CS 352
Reliability: Stop and Wait

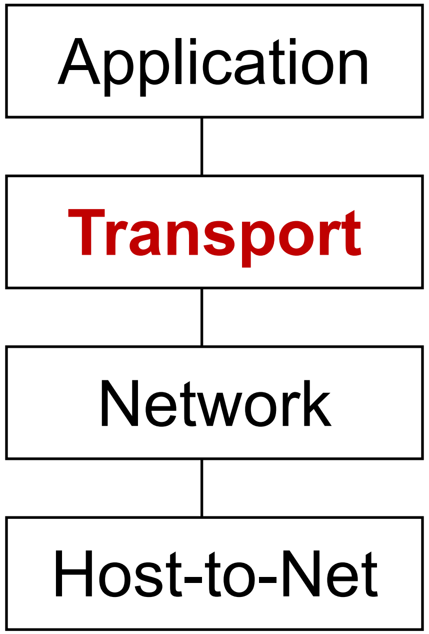
CS 352, Lecture 9.1

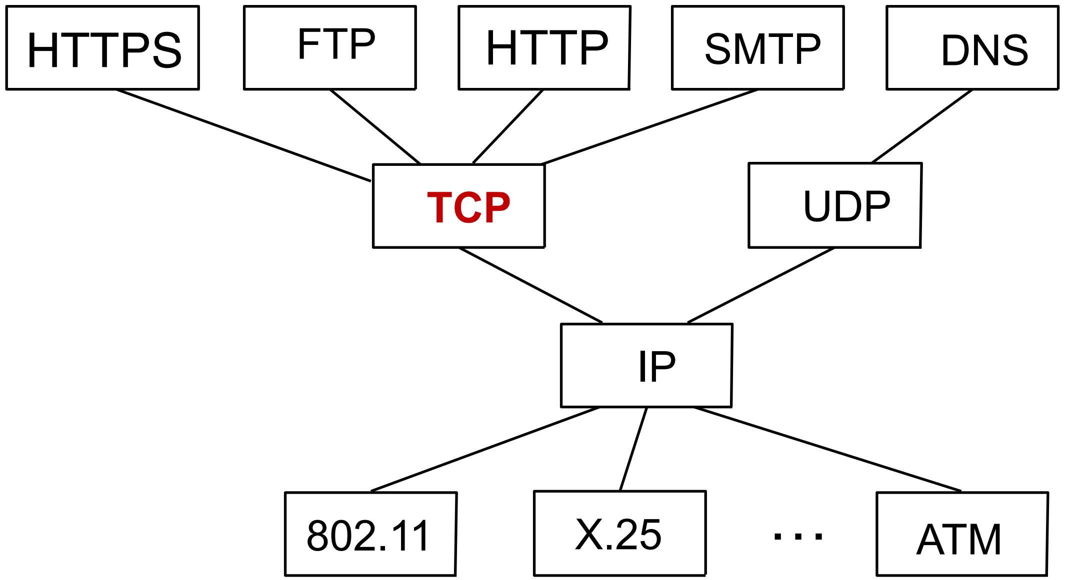
http://www.cs.rutgers.edu/~sn624/352

Srinivas Narayana

RUTGERS
UNIVERSITY | NEW BRUNSWICK

Transport


Application
Transport
Network
Host-to-Net


HTTPS
FTP
HTTP
SMTP
DNS
TCP
UDP
IP
802.11
X.25
…
ATM

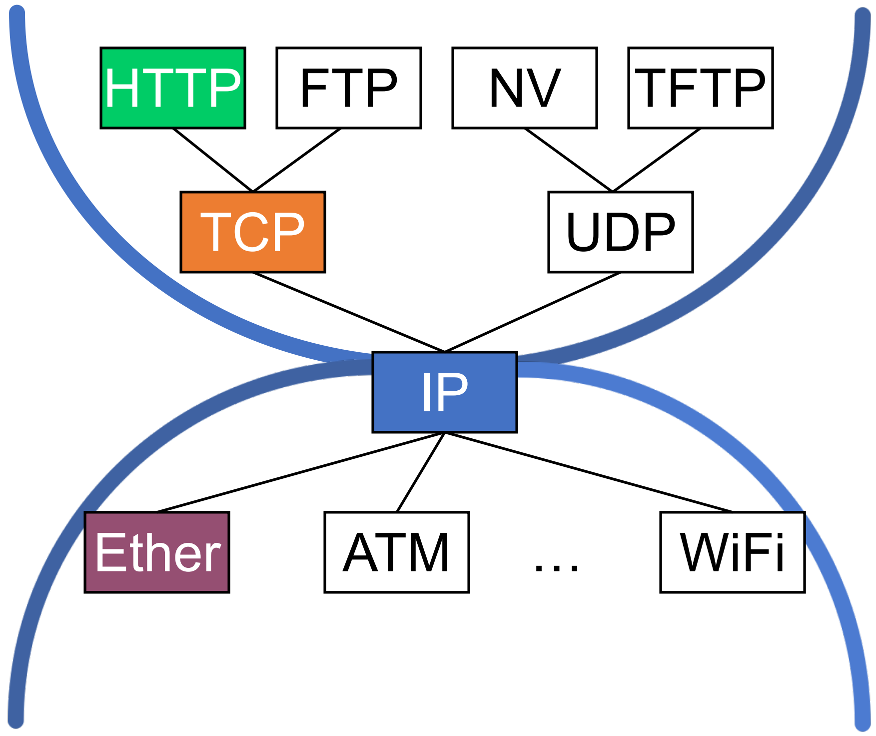
Modularity through layering

Apps: useful user-level functions

Transport: provide guarantees to apps

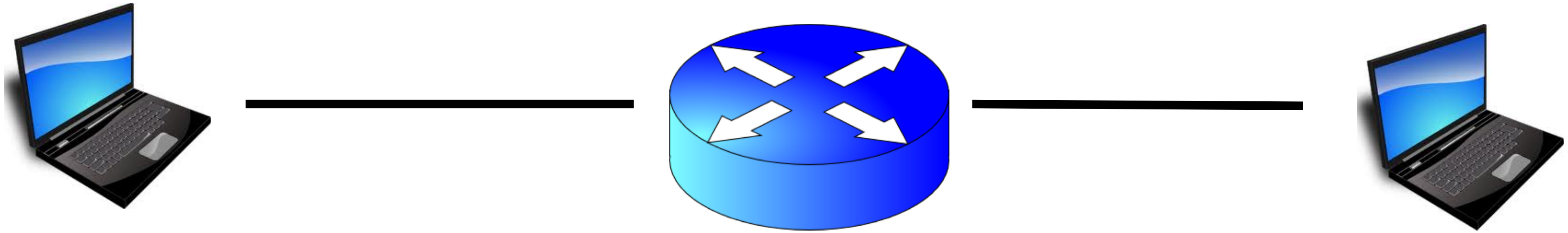
