Reliable Data Transfer Protocols (Rdt)

Lecturer: Islam Zakaria

Resources

Chapter 23: Introduction to Transport Layer

Book: **Data.Communications.and.Networking.5th.Edition**

Goal of Reliable Data Transfer

Provide a **reliable communication service** over an **unreliable channel**.

The underlying network (like IP) may:

- Lose packets
- Corrupt bits
- Duplicate packets
- Deliver packets out of order

The **Transport Layer Protocol** (like TCP) must handle all these issues so that applications see a **clean, reliable byte stream**.

Key Principles / Mechanisms

Principle	Description	Purpose
Error Detection	Detects bit errors using checksums or CRCs.	Identify corrupted packets.
Acknowledgments (ACKs)	Receiver confirms successful receipt.	Ensures sender knows data arrived correctly.
Retransmission (ARQ)	Sender retransmits lost or corrupted data.	Guarantees delivery.
Sequence Numbers	Number each packet.	Detect duplicates and ensure correct ordering.
Timers	Used with ACKs to detect losses.	Trigger retransmissions after timeout.
Flow Control	Ensures sender doesn’t overwhelm receiver.	Prevent buffer overflow.
Congestion Control	Regulates traffic when network is overloaded.	Prevents packet loss due to congestion.

Reliable Data Transfer Protocols (Conceptual Evolution)

Rdt1.0: reliable transfer over a reliable channel

- underlying channel perfectly reliable
    - no bit errors
    - no loss of packets
    - sender sends data into underlying channel
    - receiver read data from underlying channel

rdt2.0: channel with bit errors

- **Underlying channel may flip bits in packet**
  - checksum to detect bit errors
- ***Q*uestion: how to recover from errors:**
  - **Acknowledgements (ACKs):** receiver explicitly tells sender that pkt received OK
  - **Negative Acknowledgements (NAKs):** receiver explicitly tells sender that pkt had errors
  - sender retransmits pkt on receipt of NAK
- **New mechanisms in `rdt2.0` (beyond `rdt1.0`):**
  - error detection
  - receiver feedback: control msgs (ACK,NAK) rcvr->sender

rdt2.0 has a fatal flaw!

What happens if ACK/NAK corrupted?

- sender doesn't know what happened at receiver!
- can't just retransmit: possible duplicate.

Handling duplicates:

- sender retransmits current pkt if ACK/NAK misinterpreted
- sender adds sequence number to each pkt
- receiver discards (doesn't deliver up) duplicate pkt

stop and wait
Sender sends one packet, then waits for receiver response

rdt3.0: channels with errors and loss

New assumption:

underlying channel can also lose packets (data or ACKs)

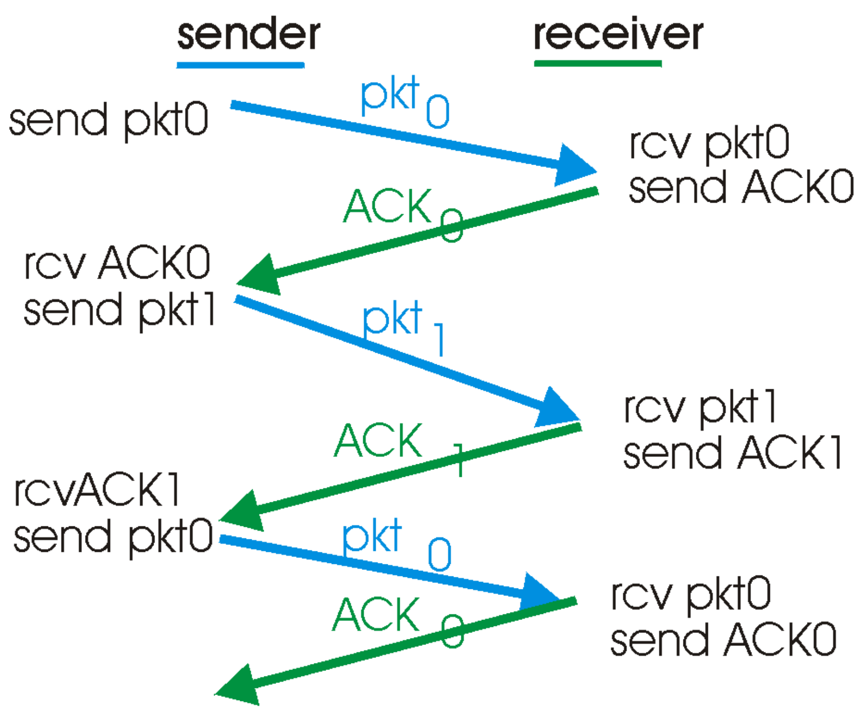
- checksum, seq. #, ACKs, retransmissions will be of help, but not enough

