Transport Layer Services and Reliable Data Transfer

Unit 07 - Hands-On Networking - 2018

Prof. Dr.-Ing. Thorsten Herfet, Andreas Schmidt, Pablo Gil Pereira

Telecommunications Lab, Saarland Informatics Campus, 21st Feb. 2018

Recap

- Application layer provides **services** to users.
- Application layer is built on top of other layers that **provide connectivity**.
- Services have certain **requirements** (throughput, loss, time).

Transport Layer Services

Transport Layer Services: Overview

- **Process-to-Process Communication**
  Transport layer uses underlying communication (host-to-host).
- **Connections**
  Information exchanges might need to maintain state.
- **Ordered Delivery**
  Packets might take different routes, which should not be visible to application.
- **Flow Control**
  Packets are discarded, when buffers at end-points overflow.
- **Congestion Control**
  Packets are lost, when buffers in intermediate devices overflow.
- **Segmentation**
  Payloads might be too big for one packet so (de-)fragmentation is required.
- **Reliable Communication**
  Physical problems might cause packets to be lost.
- **Pacing**
  The speed at which you send out packets is important.

⇄ Process-to-Process Communication

One host can provide many **different services** at the same time.

Service Identification

- Port: 16 bit, 0 - 65535
  Something like apartment number.
- Protocol: TCP, UDP, ...
- Services have well-known ports (< 1024) (IANA Mapping).
  - Server uses well-known port (e.g. port 22 for SSH).
  - Client uses ephemeral port (on Linux 32768 - 61000).
- Ports are unique.
  Two distinct applications cannot use the same port at the same time.

Multiplexing

- Done by operating system (processes bind to port).
- Protocols have different identification tuples.
- TCP: (Sender IP, Sender Port, Receiver IP, Receiver Port)
  - One socket per connection.
- UDP: (Receiver IP, Receiver Port)
  - One socket per server.
  - Incoming data is delivered along the senders address.
- Addressing scheme: `<ip:port>` (e.g. `192.168.1.1:80`)

Connections

Needed if...

- Certain parameters need to be shared / agreed upon before communication start.
- Some state has to be persisted across a session.
- Proper teardown processes are required to:
  - Make both sides aware that communication has ended.
  - Clean-up resources.

Not needed if...

- No parameters have to be synced between peers (ad-hoc communication).
- Message format is well-known.

Not desired if...

- Initial contact time should be short.
- Number of participants is high.

Quiz

You are creating the next protocol for car-to-car (C2C) communication. Which approach would you follow?

A: Connection-based.

B: Connection-less.

Answer:

A: Might also be possible, but has drawbacks.

B: Given the strict timing requirements of car-to-car, it would be good to save the connection setup latencies.

Ordered Delivery

Motivation

- Sending data to someone has **implicit temporal order**.
- Some applications require this order to be maintained as the **data shares the temporal order**.
- Network layer **cannot guarantee** that packets arrive in order, due to
    - Path diversity and instability.
    - Packet prioritization.
    - Channel characteristics (echos, multipath).

Solution

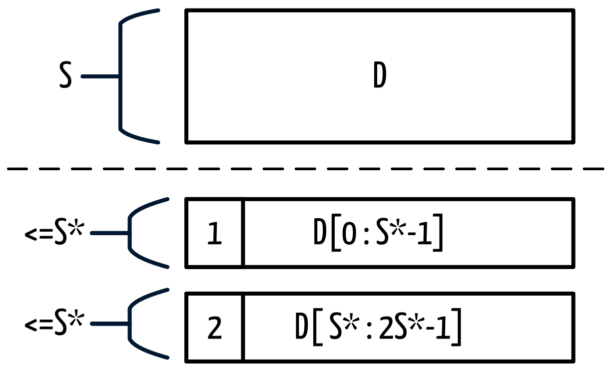
- **Sender:** Attach *sequence number*.
- **Receiver:**
    - Current sequence number packets are forwarded.
    - Higher number packets are either buffered or discarded.
- **Temporal order** from sender **restored** at receiver.

> The service should be provided by the transport layer, as network layer cannot see the complete path and *ensure the properties end-to-end.*

Segmentation

Problem: Data units of a certain size might be put into other units that have a smaller maximal size.

- Higher layer provides data unit D with size S.
- Lower layer can send unit with maximum size S^*.
- D is split into $D_1, D_2, \ldots, D_n$ with sizes $\leq S^*$.
- Lower layer header includes sequence numbers, potentially a segmentation flag.
- Lower layer at receiver reassembles packets and delivers one unit to its higher layer.



Segmentation | Examples

TCP Packetization

- Byte-stream put into socket.
- Packets sizes chosen depending on network conditions.

IPv4 Fragmentation

- Fragmentation ID (which packets belong to one unit).
- Fragmentation Offset (where does the current data belong in the original segment).
- Note: in IPv6 there is no fragmentation.

Sequence Number Pattern

Context:

- Transmission channels with reordering, loss or both.
- Packet size limits require segmentation.

Implementation:

- Provide an additional header field with increasing (cyclic) numbers.
- Ensure field size (and resulting sequence number space) is sufficient.

Benefits:

- Packets can be told apart.
- Windows (sequence number range) can be treated as one unit.

Drawbacks:

- Makes connection stateful.
- Chosing size of sequence number field can be tricky.

Congestion Control

Motivation

Network resources, as

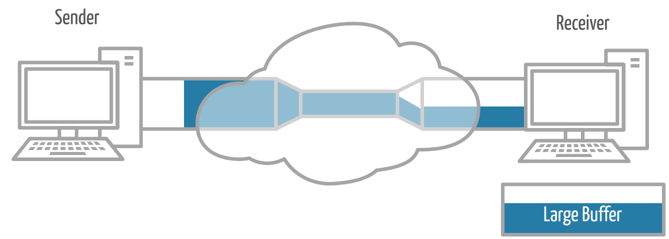
- link capacity
- buffer size

are limited and have to be shared

- effectively and
- fair

between parties using the same network / medium / channel.

> Similar to cars causing traffic jams on highways, packets can cause congestion in routers and on links.



Solution

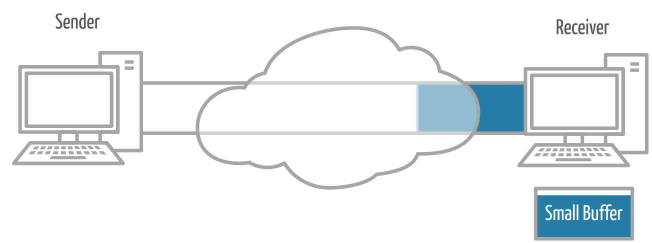
Transport layer can implement congestion control to...

