
sender/receiver desync: missing ACKs



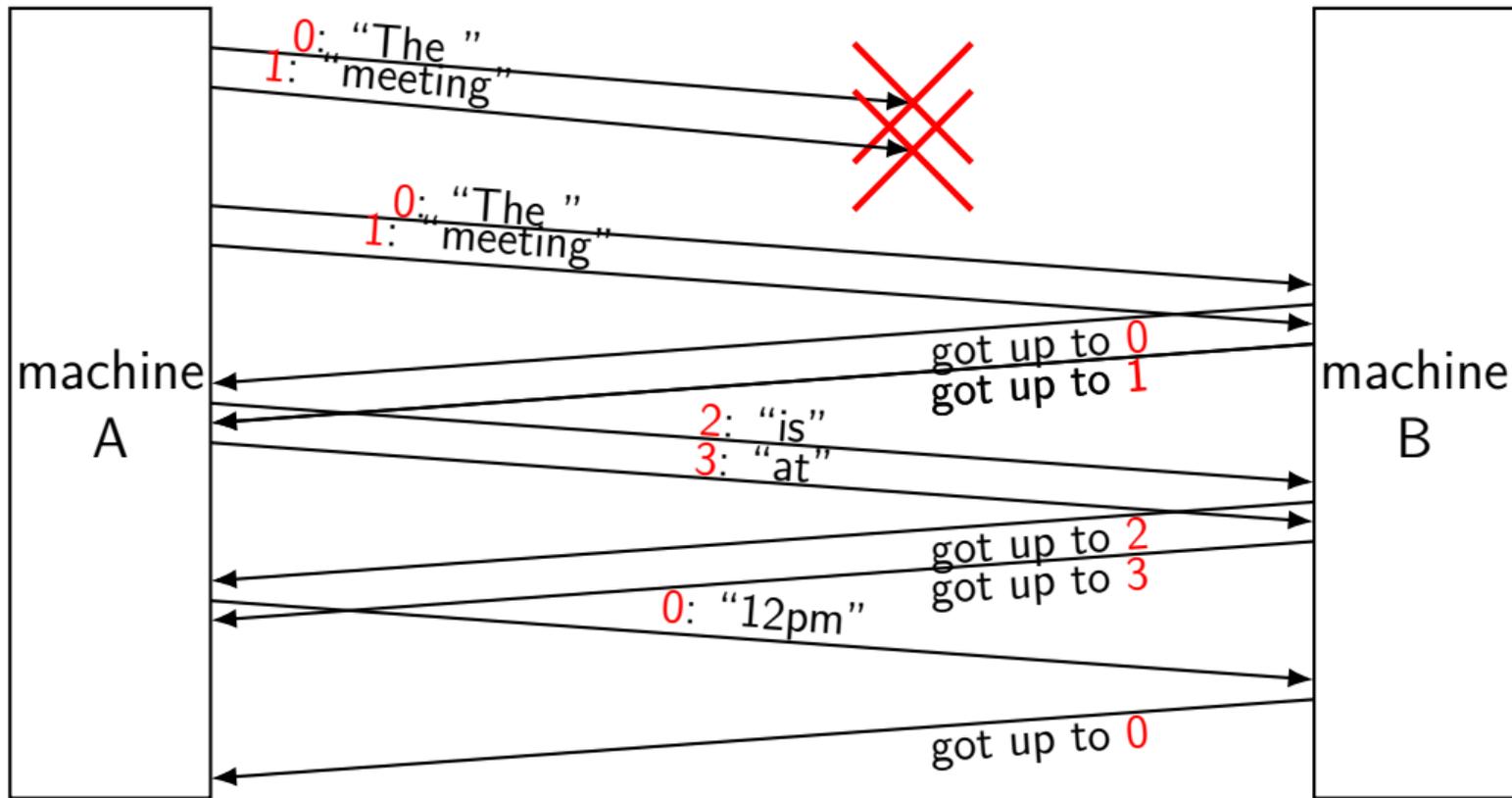
sender/receiver desync: missing ACKs



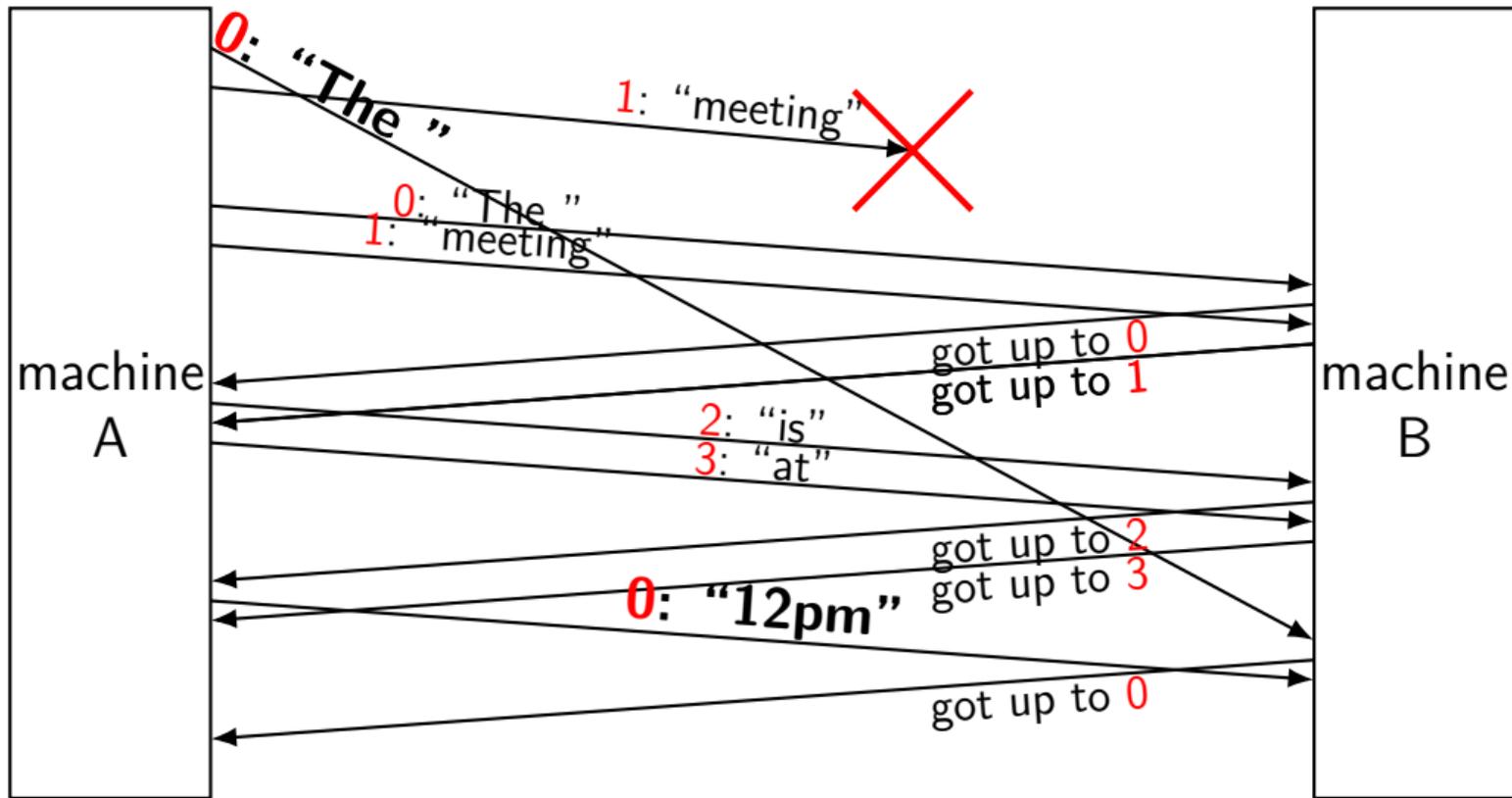
wraparound



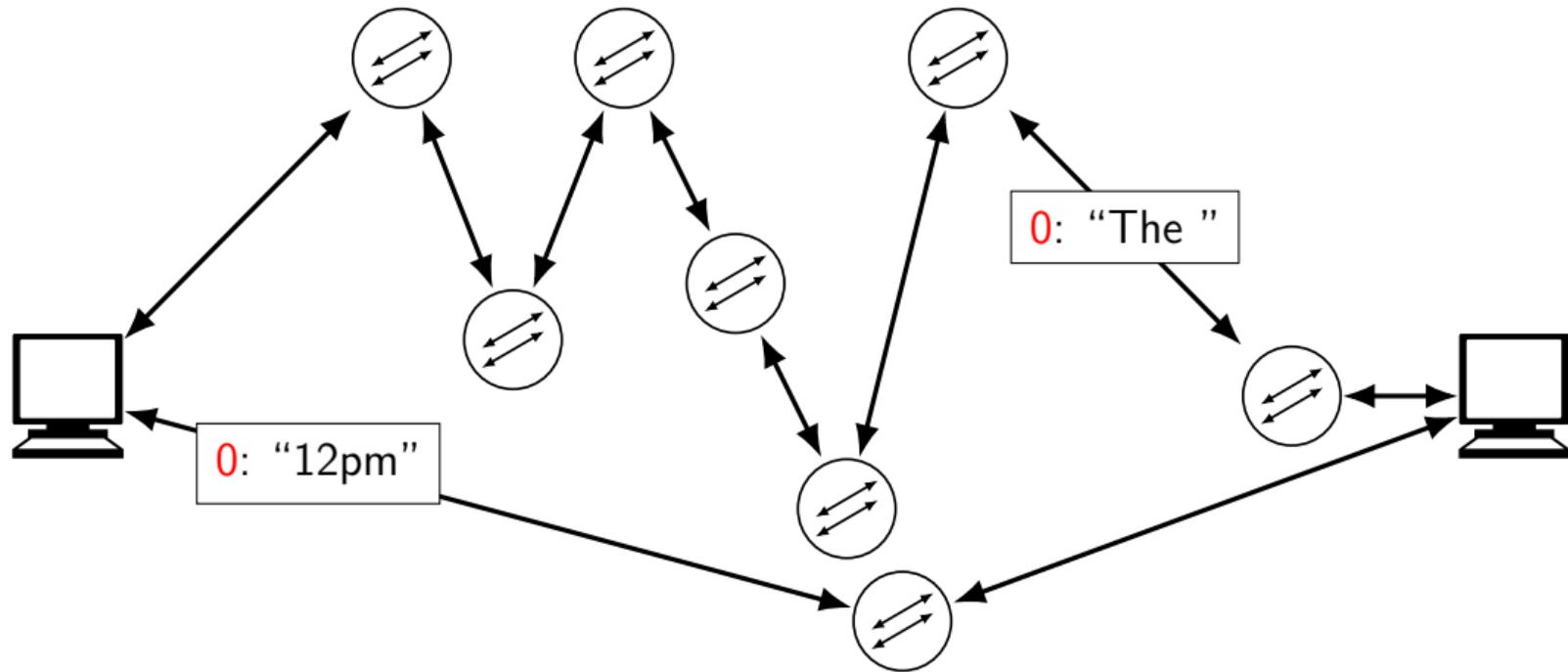
loss and resend?



very bad reordering



possible reason



sequence numbers in practice

TCP tries to assume 120 second “maximum segment lifetime”
segment = TCP's name for a packet

original TCP used 32-bit sequence number identifying *byte* number
(not segment number)

problem: means wraparound happens on modern (Gigabit+) links
in seconds!

sequence numbers in practice

TCP tries to assume 120 second “maximum segment lifetime”
segment = TCP’s name for a packet

original TCP used 32-bit sequence number identifying *byte* number
(not segment number)

problem: means wraparound happens on modern (Gigabit+) links
in seconds!

workaround: add *additional* 32-bit timestamp field
used to detect/discard duplicates
can also be used to set timeouts and/or window sizes