Approach:

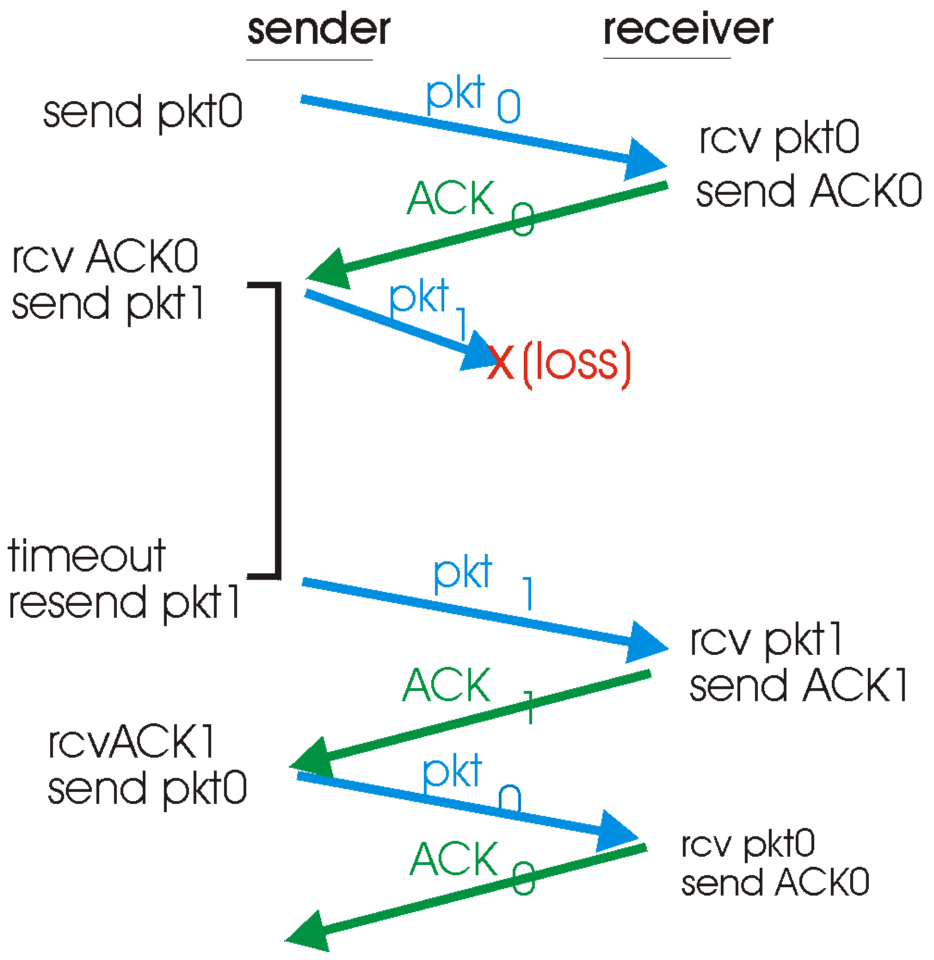
sender waits "reasonable" amount of time for ACK