- ... avoid lost packets due to buffer overflows.
- ... avoid slow downs by not exhausting maximum data rate.
- ... ensure fair share of data rate.

Flow Control

Motivation

- End-point resources are limited (receive / send buffers).
- Applications consume transport layer buffer, but on demand and not guaranteed.
- Transport layer cannot handle more packets than fit in the buffers, hence:
    - Sending more packets is wasted effort as receiver will discard.
    - Accepting more packets from application layer is impossible.



Solution

- Peers communicate their buffer sizes and fill levels via feedback.
- Rate control algorithms slow down or stop sending of packets when buffers are filled.
- Upon exhaustion of sending sides buffer, the application is informed and has to deal with it.

Pacing

Relatively new control mechanism for the speed at which packets are sent.

Problem:

- Link speeds and buffer sizes differ across the Internet.
- Sending at your line rate can be too much to fit into midway buffers.
- Packets are lost because you were sending too bursty.

Solution:

- Given a target rate R_T and a line-rate at the sender of R_L (with $R_T < R_L$).
- Send packet P of size L with line-rate R_L (takes $D_t = \frac{L}{R_L}$ seconds).
- At the target rate, it would need $D_t' = \frac{L}{R_T}$.
- After sending P, wait for $D_t' - D_t$ until you send the next packet.

Further Transport Layer Services

Error Control (see later)

Security (see U04)

- Authentication
- Encryption

Common Information Fields

- e.g. RTP RFC3550
- Timestamp
- Identifications

Message Framing

- e.g. RTP, UDP

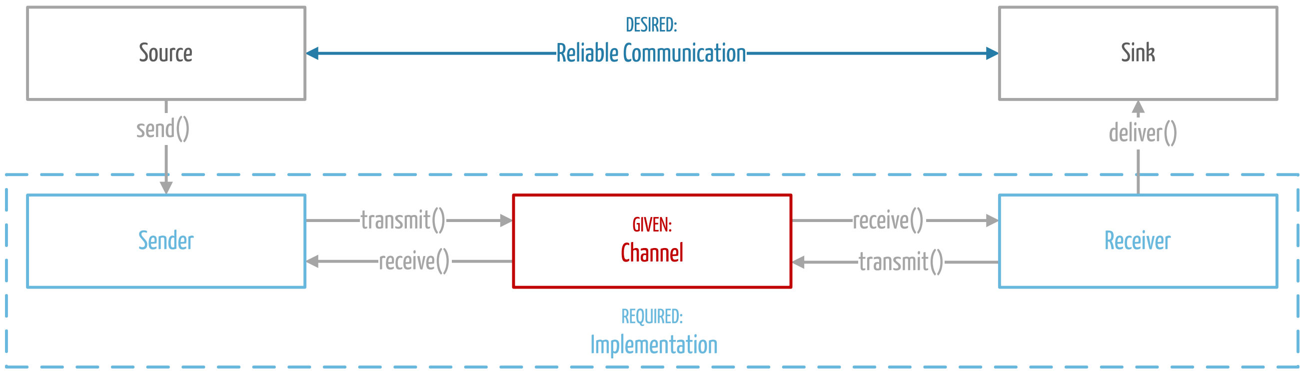
Reliable Data Transfer

Motivation

- Physical channels are never 100% reliable, so transmitted data sometimes gets ...
  - ... garbled (strange sounds, flipped bits).
  - ... lost (shadowing, connection failures, demodulation errors).
  - ... reordered (multipath, echoes).
- Logical channels aren't better:
  - Inherit errors from lower layers.
  - Resources exhaust, so data has to be discarded.
  - Software is buggy.
- Important for *Application*, *Transport* and *Link* layer.

Goal: Provide reliable transfer of data on top of an unreliable channel.

General Approach



◎ **Goal:** Data transmitted by the sender should successfully arrive at the receiver.

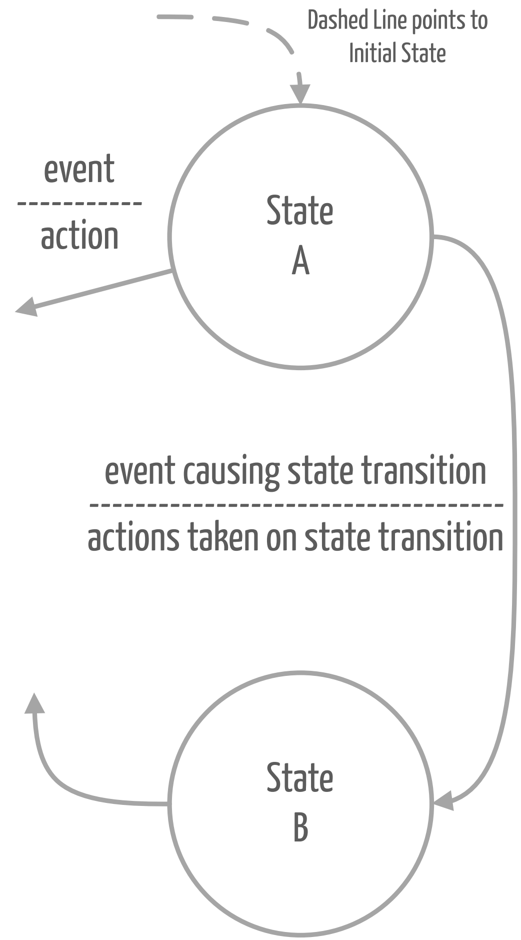
↔ **Channels:**

- Garble data (flip bits, reconstruct as something else).
- Lose data (missing packets, symbols, etc.).
- Reorder data (data still present, but at different location).

Next Steps: Incrementally develop a reliable data transfer (rdt) protocol.

