CS 43: Computer Networks

Transport Layer & Reliable Data Transfer

October 29, 2019

SWARTHMORE COLLEGE

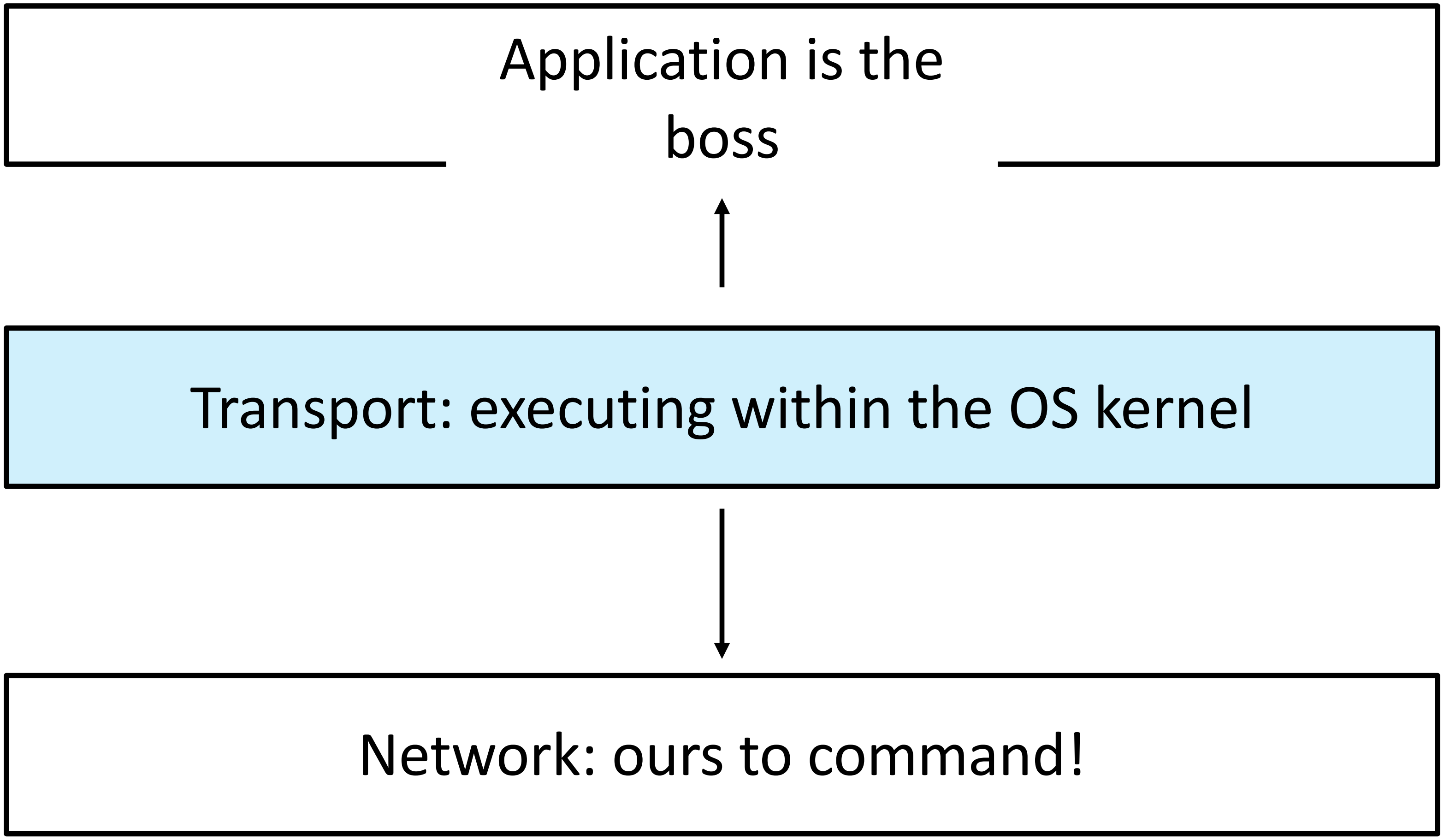
Reading Quiz

Transport Layer

Today

- Principles of reliability
- Automatic Repeat Requests

Transport Layer perspective



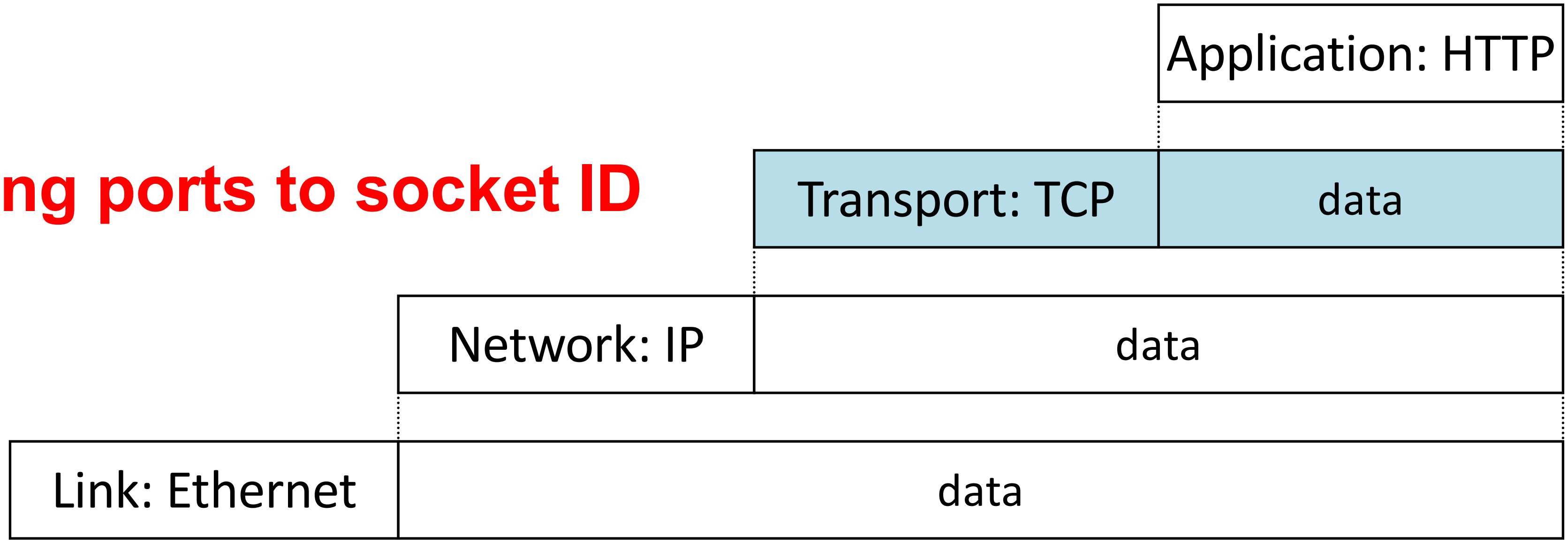
Application is the boss

Transport: executing within the OS kernel

Network: ours to command!

Transport Layer Header

Assigning ports to socket ID



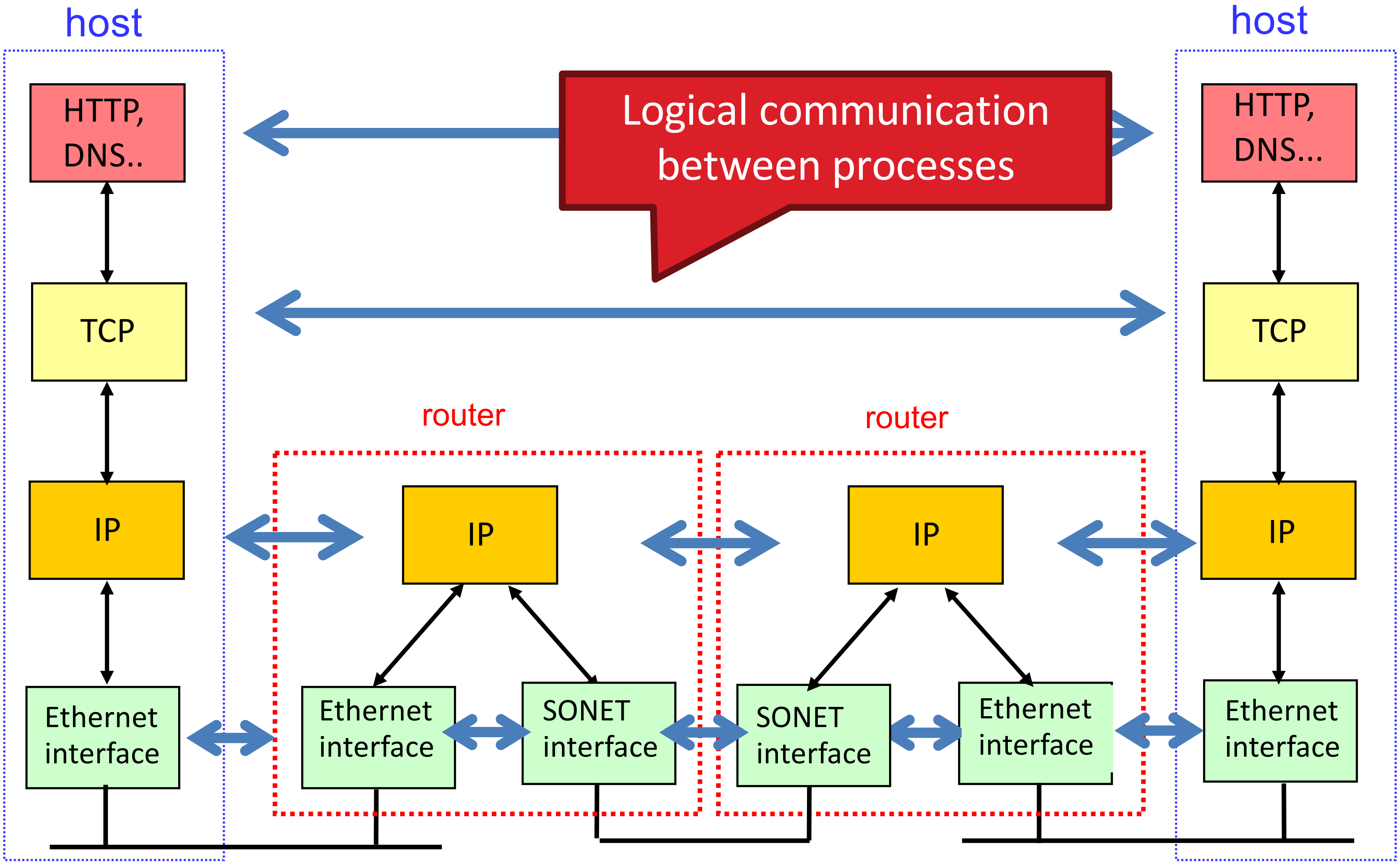
Application: HTTP

Transport: TCP | data

Network: IP | data

Link: Ethernet | data

Transport Layer: Runs on end systems



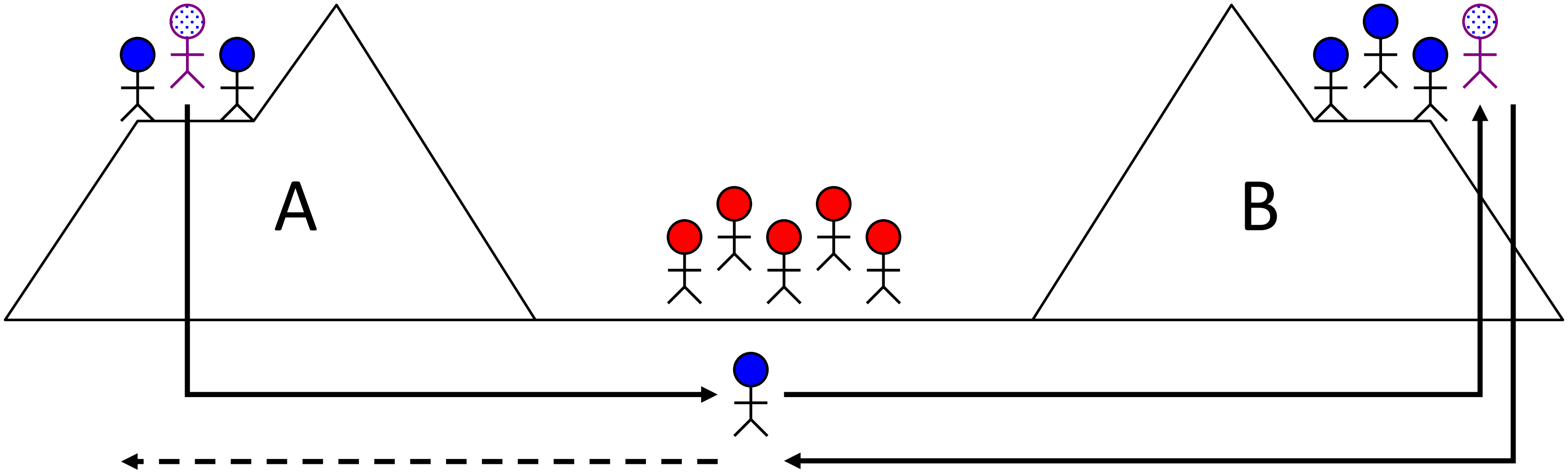
Last Class

- Principles of reliability
  - The Two Generals Problem

Today

- Automatic Repeat Requests
    - Stop and Wait
    - Go-Back-N
    - Selective Repeat

The Two Generals Problem

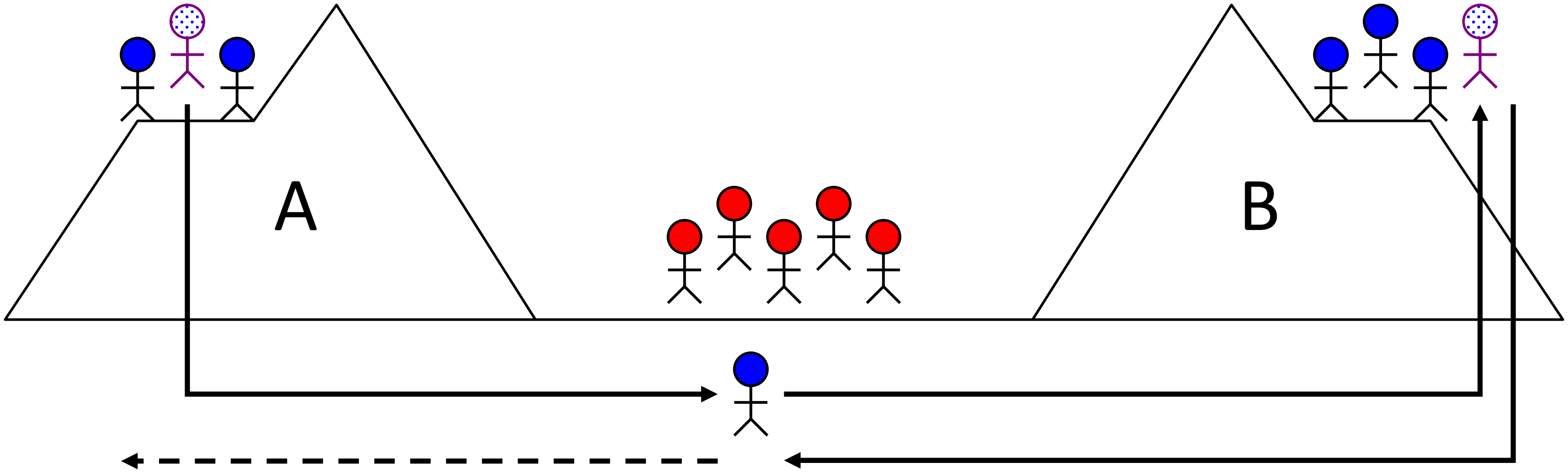


- How to be sure messenger made it?
  - Send acknowledgment: “I delivered message”

In the "two generals problem", can the two armies reliably coordinate their attack? (using what we just discussed)

- A.
- B. No: Can't create perfect channel out of faulty one
  - Can only increase probability of success

The Two Generals Problem



- Result
  - Can’t create perfect channel out of faulty one
  - Can only increase probability of success

Engineering

- Concerns
  - Message corruption
  - Message duplication
  - Message loss
  - Message reordering
  - Performance

- Our toolbox
  - Checksums
  - Timeouts
  - Acks & Nacks
  - Sequence numbering
  - Pipelining

We use these to build Automatic Repeat Request (ARQ) protocols.