- Retransmits if no ACK received in this time
- if pkt (or ACK) just delayed (not lost):
  - retransmission will be duplicate, but use of seq. #'s already handles this
  - receiver must specify seq # of pkt being ACKed
- requires countdown timer

rdt3.0 in action

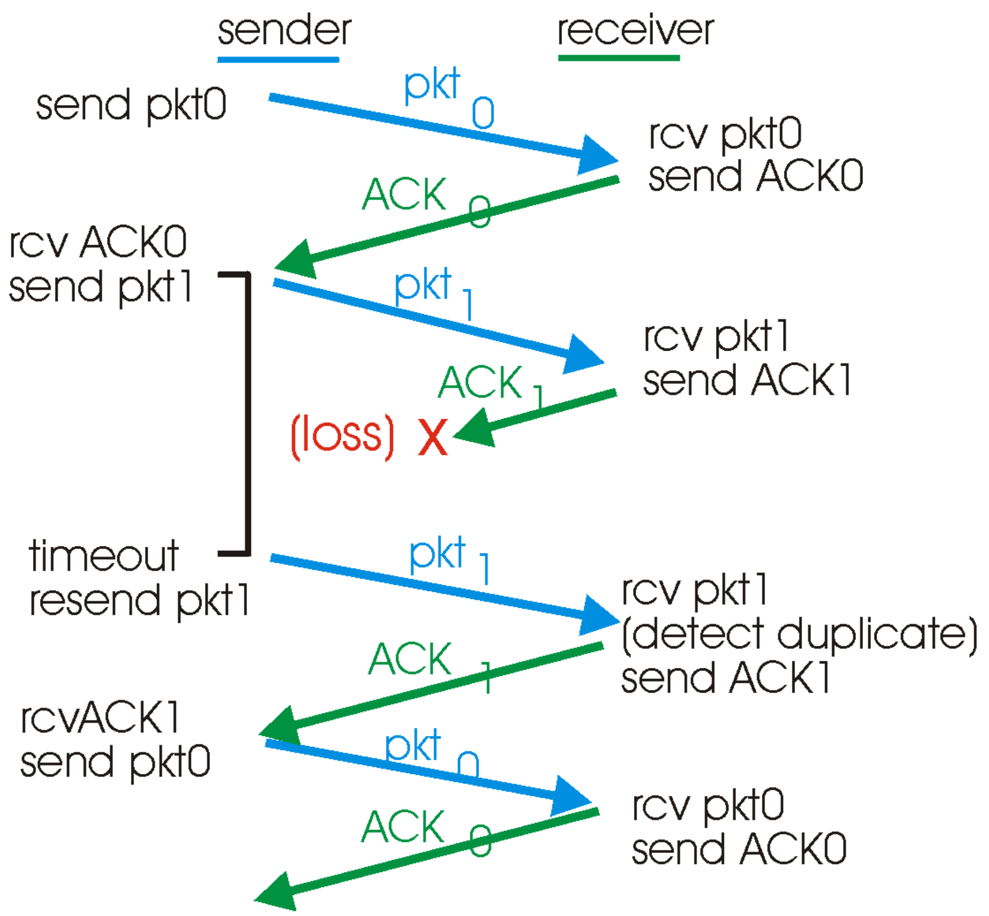