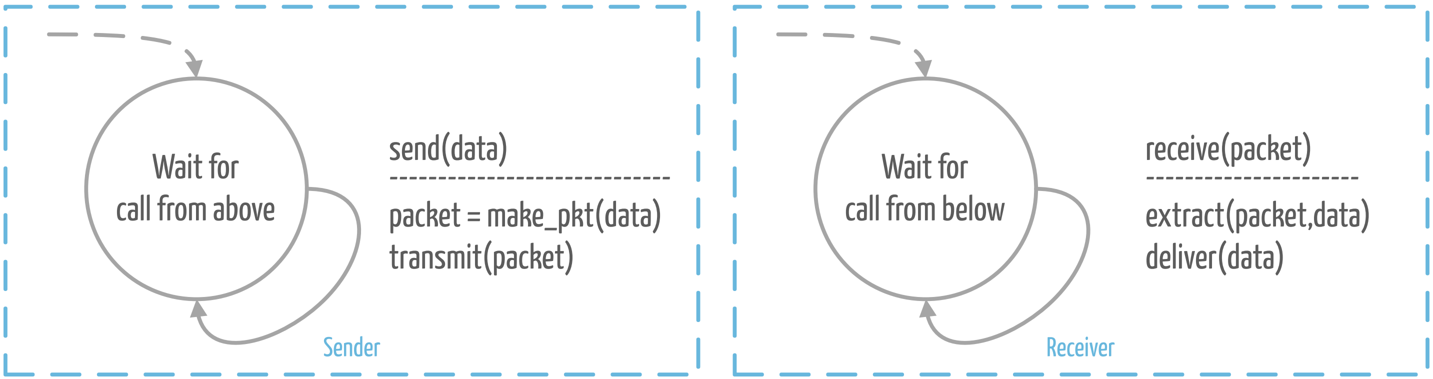
Finite State Machines

- Machine can be in state X out of the set of states S.
- In state X certain events E can happen and cause action A:
- Finite: Limited number of states.
- In our context: Deterministic.



RDT 1.0 | Reliable Channel

↔ **Channel:** Perfectly transmits data as it comes in.



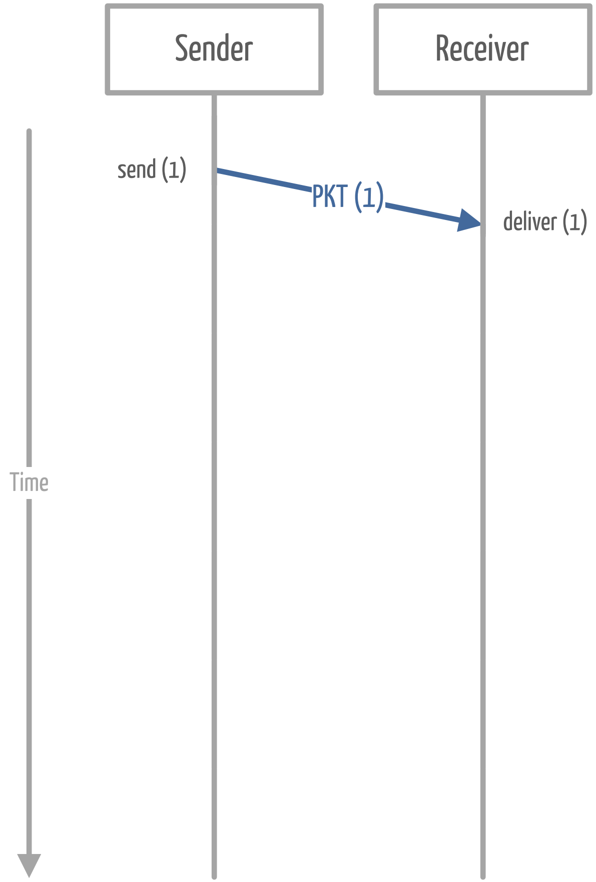
Sender

When data from sender gets pushed, sent to channel.

Receiver

When data on channel arrives, push to application layer.