Automatic Repeat Request (ARQ)

- Similar to using a cell phone with bad reception.
  - Receiver: Message garbled? Ask to repeat.
  - Sender: Didn’t hear a response? Speak again.

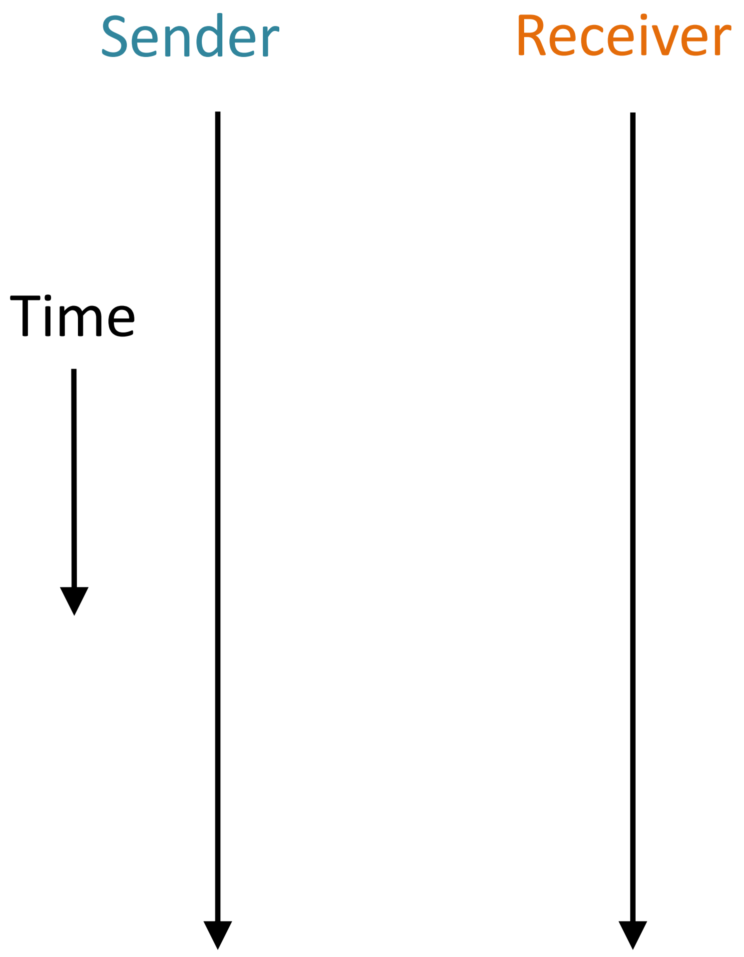
ARQ Broad Classifications

1. Stop-and-wait

Stop and Wait

We have:

- a sender
- a receiver
- time: represented by downwards arrow



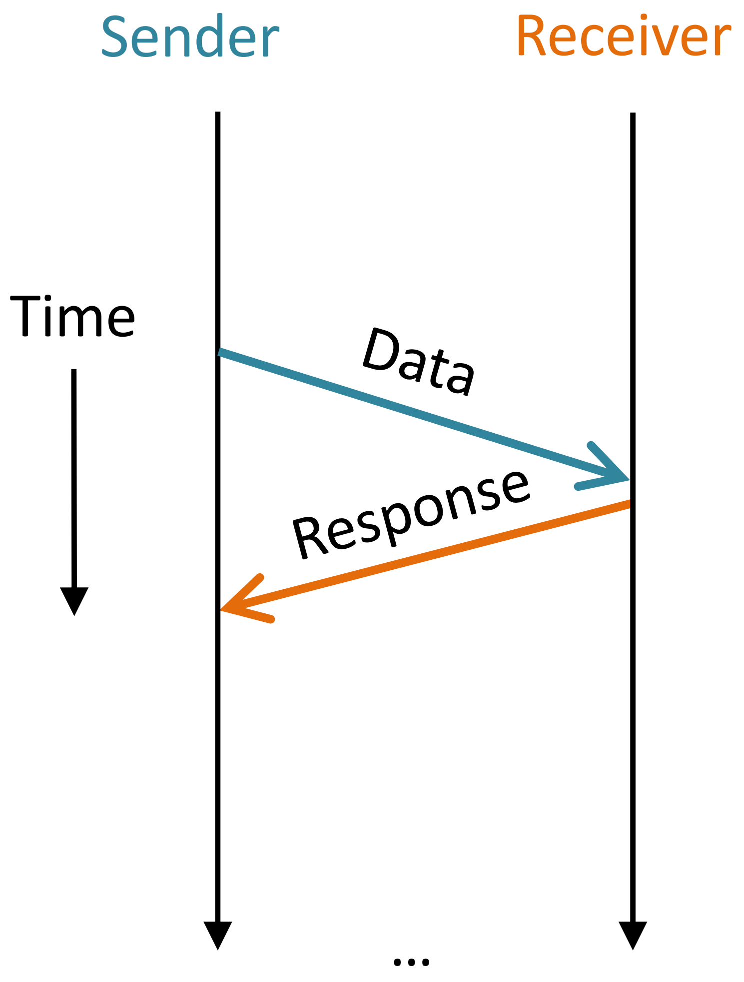
Sender

Receiver

Time

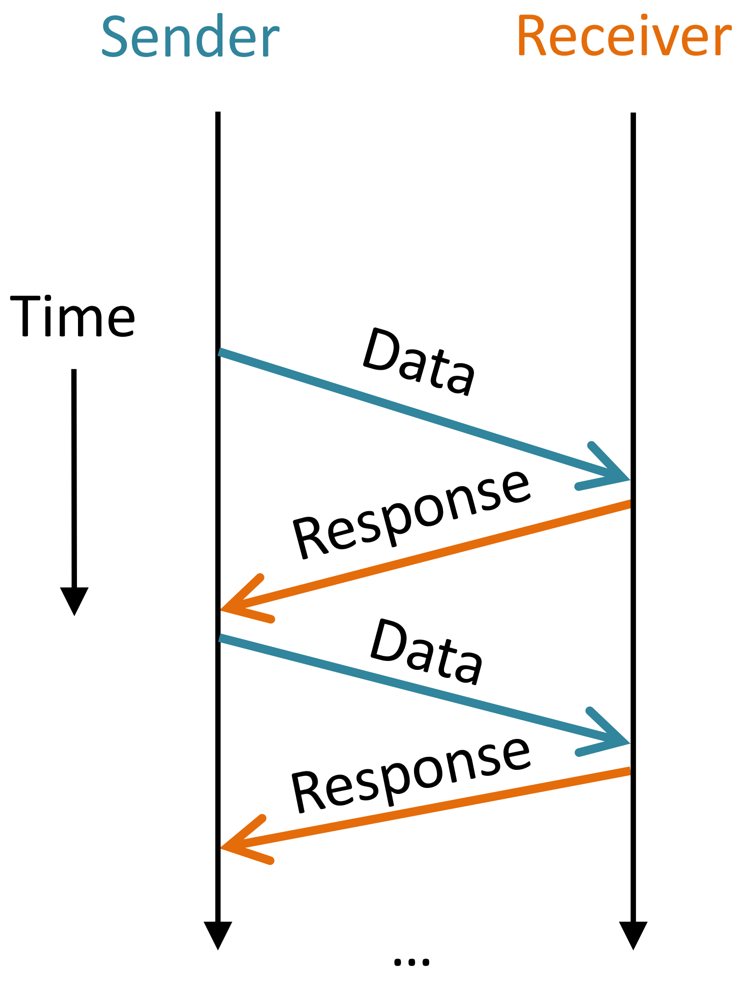
Stop and Wait

Sender sends data and waits till they get the response message from the receiver.



Stop and Wait

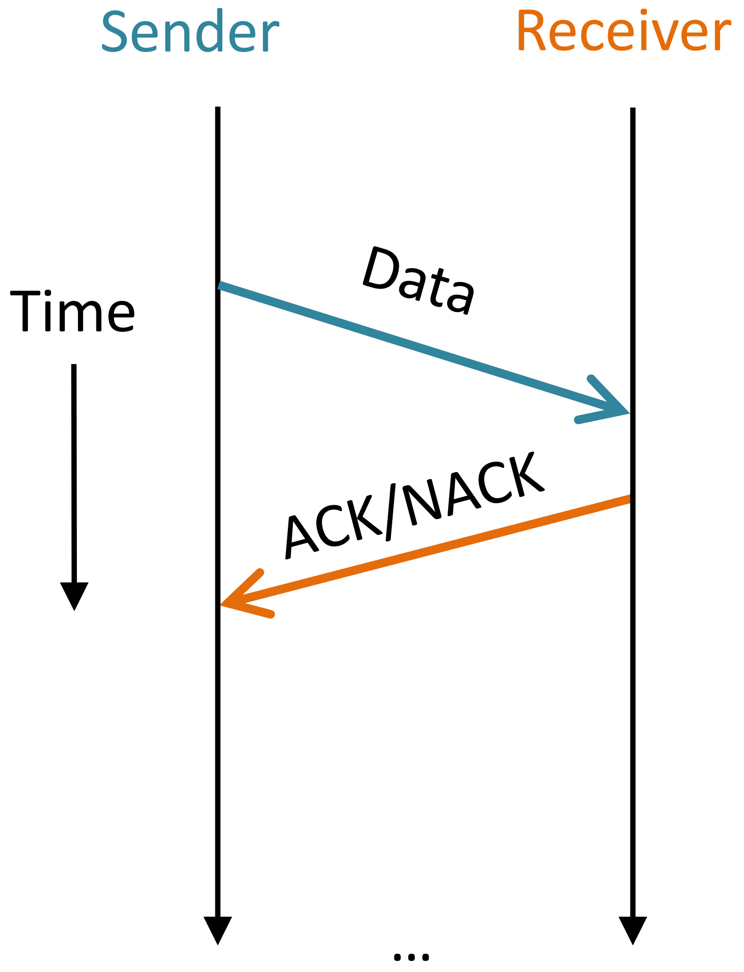
Sender sends data and waits till they get the response message from the receiver.



Buffer data, and don't send till response received

Corruption?

- Error detection mechanism: checksum
  - Data good – receiver sends back ACK
  - Data corrupt – receiver sends back NACK

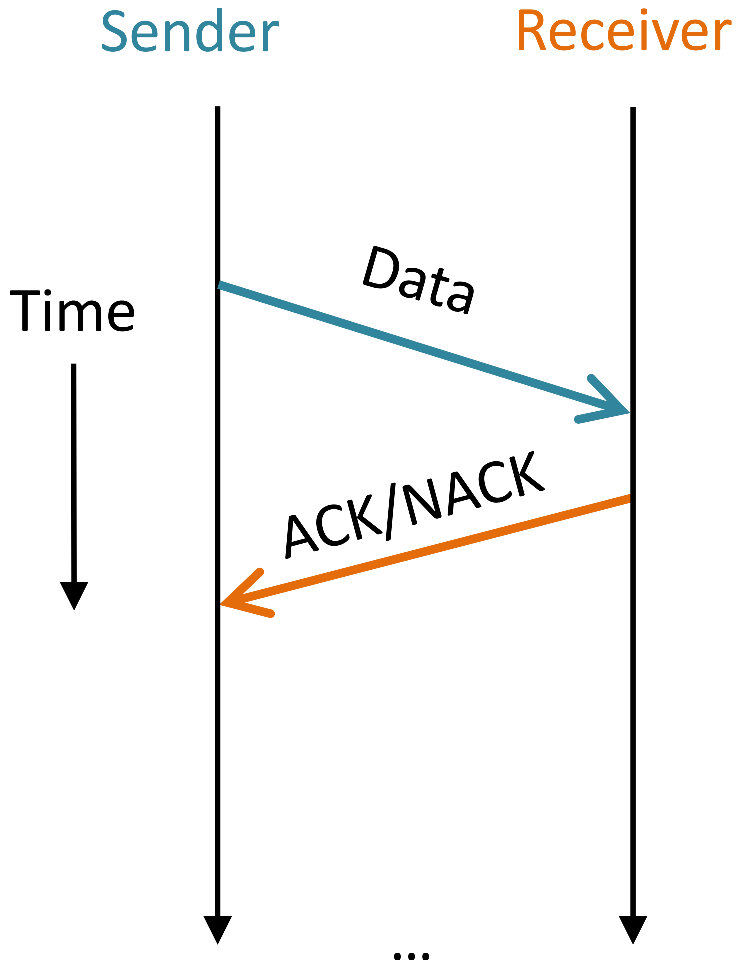


Could we do this with just ACKs or just NACKs?

Error detection mechanism: checksum

- Data good – receiver sends back ACK
- Data corrupt – receiver sends back NACK

A. No, we need them both.
B. Yes, we could do without one of them, but we'd need some other mechanism.
C. Yes, we could get by without one of them.



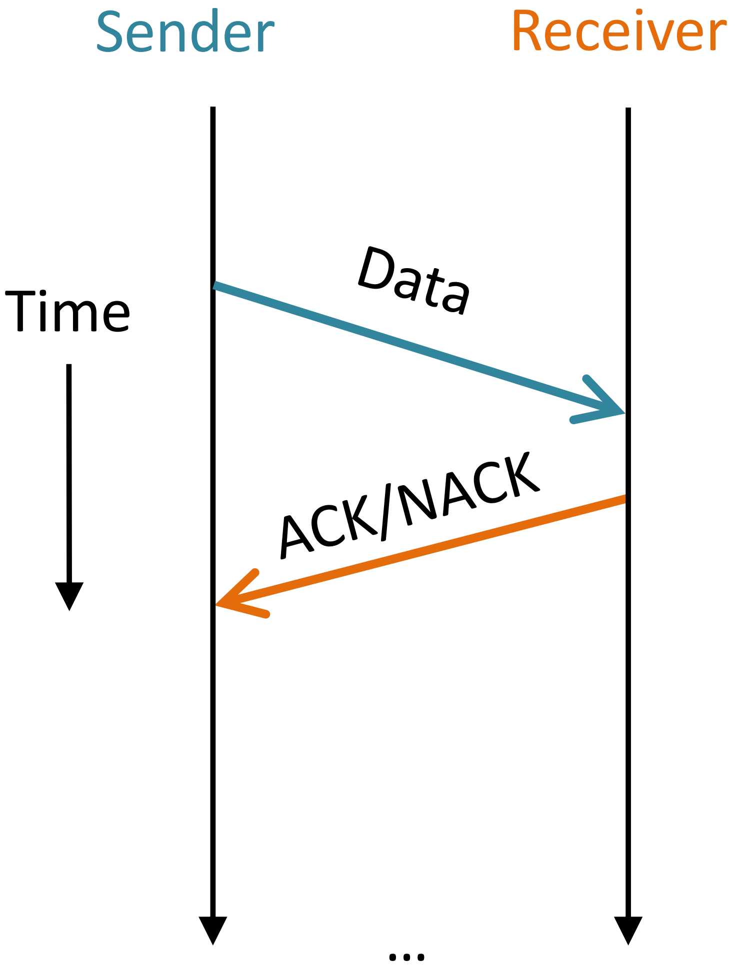
Could we do this with just ACKs or just NACKs?