(a) operation with no loss

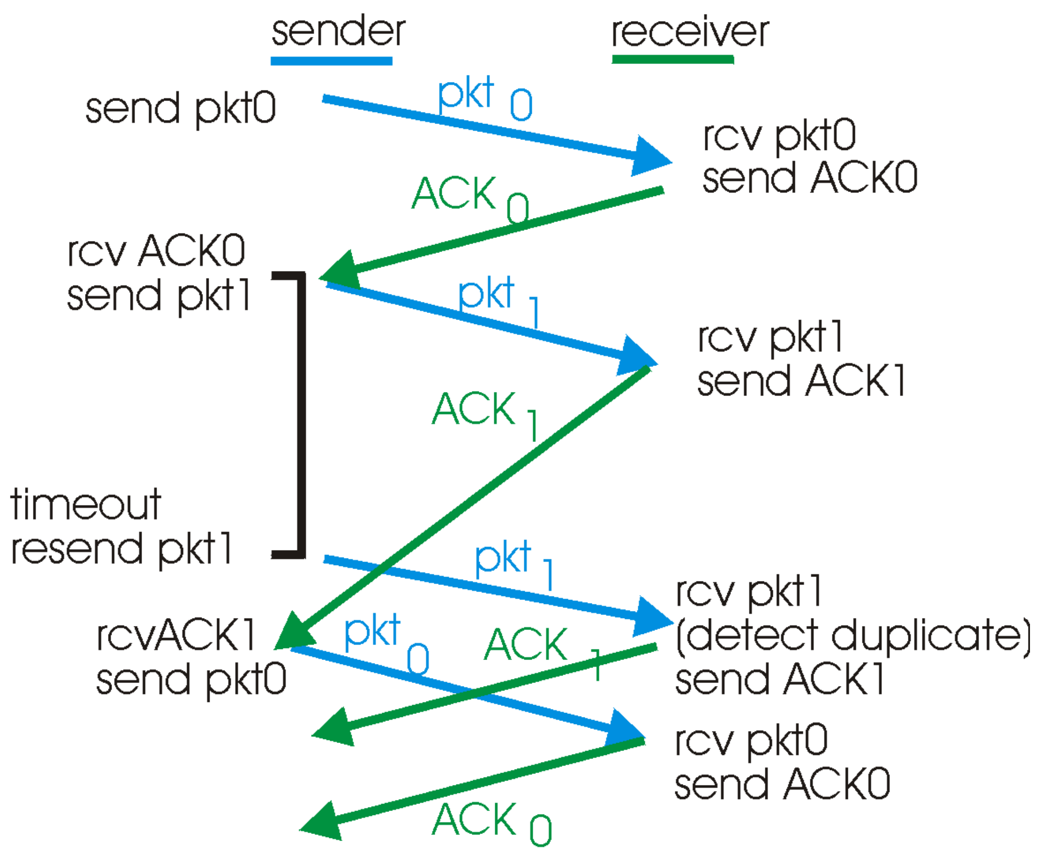


(b) lost packet

rdt3.0 in action (Cont.)



(c) lost ACK



(d) premature timeout

Performance of rdt3.0

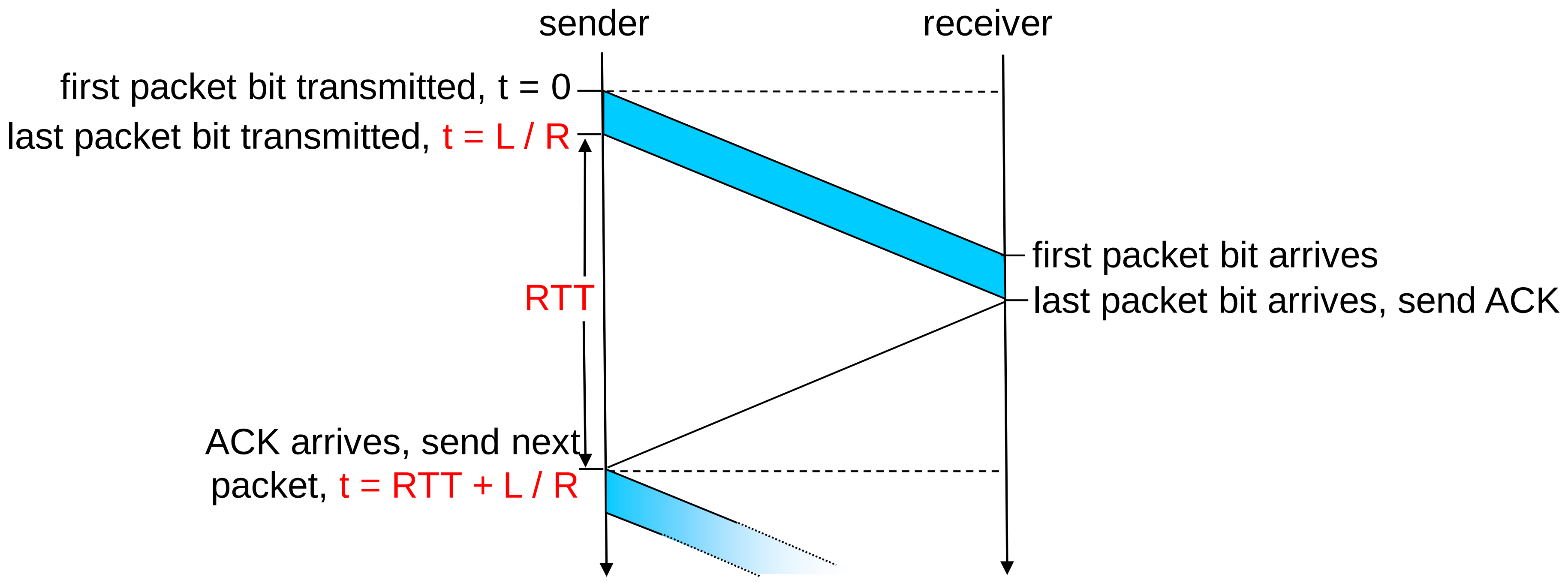
- rdt3.0 works, but performance stinks
- **ex: 1** Gbps link, **15 ms** prop. delay, **8000** bit packet:

$$d_{trans} = \frac{L}{R} = \frac{8000\text{bits}}{10^9\text{bps}} = 8\,\text{microseconds}$$

- U $_{sender}$: **utilization** – fraction of time sender busy sending

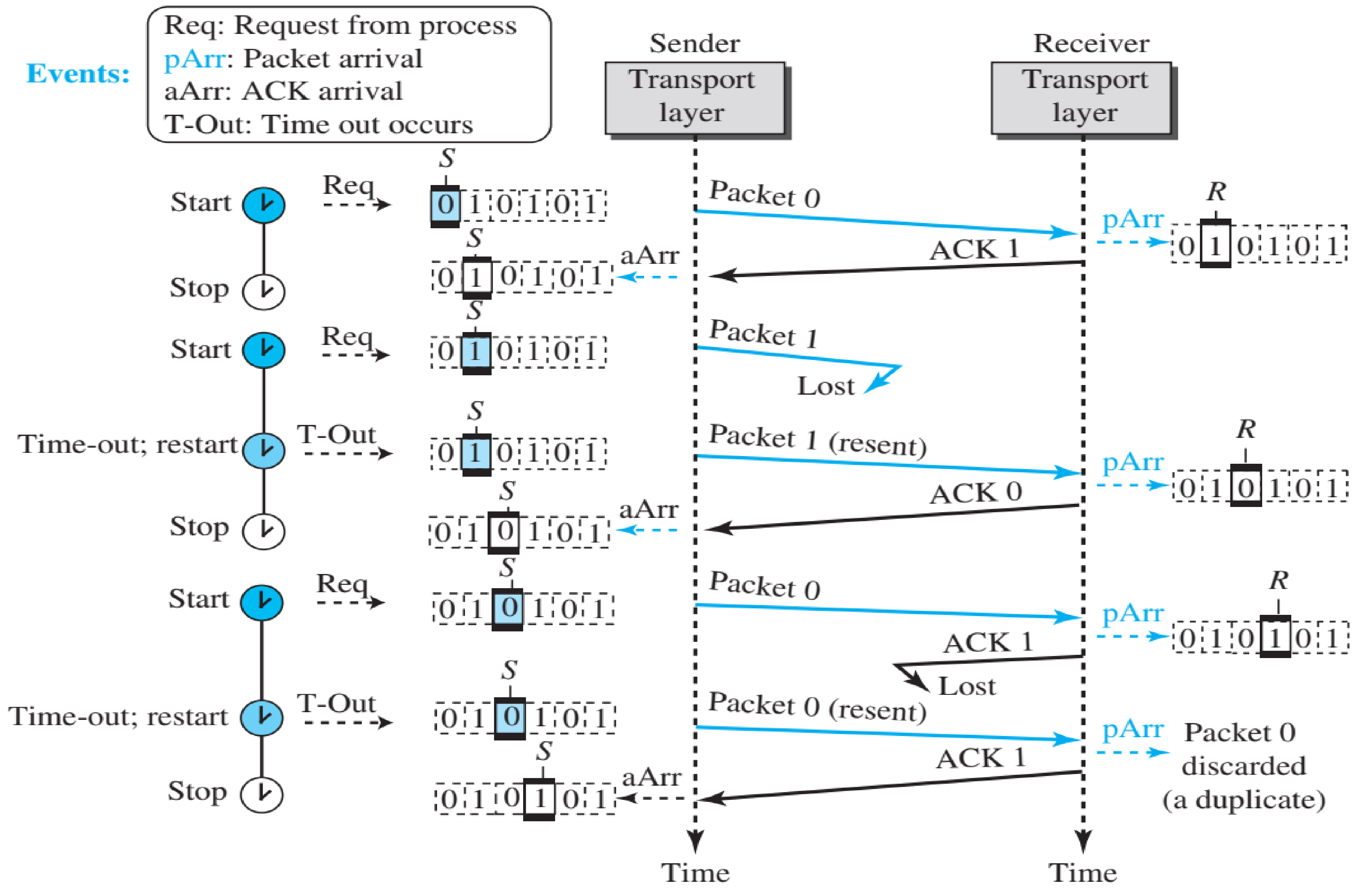
$$U_{sender} = \frac{L / R}{RTT + L / R} = \frac{.008}{30.008} = 0.00027$$