RDT 1.0 | Example

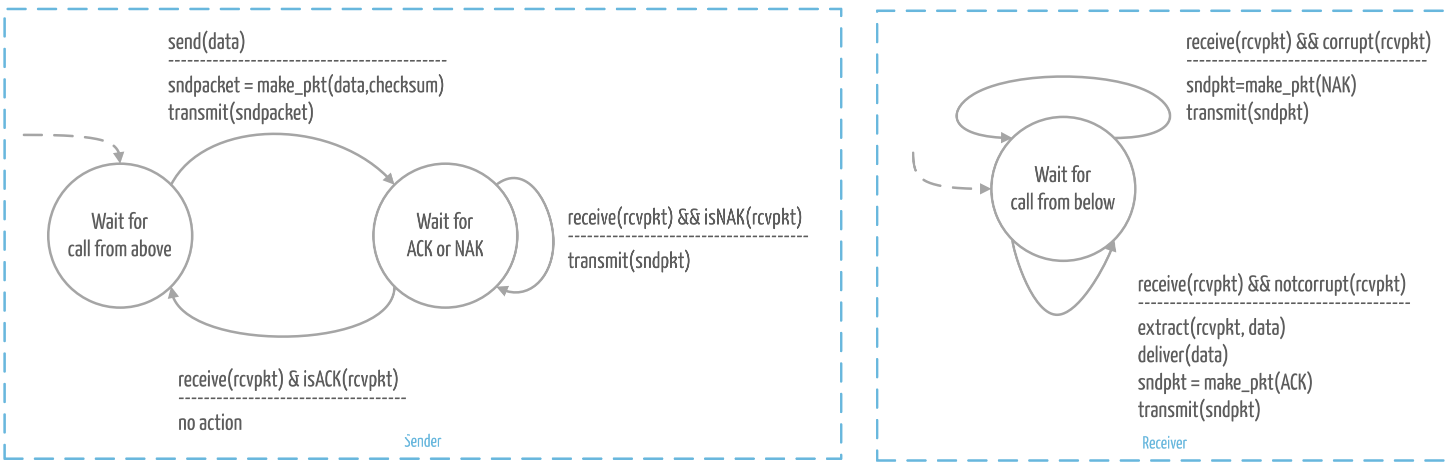


RDT 2.0 | Bit Errors

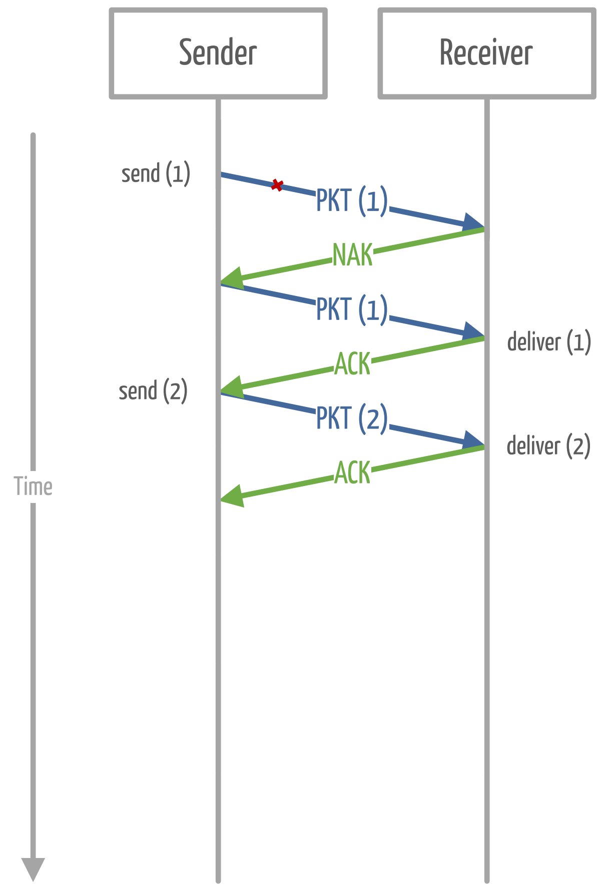
↔ **Channel:** Flip bits in packet.

Approach:

- Introduce acknowledgement (ACK) and negative acknowledgement (NAK) message to acknowledge correct reception of data or indicate error.
- Checksums are sent along and used to detect bit errors.
- New protocol features: Error Detection and Feedback (receiver talks to sender).



RDT 2.0 | Example



RDT 2.0 | Problems

What happens if ACK/NAK corrupted?

- Sender has no clue what really happened at the receiver.
- Retransmission not possible: Might cause duplicates at receiver.

Duplicate Handling

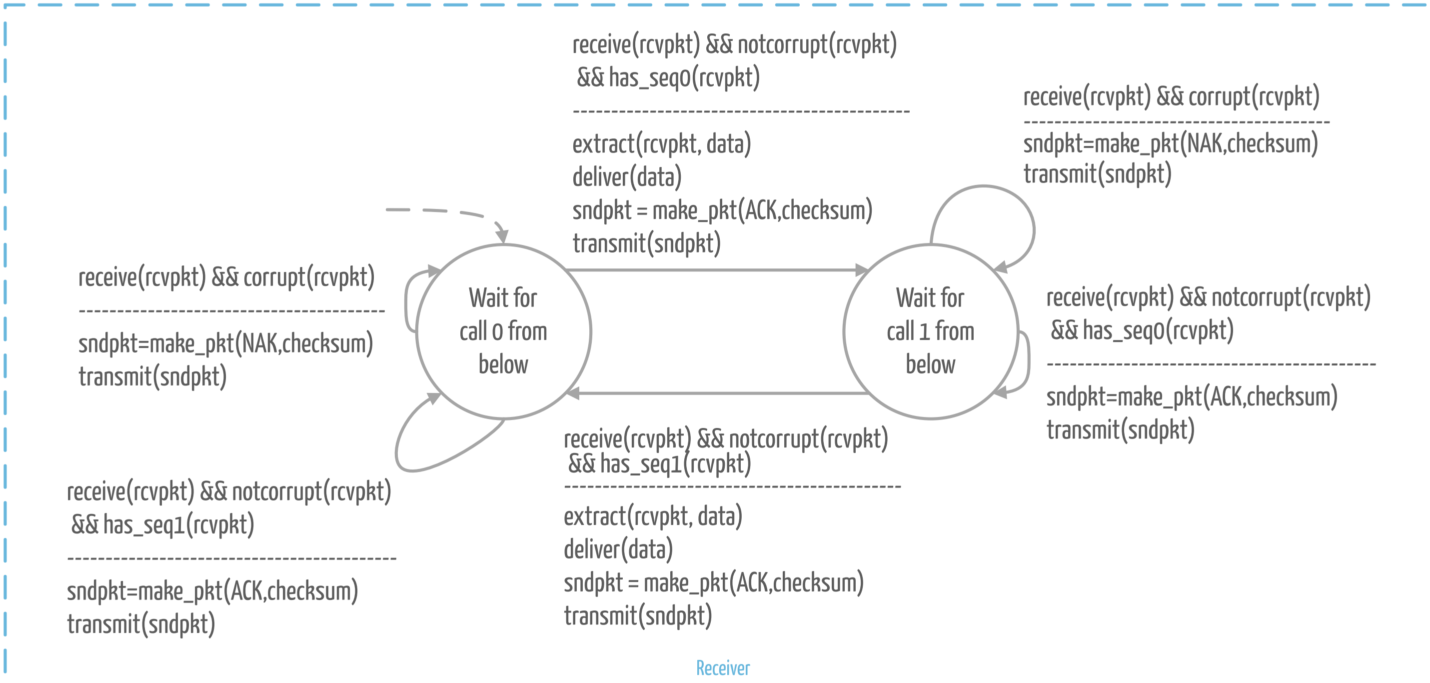
- Sender retransmits current packet if ACK/NAK is corrupted.
- Packets get an additional *sequence number* for identification.
- Receiver discards duplicate packets, not delivering them to the app.

Stop-and-Wait: Sender sends one packet and waits for receiver's response.

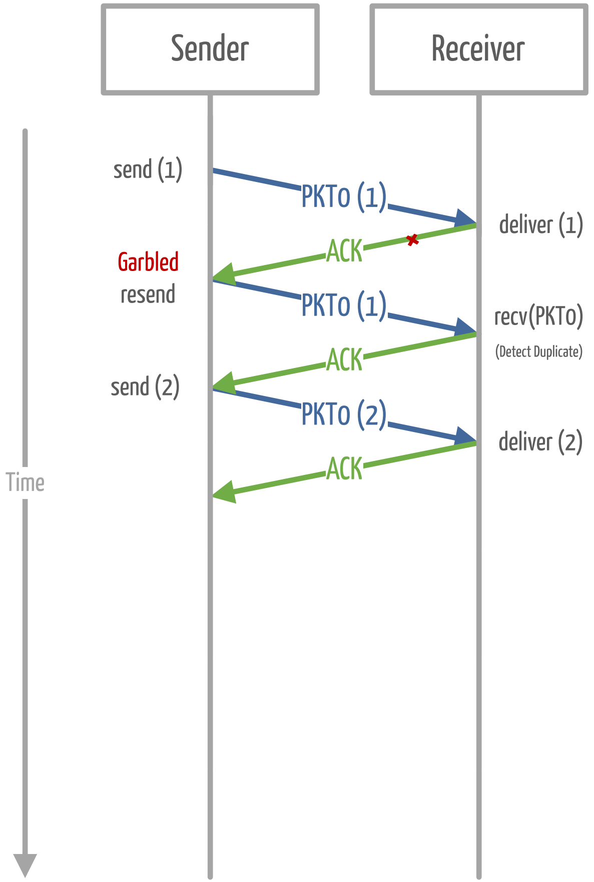
RDT 2.1 | Garbled ACK/NAKs - Sender

↔ **Channel:** Flip bits in packet and in feedback.