- With only ACK, we could get by with a timeout.
- With only NACK, we couldn't advance (no good).

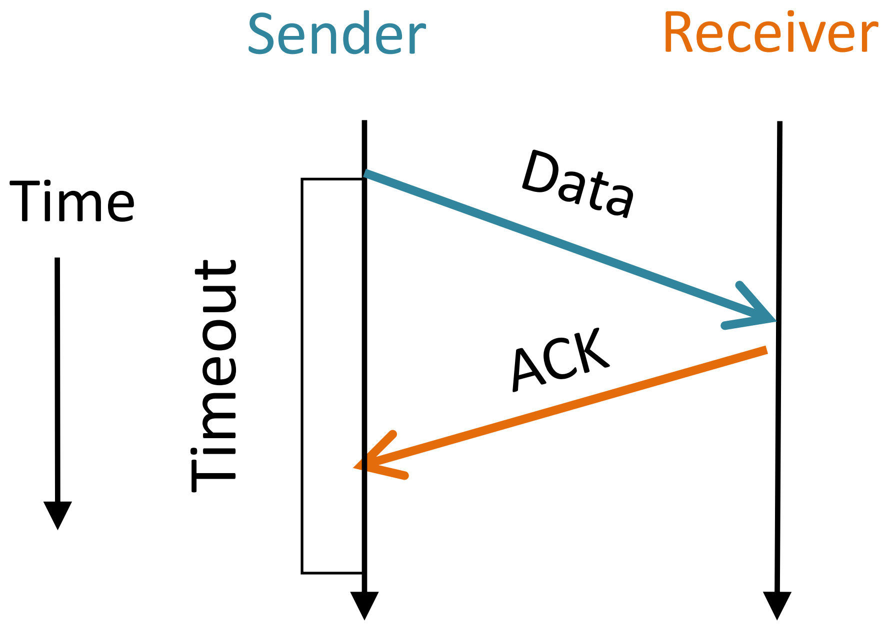
A. No, we need them both.

B. Yes, we could do without one of them, but we'd need some other mechanism.

C. Yes, we could get by without one of them.

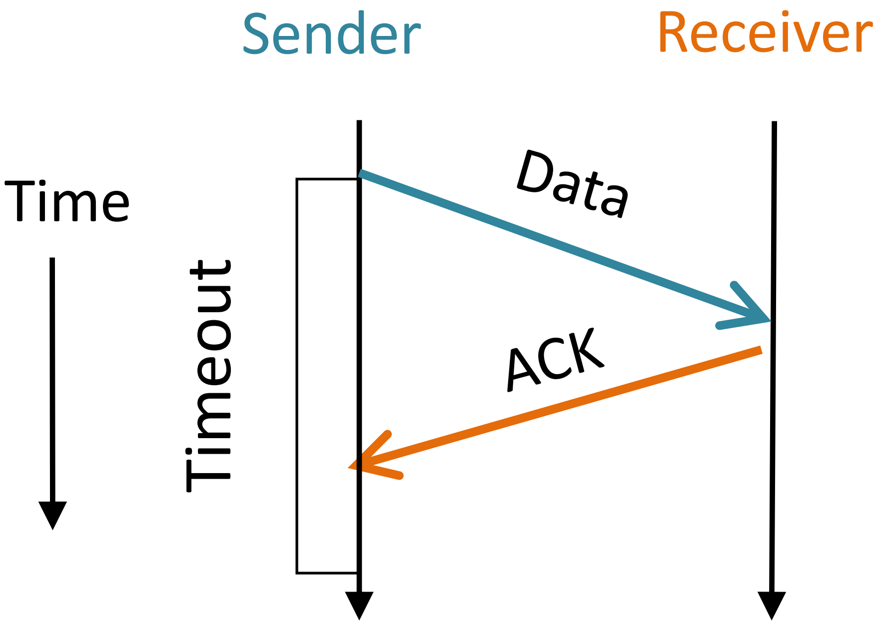


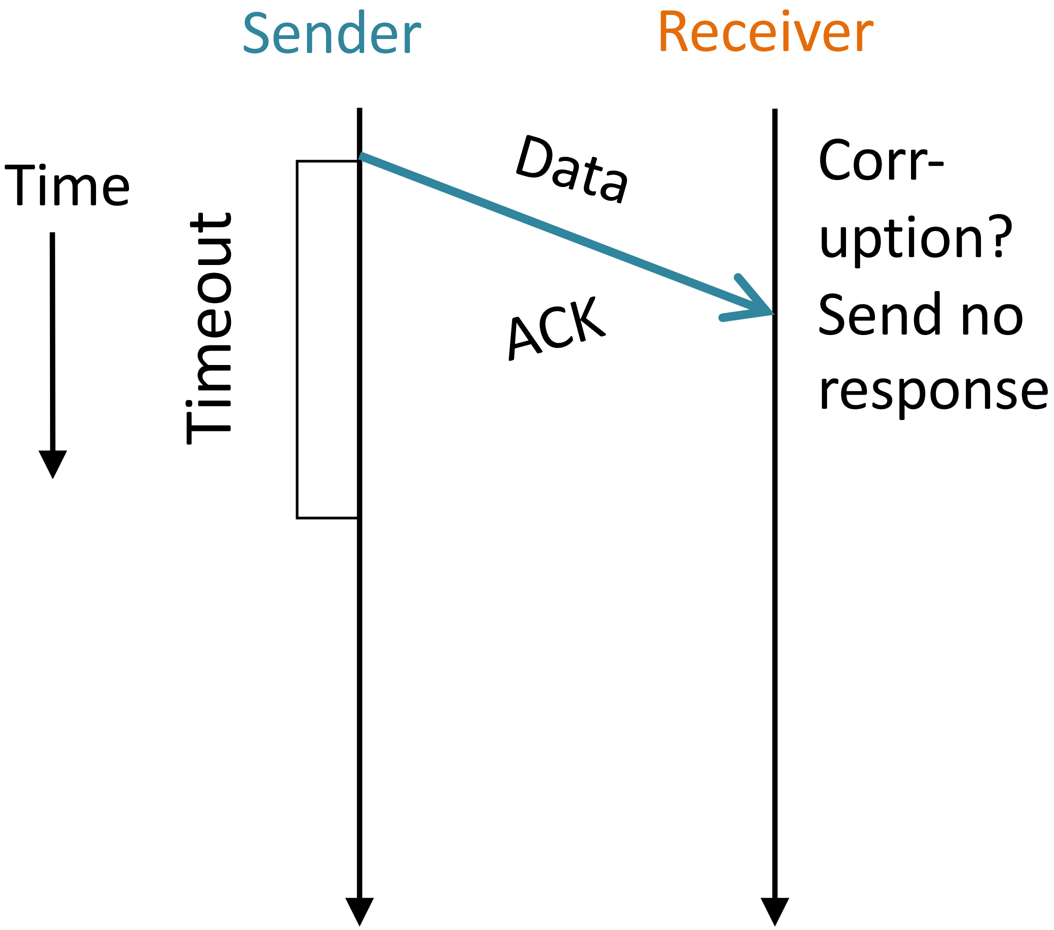
Timeouts and Losses



- Sender starts a clock. If no response, retry.

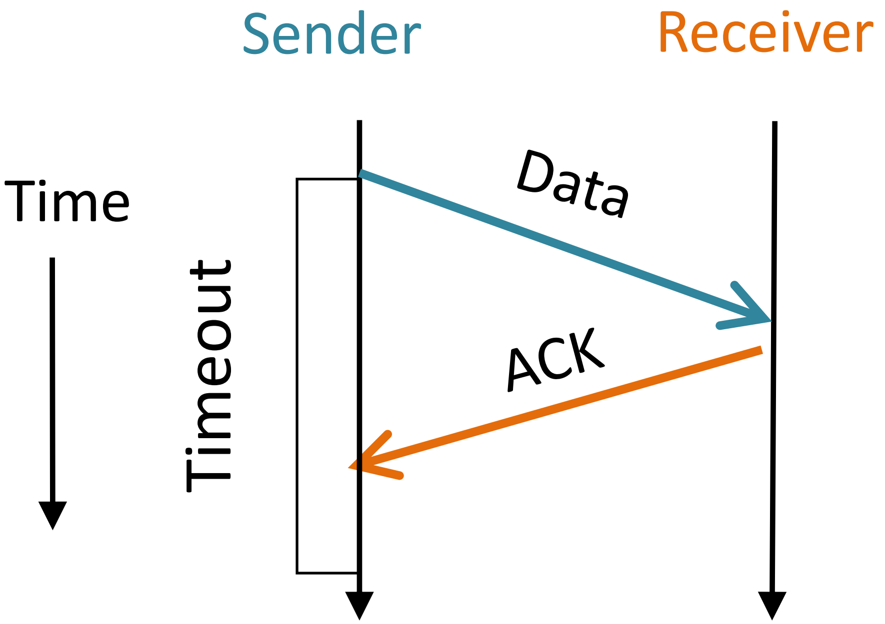
Timeouts and Losses

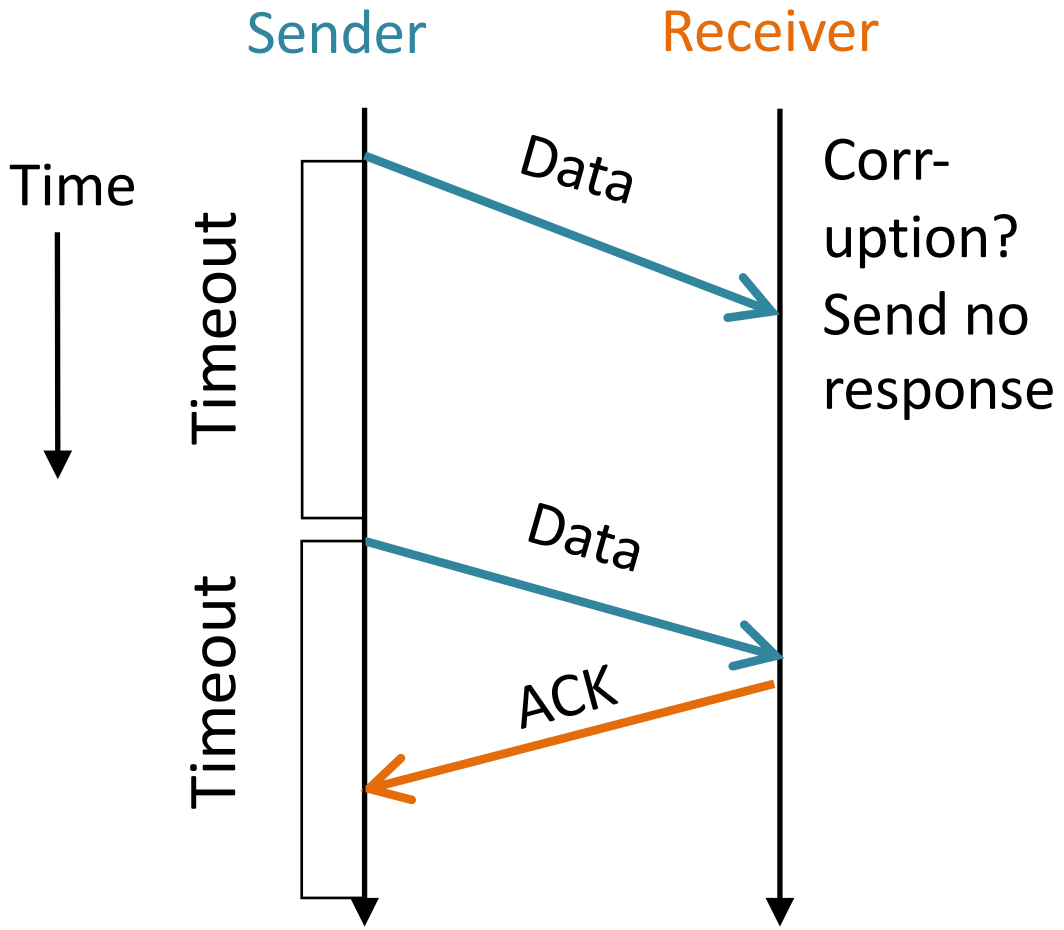




- Sender starts a clock. If no response, retry.

Timeouts and Losses

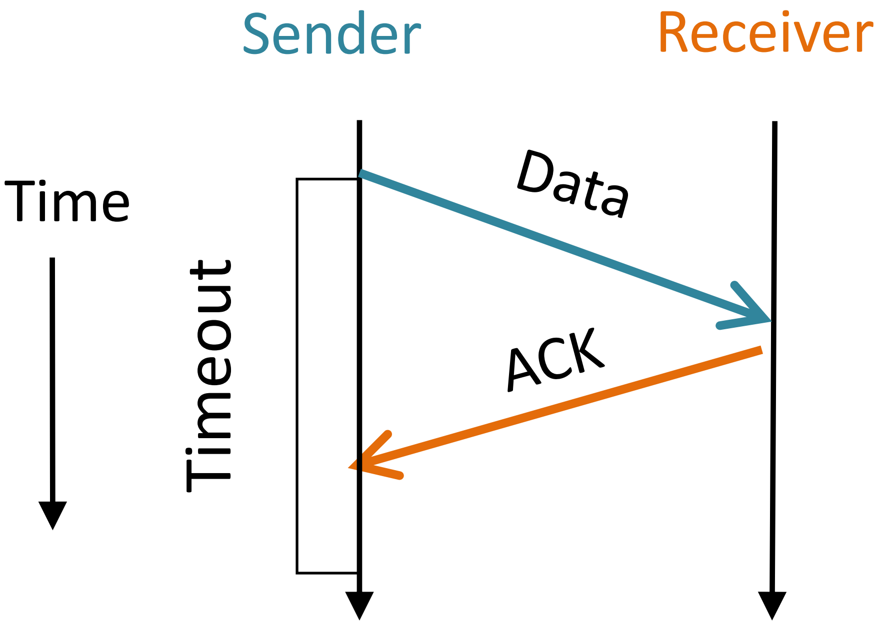




Corr-
uption?
Send no
response

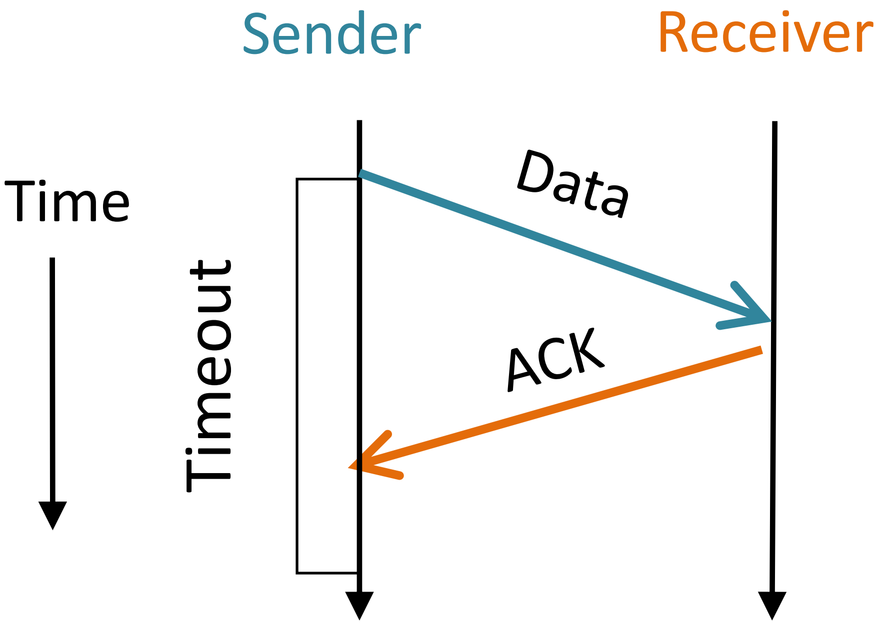
- Sender starts a clock. If no response, retry.
- Probably not a great idea for handling corruption, but it works.