rdt3.0: stop-and-wait operation


sender
receiver
first packet bit transmitted, t = 0
last packet bit transmitted, t = L / R
RTT
first packet bit arrives
last packet bit arrives, send ACK
ACK arrives, send next packet, t = RTT + L / R

$$U_{\text{sender}} = \frac{L / R}{RTT + L / R} = \frac{.008}{30.008} = 0.00027$$

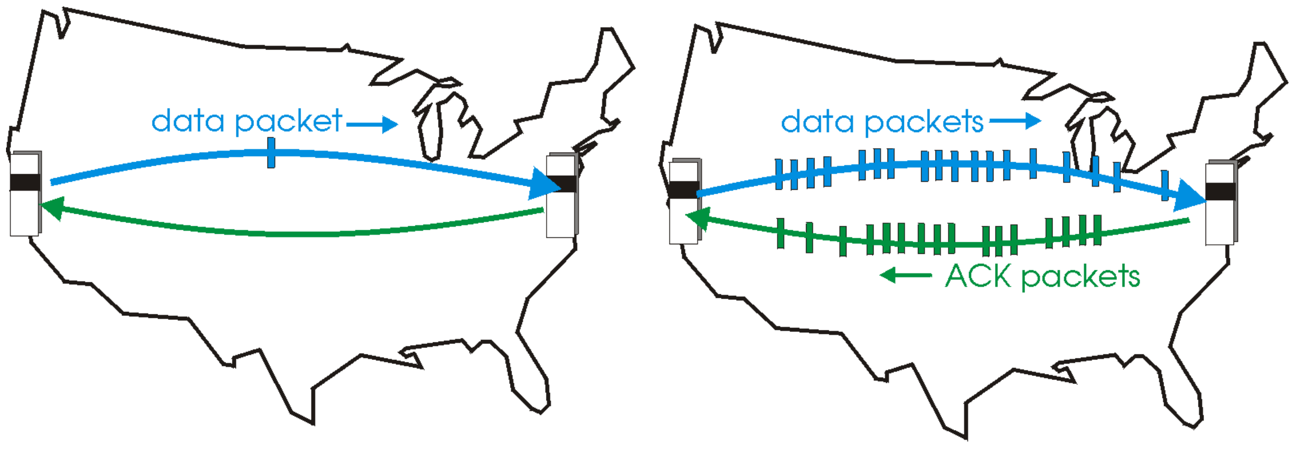
rdt3.0: stop-and-wait operation (Cont.)