Network: best-effort global pkt delivery

Link: best-effort local pkt delivery

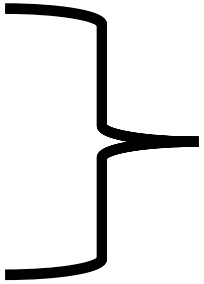


How do apps get perf guarantees?

- The network core provides no guarantees on packet delivery



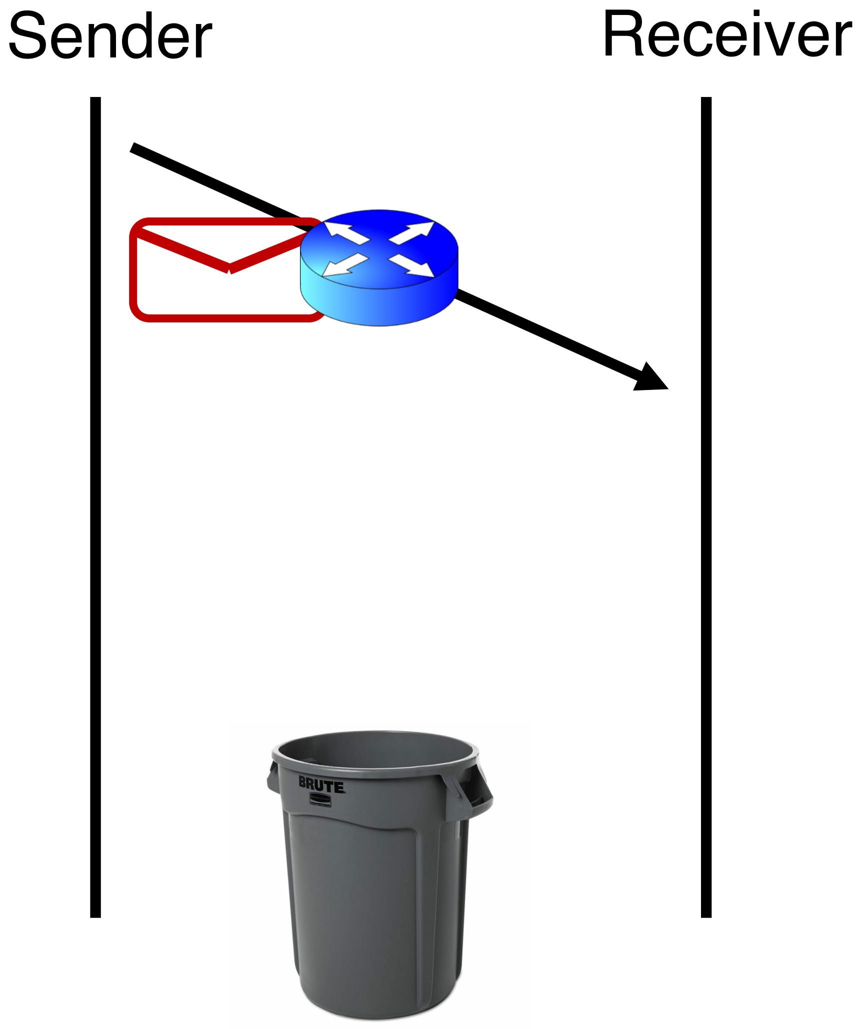
- Transport software on the endpoint oversees implementing guarantees on top of a best-effort network
- Three important kinds of guarantees
  - Reliability
  - Ordered delivery
  - Resource sharing in the network core



Transmission Control Protocol (TCP)