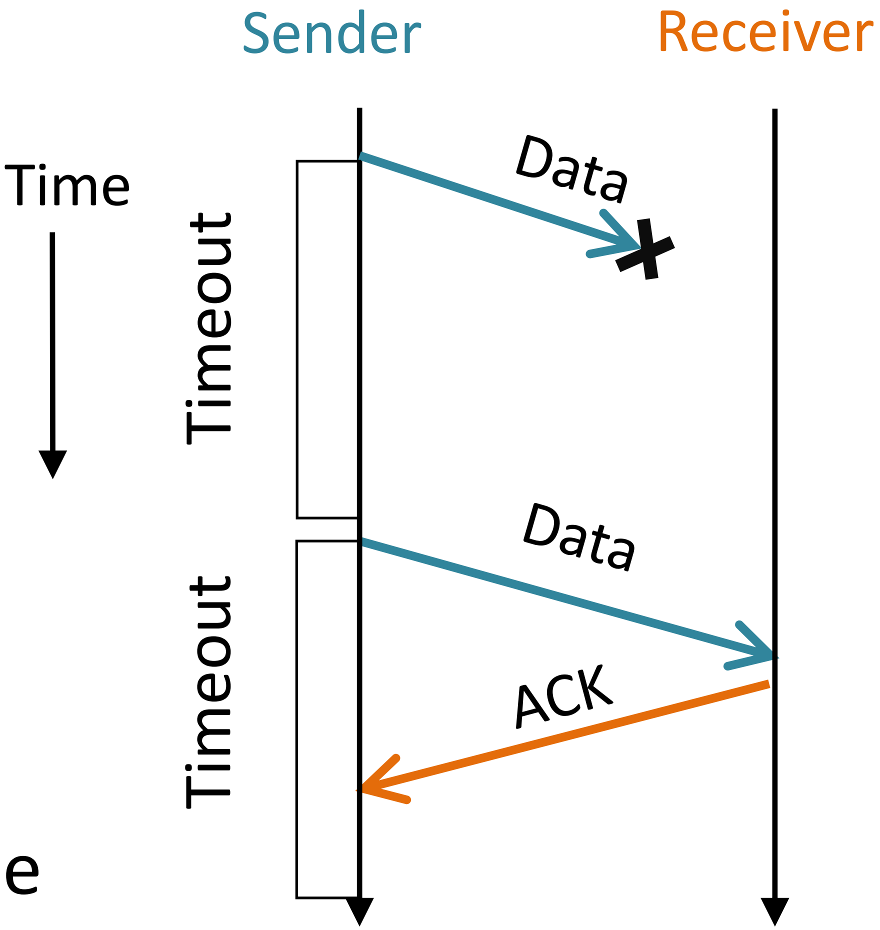
Timeouts and Losses



- Timeouts help us handle message losses too!

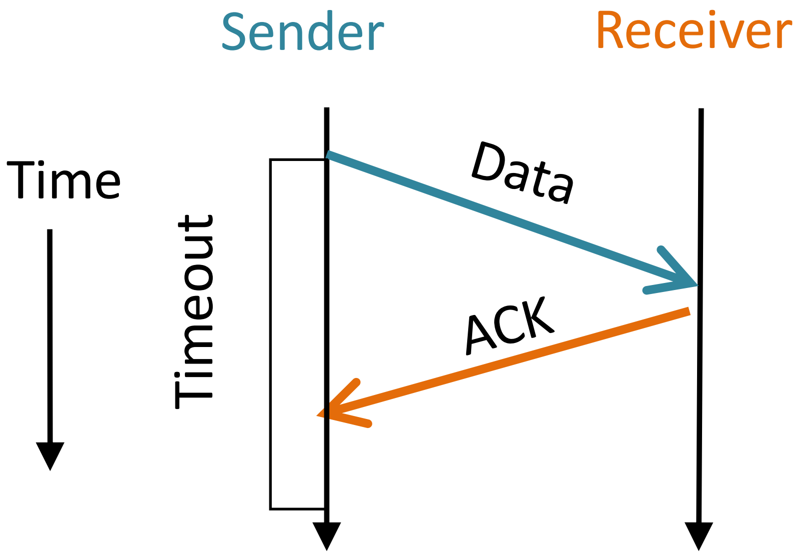
Timeouts and Losses





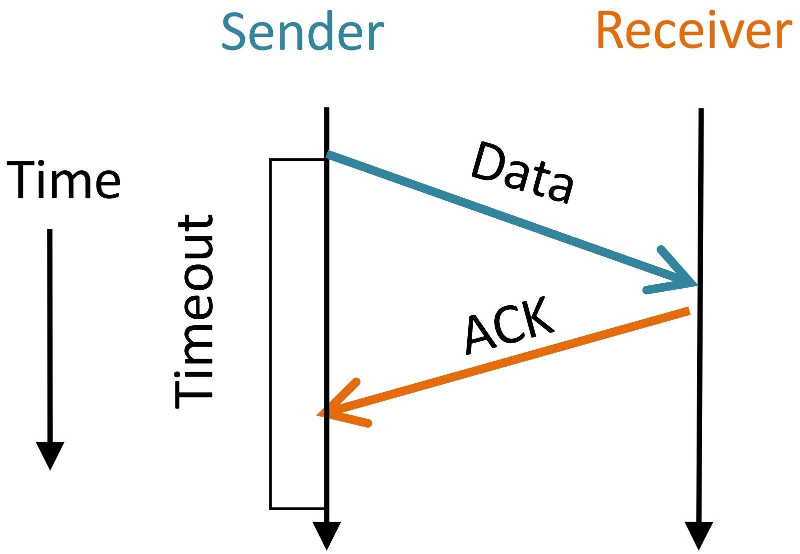
- Timeouts help us handle message losses too!

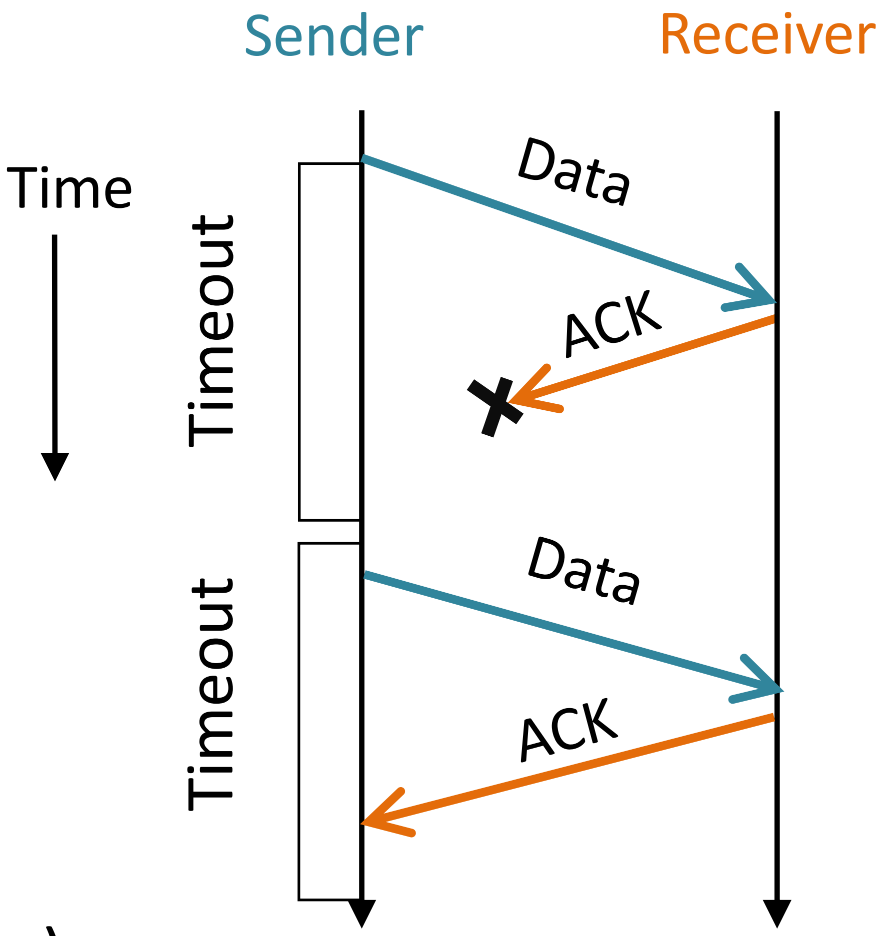
Adding timeouts might create new problems for us to worry about. How many? Examples?



A. No new problems (why not?)
B. One new problem (which is..)
C. Two new problems (which are..)
D. More than two new problems (which are..)

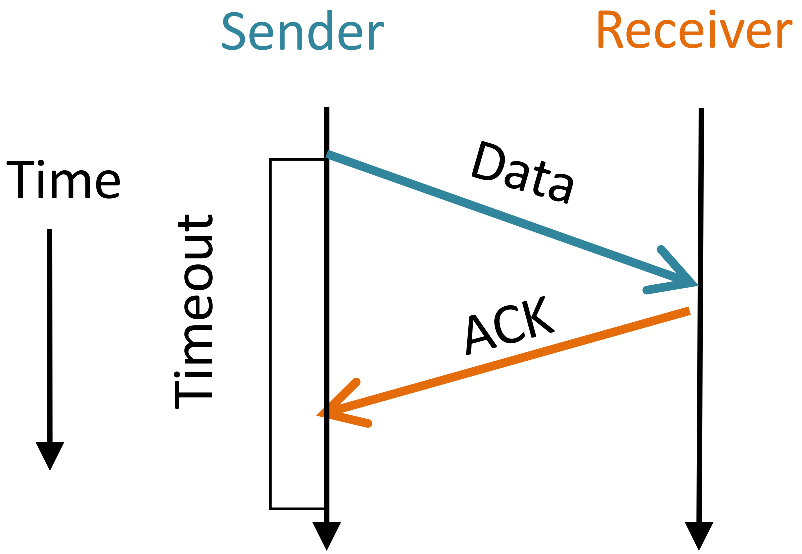
Adding timeouts might create new problems for us to worry about. How many? Examples?

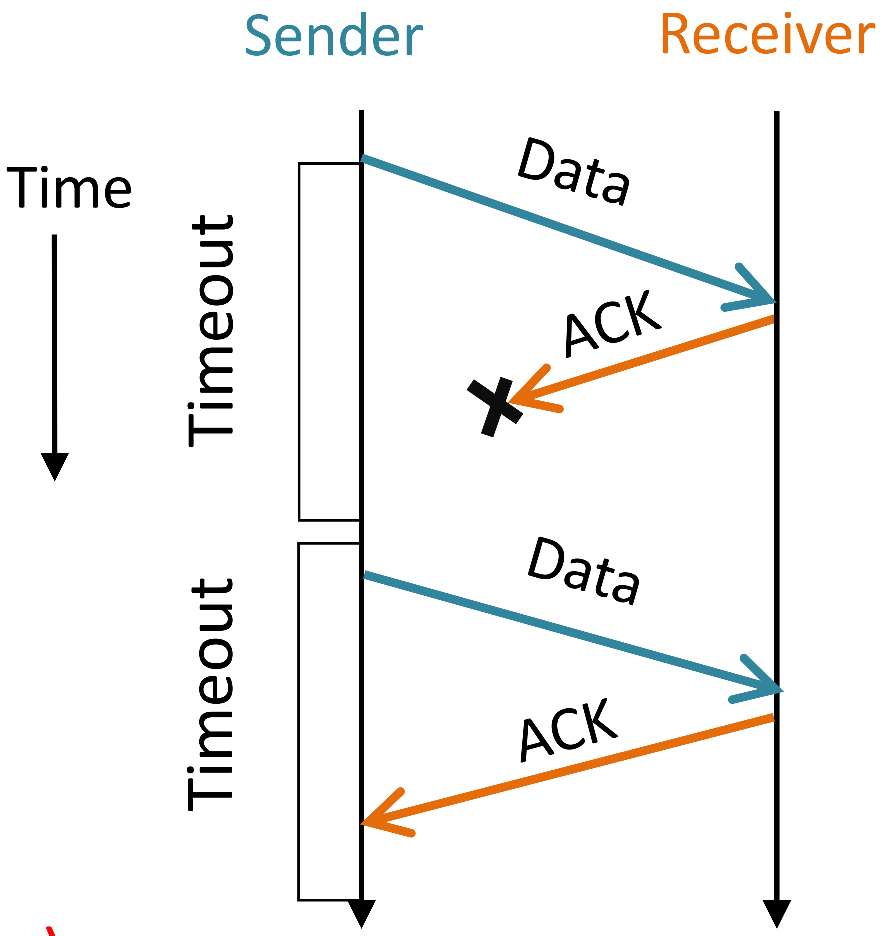




A. No new problems (why not?)
B. One new problem (which is..)
C. Two new problems (which are..)
D. More than two new problems (which are..)

Adding timeouts might create new problems for us to worry about. How many? Examples?





A. No new problems (why not?)

B. One new problem (which is..)

C. Two new problems (which are..)

D. More than two new problems (which are..)

Adding timeouts might create new problems for us to worry about. How many? Examples?

Two new problems:

1. If the data gets through but the ACK gets lost:
   - the sender's timeout will expire, since the ACK never made it across, and the sender resends a copy.
   - The receiver cannot distinguish between a repeat packet or a new packet.
2. If we decide to use a timeout – choosing how long we decide to set this timeout value is difficult!
   - really long? very slow retransmits.
   - really short? a lot of unnecessary duplicates sent that if we had waited longer we would have gotten an ACK for.
   - Choosing this timeout value has a lot of performance implications.

Sequence Numbering

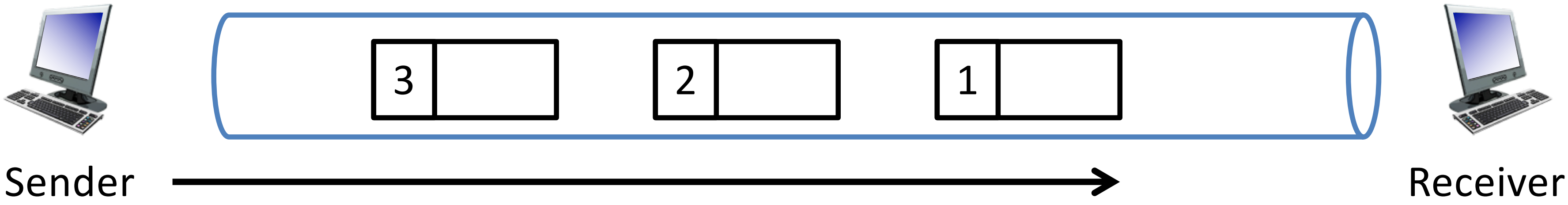
Sender

- Add a monotonically increasing label to each msg

Receiver

- Ignore messages with numbers we've seen before

- When pipelining (a few slides from now)
  - Detect gaps in the sequence (e.g., 1,2,4,5)



What is our link utilization with a stop-and-wait protocol?

System parameters:

Link rate: 8 Mbps (one megabyte per second)

RTT: 100 milliseconds

Segment size: 1024 bytes = 1kB

A. < 0.1 %
B. ≈ 0.1 %
C. ≈ 1 %
D. 1-10 %
E. > 10 %

What is our link utilization with a stop-and-wait protocol?