RDT 2.1 | Garbled ACK/NAKs - Receiver



RDT 2.1 | Example



RDT 2.1 | Discussion

Sender

- Sequence number added to packet.
- Two sequence numbers (0,1) are sufficient.
- Checks for corrupted ACK/NAK necessary.
- Double number of states:
    - State indicates which is the expected sequence number.

Receiver

- Checks for duplicates added.
    - State indicates which is the expected packet sequence number.
- Receiver unconscious if last ACK/NAK was received by sender.

NAKs can be avoided by using ACK packet that include the sequence number of last receiver packet. Duplicate ACKs at receiver cause same action as NAK: retransmit current packet.

RDT 3.0 | Erroneous and Lossy Channel

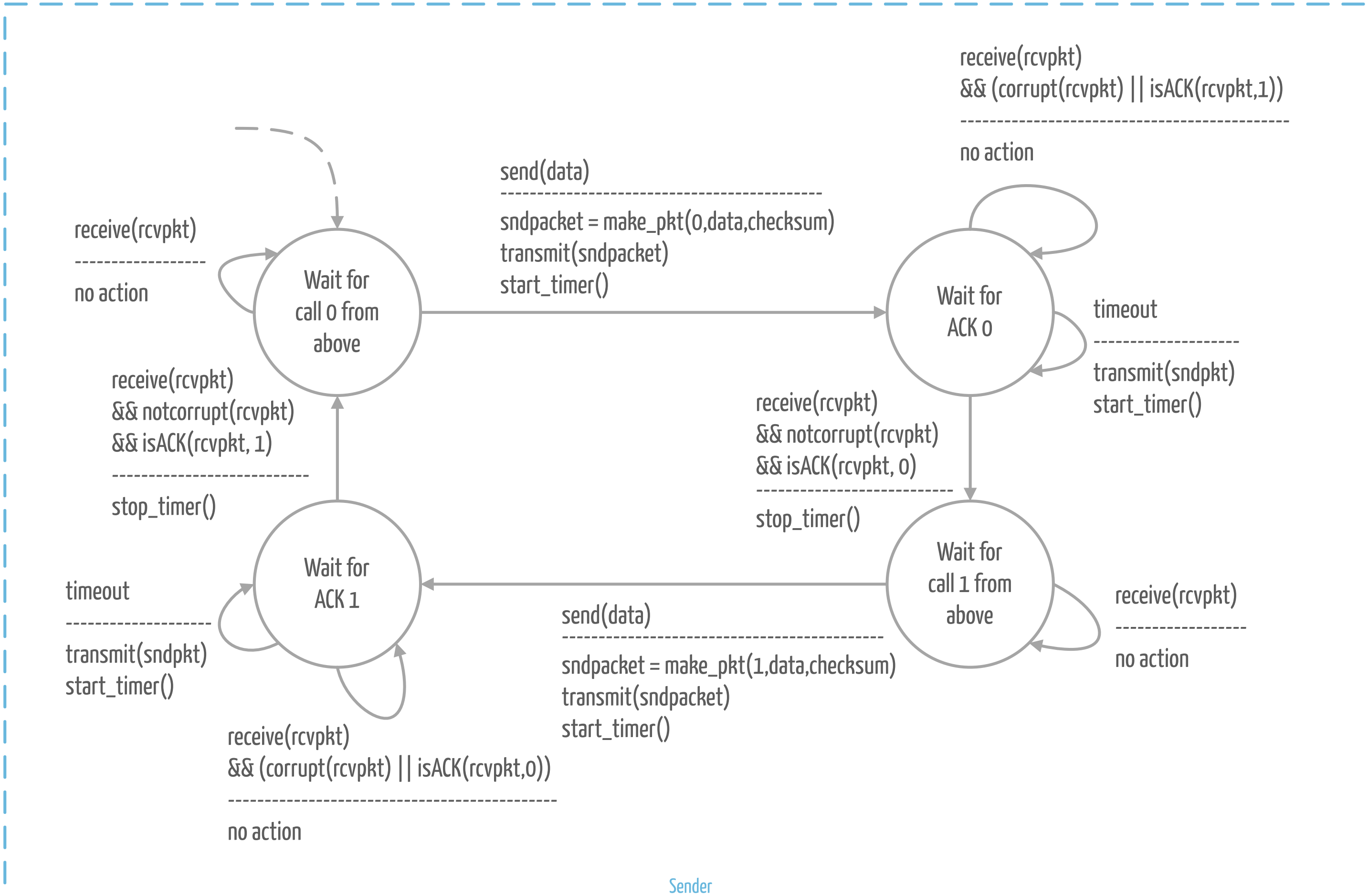
↔ **Channel:**

- Flip bits in packet or feedback.
- Lose complete packets or feedback.

❓ **What to do now?**

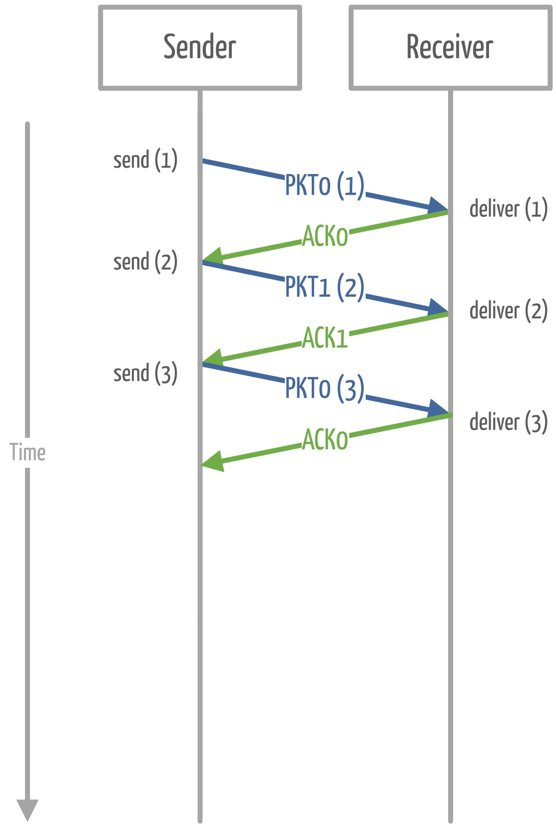
- Sender waits for "reasonable" amount of time for ACK.
- Retransmits if no ACK received in this time (assume loss).
- If packet or ACK just delayed and not lost:
  - Retransmission will be duplicate, but receiver can handle this using sequence numbers.
  - Receiver must specify which packet sequence number is ACKed.
- Requires countdown timer.