TCP

transmission control protocol (TCP)

implements reliable streams of bytes

similar mechanism to what we've described

TCP extras/differences

bidirectional —

separate sequence numbers in each direction

can combine data (from A to B) with acknowledgment (from B to A)

sequence numbers are byte numbers —

can retransmit data in different sized packets

sequence numbers = index of *first byte* sent

acknowledgment numbers = 1 + index of *last byte* acknowledged

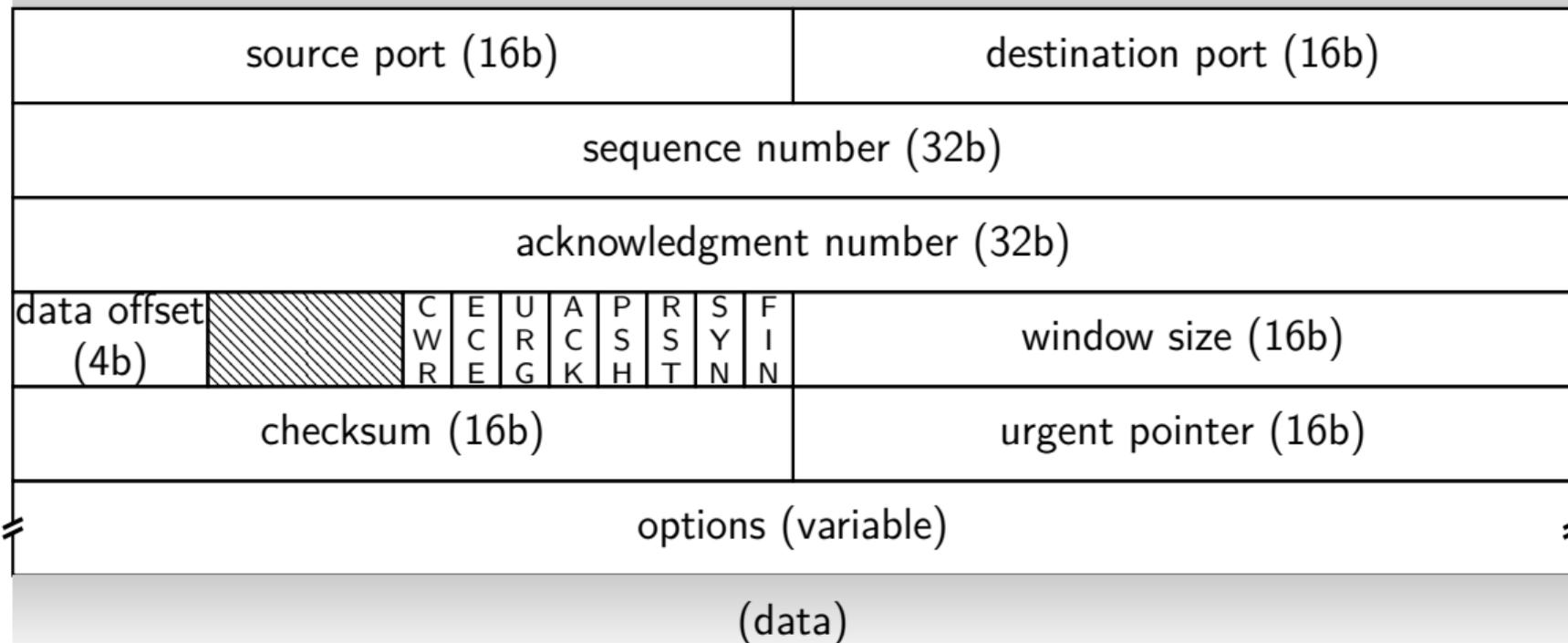
dynamic/variable window sizes

we'll discuss strategies later

official name for packets = *segments*

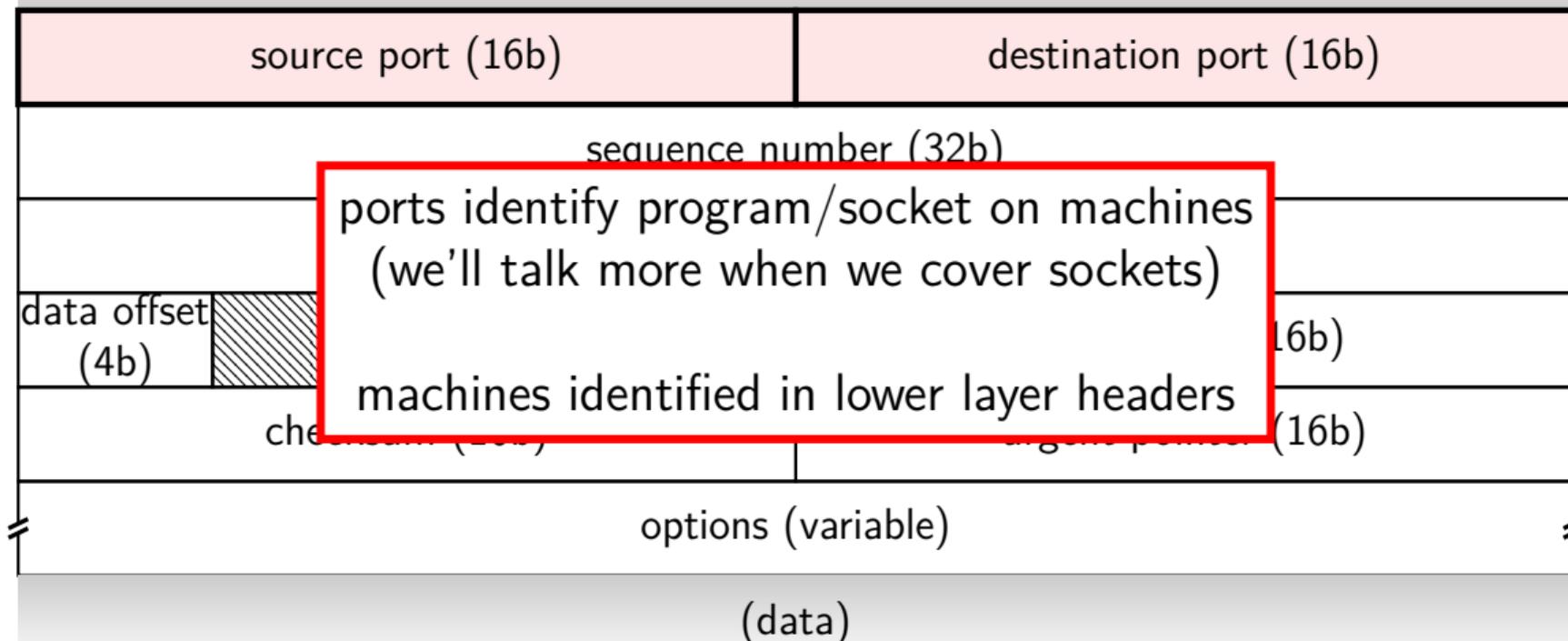
TCP segment format

(lower layer header)



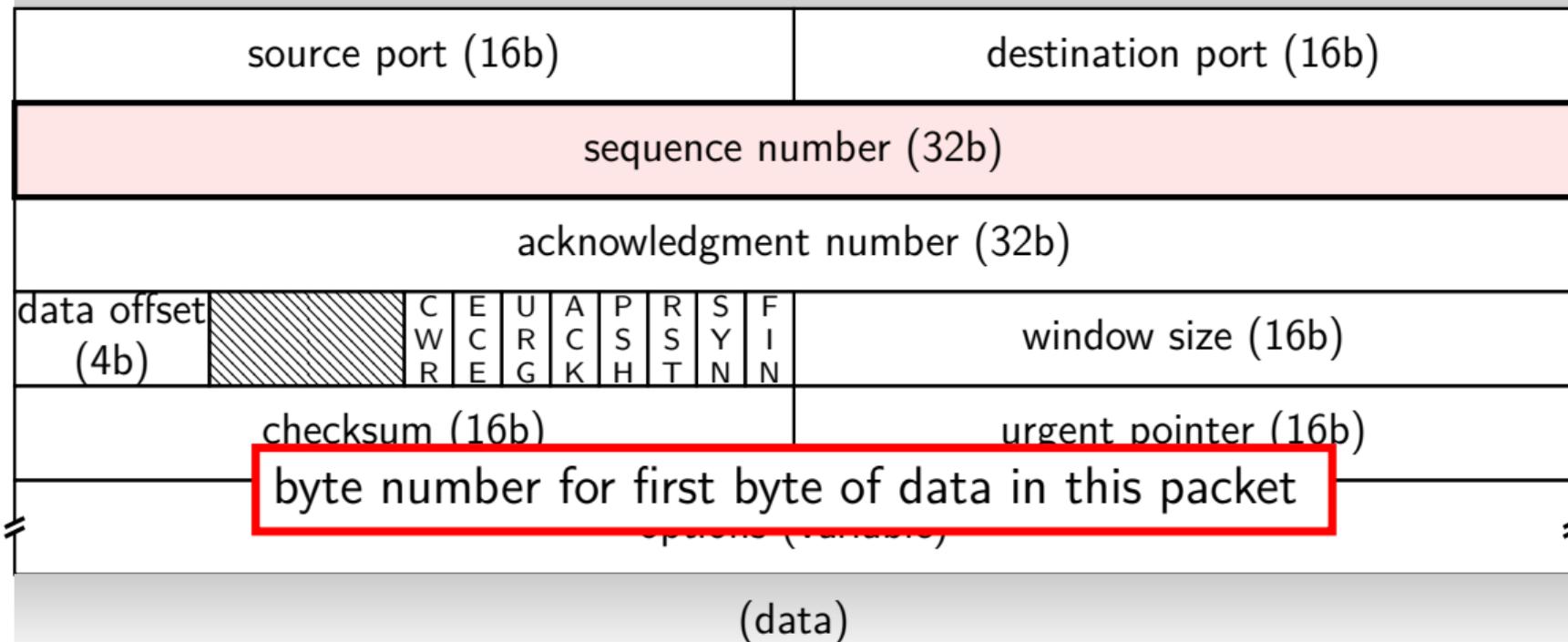
TCP segment format

(lower layer header)



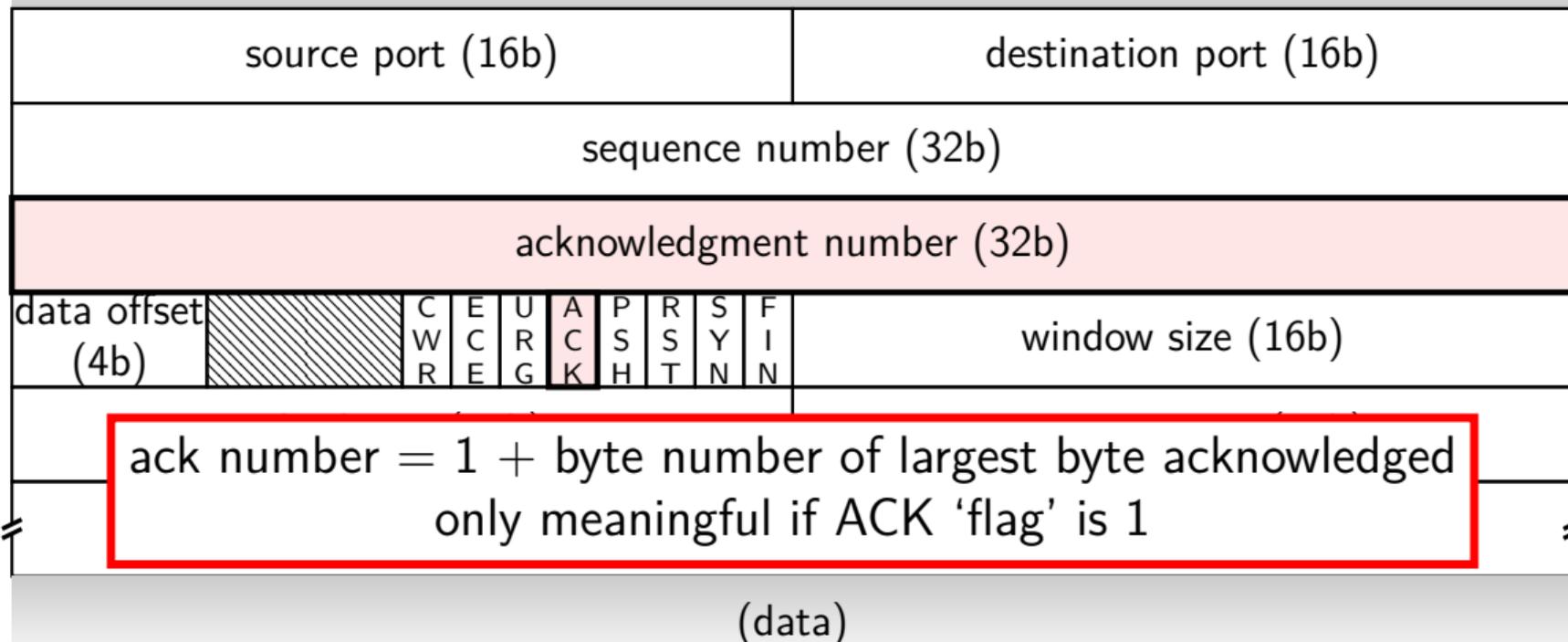
TCP segment format

(lower layer header)