Reliable data delivery

Packet loss



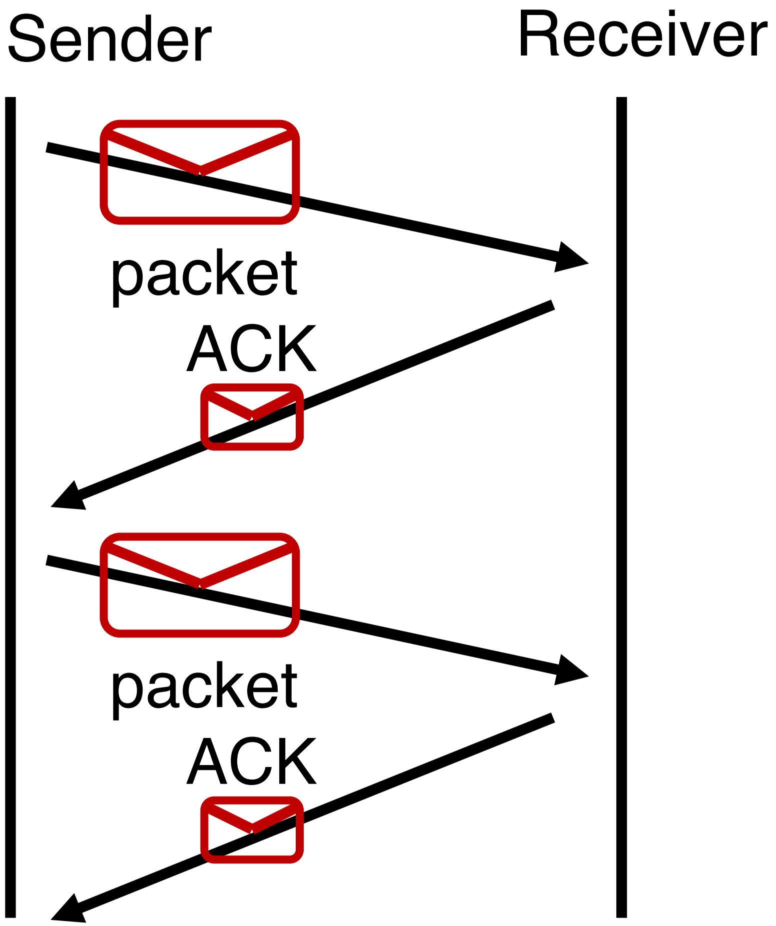
- How might a sender and receiver ensure that data is delivered reliably (despite some packets being lost)?

- TCP uses three mechanisms

Coping with packet loss: (1) ACK

- Key idea: Receiver returns an acknowledgment (ACK) per packet sent

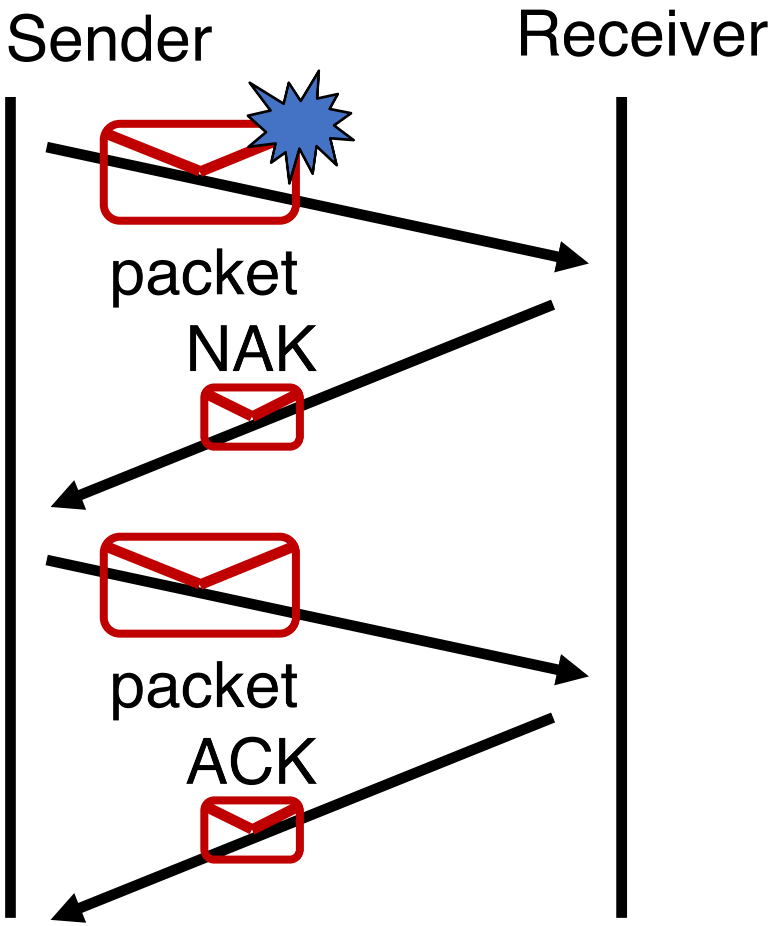
- If sender receives an ACK, it knows that the receiver got the packet.