RDT 3.0 | Sender

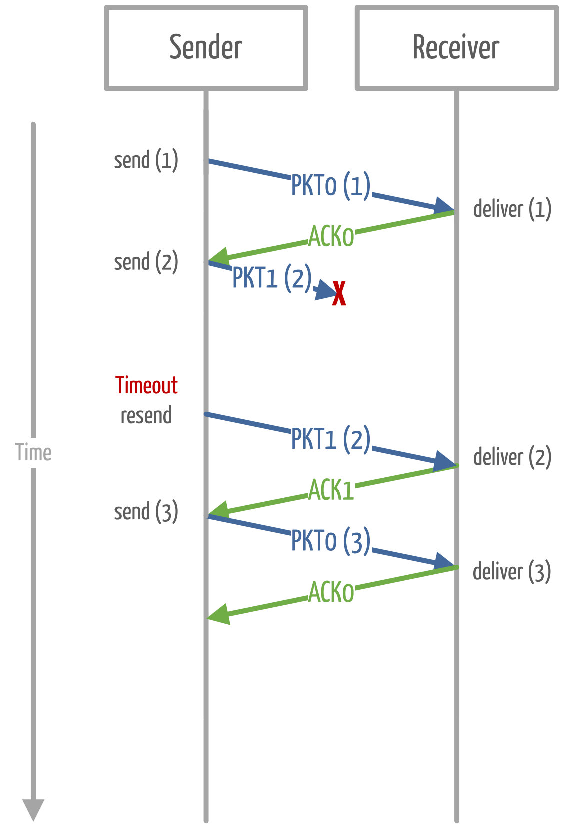


RDT 3.0 | Examples 1/2

1) No Loss

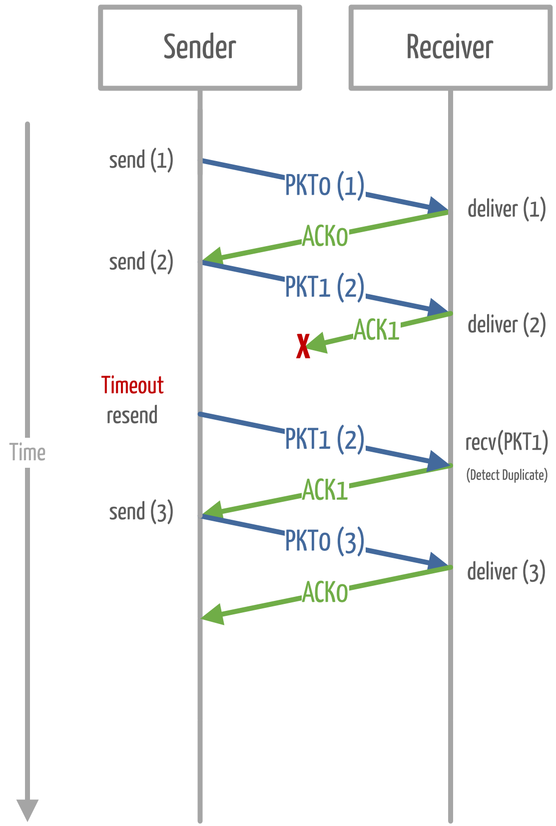


2) Packet Loss

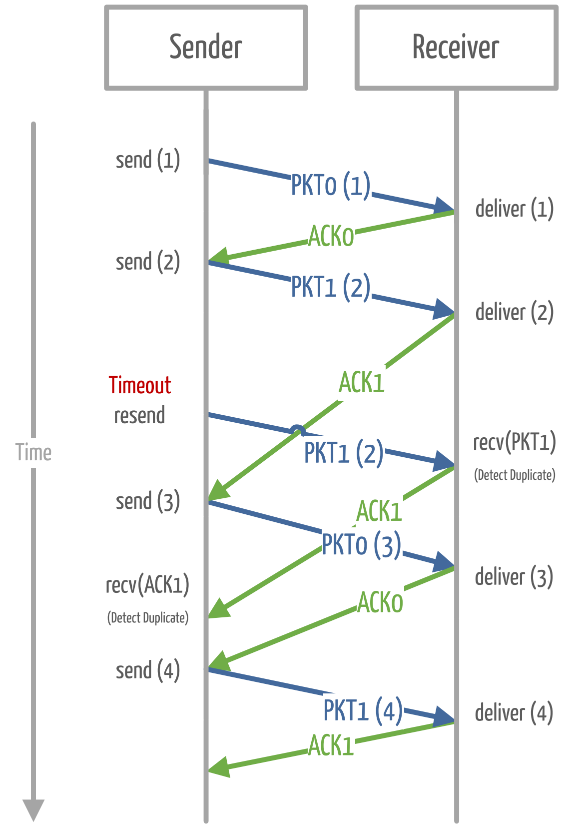


RDT 3.0 | Examples 2/2

3) Ack Loss

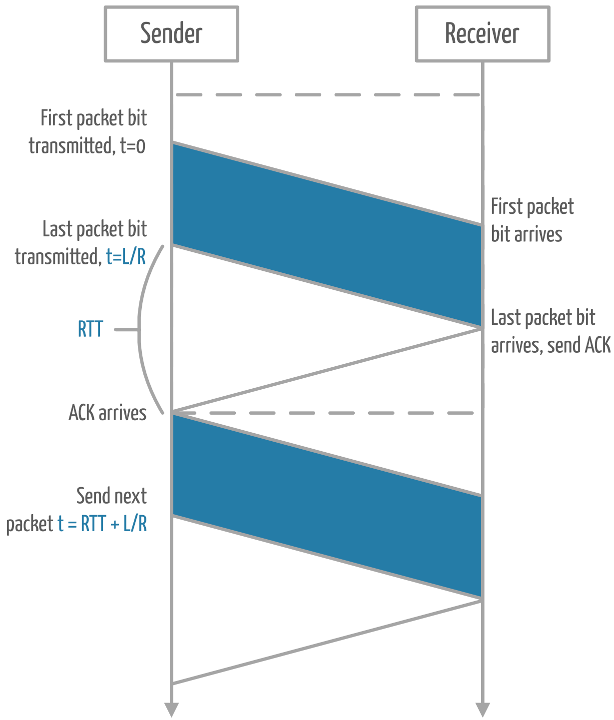


4) Premature Timeout / Delayed Ack



Performance: Stop-and-Wait

Operation



Calculation

- 1Gbps Link, 15ms propagation delay, 8000 bit packet.
- $D_{Trans} = \frac{L}{R} = \frac{8000bits}{10^9 bits/s} = 8\mu s$
- $U_{Sender} = \frac{L/R}{RTT+L/R} = \frac{0.008}{30.008} = 0.00027$
- Consequently:
    - $RTT = 30ms$, 1kB packet every 30ms.
    - 33kBps throughput over 1Gbps link.

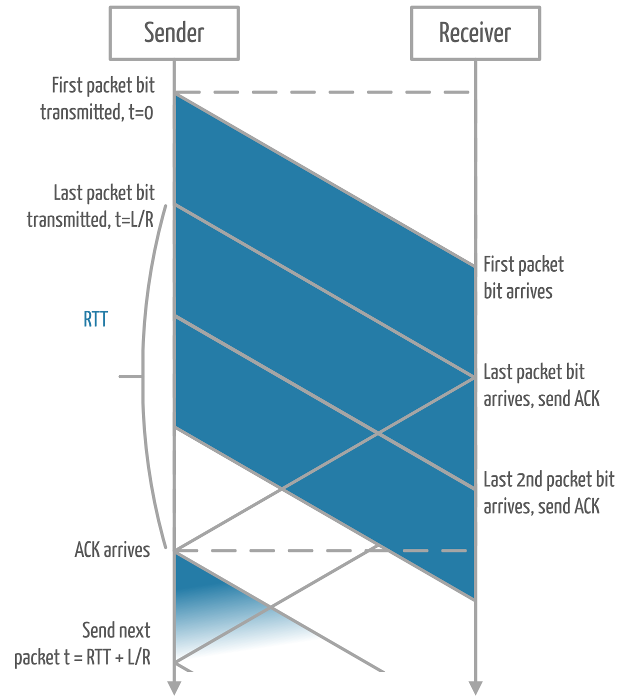
> Stop-and-wait protocols are not good performance-wise. Network resource usage limited by protocol.

Pipelining

Motivation

Problem: These protocols are very inefficient, as single packets are sent.

Idea: Multiple packets in-flight.