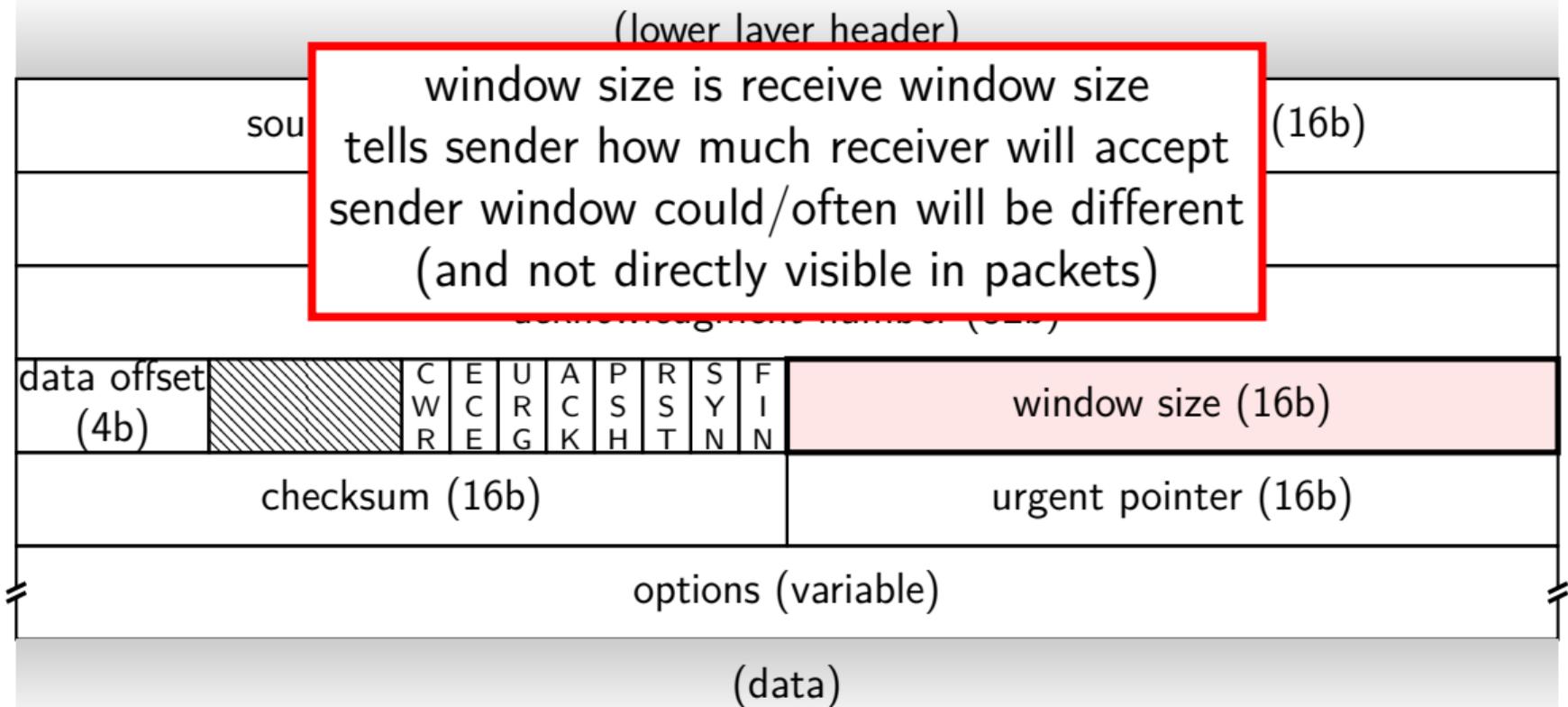


TCP segment format

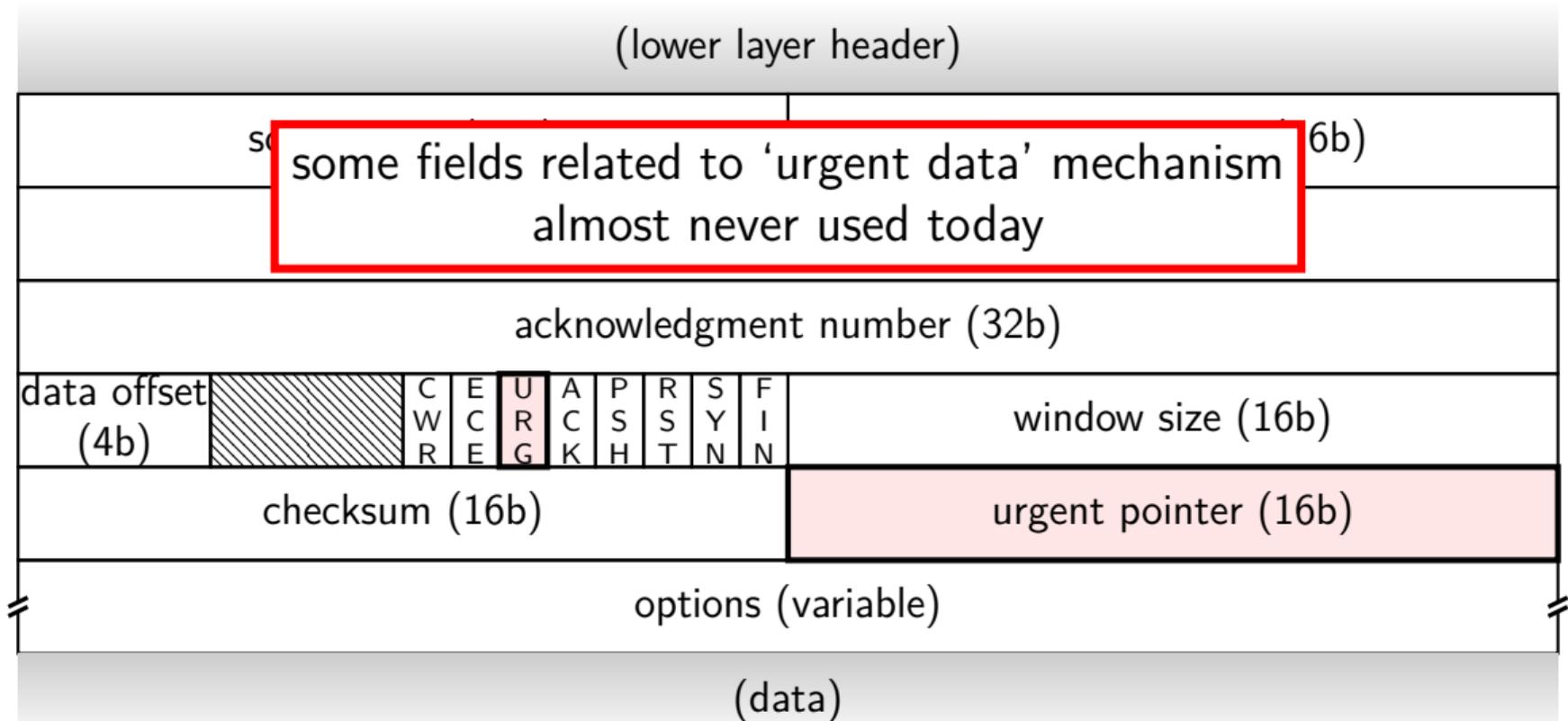
(lower layer header)



TCP segment format



TCP segment format



TCP segment format

(lower layer header)

PSH (push) 'flag' is hint that sender does not have more data to send right away

acknowledgment number (32b)

data offset
(4b)



C E U A P R S F
W C R C S S Y I
R E G K H T N N

window size (16b)

checksum (16b)

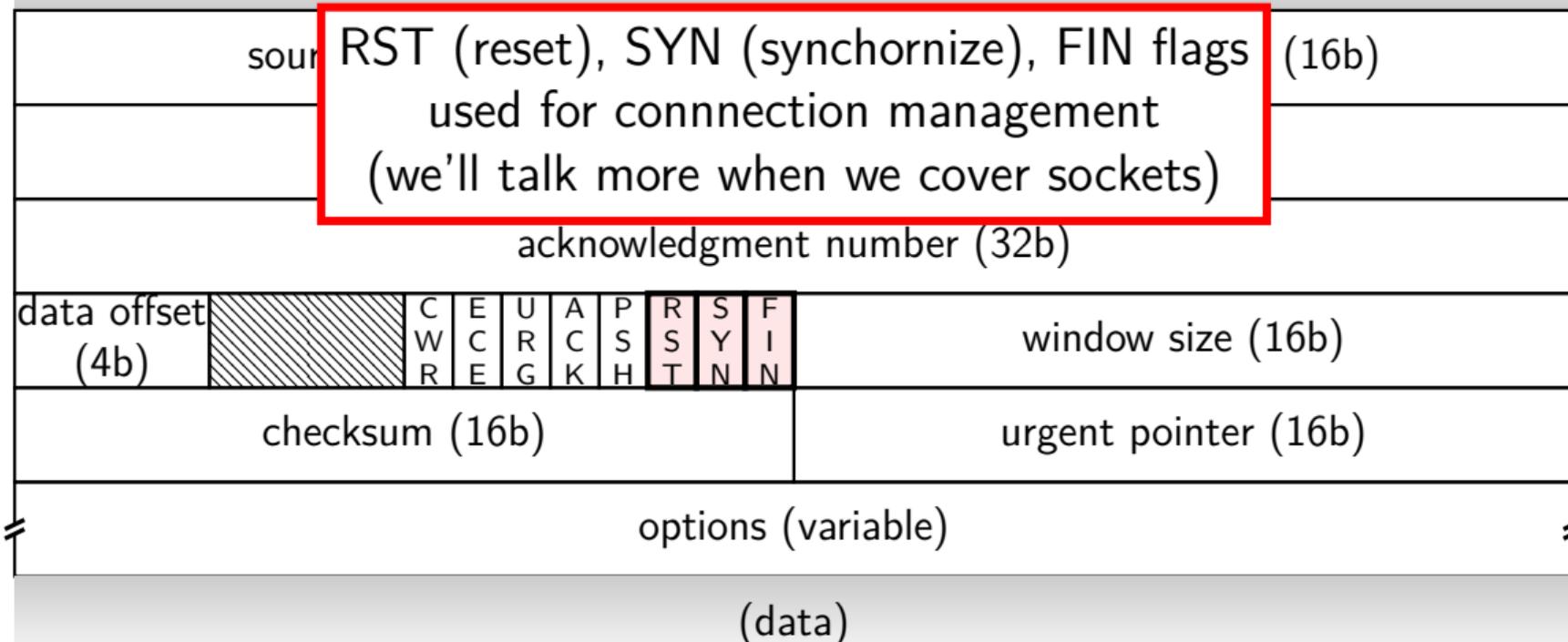
urgent pointer (16b)

options (variable)

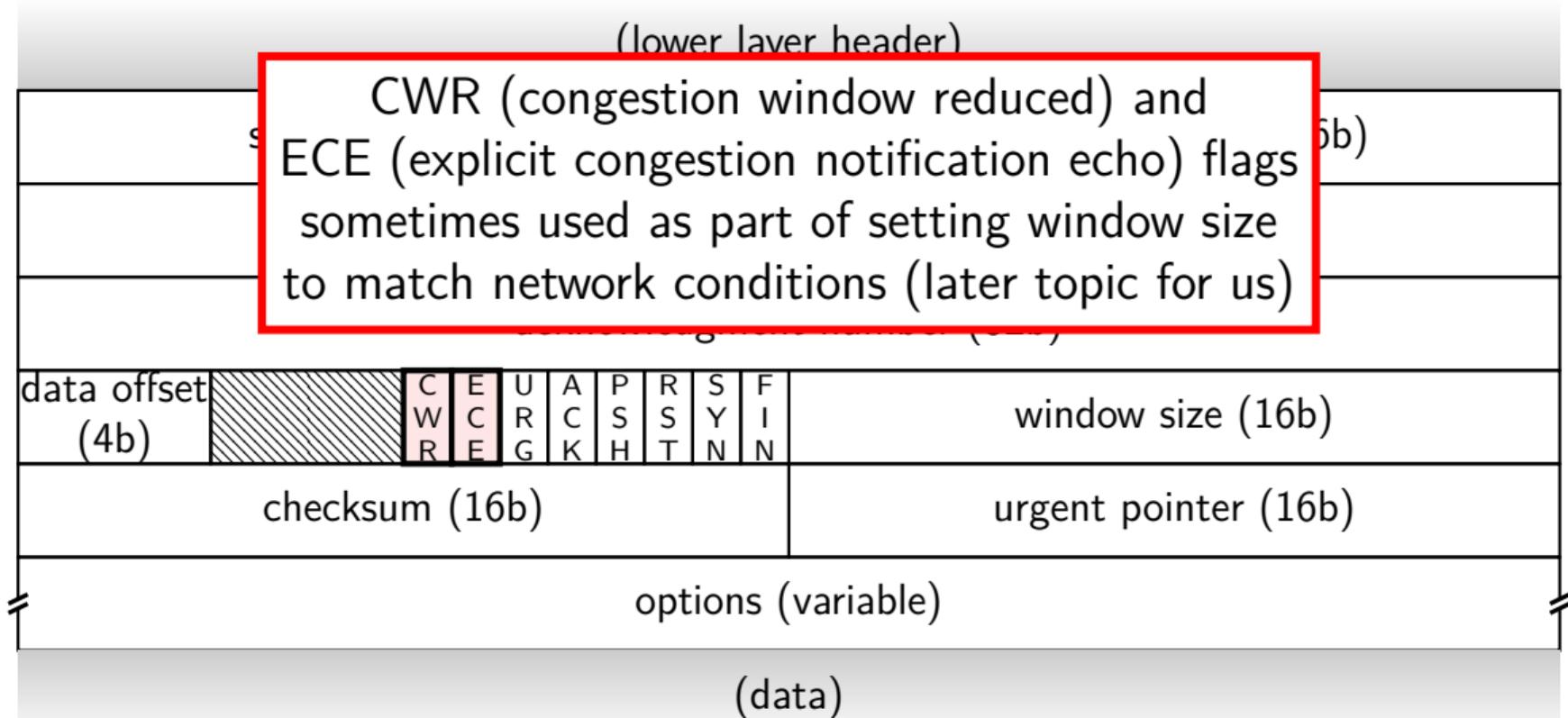
(data)

TCP segment format

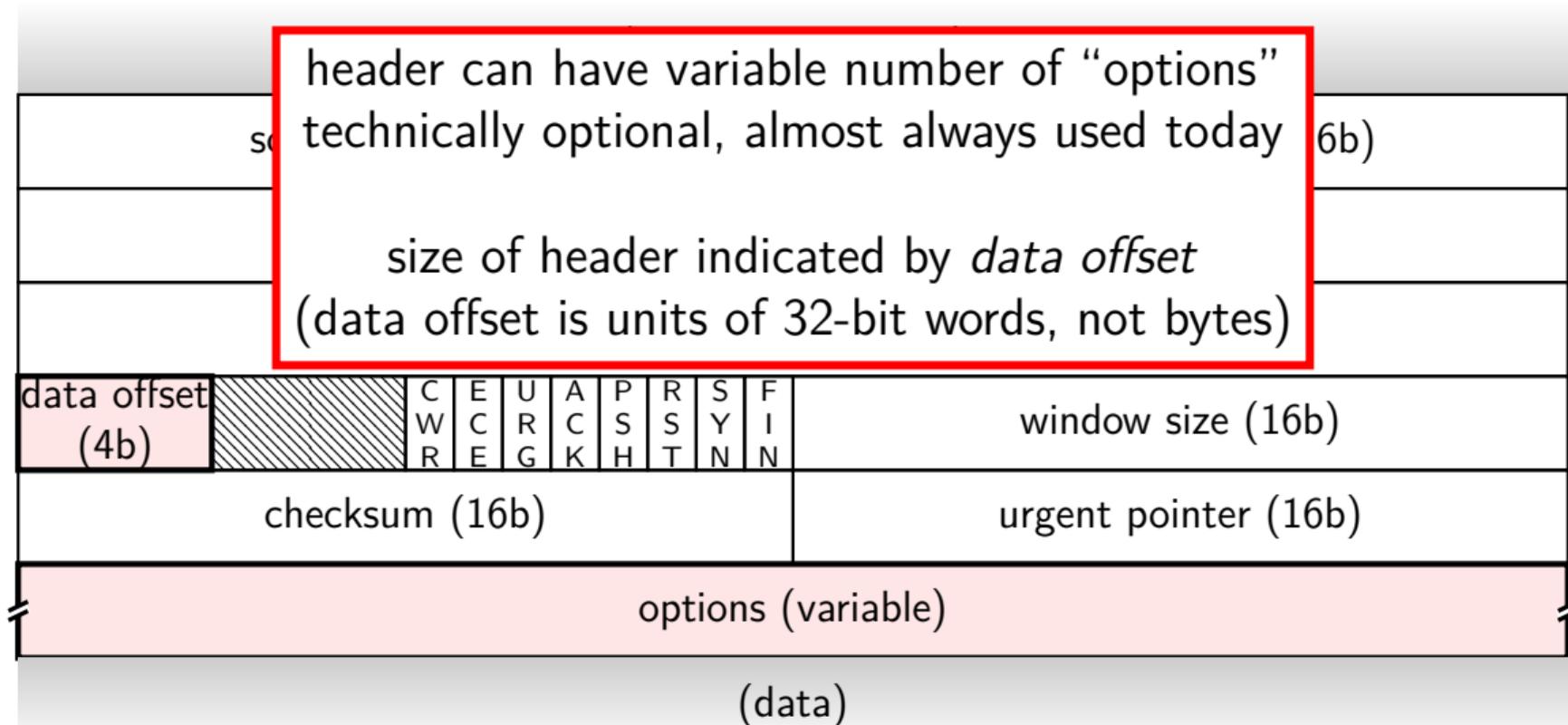
(lower layer header)



TCP segment format



TCP segment format



exercise: maximum throughput

let's say we have a receiver window size of 65535 bytes

and a round-trip time of 100 ms

if we want to avoid sending data the receiver will reject as outside its window, maximum throughput?