Coping with packet corruption: (1) ACK

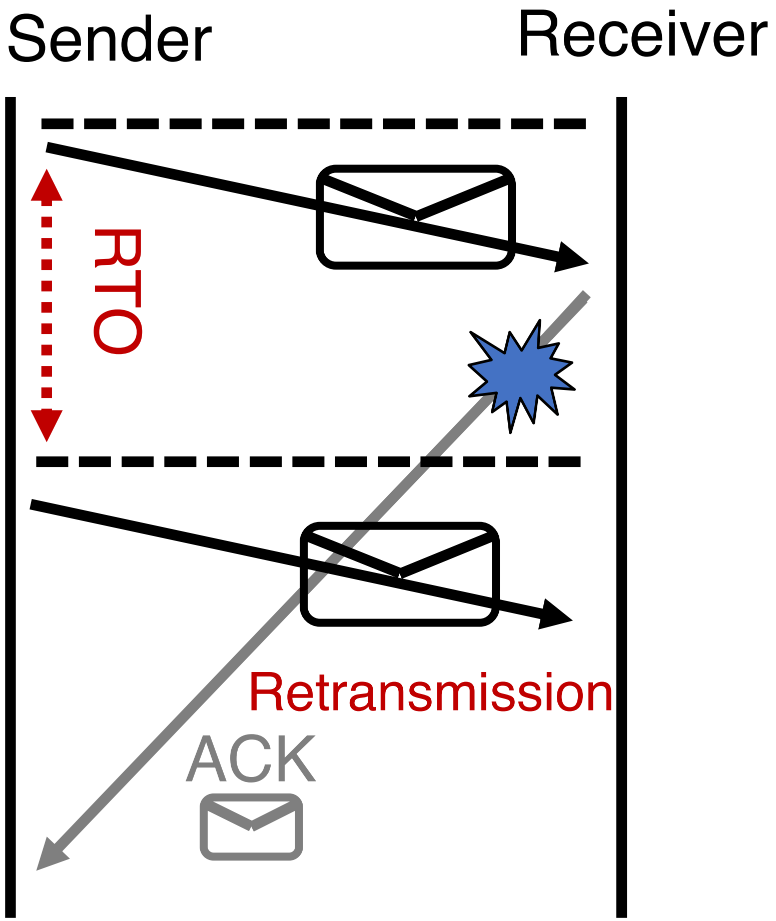
- ACKs also work to detect packet corruption on the way to the receiver
  - One possibility: A receiver could send a negative acknowledgment, or a NAK, if it receives a corrupted packet
  - Q: How to detect corrupted packet?
    - One method: Checksum!

- TCP only uses positive ACKs.



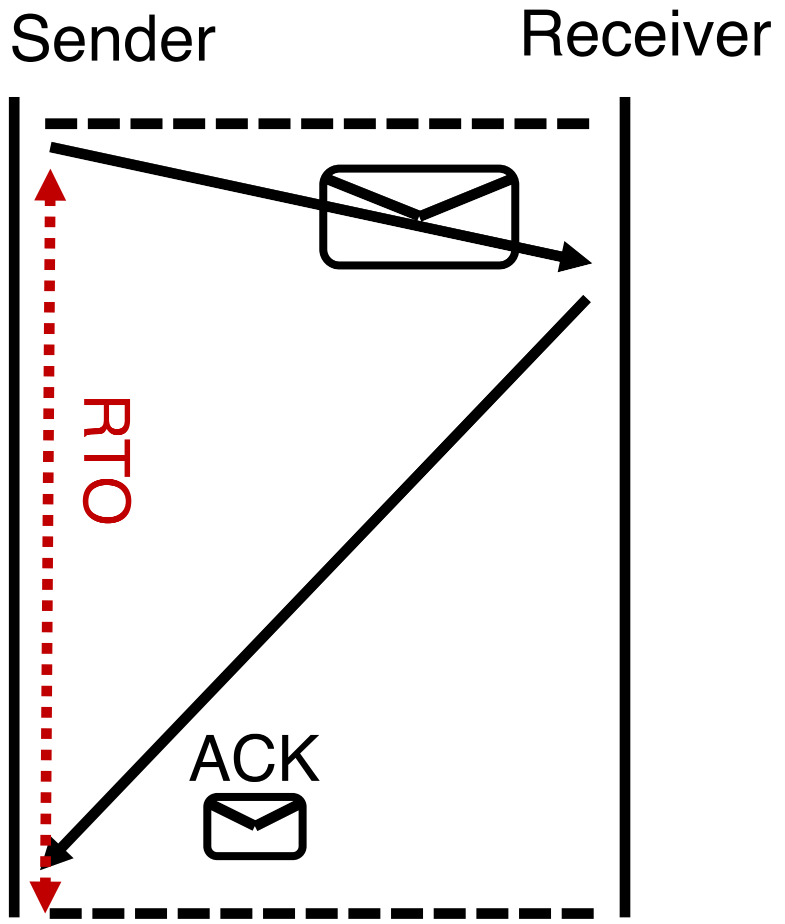
Coping with packet loss: (2) RTO

- What if a packet is dropped?
- Key idea: Wait for a duration of time (called retransmission timeout or RTO) before re-sending the packet
- In TCP, the onus is on the sender to retransmit lost data when ACKs are not received
- Note that retransmission works also if ACKs are lost or delayed