$$U_{Sender} = \frac{3 \cdot L/R}{RTT + L/R} = \frac{0.024}{30.008} = 0.00081$$

Pipeline / Streaming Pattern

Context:

- Work on data runs over multiple consecutive stages.
- Data can be segmented in parts that can be processed independently.

Implementation:

- Segment the data in blocks.
- Consider feeding in the blocks into the first step as a zeroth step.
- For each step:
  - As soon as block is processed, forward result to next step.
  - Wait for results from the previous step to arrive.

Benefits:

- Results can be provided early (when the first block is processed at the last stage).
- If steps can run in parallel, time can be saved.

Drawbacks:

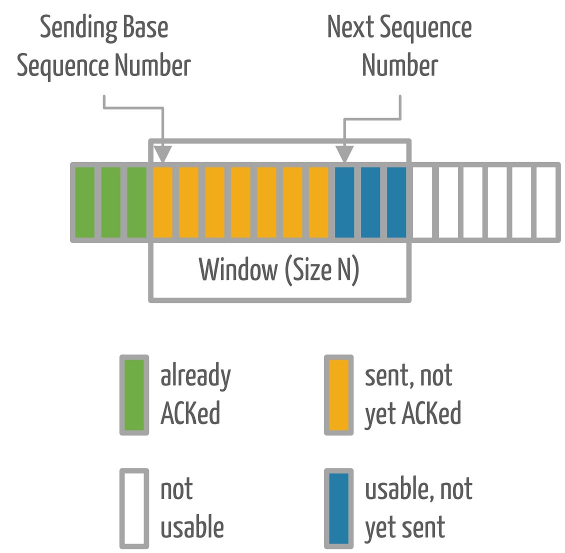
- Not necessarily speeding up process.
- Overhead due to segmenting.

Similar Patterns:

- *Full Parallelization* (which requires multiple workers per stage).
- *Batch Processing* (accumulating data units to process them at once).