- A. around 32kbyte/sec
- B. around 64kbyte/sec
- C. around 128kbyte/sec
- D. around 320kbyte/sec
- E. around 640kbyte/sec
- F. around 1280kbyte/sec
- G. something else

selected TCP options

window size scale factor

- allow receiver window sizes greater than 64k
- needed to get reasonable bandwidth on modern networks

timestamps

- allow figuring out round trip time to estimate timeout
- extend 32-bit sequence number, which is too small for multi-gigabit networks

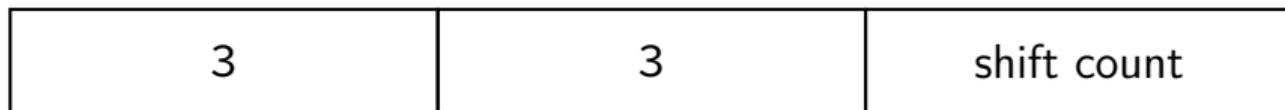
selective acknowledgements

- allow providing information about 'holes' in received data
- example: I got bytes 1–5000, 6000–7000, 8000–9000
- without it would only say 5000

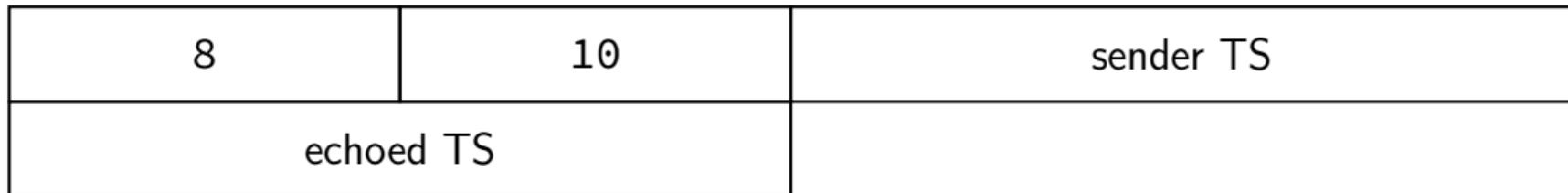
selected TCP option formats



window scale option:



timestamps:



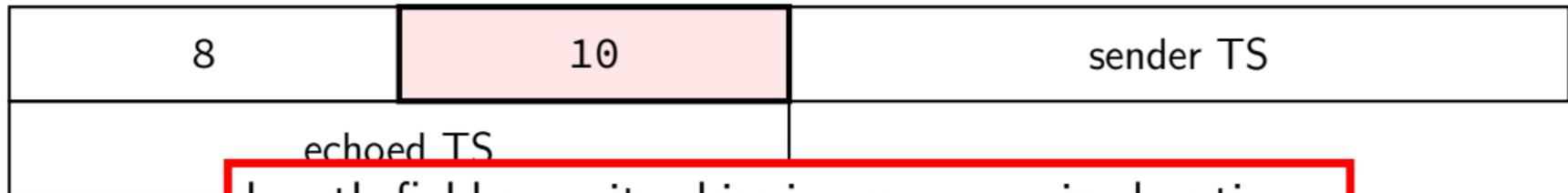
selected TCP option formats



window scale option:



timestamps:

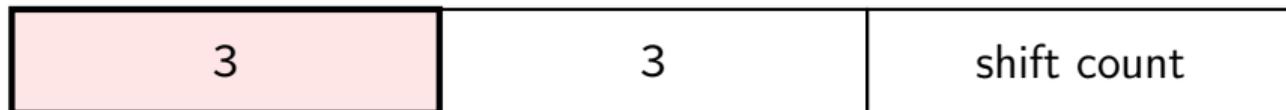


length field permits skipping unrecognized options

selected TCP option formats



window scale option:



timestamps:



ech

unique kind codes for each option
list of valid codes maintained by IANA
(Internet Assigned Numbers Authority)

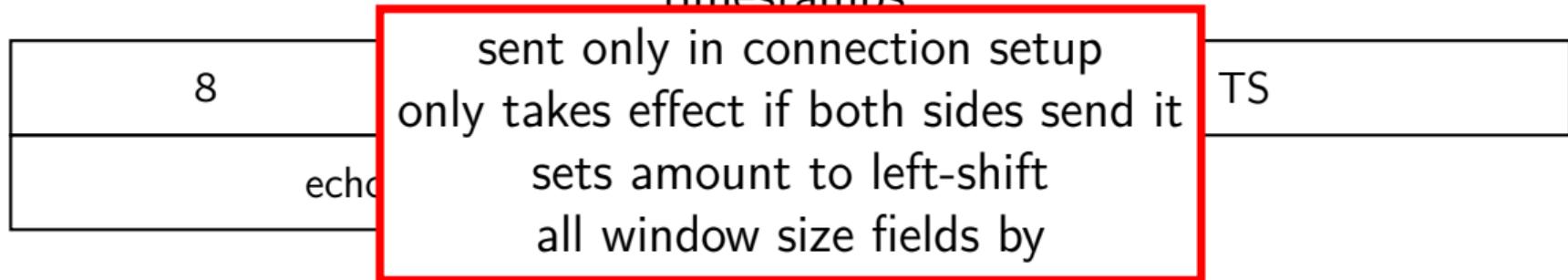
selected TCP option formats



window scale option:



timestamps:



selected TCP option formats

kind (8b)	length (8b)	option data
-----------	-------------	-------------

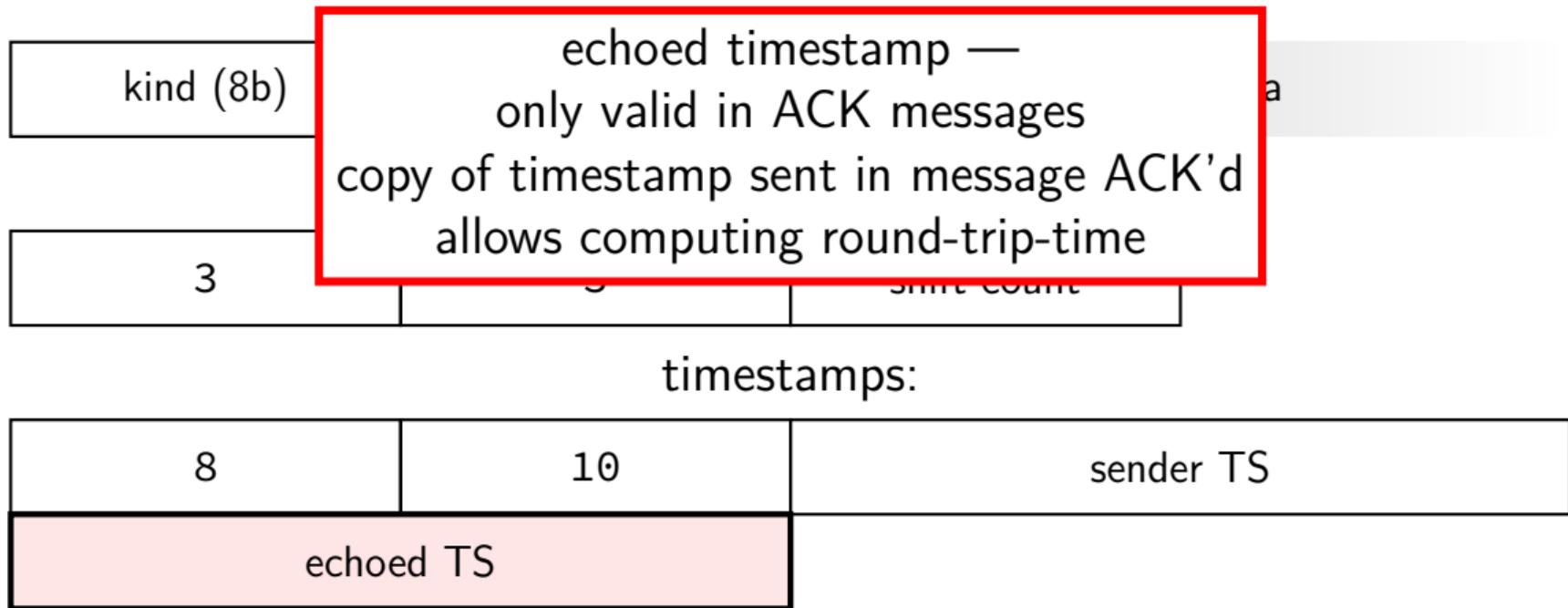
window scale option:

3	3	shift count
---	---	-------------

timestamps:

8	10	sender TS
echoed TS		

selected TCP option formats



a TCP connection

File Edit View Go Capture Analyze Statistics Telephony Wireless Tools Help

Apply a display filter ... <Ctrl->

No.	Time	Source	Destination	Protocol	Length	Seq#	Ack#	Info
1	0.000000	10.0.1.2	10.0.1.1	TCP	74	0	0	42732 → 5001 [SYN, Seq=0 Win=21900 Len=0 MSS=1460 SACK
2	0.013678	10.0.1.1	10.0.1.2	TCP	74	0	1	5001 → 42732 [SYN, ACK] Seq=0 Ack=1 Win=21720 Len=0 MSS
3	0.013731	10.0.1.2	10.0.1.1	TCP	66	1	1	42732 → 5001 [ACK] Seq=1 Ack=1 Win=22016 Len=0 TSval=1
4	0.014448	10.0.1.2	10.0.1.1	TCP	126	1	1	42732 → 5001 [PSH, ACK] Seq=1 Ack=1 Win=22016 Len=60 TS
5	0.014487	10.0.1.2	10.0.1.1	TCP	1514	61	1	42732 → 5001 [ACK] Seq=61 Ack=1 Win=22016 Len=1448 TSv
6	0.014489	10.0.1.2	10.0.1.1	TCP	1514	1509	1	42732 → 5001 [PSH, ACK] Seq=1509 Ack=1 Win=22016 Len=1
7	0.014499	10.0.1.2	10.0.1.1	TCP	1514	2957	1	42732 → 5001 [ACK] Seq=2957 Ack=1 Win=22016 Len=1448 T
8	0.014500	10.0.1.2	10.0.1.1	TCP	1514	4405	1	42732 → 5001 [PSH, ACK] Seq=4405 Ack=1 Win=22016 Len=1
9	0.014507	10.0.1.2	10.0.1.1	TCP	1514	5853	1	42732 → 5001 [ACK] Seq=5853 Ack=1 Win=22016 Len=1448 T
10	0.014508	10.0.1.2	10.0.1.1	TCP	1514	7301	1	42732 → 5001 [PSH, ACK] Seq=7301 Ack=1 Win=22016 Len=1
11	0.014514	10.0.1.2	10.0.1.1	TCP	1514	8749	1	42732 → 5001 [ACK] Seq=8749 Ack=1 Win=22016 Len=1448 T
12	0.014515	10.0.1.2	10.0.1.1	TCP	1514	10197	1	42732 → 5001 [PSH, ACK] Seq=10197 Ack=1 Win=22016 Len=
13	0.014521	10.0.1.2	10.0.1.1	TCP	1514	11645	1	42732 → 5001 [PSH, ACK] Seq=11645 Ack=1 Win=22016 Len=
14	0.045412	10.0.1.2	10.0.1.1	TCP	1514	13093	1	42732 → 5001 [ACK] Seq=13093 Ack=1 Win=22016 Len=1448
15	0.132429	10.0.1.1	10.0.1.2	TCP	66	1	61	5001 → 42732 [ACK] Seq=1 Ack=61 Win=22016 Len=0 TSval=
16	0.132462	10.0.1.2	10.0.1.1	TCP	1514	14541	1	42732 → 5001 [ACK] Seq=14541 Ack=1 Win=22016 Len=1448
17	0.153499	10.0.1.1	10.0.1.2	TCP	66	29	2957	[TCP Previous segment not captured] 5001 → 42732 [ACK]
18	0.153549	10.0.1.2	10.0.1.1	TCP	1514	15989	1	42732 → 5001 [ACK] Seq=15989 Ack=1 Win=22016 Len=1448
19	0.153557	10.0.1.2	10.0.1.1	TCP	1514	17437	1	42732 → 5001 [PSH, ACK] Seq=17437 Ack=1 Win=22016 Len=
20	0.153576	10.0.1.2	10.0.1.1	TCP	1514	18885	1	42732 → 5001 [ACK] Seq=18885 Ack=1 Win=22016 Len=1448
21	0.153577	10.0.1.2	10.0.1.1	TCP	1514	20333	1	42732 → 5001 [PSH, ACK] Seq=20333 Ack=1 Win=22016 Len=
22	0.166571	10.0.1.1	10.0.1.2	TCP	66	29	5853	5001 → 42732 [ACK] Seq=29 Ack=5853 Win=20992 Len=0 TSv
23	0.166622	10.0.1.2	10.0.1.1	TCP	1514	21781	1	42732 → 5001 [ACK] Seq=21781 Ack=1 Win=22016 Len=1448
24	0.166630	10.0.1.2	10.0.1.1	TCP	1514	23229	1	42732 → 5001 [ACK] Seq=23229 Ack=1 Win=22016 Len=1448
25	0.166632	10.0.1.2	10.0.1.1	TCP	1514	24677	1	42732 → 5001 [PSH, ACK] Seq=24677 Ack=1 Win=22016 Len=
26	0.173766	10.0.1.1	10.0.1.2	TCP	66	29	8749	5001 → 42732 [ACK] Seq=29 Ack=8749 Win=20992 Len=0 TSv
27	0.173815	10.0.1.2	10.0.1.1	TCP	1514	26125	1	42732 → 5001 [ACK] Seq=26125 Ack=1 Win=22016 Len=1448

[Calculated window size: 21900]

0030 55 8c 5a 1a 00 00 02 04 05 b4 04 02 08 0a 45 b0 U-Z.....