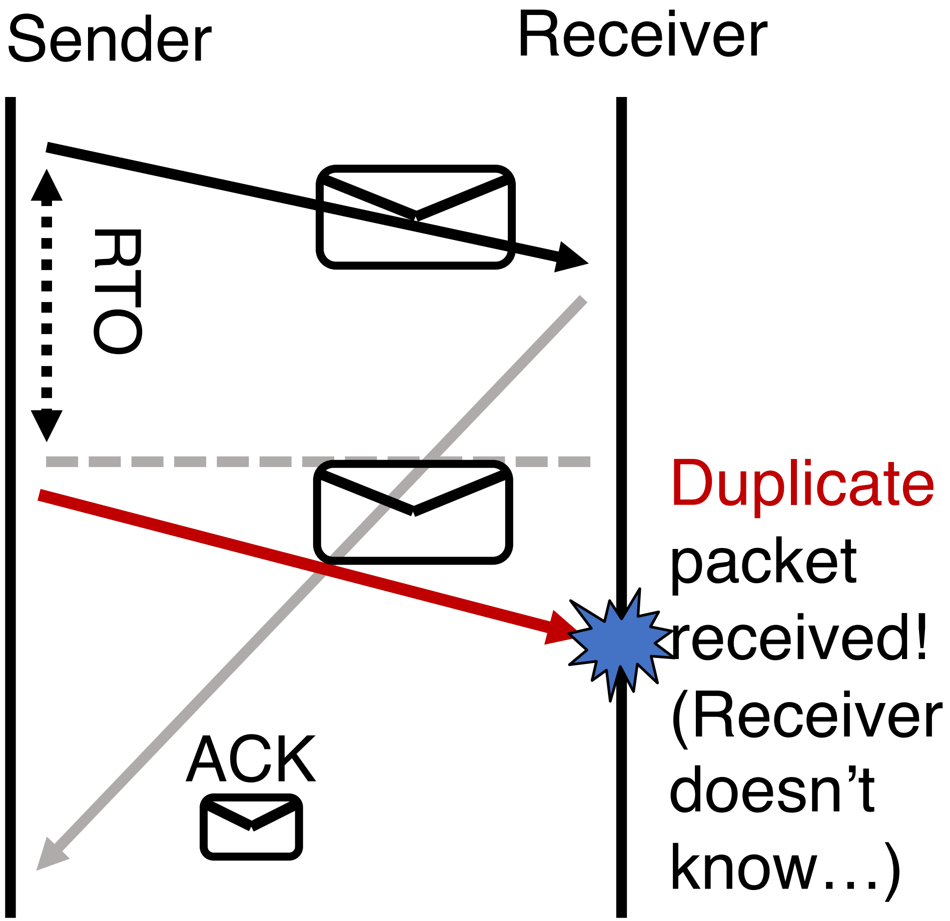
How should the RTO be set?

- A good RTO must predict the round-trip time (RTT) between the sender and receiver
  - RTT: the time to send a single packet and receive a (corresponding) single ACK at the sender
- Intuition: If an ACK hasn't returned, and our (best estimate of) RTT has elapsed, the packet was likely dropped.
- RTT can be measured directly at the sender. No receiver involvement needed.



Coping with packet duplication

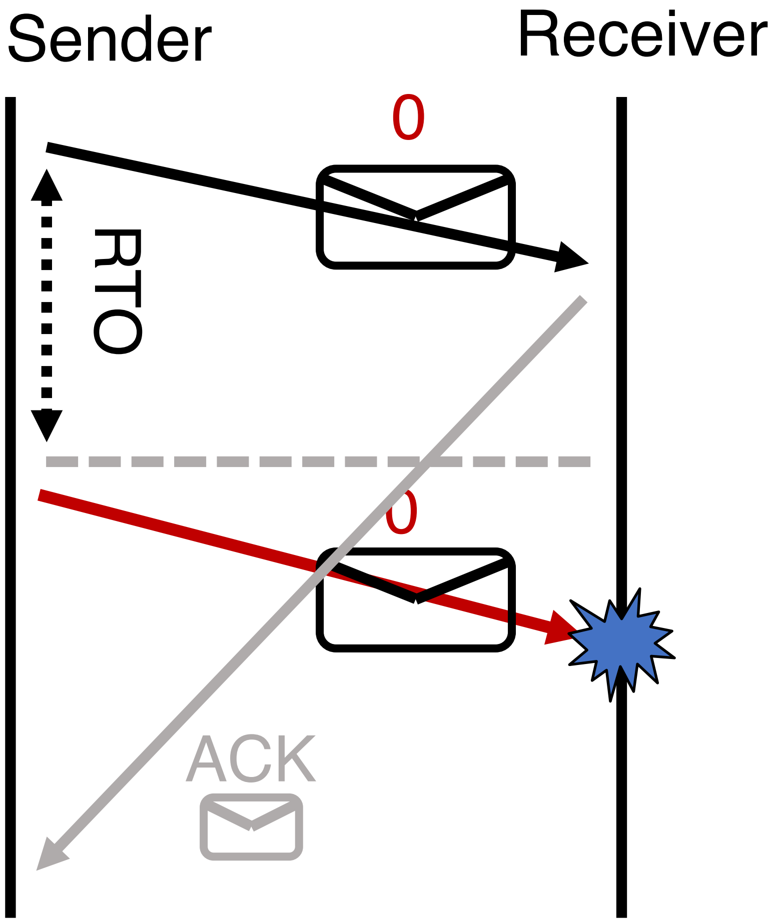
- If ACKs delayed beyond the RTO, sender may retransmit the same data
  - Receiver wouldn't know that it just received duplicate data from this retransmitted packet
- Add some identification to each packet to help distinguish between adjacent transmissions
  - This is known as the sequence number



Coping with packet loss: (3) Sequence #s

- A bad scenario: Suppose an ACK was delayed beyond the RTO; sender ended up retransmitting the packet.