A. < 0.1 %
B. ≈ 0.1 %
C. ≈ 1 %
D. 1-10 %
E. > 10 %

Link Utilization:

= Protocol Sending Rate/Link Rate

Protocol Sending Rate in seconds:

= 1 segment (1kB) in 1 RTT

= 1 segment in 100ms or 0.1 seconds

= 10 segments in 1 second

Link Rate = 1 megabyte = 1000kB

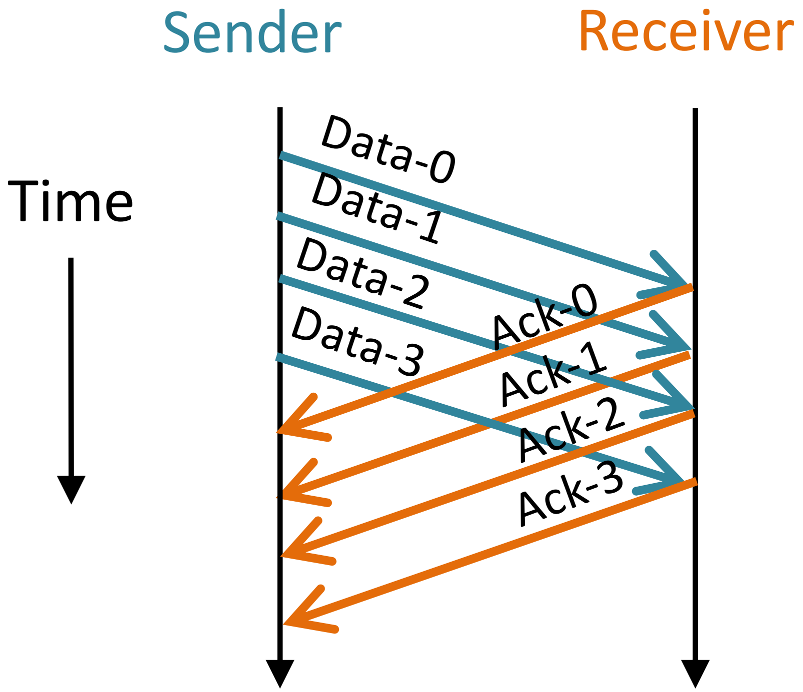
Link Utilization:

= 10 kBps /1000 kBps (1 megabyte = 1000kB)

= 1%

Big Problem: Performance is determined by RTT, not channel capacity!

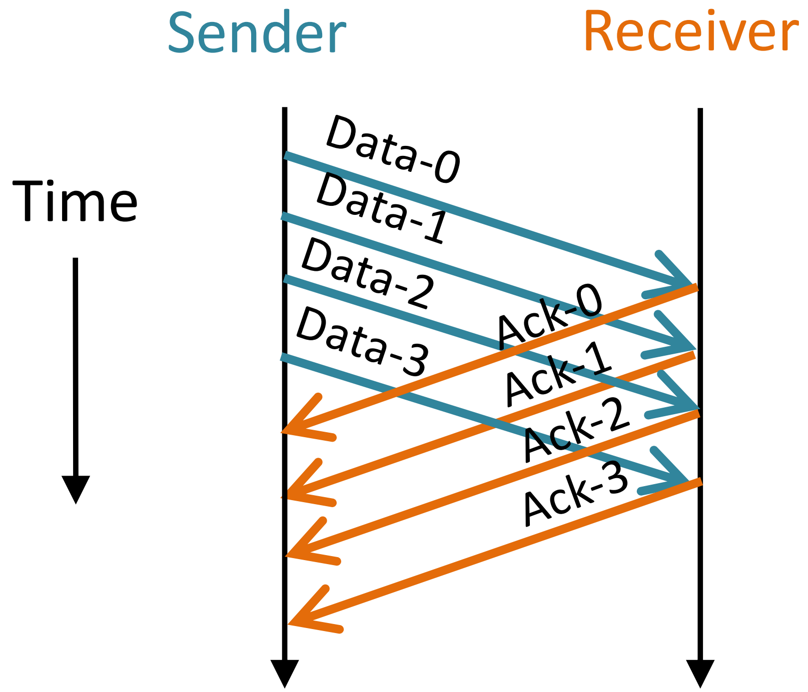
Pipelined Transmission

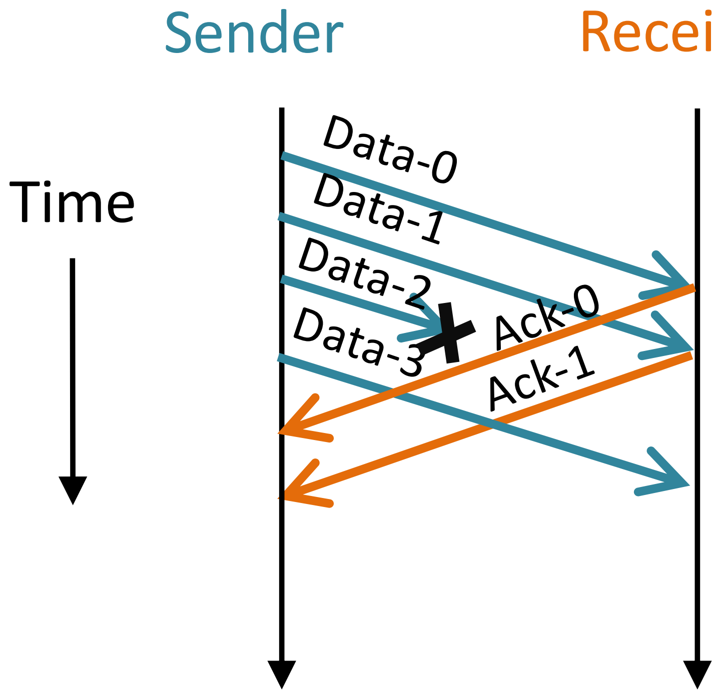


<u>Keep multiple segments "in flight"</u>

- Allows sender to make efficient use of the link
- Sequence numbers ensure receiver can distinguish segments
- We'll talk about "how many" next time (windowing).

Pipelined Transmission



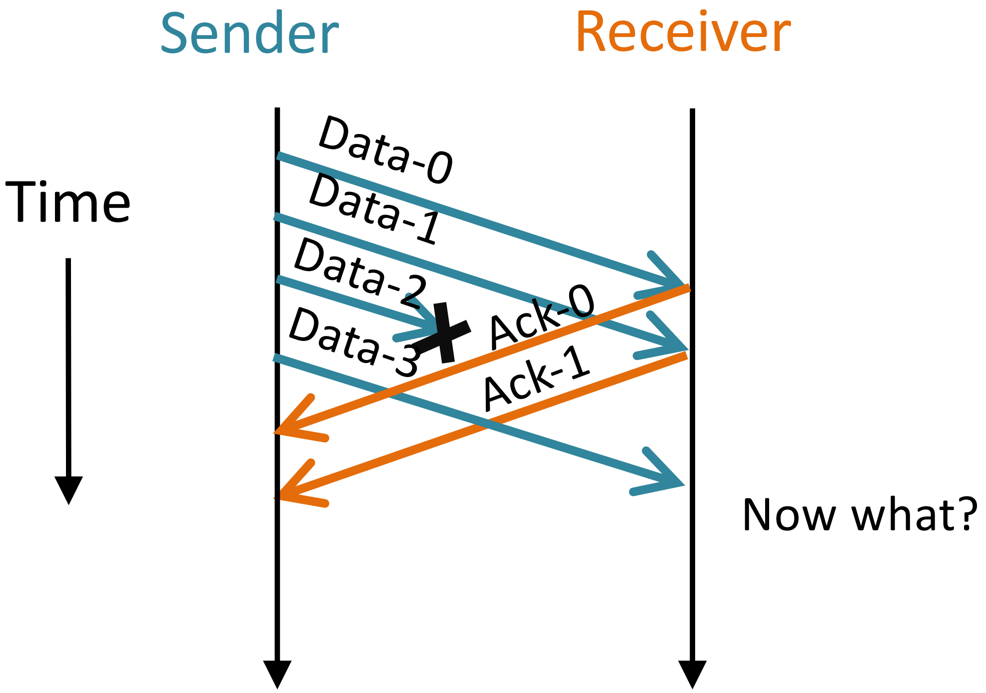


Now what?

Keep multiple segments "in flight"

- Allows sender to make efficient use of the link
- Sequence numbers ensure receiver can distinguish segments
- We'll talk about "how many" next time (windowing).

What should the sender do here?

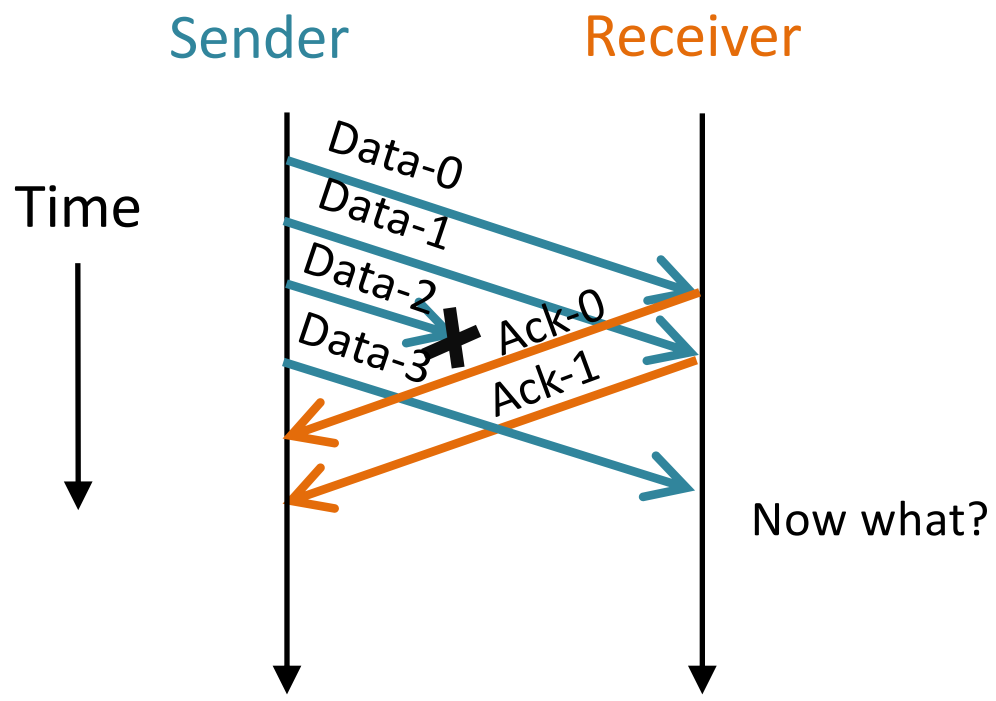


What information does the sender need to make that decision?

What is required by either party to keep track?

A. Start sending all data again from 0.
B. Start sending all data again from 2.
C. Resend just 2, then continue with 4 afterwards.

What should the sender do here?



What information does the sender need to make that decision?

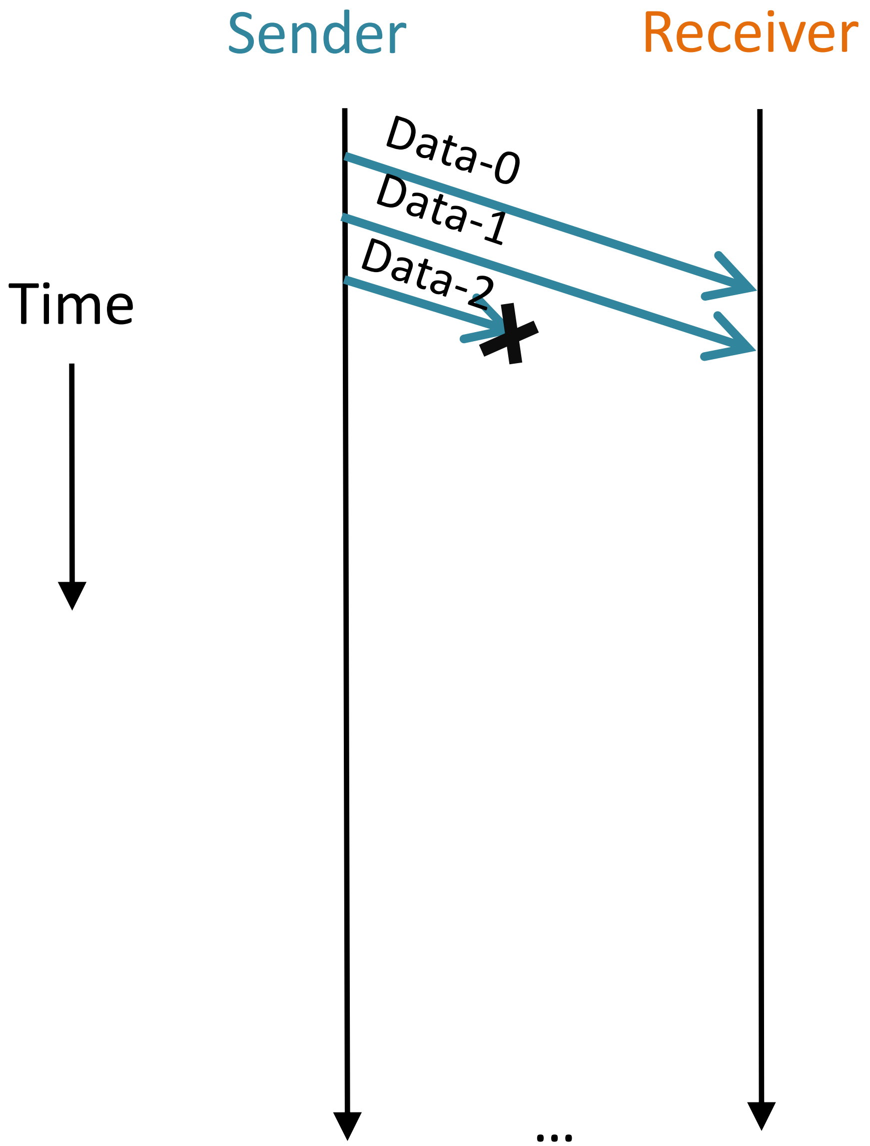
What is required by either party to keep track?