The scaled window size (if scaling has been used) (tcp.window_size), 2 bytes

Packets: 1313 · Displayed: 1313 (100.0%) Profile: Default

a TCP connection

Apply a display filter ... <Ctrl->

No.	Time	Source	Destination	Protocol	Length	Seq#	Ack#	Info
1	0.000000	10.0.1.2	10.0.1.1	TCP	74	0	0	42732 → 5001 [SYN] Seq=0 Win=21900 Len=0 MSS=1460 SACK...
2	0.013678	10.0.1.1	10.0.1.2	TCP	74	0	1	5001 → 42732 [SYN, ACK] Seq=0 Ack=1 Win=21720 Len=0 MSS...
3	0.013731	10.0.1.2	10.0.1.1	TCP	66	1	1	42732 → 5001 [ACK] Seq=1 Ack=1 Win=22016 Len=0 TSval=1...
4	0.014448	10.0.1.1	10.0.1.2	TCP	66	1	1	5001 → 42732 [ACK] Seq=1 Ack=1 Win=22016 Len=0 TSval=1...
5	0.014487	10.0.1.1	10.0.1.2	TCP	66	1	1	5001 → 42732 [ACK] Seq=1 Ack=1 Win=22016 Len=0 TSval=1...
6	0.014489	10.0.1.1	10.0.1.2	TCP	66	1	1	5001 → 42732 [ACK] Seq=1 Ack=1 Win=22016 Len=0 TSval=1...
7	0.014499	10.0.1.1	10.0.1.2	TCP	66	1	1	5001 → 42732 [ACK] Seq=1 Ack=1 Win=22016 Len=0 TSval=1...
8	0.014500	10.0.1.2	10.0.1.1	TCP	1514	4405	1	42732 → 5001 [PSH, ACK] Seq=4405 Ack=1 Win=22016 Len=1...
9	0.014507	10.0.1.2	10.0.1.1	TCP	1514	5853	1	42732 → 5001 [ACK] Seq=5853 Ack=1 Win=22016 Len=1448 T...
10	0.014508	10.0.1.2	10.0.1.1	TCP	1514	7301	1	42732 → 5001 [PSH, ACK] Seq=7301 Ack=1 Win=22016 Len=1...
11	0.014514	10.0.1.2	10.0.1.1	TCP	1514	8749	1	42732 → 5001 [ACK] Seq=8749 Ack=1 Win=22016 Len=1448 T...
12	0.014515	10.0.1.2	10.0.1.1	TCP	1514	10197	1	42732 → 5001 [PSH, ACK] Seq=10197 Ack=1 Win=22016 Len=...
13	0.014521	10.0.1.2	10.0.1.1	TCP	1514	11645	1	42732 → 5001 [PSH, ACK] Seq=11645 Ack=1 Win=22016 Len=...
14	0.045412	10.0.1.2	10.0.1.1	TCP	1514	13093	1	42732 → 5001 [ACK] Seq=13093 Ack=1 Win=22016 Len=1448...
15	0.132429	10.0.1.1	10.0.1.2	TCP	66	1	61	5001 → 42732 [ACK] Seq=1 Ack=61 Win=22016 Len=0 TSval=...
16	0.132462	10.0.1.2	10.0.1.1	TCP	1514	14541	1	42732 → 5001 [ACK] Seq=14541 Ack=1 Win=22016 Len=1448...
17	0.153499	10.0.1.1	10.0.1.2	TCP	66	29	2957	[TCP Previous segment not captured] 5001 → 42732 [ACK]...
18	0.153549	10.0.1.2	10.0.1.1	TCP	1514	15989	1	42732 → 5001 [ACK] Seq=15989 Ack=1 Win=22016 Len=1448...
19	0.153557	10.0.1.2	10.0.1.1	TCP	1514	17437	1	42732 → 5001 [PSH, ACK] Seq=17437 Ack=1 Win=22016 Len=...
20	0.153576	10.0.1.2	10.0.1.1	TCP	1514	18885	1	42732 → 5001 [ACK] Seq=18885 Ack=1 Win=22016 Len=1448...
21	0.153577	10.0.1.2	10.0.1.1	TCP	1514	20333	1	42732 → 5001 [PSH, ACK] Seq=20333 Ack=1 Win=22016 Len=...
22	0.166571	10.0.1.1	10.0.1.2	TCP	66	29	5853	5001 → 42732 [ACK] Seq=29 Ack=5853 Win=20992 Len=0 TSv...
23	0.166622	10.0.1.2	10.0.1.1	TCP	1514	21781	1	42732 → 5001 [ACK] Seq=21781 Ack=1 Win=22016 Len=1448...
24	0.166630	10.0.1.2	10.0.1.1	TCP	1514	23229	1	42732 → 5001 [ACK] Seq=23229 Ack=1 Win=22016 Len=1448...
25	0.166632	10.0.1.2	10.0.1.1	TCP	1514	24677	1	42732 → 5001 [PSH, ACK] Seq=24677 Ack=1 Win=22016 Len=...
26	0.173766	10.0.1.1	10.0.1.2	TCP	66	29	8749	5001 → 42732 [ACK] Seq=29 Ack=8749 Win=20992 Len=0 TSv...
27	0.173815	10.0.1.2	10.0.1.1	TCP	1514	26125	1	42732 → 5001 [ACK] Seq=26125 Ack=1 Win=22016 Len=1448...

[Calculated window size: 21900]

0030 55 8c 5a 1a 00 00 02 04 05 b4 04 02 08 0a 45 b0 U-Z.....

The scaled window size (if scaling has been used) (tcp.window_size), 2 bytes

Packets: 1313 · Displayed: 1313 (100.0%)

Profile: Default

a TCP connection

server+client sequence numbers
advance by 1 to indicate where in setup

The main window shows a list of packets with the following columns: No., Time, Source, Destination, Protocol, Length, Seq#, Ack#, and Info. The first three packets are highlighted with a red box:

No.	Time	Source	Destination	Protocol	Length	Seq#	Ack#	Info
1	0.000000	10.0.1.2	10.0.1.1	TCP	74	0	0	42732 → 5001 [SYN, Seq=0 Win=21900 Len=0 MSS=1460 SACK...
2	0.013678	10.0.1.1	10.0.1.2	TCP	74	0	1	5001 → 42732 [SYN, ACK] Seq=0 Ack=1 Win=21720 Len=0 MSS...
3	0.013731	10.0.1.2	10.0.1.1	TCP	66	1	1	42732 → 5001 [ACK] Seq=1 Ack=1 Win=22016 Len=0 TSval=1...

The inset window shows a zoomed-in view of the first three packets with the following columns: Length, Seq#, Ack#, and Info. The first three packets are highlighted with a red box:

Length	Seq#	Ack#	Info
74	0	0	42732 → 5001 [SYN, Seq=0 Win=21900 Len=0 MSS=1460 SACK...
74	0	1	5001 → 42732 [SYN, ACK] Seq=0 Ack=1 Win=21720 Len=0 MSS...
66	1	1	42732 → 5001 [ACK] Seq=1 Ack=1 Win=22016 Len=0 TSval=1...

connection setup, no data transferred

[Calculated window size: 21900]

0030 55 8c 5a 1a 00 00 02 04 05 b4 04 02 08 0a 45 b0 U-Z.....

The scaled window size (if scaling has been used) (tcp.window_size), 2 bytes

Packets: 1313 · Displayed: 1313 (100.0%) Profile: Default

a TCP connection

connection is bidirectional
from now, using olive color to show 'backwards' packets

No.	Time	Source	Destination	Protocol	Length	Seq#	Ack#	Info
1	0.000000	10.0.1.2	10.0.1.1	TCP	74	0	0	42732 → 5001 [SYN] Seq=0 Win=21900 Len=0 MSS=1460 SACK
2	0.013678	10.0.1.1	10.0.1.2	TCP	74	0	1	5001 → 42732 [SYN, ACK] Seq=0 Ack=1 Win=21720 Len=0 MSS=
3	0.013731	10.0.1.2	10.0.1.1	TCP	66	1	1	42732 → 5001 [ACK] Seq=1 Ack=1 Win=22016 Len=0 TSval=1
4	0.014448	10.0.1.2	10.0.1.1	TCP	126	1	1	42732 → 5001 [PSH, ACK] Seq=1 Ack=1 Win=22016 Len=60 TSv
5	0.014487	10.0.1.2	10.0.1.1	TCP	1514	61	1	42732 → 5001 [ACK] Seq=61 Ack=1 Win=22016 Len=1448 TSv
6								
7								
8								
9								
10								
11								
12								
13								
14	0.045412	10.0.1.2	10.0.1.1	TCP	1514	13093	1	42732 → 5001 [ACK] Seq=13093 Ack=1 Win=22016 Len=1448
15	0.132429	10.0.1.1	10.0.1.2	TCP	66	1	61	5001 → 42732 [ACK] Seq=1 Ack=61 Win=22016 Len=0 TSval=1
16	0.132462	10.0.1.2	10.0.1.1	TCP	1514	14541	1	42732 → 5001 [ACK] Seq=14541 Ack=1 Win=22016 Len=1448
17	0.153499	10.0.1.1	10.0.1.2	TCP	66	29	2957	[TCP Previous segment not captured] 5001 → 42732 [ACK]
18	0.153549	10.0.1.2	10.0.1.1	TCP	1514	15989	1	42732 → 5001 [ACK] Seq=15989 Ack=1 Win=22016 Len=1448
19	0.153557	10.0.1.2	10.0.1.1	TCP	1514	17437	1	42732 → 5001 [PSH, ACK] Seq=17437 Ack=1 Win=22016 Len=
20	0.153576	10.0.1.2	10.0.1.1	TCP	1514	18885	1	42732 → 5001 [ACK] Seq=18885 Ack=1 Win=22016 Len=1448
21	0.153577	10.0.1.2	10.0.1.1	TCP	1514	20333	1	42732 → 5001 [PSH, ACK] Seq=20333 Ack=1 Win=22016 Len=
22	0.166571	10.0.1.1	10.0.1.2	TCP	66	29	5853	5001 → 42732 [ACK] Seq=29 Ack=5853 Win=20992 Len=0 TSv
23	0.166622	10.0.1.2	10.0.1.1	TCP	1514	21781	1	42732 → 5001 [ACK] Seq=21781 Ack=1 Win=22016 Len=1448
24	0.166630	10.0.1.2	10.0.1.1	TCP	1514	23229	1	42732 → 5001 [ACK] Seq=23229 Ack=1 Win=22016 Len=1448
25	0.166632	10.0.1.2	10.0.1.1	TCP	1514	24677	1	42732 → 5001 [PSH, ACK] Seq=24677 Ack=1 Win=22016 Len=
26	0.173766	10.0.1.1	10.0.1.2	TCP	66	29	8749	5001 → 42732 [ACK] Seq=29 Ack=8749 Win=20992 Len=0 TSv
27	0.173815	10.0.1.2	10.0.1.1	TCP	1514	26125	1	42732 → 5001 [ACK] Seq=26125 Ack=1 Win=22016 Len=1448
28	0.173828	10.0.1.2	10.0.1.1	TCP	1514	27573	1	42732 → 5001 [PSH, ACK] Seq=27573 Ack=1 Win=22016 Len=

[Calculated window size: 21900]

0030 55 8c 5a 1a 00 00 02 04 05 b4 04 02 08 0a 45 b0 U·Z·····

The scaled window size (if scaling has been used) (tcp.window_size), 2 bytes

Packets: 1313 · Displayed: 1313 (100.0%)

Profile: Default

a TCP connection

The image shows a Wireshark capture of a TCP connection. The interface includes a menu bar (File, Edit, View, Go, Capture, Analyze, Statistics, Telephony, Wireless, Tools, Help), a toolbar, and a packet list pane. The packet list pane shows the following data:

No.	Time	Source	Destination	Protocol	Length	Seq#	Ack#	Info
1	0.000000	10.0.1.2	10.0.1.1	TCP	74	0	0	42732 → 5001 [ACK] Seq=0 Win=21900 Len=0 MSS=1460 SACK
2	0.013678	10.0.1.2	10.0.1.1	TCP	60	1	0	5001 → 42732 [ACK] Seq=0 Ack=1 Win=21720 Len=0 MSS=1460
3	0.013731	10.0.1.2	10.0.1.1	TCP	60	1	1	42732 → 5001 [ACK] Seq=1 Ack=1 Win=22016 Len=0 TSval=1
4	0.014448	10.0.1.2	10.0.1.1	TCP	60	1	1	42732 → 5001 [PSH, ACK] Seq=1 Ack=1 Win=22016 Len=60 TSv
5	0.014487	10.0.1.2	10.0.1.1	TCP	60	1	1	42732 → 5001 [ACK] Seq=61 Ack=1 Win=22016 Len=1448 TSv
6	0.014489	10.0.1.2	10.0.1.1	TCP	60	1509	1	42732 → 5001 [PSH, ACK] Seq=1509 Ack=1 Win=22016 Len=1
7	0.014499	10.0.1.2	10.0.1.1	TCP	60	2957	1	42732 → 5001 [ACK] Seq=2957 Ack=1 Win=22016 Len=1448 TSv
8	0.014500	10.0.1.2	10.0.1.1	TCP	60	4405	1	42732 → 5001 [PSH, ACK] Seq=4405 Ack=1 Win=22016 Len=1
9	0.014507	10.0.1.2	10.0.1.1	TCP	1514	5853	1	42732 → 5001 [ACK] Seq=5853 Ack=1 Win=22016 Len=1448 TSv
10	0.014508	10.0.1.2	10.0.1.1	TCP	1514	7301	1	42732 → 5001 [PSH, ACK] Seq=7301 Ack=1 Win=22016 Len=1
11	0.014514	10.0.1.2	10.0.1.1	TCP	1514	8749	1	42732 → 5001 [ACK] Seq=8749 Ack=1 Win=22016 Len=1448 TSv
12	0.014515	10.0.1.2	10.0.1.1	TCP	1514	10197	1	42732 → 5001 [PSH, ACK] Seq=10197 Ack=1 Win=22016 Len=1
13	0.014521	10.0.1.2	10.0.1.1	TCP	1514	11645	1	42732 → 5001 [PSH, ACK] Seq=11645 Ack=1 Win=22016 Len=1
14	0.045412	10.0.1.2	10.0.1.1	TCP	1514	13093	1	42732 → 5001 [ACK] Seq=13093 Ack=1 Win=22016 Len=1448 TSv
15	0.132429	10.0.1.1	10.0.1.2	TCP	60	1	61	5001 → 42732 [ACK] Seq=1 Ack=61 Win=22016 Len=0 TSval=1
16	0.132462	10.0.1.1	10.0.1.2	TCP	60	14541	1	42732 → 5001 [ACK] Seq=14541 Ack=1 Win=22016 Len=1448 TSv
17	0.153499	10.0.1.1	10.0.1.2	TCP	66	29	2957	[TCP Previous segment not captured] 5001 → 42732 [ACK]
18	0.153549	10.0.1.1	10.0.1.2	TCP	66	15989	1	42732 → 5001 [ACK] Seq=15989 Ack=1 Win=22016 Len=1448 TSv
19	0.153557	10.0.1.2	10.0.1.1	TCP	1514	17437	1	42732 → 5001 [PSH, ACK] Seq=17437 Ack=1 Win=22016 Len=1
20	0.153576	10.0.1.2	10.0.1.1	TCP	1514	18885	1	42732 → 5001 [ACK] Seq=18885 Ack=1 Win=22016 Len=1448 TSv
21	0.153577	10.0.1.2	10.0.1.1	TCP	1514	20333	1	42732 → 5001 [PSH, ACK] Seq=20333 Ack=1 Win=22016 Len=1
22	0.166571	10.0.1.1	10.0.1.2	TCP	66	29	5853	5001 → 42732 [ACK] Seq=2957 Ack=5853 Win=22016 Len=0 TSv
23	0.166622	10.0.1.2	10.0.1.1	TCP	1514	21781	1	42732 → 5001 [ACK] Seq=21781 Ack=1 Win=22016 Len=1448 TSv
24	0.166630	10.0.1.2	10.0.1.1	TCP	1514	23229	1	42732 → 5001 [ACK] Seq=23229 Ack=1 Win=22016 Len=1448 TSv
25	0.166632	10.0.1.2	10.0.1.1	TCP	1514	24677	1	42732 → 5001 [PSH, ACK] Seq=24677 Ack=1 Win=22016 Len=1
26	0.173766	10.0.1.1	10.0.1.2	TCP	66	29	8749	5001 → 42732 [ACK] Seq=2957 Ack=8749 Win=22016 Len=0 TSv
27	0.173815	10.0.1.2	10.0.1.1	TCP	1514	26125	1	42732 → 5001 [ACK] Seq=26125 Ack=1 Win=22016 Len=1448 TSv

Annotations in the image:

- A red box highlights the sequence number **1** in the first packet's info field.
- A red box highlights the sequence number **61** in the 15th packet's info field.
- A red box highlights the sequence number **2957** in the 7th packet's info field.
- A red box highlights the sequence number **61** in the 15th packet's info field.
- Red text annotations: "data packet with client bytes 1-60" (pointing to packet 4) and "acknowledgement of client bytes up to 60" (pointing to packet 15).

At the bottom, the status bar shows: [Calculated window size: 21900], 0030 55 8c 5a 1a 00 00 02 04 05 b4 04 02 08 0a 45 b0 U:Z... Profile: Default

a TCP connection

File Edit View Go Capture Analyze Statistics Telephony Wireless Tools Help

ip.src = 10.0.1.1

No.	Time	Source	Destination	Protocol	Length	Seq#	Ack#	Info
1	0.000000	10.0.1.2	10.0.1.1	TCP	74	0	0	42732 → 5001 [SYN] Seq=0 Win=21900 Len=0 MSS=1460 SACK
2	0.013678	10.0.1.1	10.0.1.2	TCP	74	1	1	5001 → 42732 [ACK] Seq=1 Ack=1 Win=21720 Len=0 MSS=1460 SACK
3	0.013731	10.0.1.1	10.0.1.2	TCP	74	1	1	5001 → 42732 [ACK] Seq=1 Ack=1 Win=22016 Len=0 MSS=1460 SACK
4	0.014448	10.0.1.1	10.0.1.2	TCP	74	1	1	5001 → 42732 [ACK] Seq=1 Ack=1 Win=22016 Len=60 MSS=1460 SACK
5	0.014487	10.0.1.1	10.0.1.2	TCP	74	1	1	5001 → 42732 [ACK] Seq=1 Ack=1 Win=22016 Len=1448 MSS=1460 SACK
6	0.014489	10.0.1.1	10.0.1.2	TCP	74	1	1	5001 → 42732 [ACK] Seq=1 Ack=1 Win=22016 Len=1448 MSS=1460 SACK
7	0.014499	10.0.1.1	10.0.1.2	TCP	74	1	1	5001 → 42732 [ACK] Seq=1 Ack=1 Win=22016 Len=1448 MSS=1460 SACK
8	0.014500	10.0.1.1	10.0.1.2	TCP	74	1	1	5001 → 42732 [ACK] Seq=1 Ack=1 Win=22016 Len=1448 MSS=1460 SACK
9	0.014507	10.0.1.1	10.0.1.2	TCP	74	1	1	5001 → 42732 [ACK] Seq=1 Ack=1 Win=22016 Len=1448 MSS=1460 SACK
10	0.014508	10.0.1.1	10.0.1.2	TCP	74	1	1	5001 → 42732 [ACK] Seq=1 Ack=1 Win=22016 Len=1448 MSS=1460 SACK
11	0.014514	10.0.1.1	10.0.1.2	TCP	74	1	1	5001 → 42732 [ACK] Seq=1 Ack=1 Win=22016 Len=1448 MSS=1460 SACK
12	0.014515	10.0.1.2	10.0.1.1	TCP	74	10197	1	42732 → 5001 [PSH, ACK] Seq=10197 Ack=1 Win=22016 Len=0 MSS=1460 SACK
13	0.014521	10.0.1.2	10.0.1.1	TCP	74	11645	1	42732 → 5001 [PSH, ACK] Seq=11645 Ack=1 Win=22016 Len=0 MSS=1460 SACK
14	0.045412	10.0.1.2	10.0.1.1	TCP	74	11645	1	42732 → 5001 [PSH, ACK] Seq=11645 Ack=1 Win=22016 Len=0 MSS=1460 SACK
15	0.132429	10.0.1.1	10.0.1.2	TCP	74	1	1	5001 → 42732 [ACK] Seq=1 Ack=61 Win=22016 Len=0 MSS=1460 SACK
16	0.132462	10.0.1.2	10.0.1.1	TCP	74	14541	1	42732 → 5001 [ACK] Seq=14541 Ack=1 Win=22016 Len=0 MSS=1460 SACK
17	0.153499	10.0.1.1	10.0.1.2	TCP	74	1	1	5001 → 42732 [ACK] Seq=1 Ack=2957 Win=22016 Len=0 MSS=1460 SACK
18	0.153549	10.0.1.2	10.0.1.1	TCP	74	2957	1	42732 → 5001 [TCP Previous segment not captured] Seq=2957 Ack=1 Win=22016 Len=0 MSS=1460 SACK
19	0.153557	10.0.1.2	10.0.1.1	TCP	74	15000	1	42732 → 5001 [ACK] Seq=15000 Ack=1 Win=22016 Len=0 MSS=1460 SACK
20	0.153576	10.0.1.2	10.0.1.1	TCP	74	17437	1	42732 → 5001 [PSH, ACK] Seq=17437 Ack=1 Win=22016 Len=1448 MSS=1460 SACK
21	0.153577	10.0.1.2	10.0.1.1	TCP	74	18885	1	42732 → 5001 [ACK] Seq=18885 Ack=1 Win=22016 Len=1448 MSS=1460 SACK
22	0.166571	10.0.1.1	10.0.1.2	TCP	66	29	5853	5001 → 42732 [ACK] Seq=29 Ack=5853 Win=20992 Len=0 MSS=1460 SACK
23	0.166622	10.0.1.2	10.0.1.1	TCP	74	21781	1	42732 → 5001 [ACK] Seq=21781 Ack=1 Win=22016 Len=1448 MSS=1460 SACK
24	0.166630	10.0.1.2	10.0.1.1	TCP	74	23229	1	42732 → 5001 [ACK] Seq=23229 Ack=1 Win=22016 Len=1448 MSS=1460 SACK
25	0.166632	10.0.1.2	10.0.1.1	TCP	74	24677	1	42732 → 5001 [PSH, ACK] Seq=24677 Ack=1 Win=22016 Len=1448 MSS=1460 SACK
26	0.173766	10.0.1.1	10.0.1.2	TCP	66	29	8749	5001 → 42732 [ACK] Seq=29 Ack=8749 Win=20992 Len=0 MSS=1460 SACK
27	0.173815	10.0.1.2	10.0.1.1	TCP	74	26125	1	42732 → 5001 [ACK] Seq=26125 Ack=1 Win=22016 Len=1448 MSS=1460 SACK

[Calculated window size: 21900]

0030 55 8c 5a 1a 00 00 02 04 05 b4 04 02 08 0a 45 b0 U·Z·····

The scaled window size (if scaling has been used) (tcp.window_size), 2 bytes

Packets: 1313 · Displayed: 1313 (100.0%)

Profile: Default

a TCP connection

The image shows a Wireshark capture of a TCP connection. The main pane displays a list of network packets. A red text overlay reads "scrolling down reveals retransmission later". A black box highlights a specific packet (No. 54) with the following details: "1 23229 [TCP Retransmission] 5001 -> 42732 [PSH, ACK] Seq=53637". Another black box highlights packet 55: "29 42732 -> 5001 [ACK] Seq=56533 Ack=29 Win=22016 Len=1448". A third black box highlights packet 63: "29 [TCP Retransmission] 42732 -> 5001 [ACK] Seq=27573 Ack=63 Win=22016 Len=1448". A text overlay reads "wireshark knows it's retransmission because sequence number sent by server went backwards". The status bar at the bottom shows "Packets: 1313 · Displayed: 1313 (100.0%) Profile: Default".