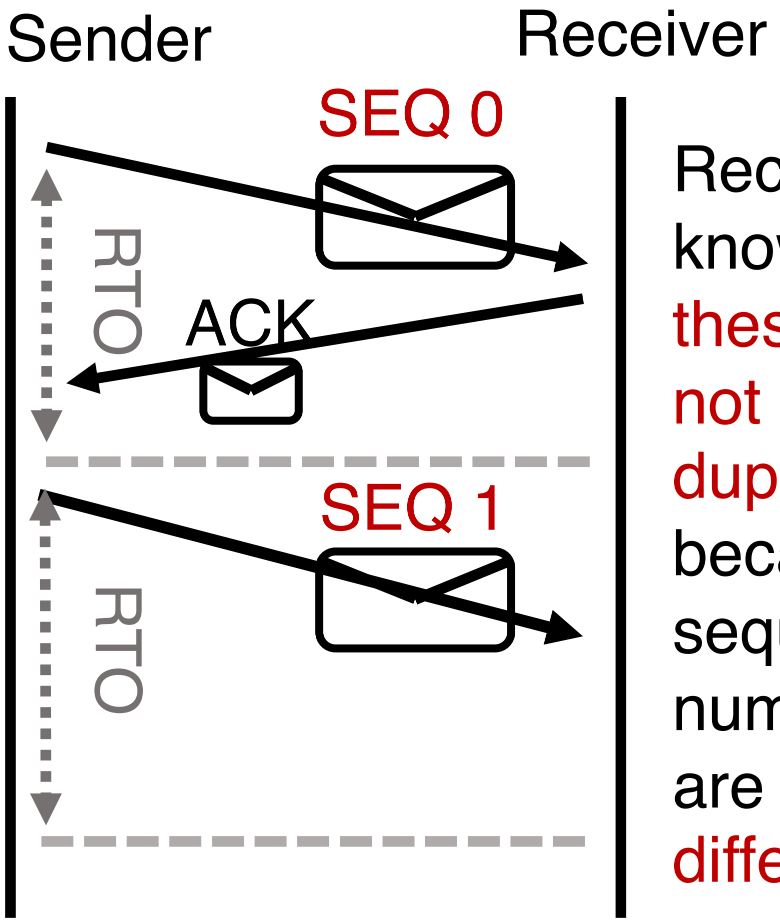
- At the receiver: sequence number helps disambiguate a fresh transmission from a retransmission
  - Sequence number same as earlier: retransmission
  - Fresh sequence number: fresh data



Coping with packet loss: (3) Sequence #s

- A good scenario: packet successfully received and ACK returned within RTO

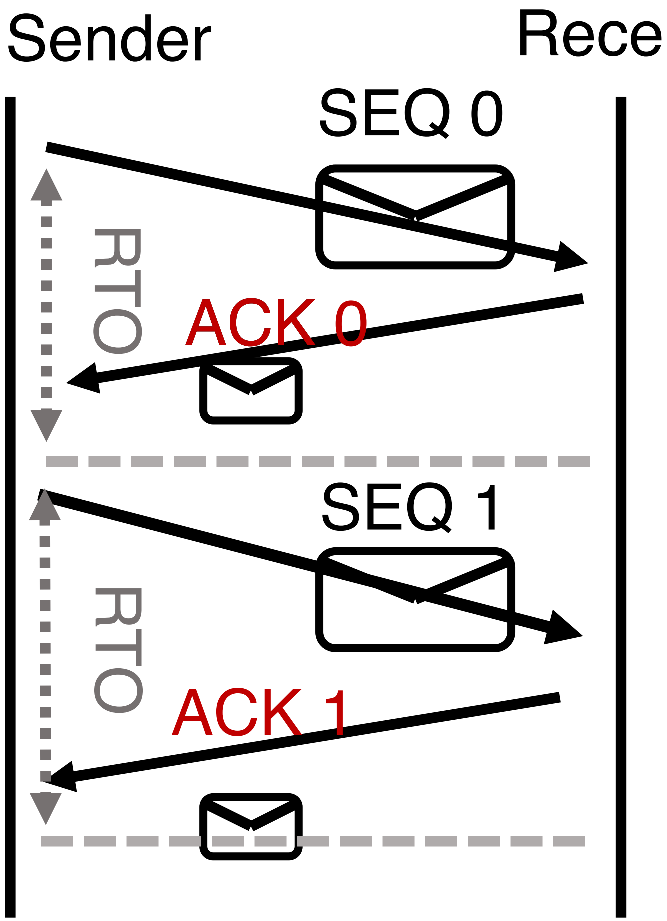
- Sequence numbers of successively transmitted packets are different