A. Start sending all data again from 0.
B. Start sending all data again from 2 (GBN)
C. Resend just 2, then continue with 4 afterwards (Selective Repeat)

ARQ Broad Classifications

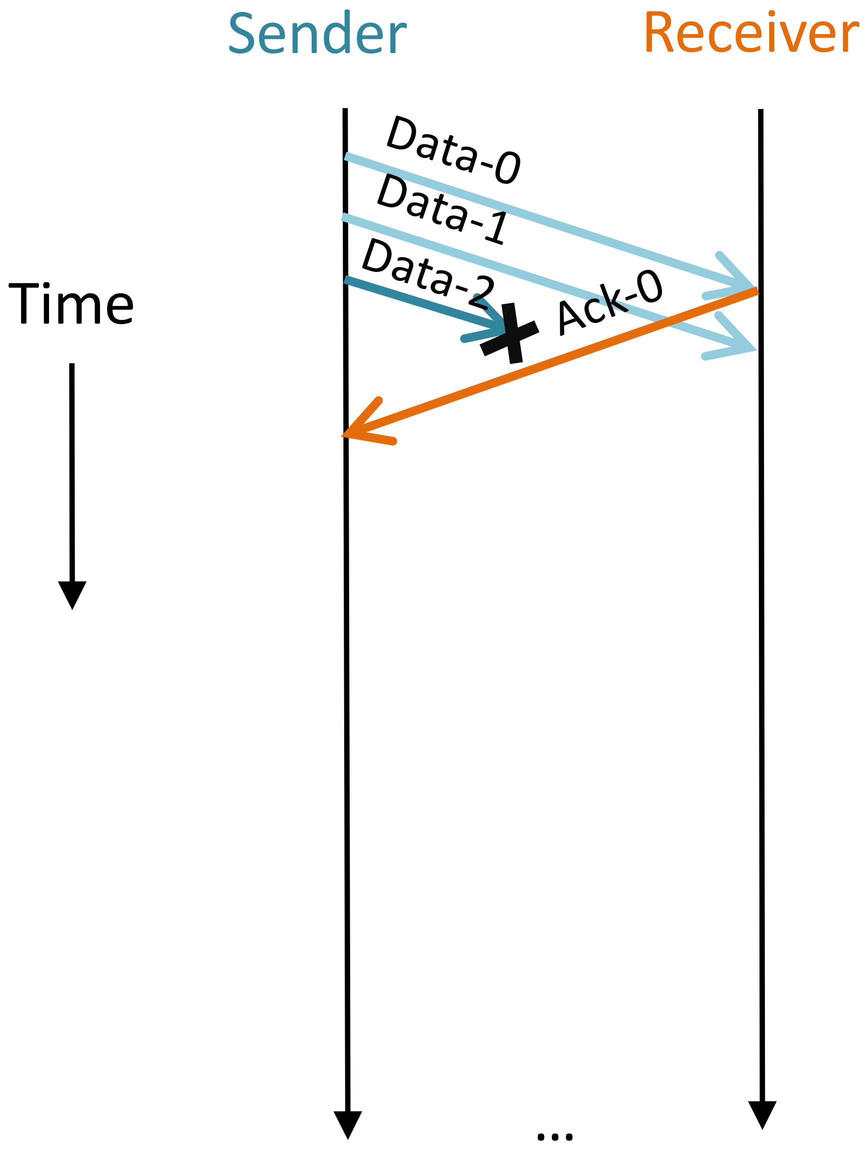
1. Stop-and-wait
2. Go-back-N

Go-Back-N



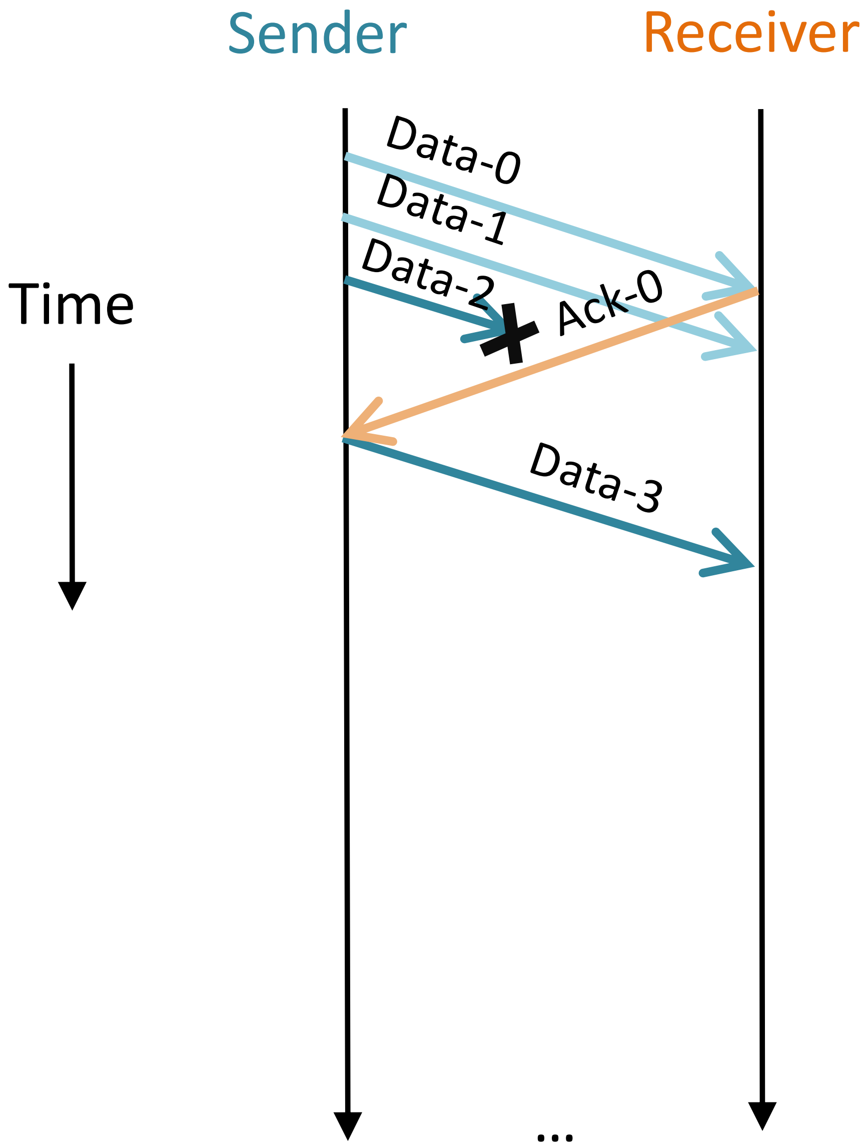
- Retransmit from point of loss
  - Segments between loss event and retransmission are ignored
  - “Go-back-N” if a timeout event occurs

Go-Back-N

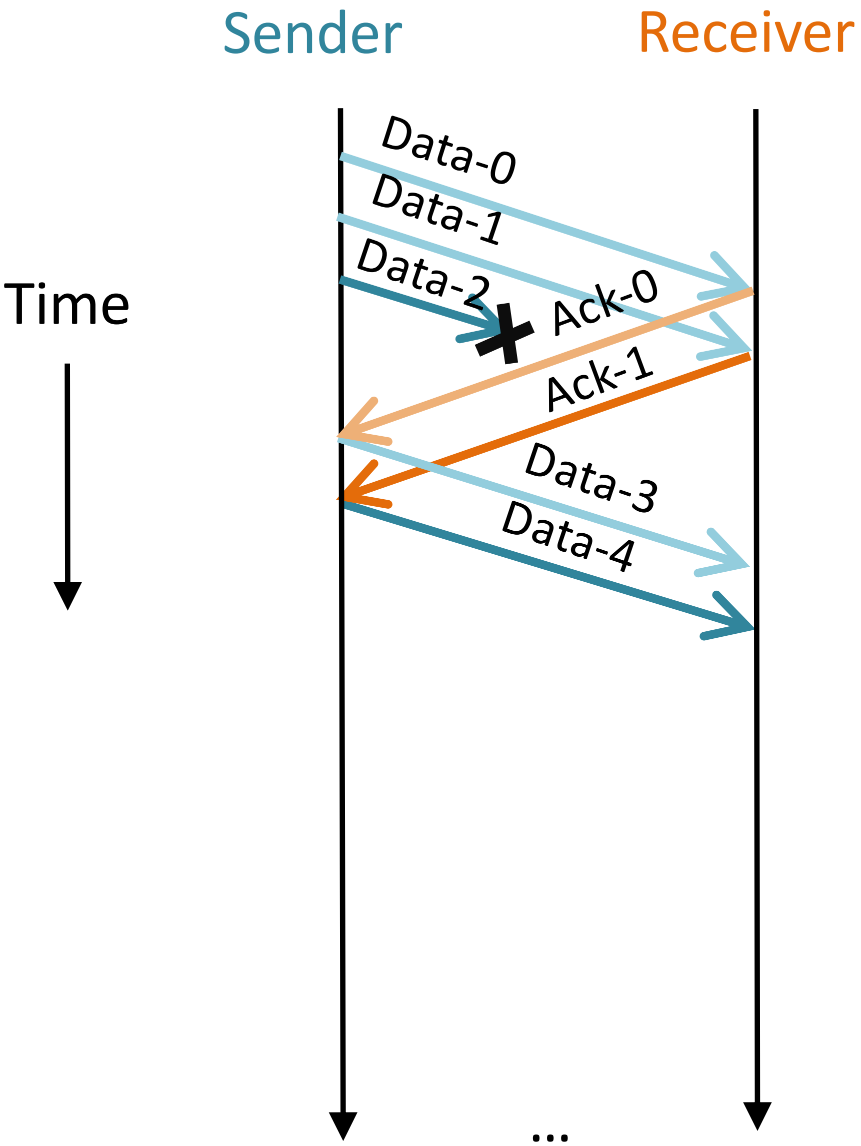


- Retransmit from point of loss
  - Segments between loss event and retransmission are ignored
  - “Go-back-N” if a timeout event occurs

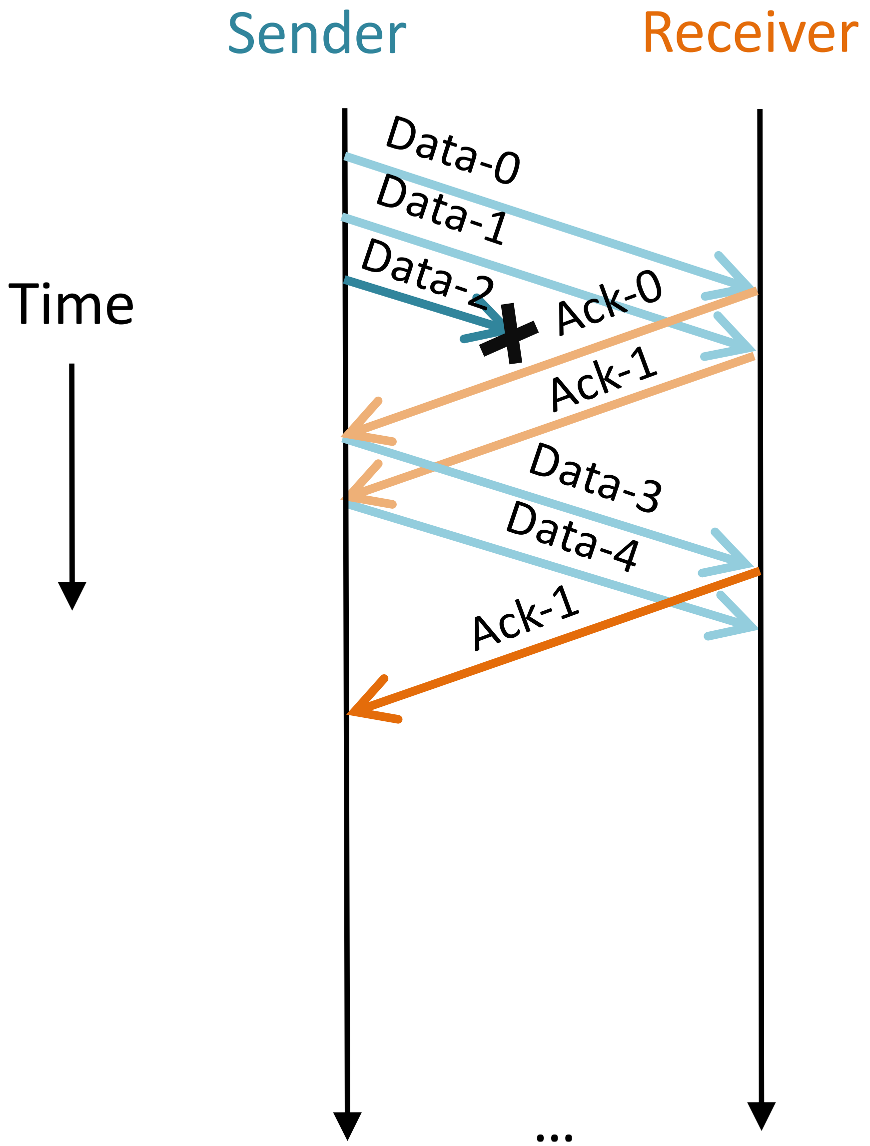
Go-Back-N



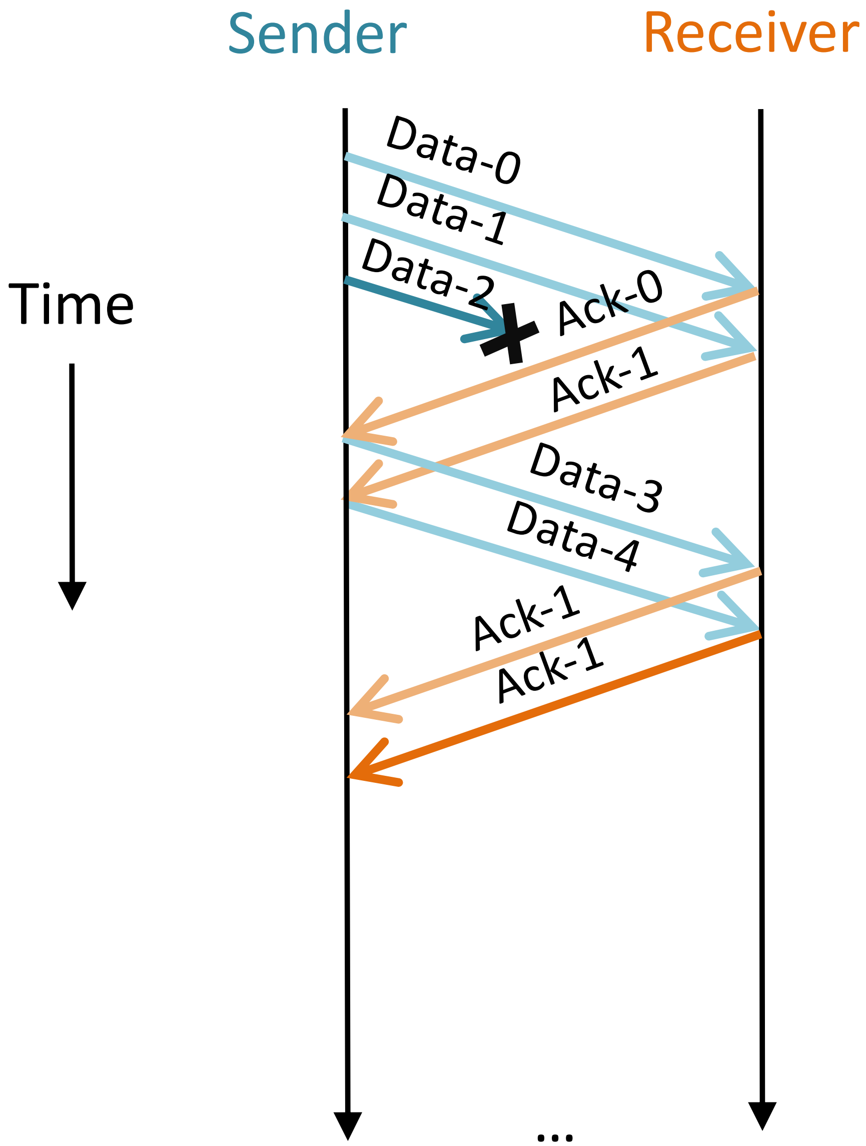
Go-Back-N


Sender
Receiver
Time
Data-0
Data-1
Data-2
Ack-0
Ack-1
Data-3
Data-4
…

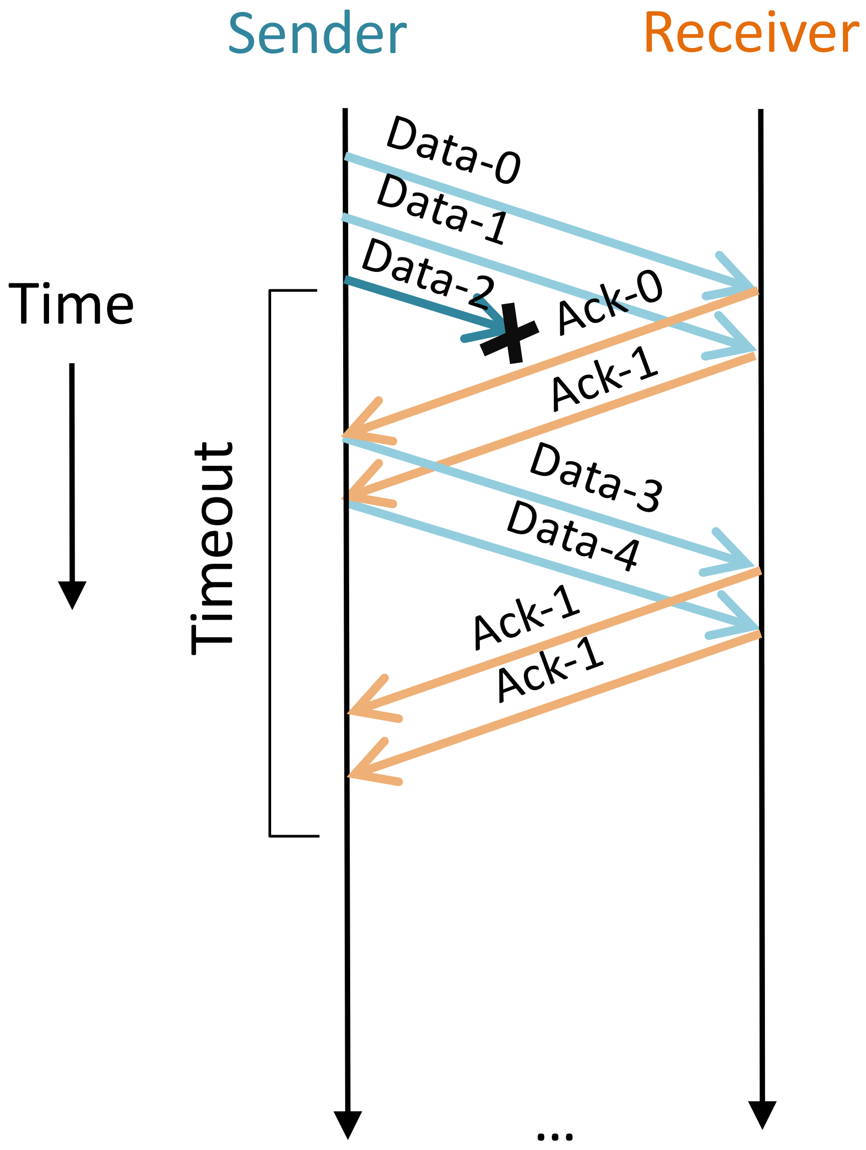
Go-Back-N


Sender
Receiver
Time
Data-0
Data-1
Data-2
Ack-0
Ack-1
Data-3
Data-4
Ack-1
...