No.	Time	Source	Destination	Protocol	Length	Seq#	Ack#	Info
40	0.313333	10.0.1.2	10.0.1.1	TCP	1514	39157	1 42732	→ 5001 [ACK] Seq=39157 Ack=1 Win=22016 Len=1448
41	0.313338	10.0.1.2	10.0.1.1	TCP	1514	40605	1 42732	→ 5001 [PSH, ACK] Seq=40605 Ack=1 Win=22016 Len=1448
42	0.313379	10.0.1.2	10.0.1.1	TCP	1514	42053	1 42732	→ 5001 [ACK] Seq=42053 Ack=1 Win=22016 Len=1448
43	0.313382	10.0.1.2	10.0.1.1	TCP	1514	43501	1 42732	→ 5001 [ACK] Seq=43501 Ack=1 Win=22016 Len=1448
44	0.313388	10.0.1.2	10.0.1.1	TCP	1514	44949	1 42732	→ 5001 [PSH, ACK] Seq=44949 Ack=1 Win=22016 Len=1448
45	0.313402	10.0.1.2	10.0.1.1	TCP	1514	46397	1 42732	→ 5001 [ACK] Seq=46397 Ack=1 Win=22016 Len=1448
46	0.313403	10.0.1.2	10.0.1.1	TCP	1514	47845	1 42732	→ 5001 [ACK] Seq=47845 Ack=1 Win=22016 Len=1448
47	0.325448	10.0.1.2	10.0.1.1	TCP	1514	23229	1 42732	→ 5001 [ACK] Seq=23229 Ack=1 Win=22016 Len=1448
48	0.325513	10.0.1.2	10.0.1.1	TCP	1514	33229	1 42732	→ 5001 [ACK] Seq=33229 Ack=1 Win=22016 Len=1448
49	0.325520	10.0.1.2	10.0.1.1	TCP	1514	50741	1 42732	→ 5001 [ACK] Seq=50741 Ack=1 Win=22016 Len=1448
50	0.325537	10.0.1.2	10.0.1.1	TCP	1514	53637	1 42732	→ 5001 [ACK] Seq=53637 Ack=1 Win=22016 Len=1448
51	0.325538	10.0.1.2	10.0.1.1	TCP	1514	55085	1 42732	→ 5001 [PSH, ACK] Seq=55085 Ack=1 Win=22016 Len=1448
52	0.325540	10.0.1.2	10.0.1.1	TCP	1514	1	23229	[TCP Retransmission] 5001 -> 42732 [PSH, ACK] Seq=53637
53	0.333364	10.0.1.1	10.0.1.2	TCP	1514	29	42732	-> 5001 [ACK] Seq=56533 Ack=29 Win=22016 Len=1448
54	0.333403	10.0.1.2	10.0.1.1	TCP	1514	1	23229	[TCP Retransmission] 5001 -> 42732 [PSH, ACK] Seq=53637
55	0.343063	10.0.1.1	10.0.1.2	TCP	1514	56533	29 42732	-> 5001 [ACK] Seq=56533 Ack=29 Win=22016 Len=1448
56	0.343108	10.0.1.1	10.0.1.2	TCP	1514	6533	29 42732	-> 5001 [ACK] Seq=6533 Ack=29 Win=22016 Len=1448
57	0.343115	10.0.1.1	10.0.1.2	TCP	1514	57981	29 42732	-> 5001 [ACK] Seq=57981 Ack=29 Win=22016 Len=1448
58	0.343124	10.0.1.1	10.0.1.2	TCP	1514	9429	29 42732	-> 5001 [ACK] Seq=9429 Ack=29 Win=22016 Len=1448
59	0.343125	10.0.1.1	10.0.1.2	TCP	1514	60877	29 42732	-> 5001 [ACK] Seq=60877 Ack=29 Win=22016 Len=1448
60	0.343132	10.0.1.1	10.0.1.2	TCP	1514	2325	29 42732	-> 5001 [ACK] Seq=2325 Ack=29 Win=22016 Len=1448
61	0.349314	10.0.1.1	10.0.1.2	TCP	1514	32	29 42732	-> 5001 [ACK] Seq=32 Ack=29 Win=22016 Len=1448
62	0.352884	10.0.1.1	10.0.1.2	TCP	86	29	24677	[TCP Window Update] 5001 -> 42732 [ACK] Seq=29 Ack=24677
63	0.352919	10.0.1.2	10.0.1.1	TCP	1514	27573	29	[TCP Retransmission] 42732 -> 5001 [ACK] Seq=27573 Ack=63 Win=22016 Len=1448
64	0.363404	10.0.1.1	10.0.1.2	TCP	94	29	24677	[TCP Window Update] 5001 -> 42732 [ACK] Seq=29 Ack=24677
65	0.363445	10.0.1.2	10.0.1.1	TCP	1514	30469	29	[TCP Retransmission] 42732 -> 5001 [ACK] Seq=30469 Ack=65 Win=22016 Len=1448
66	0.472622	10.0.1.1	10.0.1.2	TCP	94	29	24677	[TCP Window Update] 5001 -> 42732 [ACK] Seq=29 Ack=24677
67	0.483298	10.0.1.1	10.0.1.2	TCP	94	29	24677	[TCP Window Update] 5001 -> 42732 [ACK] Seq=29 Ack=24677

first data packet

▶ Frame 4: 126 bytes on wire (1008 bits), 126 bytes captured (1008 bits)	0000	08 0
▶ Ethernet II, Src: 08:00:00:00:01:02 (08:00:00:00:01:02), Dst: 08:00:00:00:01:01 (08:00:00:00:01:01)	0010	00 7
▶ Internet Protocol Version 4, Src: 10.0.1.2, Dst: 10.0.1.1	0020	01 0
▼ Transmission Control Protocol, Src Port: 42732, Dst Port: 5001, Seq: 1, Ack: 1, Len: 60	0030	00 2
Source Port: 42732	0040	43 e
Destination Port: 5001	0050	00 0
[Stream index: 0]	0060	00 0
▶ [Conversation completeness: Complete, WITH_DATA (31)]	0070	00 0
[TCP Segment Len: 60]		
Sequence Number: 1 (relative sequence number)		
Sequence Number (raw): 3465579712		
[Next Sequence Number: 61 (relative sequence number)]		
Acknowledgment Number: 1 (relative ack number)		
Acknowledgment number (raw): 3771659014		
1000 = Header Length: 32 bytes (8)		
▶ Flags: 0x018 (PSH, ACK)		
Window: 43		
[Calculated window size: 22016]		
[Window size scaling factor: 512]		
Checksum: 0x4173 [unverified]		
[Checksum Status: Unverified]		
Urgent Pointer: 0		
▼ Options: (12 bytes), No-Operation (NOP), No-Operation (NOP), Timestamps		
▶ TCP Option - No-Operation (NOP)		
▶ TCP Option - No-Operation (NOP)		
▶ TCP Option - Timestamps		
[Timestamps]		

first data packet

- ▶ Frame 4: 126 bytes on wire (1008 bits), 126 bytes captured (1008 bits)
- ▶ Ethernet II, Src: 08:00:00:00:01:02 (08:00:00:00:01:02), Dst: 08:00:00:00:01:01 (08:00:00:00:01:01)
- ▶ Internet Protocol Version 4, Src: 10.0.1.2, Dst: 10.0.1.1
- ▼ Transmission Control Protocol, Src Port: 42732, Dst Port: 5001, Seq: 1, Ack: 1, Len: 60
 - Source Port: 42732
 - Destination Port: 5001
 - [Stream index: 0]
 - ▶ [Conversation complete] not actually part of header
 - [TCP Segment Len: 60] computed using length from lower layer
 - Sequence Number: 1
 - Sequence Number (raw): 3483374712
 - [Next Sequence Number: 61 (relative sequence number)]
 - Acknowledgment Number: 1 (relative ack number)
 - Acknowledgment number (raw): 3771659014
 - 1000 ... = Header Length: 32 bytes (8)
 - ▶ Flags: 0x018 (PSH, ACK)
 - Window: 43
 - [Calculated window size: 22016]
 - [Window size scaling factor: 512]
 - Checksum: 0x4173 [unverified]
 - [Checksum Status: Unverified]
 - Urgent Pointer: 0
 - ▼ Options: (12 bytes), No-Operation (NOP), No-Operation (NOP), Timestamps
 - ▶ TCP Option - No-Operation (NOP)
 - ▶ TCP Option - No-Operation (NOP)
 - ▶ TCP Option - Timestamps

0000	08 0
0010	00 7
0020	01 0
0030	00 2
0040	43 e
0050	00 0
0060	00 0
0070	00 0

first data packet

- ▶ Frame 4: 126 bytes on wire (1008 bits), 126 bytes captured (1008 bits)
- ▶ Ethernet II, Src: 08:00:00:00:01:02 (08:00:00:00:01:02), Dst: 08:00:00:00:01:01 (08:00:00:00:01:01)
- ▶ Internet Protocol Version 4, Src: 10.0.1.2, Dst: 10.0.1.1
- ▼ Transmission Control Protocol, Src Port: 42732, Dst Port: 5001, Seq: 1, Ack: 1, Len: 60
 - Source Port: 42732
 - Destination Port: 5001
 - [Stream index: 0]
 - ▶ [Conversation completeness: Complete, WITH_DATA (31)]
 - [TCP Segment Len: 60]
 - Sequence Number: 1 (relative sequence number)
 - Sequence Number (raw): 3465579712
 - [Next Sequence Number: 61 (relative sequence number)]
 - Acknowledgment number (raw): 3771659814
 - Flags: 0x016 (PSH, ACK)
 - Window: 43
 - [Calculated window size: 22016]
 - [Window size scaling factor: 512]
 - Checksum: 0x4173 [unverified]
 - [Checksum Status: Unverified]
 - Urgent Pointer: 0
 - ▼ Options: (12 bytes), No-Operation (NOP), No-Operation (NOP), Timestamps
 - ▶ TCP Option - No-Operation (NOP)
 - ▶ TCP Option - No-Operation (NOP)
 - ▶ TCP Option - Timestamps

0000	08 0
0010	00 7
0020	01 0
0030	00 2
0040	43 e
0050	00 0
0060	00 0
0070	00 0

sequence numbers in header don't start at 0
wireshark converts to 0-based indices

first data packet

- ▶ Frame 4: 126 bytes on wire (1008 bits), 126 bytes captured (1008 bits)
- ▶ Ethernet II, Src: 08:00:00:00:01:02 (08:00:00:00:01:02), Dst: 08:00:00:00:01:01 (08:00:00:00:01:01)
- ▶ Internet Protocol Version 4, Src: 10.0.1.2, Dst: 10.0.1.1
- ▼ Transmission Control Protocol, Src Port: 42732, Dst Port: 5001, Seq: 1, Ack: 1, Len: 60
 - Source Port: 42732
 - Destination Port: 5001
 - [Stream index: 0]
 - ▶ [Conversation completeness: Complete, WITH_DATA (31)]
 - [TCP Segment Len: 60]
 - Sequence Number: 1 (relative sequence number)
 - Sequence Number (raw): 3465579712
 - [Next Sequence Number: 61 (relative sequence number)]
 - Acknowledgment Number: 1 (relative sequence number)
 - [Offset: 0]
 - [Data Offset: 5 = Header Length: 32 bytes (8)]
 - [Window: 65535]
 - [Checksum: 0x4173 [unverified]]
 - [Checksum Status: Unverified]
 - Urgent Pointer: 0
 - ▼ Options: (12 bytes), No-Operation (NOP), No-Operation (NOP), Timestamps
 - ▶ TCP Option - No-Operation (NOP)
 - ▶ TCP Option - No-Operation (NOP)
 - ▶ TCP Option - Timestamps

0000	08 0
0010	00 7
0020	01 0
0030	00 2
0040	43 e
0050	00 0
0060	00 0
0070	00 0

sequence number is *first* byte being sent

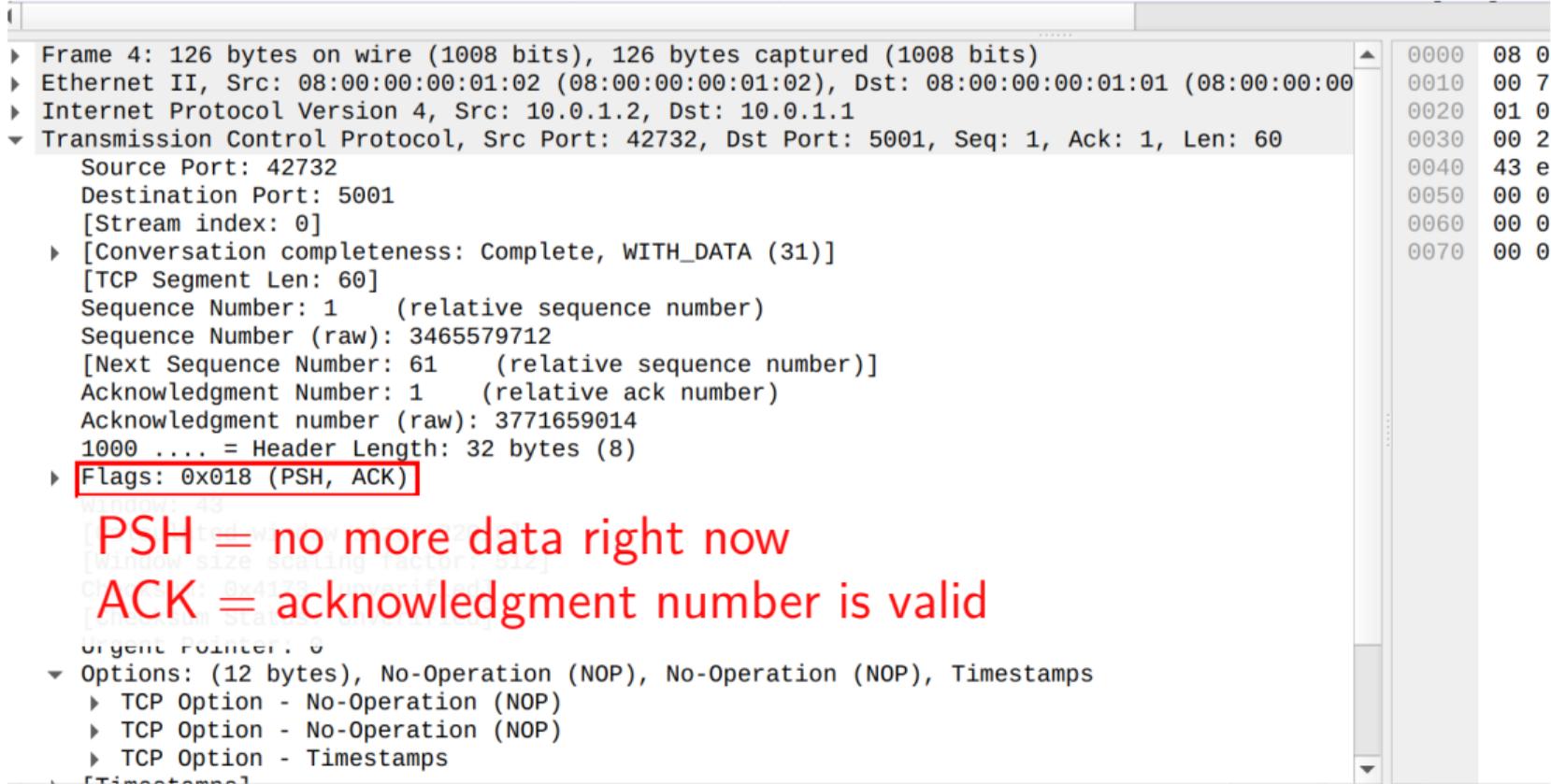
- ▶ need to use segment length to know last byte's number
(= what to ACK if receiving this)

first data packet

- ▶ Frame 4: 126 bytes on wire (1008 bits), 126 bytes captured (1008 bits)
- ▶ Ethernet II, Src: 08:00:00:00:01:02 (08:00:00:00:01:02), Dst: 08:00:00:00:01:01 (08:00:00:00:01:01)
- ▶ Internet Protocol Version 4, Src: 10.0.1.2, Dst: 10.0.1.1
- ▼ Transmission Control Protocol, Src Port: 42732, Dst Port: 5001, Seq: 1, Ack: 1, Len: 60
 - Source Port: 42732
 - Destination Port: 5001
 - [Stream index: 0]
 - ▶ [Conversation completeness: Complete, WITH_DATA (31)]
 - [TCP Segment Len: 60]
 - Sequence Number: 1 (relative sequence number)
 - Sequence Number (raw): 3465579712
 - [Next Sequence Number: 61 (relative sequence number)]
 - Acknowledgment Number: 1 (relative ack number)**
 - Acknowledgment number (raw): 3771659014**
 - ▶ **ack number indicates received start-of-connection stuff and nothing else (in case server sent something)**
 - Checksum: 0x4173 [unverified]
 - [Checksum Status: Unverified]
 - Urgent Pointer: 0
 - ▼ Options: (12 bytes), No-Operation (NOP), No-Operation (NOP), Timestamps
 - ▶ TCP Option - No-Operation (NOP)
 - ▶ TCP Option - No-Operation (NOP)
 - ▶ TCP Option - Timestamps