Receiver knows these are not duplicate, because sequence numbers are different

Coping with packet loss: (3) Sequence #s

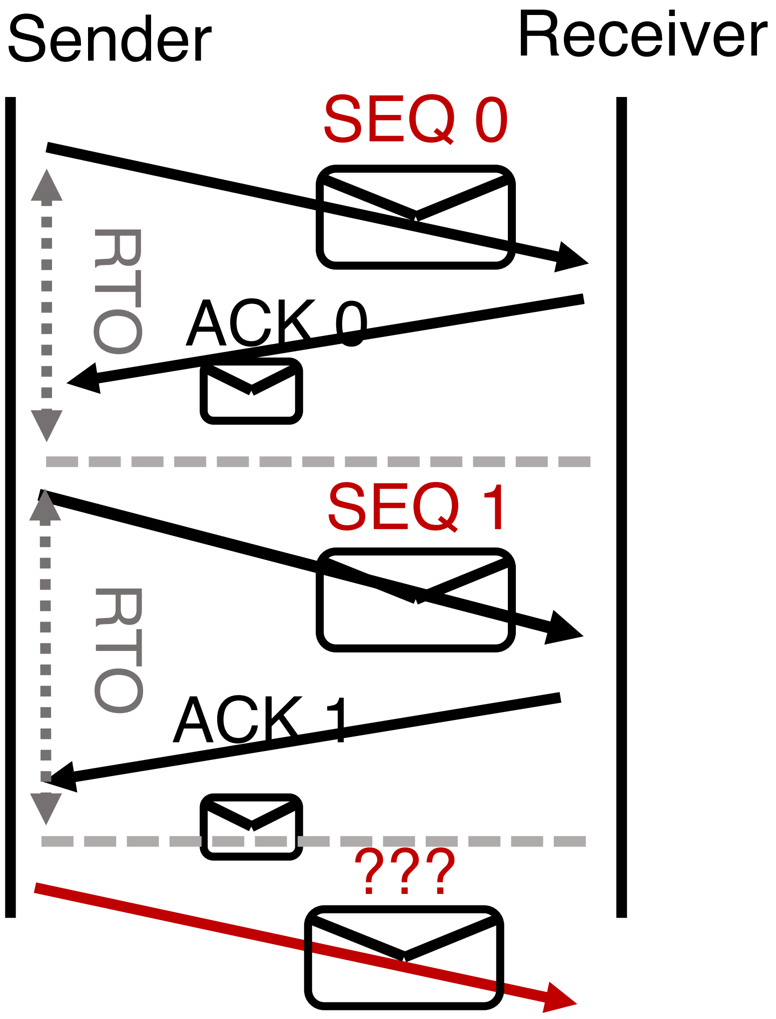
- A good scenario: packet successfully received and ACK returned within RTO
- Sequence numbers of successively transmitted packets are different
- Further, the receiver informs the sender which packet was ACK'ed using an ACK sequence number



Receiver knows these are not duplicate, because sequence numbers are different

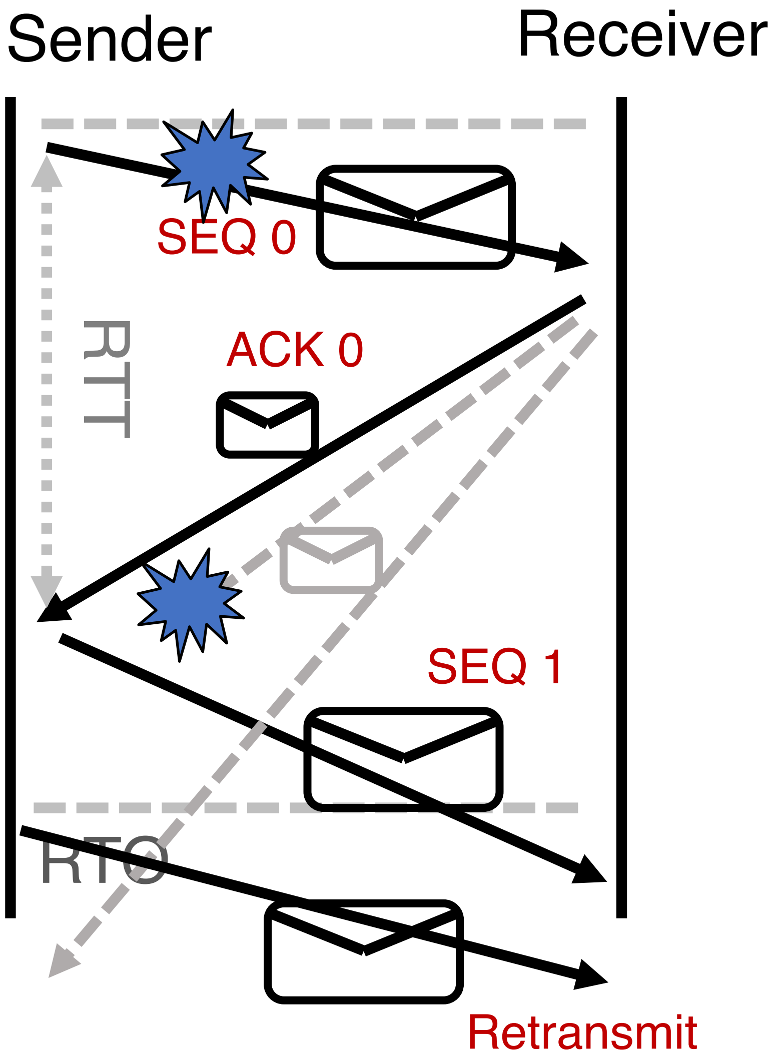
Q: What is the seq# of third packet?