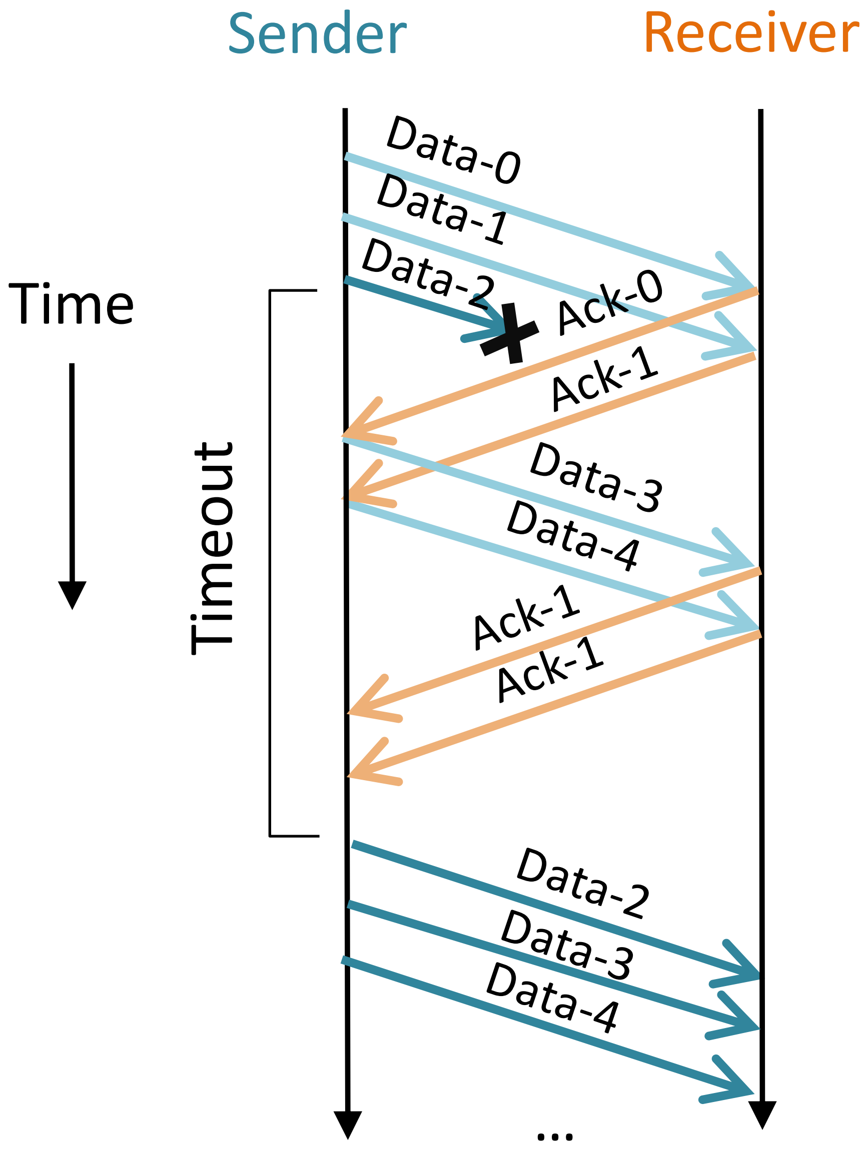
Go-Back-N


Sender
Receiver
Time
Data-0
Data-1
Data-2
Ack-0
Ack-1
Data-3
Data-4
Ack-1
Ack-1
...

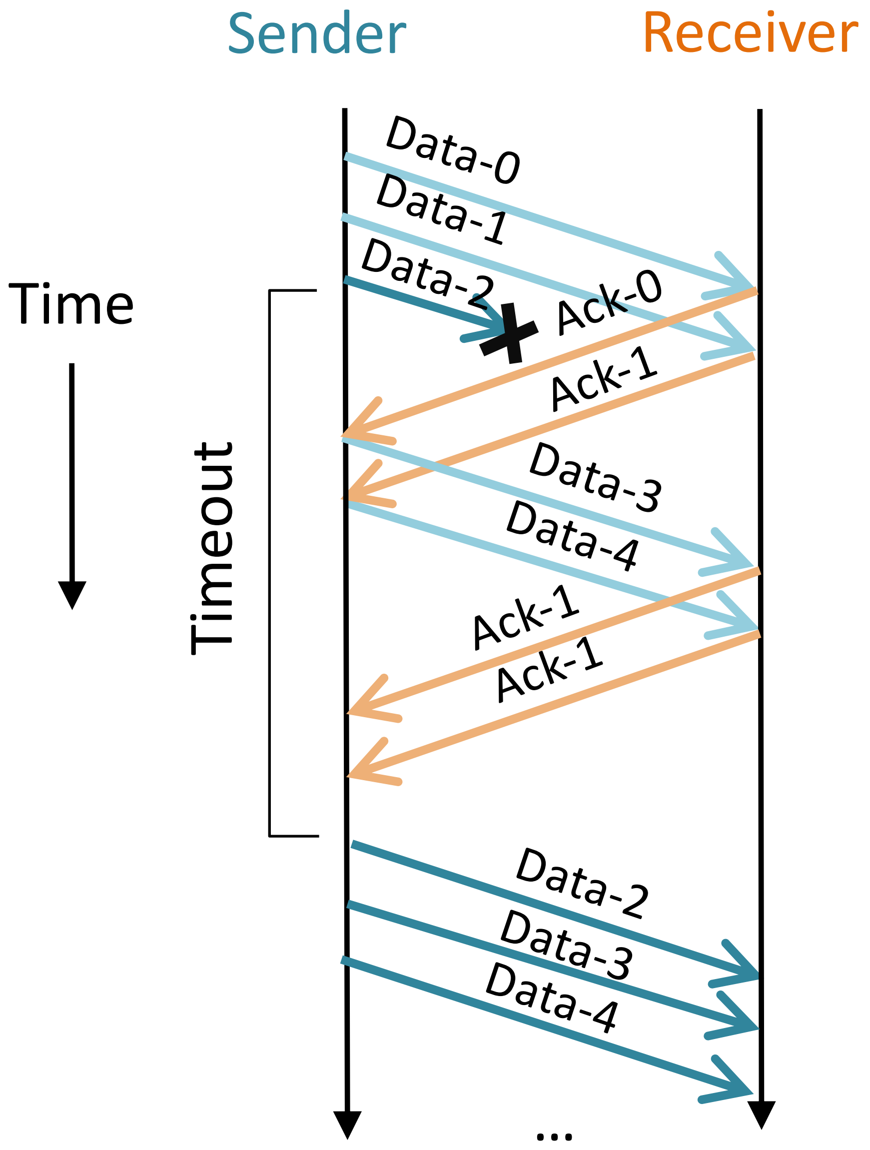
Go-Back-N


Sender
Receiver
Data-0
Data-1
Data-2
Ack-0
Ack-1
Time
Timeout
Data-3
Data-4
Ack-1
Ack-1
…

Go-Back-N

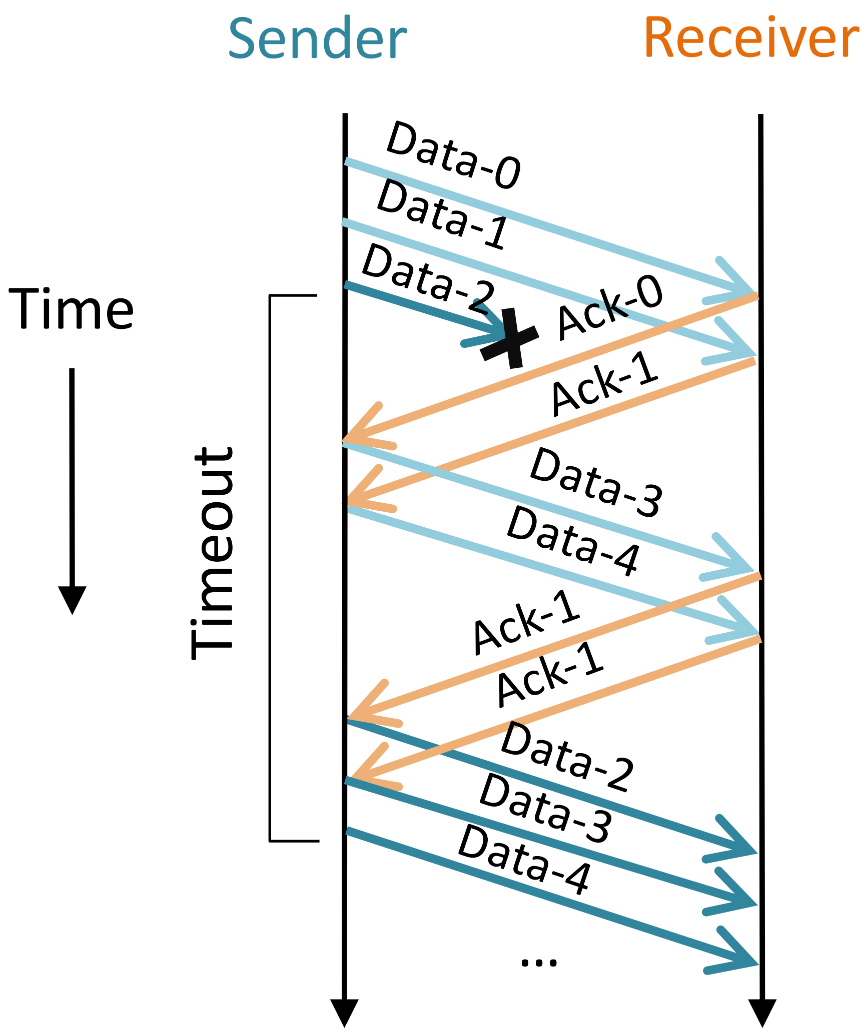

Sender
Receiver
Time
Timeout
Data-0
Data-1
Data-2
Ack-0
Ack-1
Data-3
Data-4
Ack-1
Ack-1
Data-2
Data-3
Data-4
...

Go-Back-N



- Retransmit from point of loss
  - – Segments between loss event and retransmission are ignored
  - – "Go-back-N" if a timeout event occurs
- Fast retransmit
  - – Don't wait for timeout if we get N duplicate ACKs

Go-Back-N

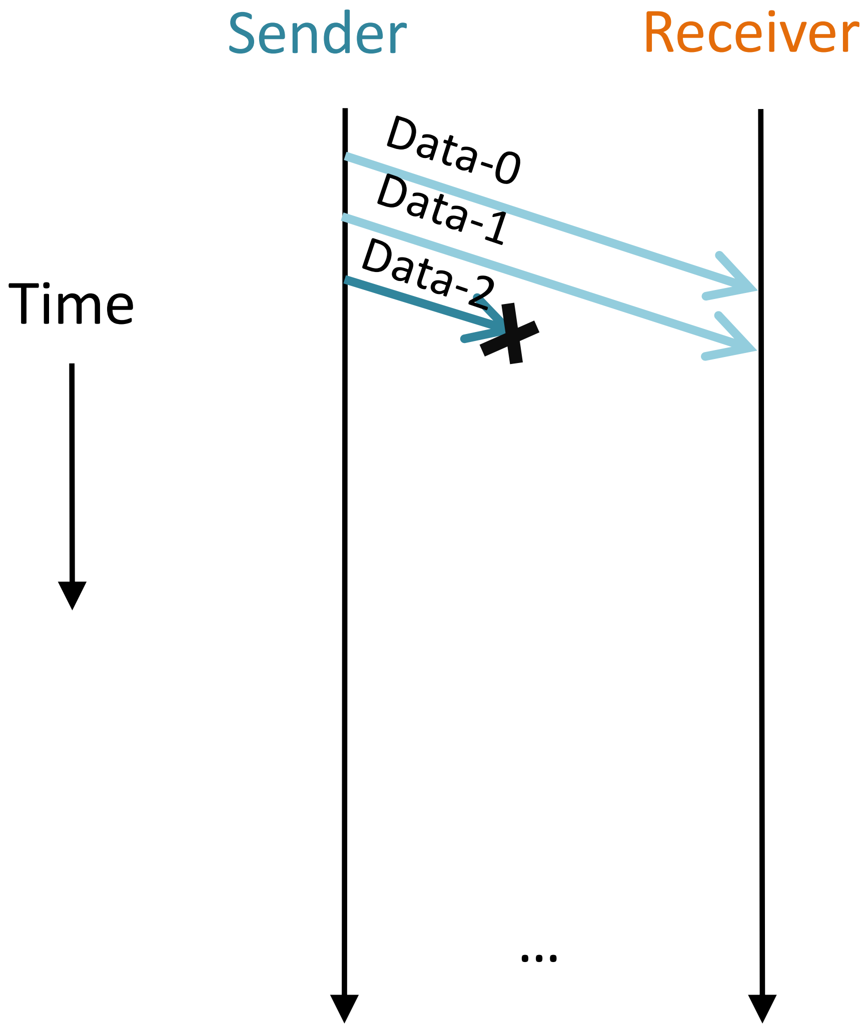


- Retransmit from point of loss
  - Segments between loss event and retransmission are ignored
  - "Go-back-N" if a timeout event occurs
- Fast retransmit
  - Don't wait for timeout if we get N duplicate ACKs