0000	08 0
0010	00 7
0020	01 0
0030	00 2
0040	43 e
0050	00 0
0060	00 0
0070	00 0

first data packet

- 
- ```
▶ Frame 4: 126 bytes on wire (1008 bits), 126 bytes captured (1008 bits)
▶ Ethernet II, Src: 08:00:00:00:01:02 (08:00:00:00:01:02), Dst: 08:00:00:00:01:01 (08:00:00:00:01:01)
▶ Internet Protocol Version 4, Src: 10.0.1.2, Dst: 10.0.1.1
▼ Transmission Control Protocol, Src Port: 42732, Dst Port: 5001, Seq: 1, Ack: 1, Len: 60
 Source Port: 42732
 Destination Port: 5001
 [Stream index: 0]
 ▶ [Conversation completeness: Complete, WITH_DATA (31)]
 [TCP Segment Len: 60]
 Sequence Number: 1 (relative sequence number)
 Sequence Number (raw): 3465579712
 [Next Sequence Number: 61 (relative sequence number)]
 Acknowledgment Number: 1 (relative ack number)
 Acknowledgment number (raw): 3771659014
 1000 = Header Length: 32 bytes (8)
 ▶ Flags: 0x018 (PSH, ACK)
 Window size scaling factor: 32
 Urgent Pointer: 0
 ▼ Options: (12 bytes), No-Operation (NOP), No-Operation (NOP), Timestamps
 ▶ TCP Option - No-Operation (NOP)
 ▶ TCP Option - No-Operation (NOP)
 ▶ TCP Option - Timestamps
 [Timestamps]
```
- |      |      |
|------|------|
| 0000 | 08 0 |
| 0010 | 00 7 |
| 0020 | 01 0 |
| 0030 | 00 2 |
| 0040 | 43 e |
| 0050 | 00 0 |
| 0060 | 00 0 |
| 0070 | 00 0 |

PSH = no more data right now

ACK = acknowledgment number is valid

# first data packet

- ▶ Frame 4: 126 bytes on wire (1008 bits), 126 bytes captured (1008 bits)
- ▶ Ethernet II, Src: 08:00:00:00:01:02 (08:00:00:00:01:02), Dst: 08:00:00:00:01:01 (08:00:00:00:01:01)
- ▶ Internet Protocol Version 4, Src: 10.0.1.2, Dst: 10.0.1.1
- ▼ Transmission Control Protocol, Src Port: 42732, Dst Port: 5001, Seq: 1, Ack: 1, Len: 60
  - Source Port: 42732
  - Destination Port: 5001
  - [Stream index: 0]
  - ▶ [Conversation completeness: Complete, WITH\_DATA (31)]
  - [TCP Segment Len: 60]
  - Sequence Number: 1 (relative sequence number)
  - Sequence Number (raw): 3485579712
  - Initial Sequence Number (relative sequence number)
  - Acknowledgment Number (relative ack number)
  - Acknowledgment Number (raw): 37758904
  - Window: 43
  - [Calculated window size: 22016]
  - [Window size scaling factor: 512]
  - Checksum: 0x4173 [unverified]
  - [Checksum Status: Unverified]
  - Urgent Pointer: 0
  - ▼ Options: (12 bytes), No-Operation (NOP), No-Operation (NOP), Timestamps
    - ▶ TCP Option - No-Operation (NOP)
    - ▶ TCP Option - No-Operation (NOP)
    - ▶ TCP Option - Timestamps

|      |      |
|------|------|
| 0000 | 08 0 |
| 0010 | 00 7 |
| 0020 | 01 0 |
| 0030 | 00 2 |
| 0040 | 43 e |
| 0050 | 00 0 |
| 0060 | 00 0 |
| 0070 | 00 0 |

window scaling option in use  
(scaling factor only sent in connection setup)

# first data packet

```
▶ Frame 4: 126 bytes on wire (1008 bits), 126 bytes captured (1008 bits)
▶ Ethernet II, Src: 08:00:00:00:01:02 (08:00:00:00:01:02), Dst: 08:00:00:00:01:01 (08:00:00:00:01:01)
▶ Internet Protocol Version 4, Src: 10.0.1.2, Dst: 10.0.1.1
▼ Transmission Control Protocol, Src Port: 42732, Dst Port: 5001, Seq: 1, Ack: 1, Len: 60
 Source Port: 42732
 Destination Port: 5001
 [Stream index: 0]
 ▶ [Conversation completeness: Complete, WITH_DATA (31)]
 [TCP Segment Len: 60]
 Sequence Number: 1 (relative sequence number)
 Sequence Number (raw): 3465579712
 [Next Sequence Number: 61 (relative sequence number)]
 Acknowledgment Number: 1 (relative ack number)
 Acknowledgment number (raw): 3771659014
 1000 = Header Length: 32 bytes (8)
 ▶ Flags: 0x018 (PSH, ACK)
 Window: 43
 [Calculated window size: 22016]
 [Window size scaling factor: 512]
 Options: (12 bytes), No-Operation (NOP), No-Operation (NOP), Timestamps
 ▶ TCP Option - No-Operation (NOP)
 ▶ TCP Option - No-Operation (NOP)
 ▶ TCP Option - Timestamps
```

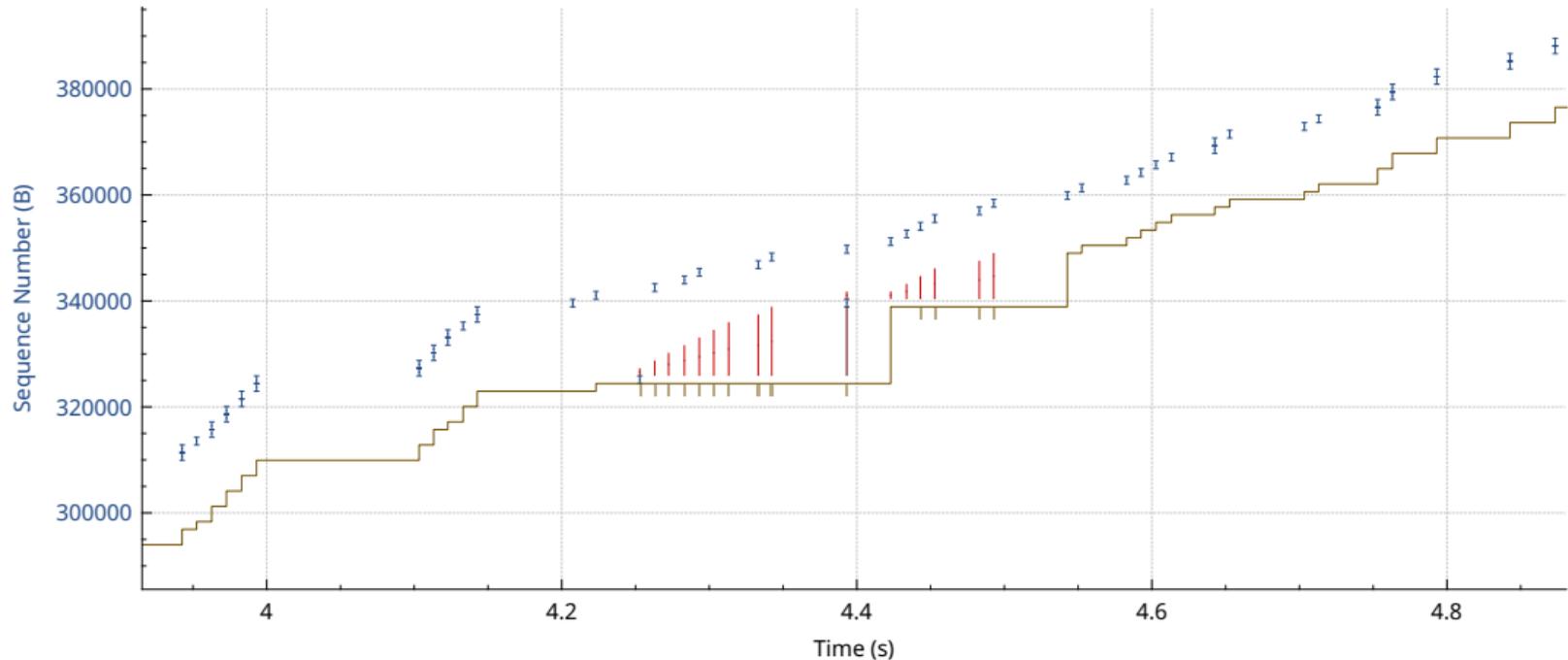
|      |      |
|------|------|
| 0000 | 08 0 |
| 0010 | 00 7 |
| 0020 | 01 0 |
| 0030 | 00 2 |
| 0040 | 43 e |
| 0050 | 00 0 |
| 0060 | 00 0 |
| 0070 | 00 0 |

no-operation options used to make TCP header size multiple of 4

# sequence numbers graph

Sequence Numbers (tcptrace) for 10.0.1.2:42732 → 10.0.1.1:5001

tcp-only-from-2.pcap



# reading thigs graph

bottom line = last ack number

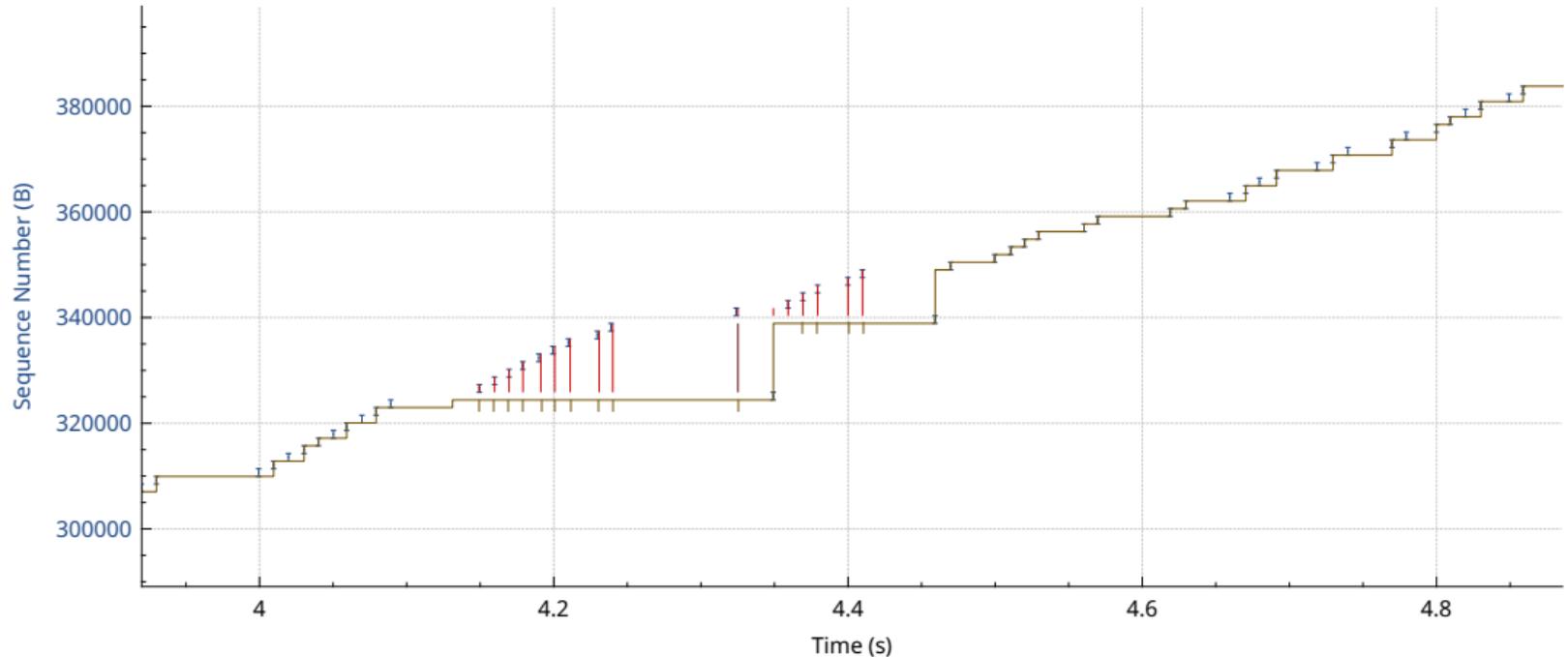
notches on bottom line = duplicate acks

red lines = selective ACK info

# diff. timing in opposite direction

Sequence Numbers (tcptrace) for 10.0.1.2:42732 → 10.0.1.1:5001

tcp-only-from-1.pcap



# backup slides