- Goal: Avoid ambiguity on which packet was received/ACK'ed from both the sender and receiver's perspective
- One possibility: keep incrementing the seq #: 2, 3, …
- Alternative: since seq # 0 was successfully ACK'ed earlier, it is OK to reuse seq #0 for next transmission.
  - Seq #s reused if enough time elapsed



Summary: Stop-and-Wait Reliability

- Sender sends a single packet, then waits for an ACK to know the packet was successfully received. Then the sender transmits the next packet.
- If ACK is not received until a timeout (RTO), sender retransmits the packet
- Disambiguate duplicate vs. fresh packets using sequence numbers that change on "adjacent" packets



CS 352
Reliability: TCP Metadata

CS 352, Lecture 9.2

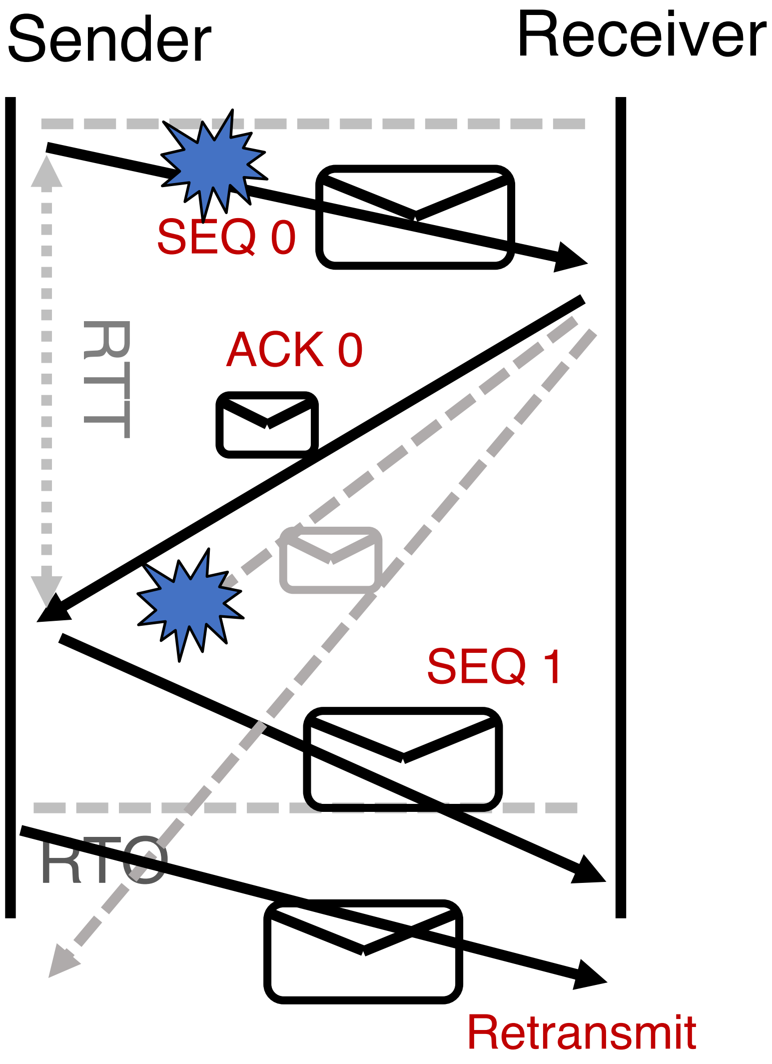
http://www.cs.rutgers.edu/~sn624/352

Srinivas Narayana

RUTGERS
UNIVERSITY | NEW BRUNSWICK

Review: Stop-and-Wait Reliability

- Sender sends a single packet, then waits for an ACK to know the packet was successfully received. Then the sender transmits the next packet.
- If ACK is not received until a timeout (RTO), sender retransmits the packet
- Disambiguate duplicate vs. fresh packets using sequence numbers that change on “adjacent” packets



Q1: Where are seq & ACK #s written to?

- Naturally, in the packet header!

TCP header structure

Source port, destination port (connection demultiplexing)

Size of the TCP header (in 32-bit words)

Basic error detection through checksums (similar to UDP)

```
 0                   1                   2                   3
 0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|          Source Port          |       Destination Port        |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|                        Sequence Number                        |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|                    Acknowledgment Number                      |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|  Data |           |U|A|P|R|S|F|                               |
| Offset| Reserved  |R|C|S|S|Y|I|            Window             |
|       |           |G|K|H|T|N|N|                               |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|           Checksum            |         Urgent Pointer        |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|                    Options                    |    Padding    |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|                             data                              |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+

                    TCP Header Format

      Note that one tick mark represents one bit position.
```

TCP header structure

Identifies data in the packet from sender's perspective

TCP uses byte seq #s

Identifies the data being ACKed from the receiver's perspective.

TCP uses next seq # that the receiver is expecting.

```
 0                   1                   2                   3
 0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|          Source Port          |       Destination Port        |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|                        Sequence Number                        |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|                    Acknowledgment Number                      |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|  Data |           |U|A|P|R|S|F|                               |
| Offset| Reserved  |R|C|S|S|Y|I|            Window             |
|       |           |G|K|H|T|N|N|                               |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|           Checksum            |         Urgent Pointer        |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|                    Options                    |    Padding    |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|                             data                              |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+

                        TCP Header Format

      Note that one tick mark represents one bit position.
```

A TCP exchange

- A small demo