ARQ Broad Classifications

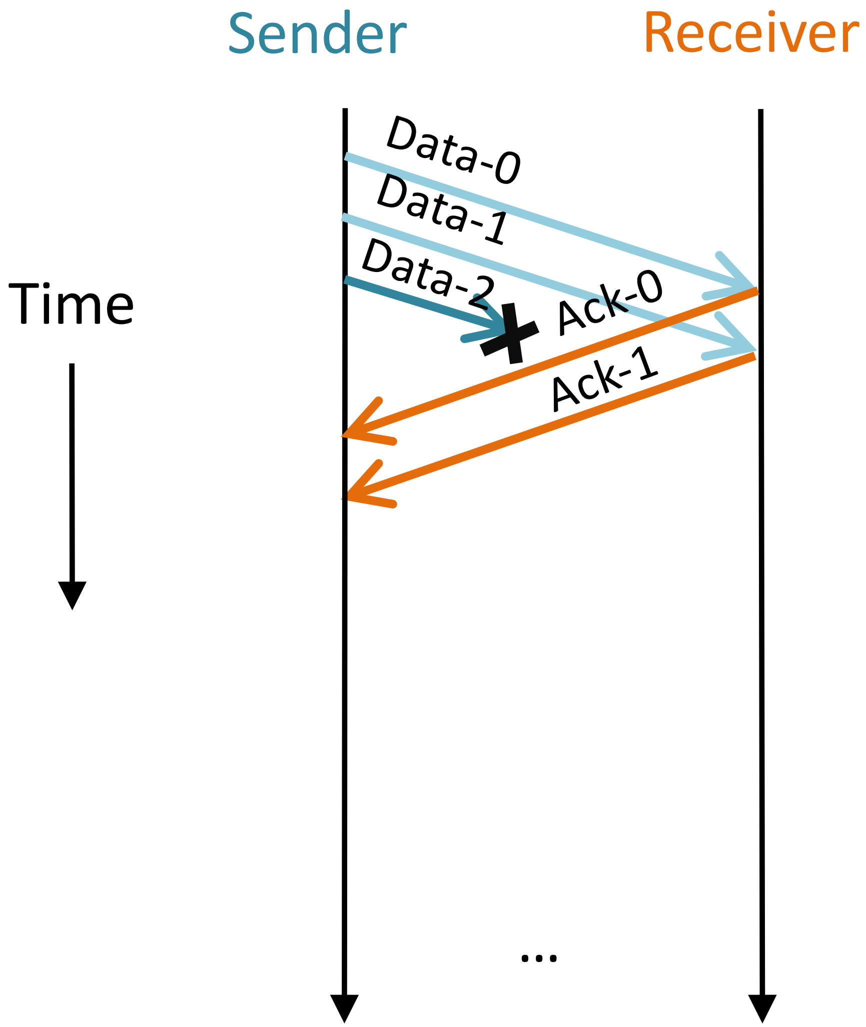
1. Stop-and-wait
2. Go-back-N
3. Selective repeat
    - a.k.a selective reject, selective acknowledgement

Selective Repeat

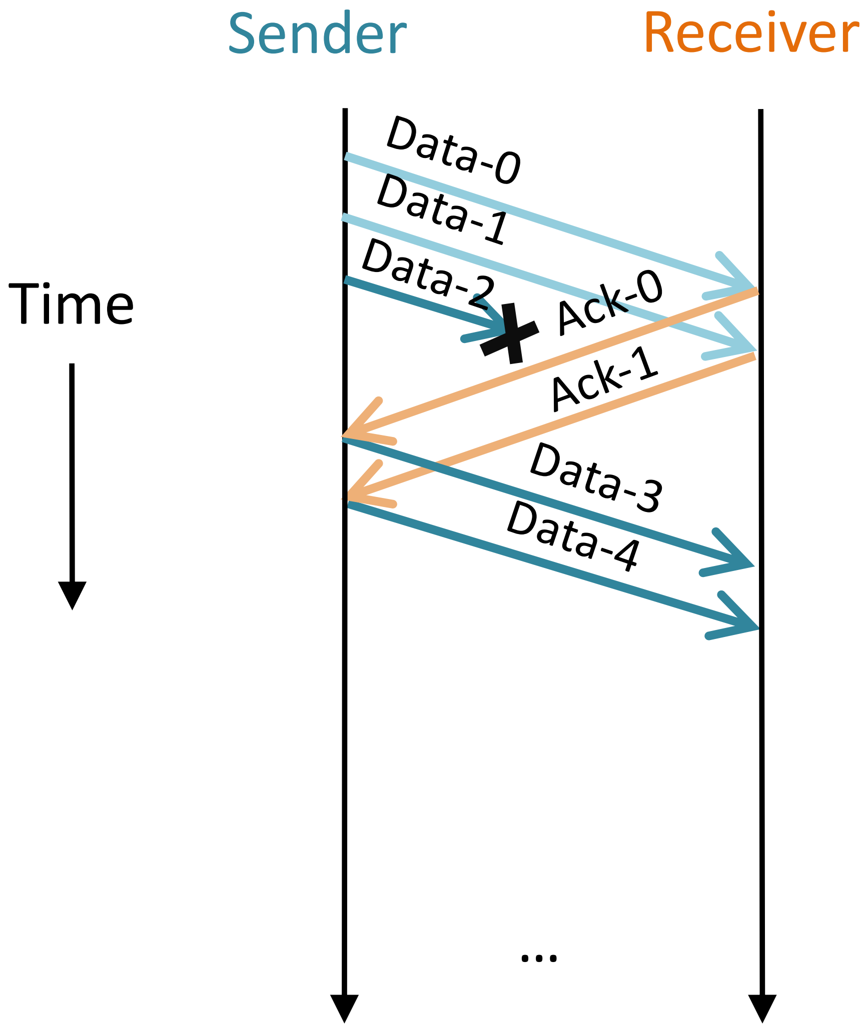


- Receiver ACKs each segment individually (not cumulative)
- Sender only resends those not ACKed
- Requires extra buffering and state on the receiver

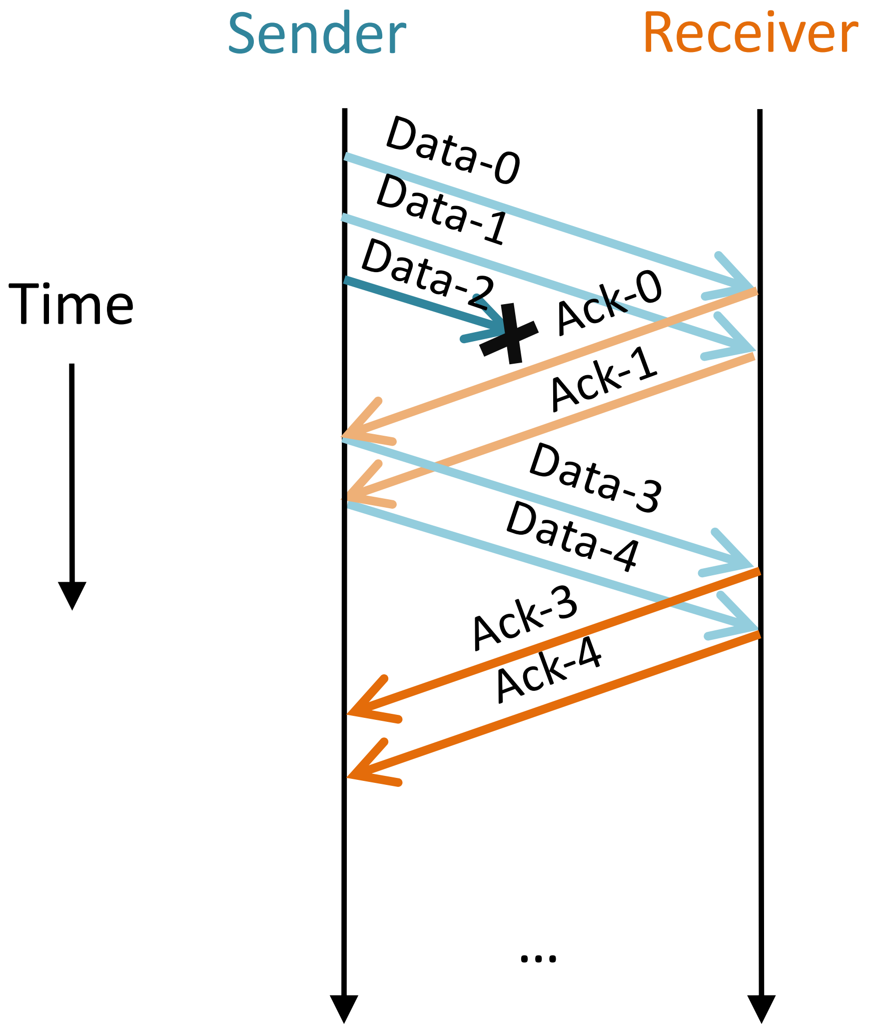
Selective Repeat



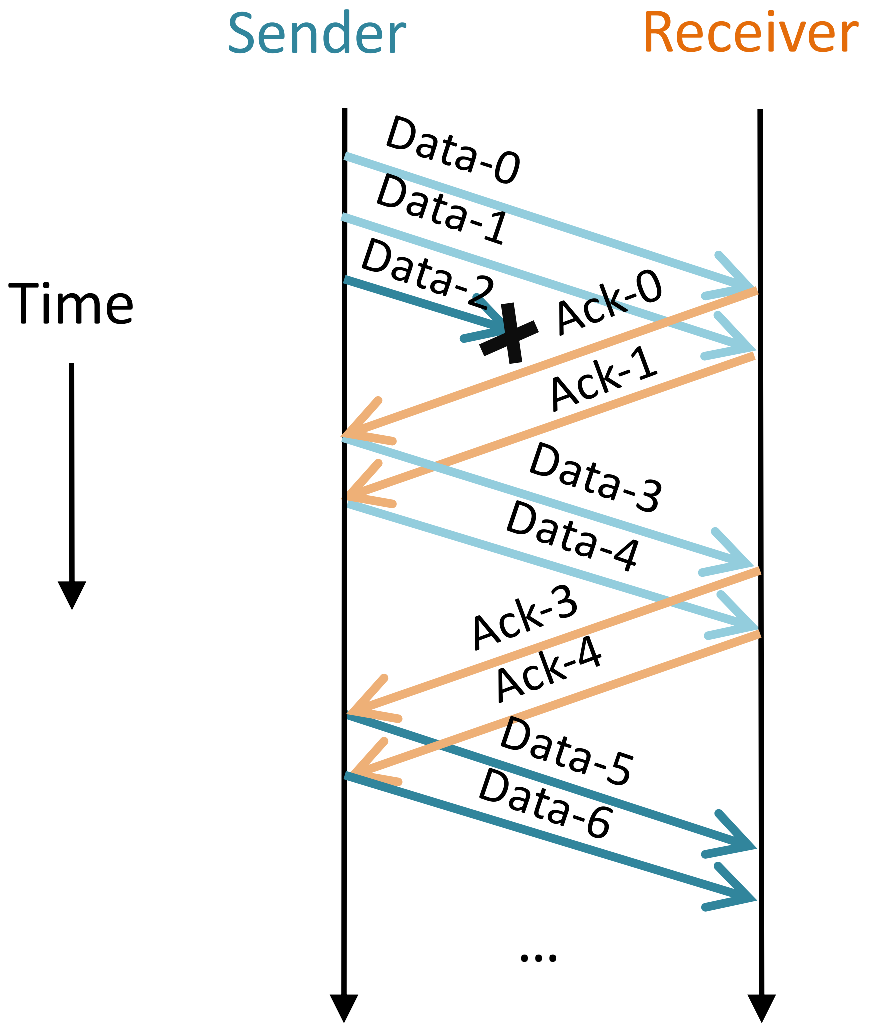
Selective Repeat



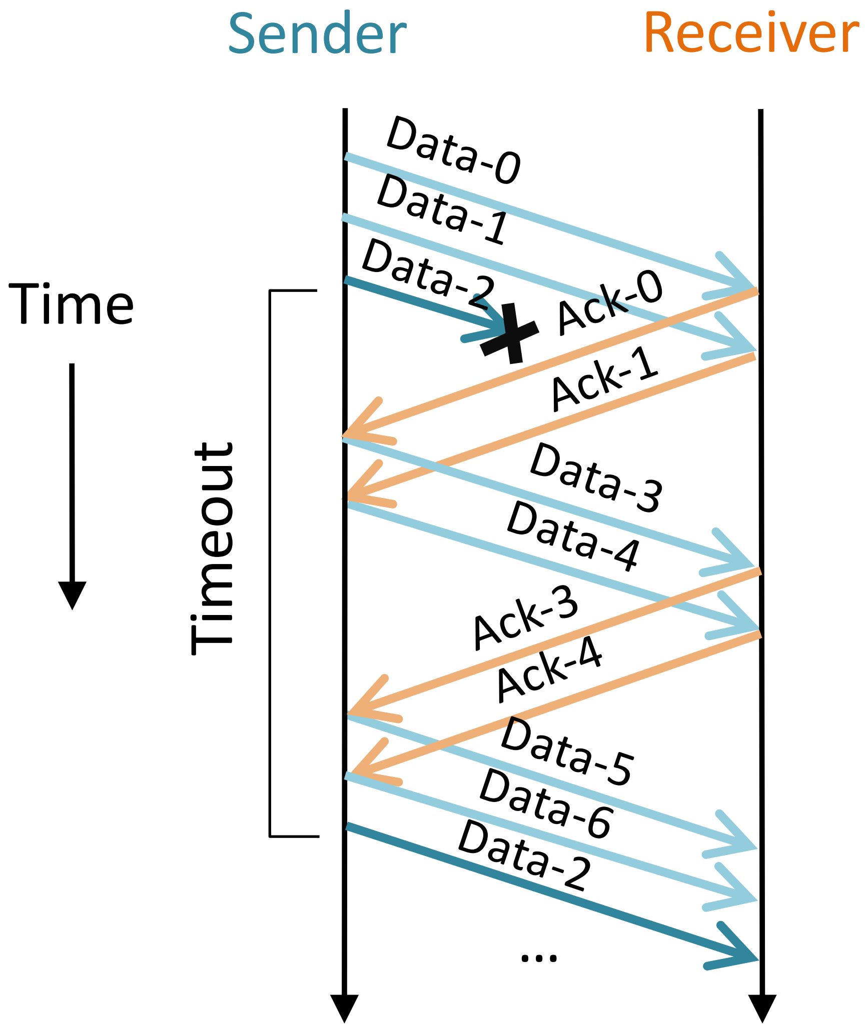
Selective Repeat



Selective Repeat



Selective Repeat



- Receiver ACKs each segment individually (not cumulative)
- Sender only resends those not ACKed
- Requires extra buffering and state on the receiver

ARQ Alternatives

- Can’t afford the RTT’s or timeouts?
- When?
  - Broadcasting, with lots of receivers
  - Very lossy or long-delay channels (e.g., space)
- Use redundancy – send more data
  - Simple form: send the same message N times
  - More efficient: use “erasure coding”
  - For example, encode your data in 10 pieces such that the receiver can piece it together with any subset of size 8.

Summary

ARQ Protocol format:

- Message garbled? Ask to repeat. Didn't hear a response? Speak again.

Reliability at the transport layer:

- Can't create perfect channel out of faulty one, we <u>can only increase probability of success</u>

Stop-and-wait:

- ACKs/NACKs: help with message corruption
- ACKs/Timeouts: help with message corruption + loss
- <u>Stop and wait link utilization depends completely on RTT and not channel capacity</u>

Summary (2)

Pipelining: <u>Keep multiple segments "in flight"</u>
  - Allows sender to make efficient use of the link
  - Sequence numbers ensure receiver can distinguish segments

- Go-Back-N:
  - Retransmit from point of loss
  - ACK cumulatively
  - Fast retransmit
- Selective repeat:
  - ACKs each segment individually
  - Retransmit lost packet
  - extra buffering, state at Receiver