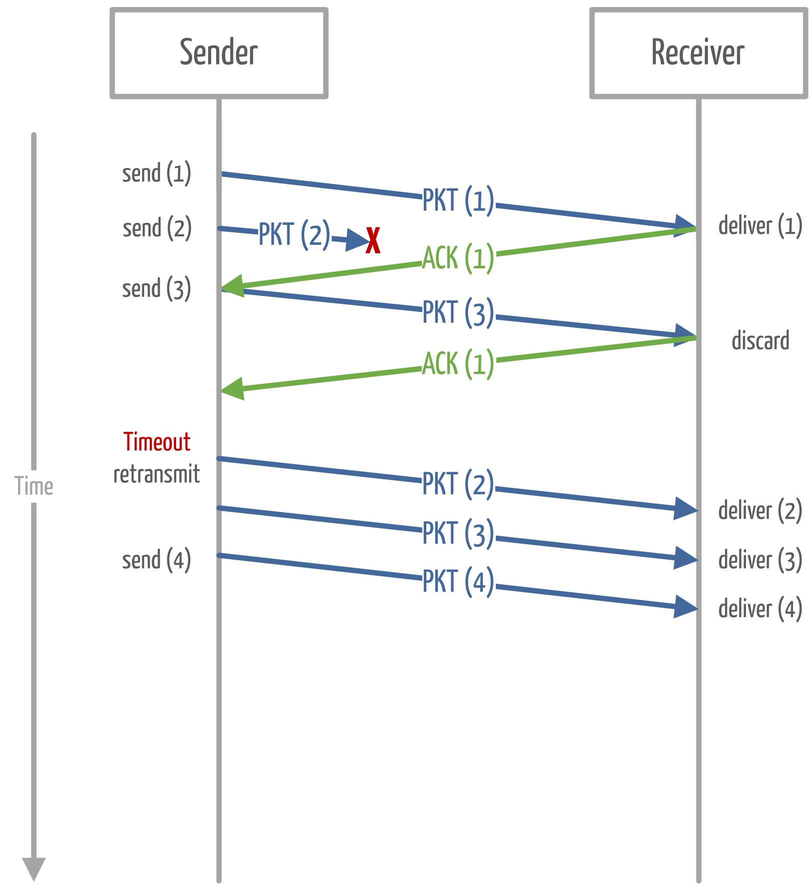
Go Back N

- Sender sends up to N non-acknowledged segments.
- N is the *window size*.
- Recipient sends *cumulative ACK* ("all until X is received").
- If segment X is lost, receiver discards all following (X+1...N) segments.
- Sender repeats sending from X (after timeout, NAK or duplicate ACKs).



Good for **high-bandwidth low-delay** networks.

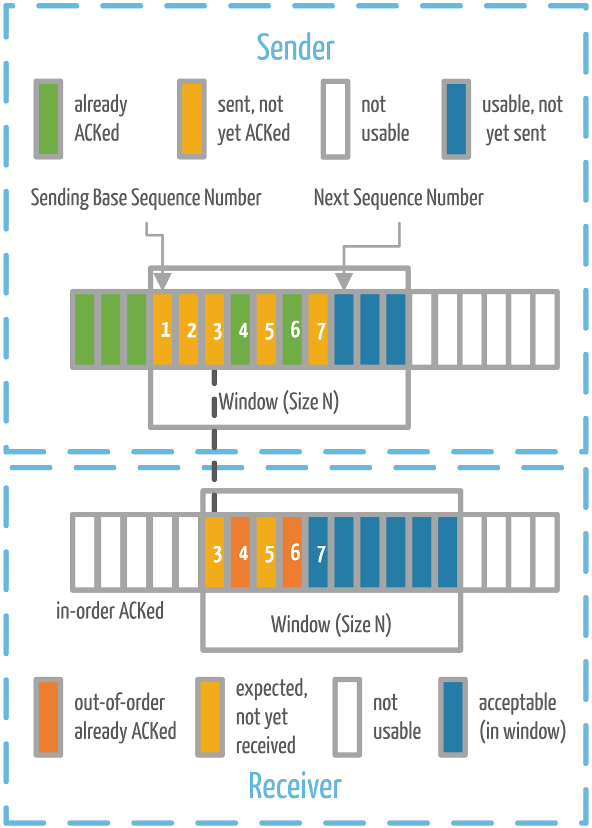
Go Back N | Example



Selective Repeat

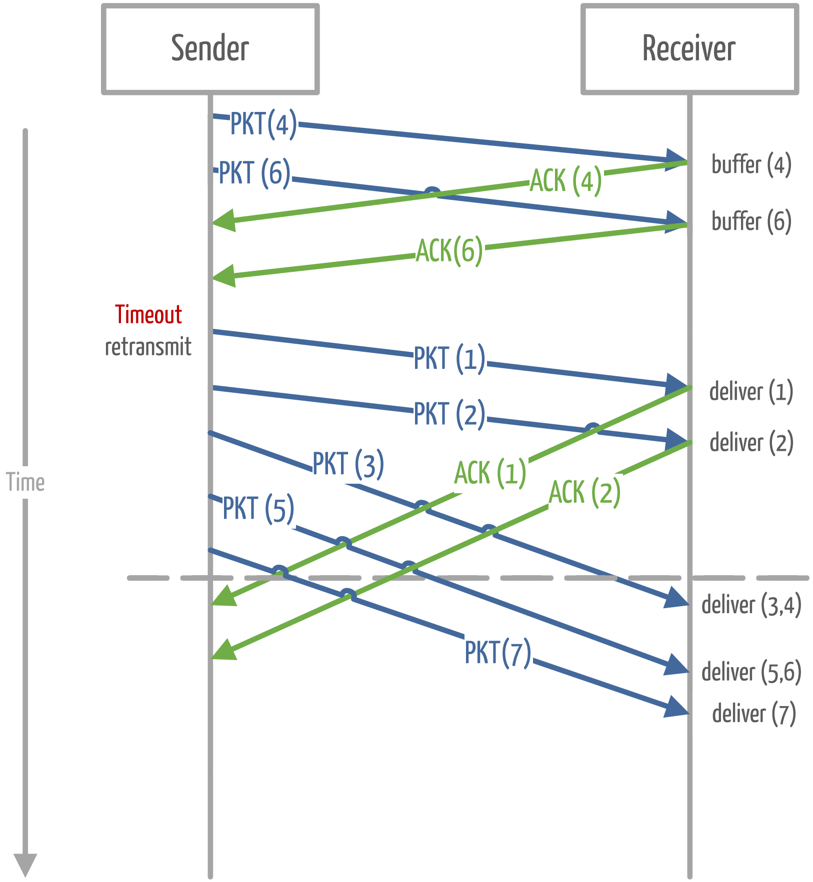
- Sender sends up to N non-acknowledged segments.
- Receiver acknowledges received segments and buffers them (including out-of-order segments).
- Sender repeats non-ACKed segments.

Good for **low-bandwidth** networks.



HON

Selective Repeat | Example



Quiz

Assume receiver gets a duplicate segment, which it already ACKed. Should it send another ACK for this segment?

A: Yes.

B: No.

C: Don't care.

Answer:

A: It has to! ACKs might get lost so the sender might not know that something got lost.

B: Even though you might think this adds additional bandwidth... it is a must!

C: Nope, it has to!

Wrap-Up

Questions?

Take-Home Messages

- Transport layer provides process-to-process multiplexing.
- Transport layer implements important network functions.
- Reliable Data Transfer (rdt) is an important topic and has to be done properly.
- Pipelining is a helpful approach to make protocols efficient.

Further Reading

- Kurose-Ross "Computer Networking"
  - Sec. 3.1 - 3.2 (Transport Layer Services)
  - Sec. 3.4 (Reliable Data Transfer)

Copyright and Acknowledgement

- Some examples and parts of the content are taken from the book Computer Networking as well as the slide deck by James Kurose and Keith Ross.
- The **material is copyrighted**. Please treat the slides accordingly and do not share them.