Pipelined protocols

Pipelining: sender allows multiple, "**in-flight**", yet-to-be-acknowledged pkts

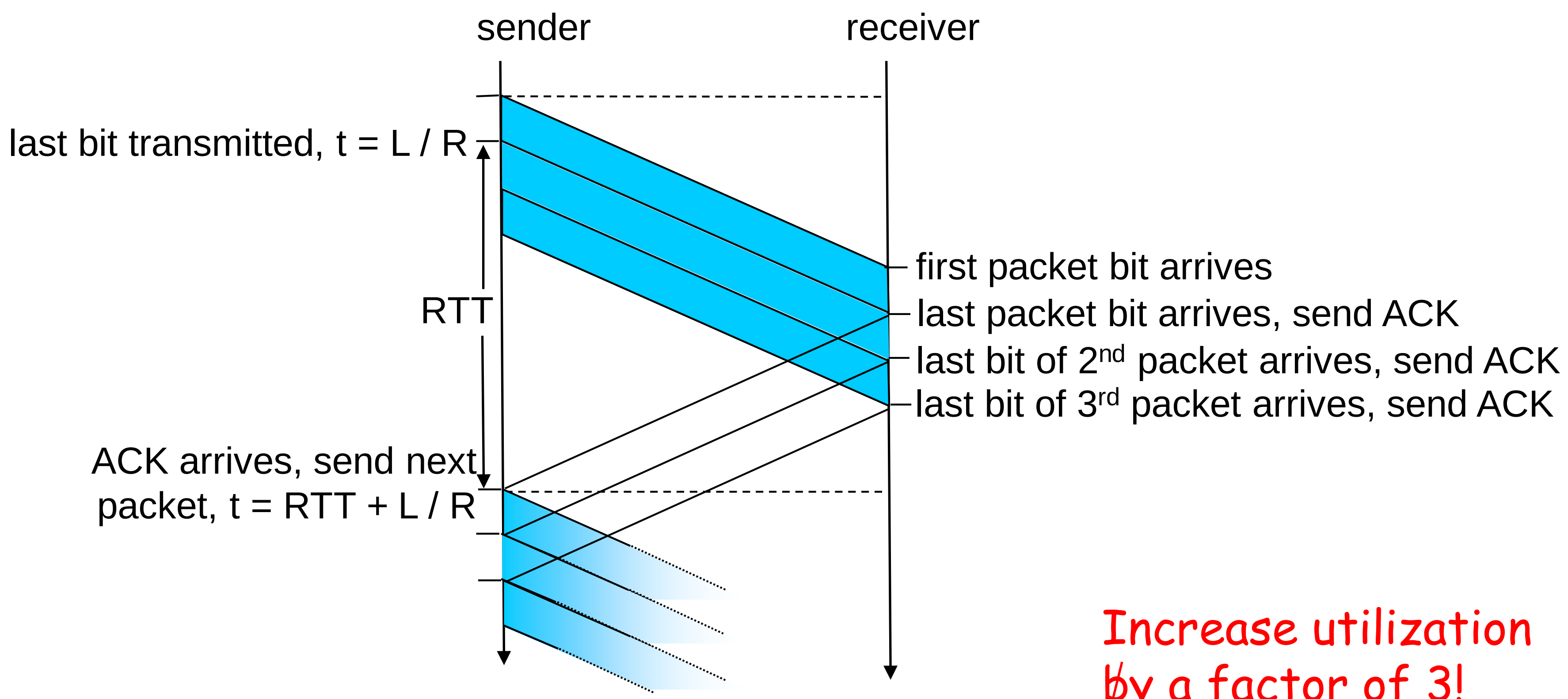
- ➢range of sequence numbers must be increased
- ➢buffering at sender and/or receiver



(a) a stop-and-wait protocol in operation (b) a pipelined protocol in operation

- Two generic forms of pipelined protocols: **go-Back-N**, **selective repeat**

Pipelining: increased utilization


sender
receiver
last bit transmitted, t = L / R
RTT
ACK arrives, send next packet, t = RTT + L / R
first packet bit arrives
last packet bit arrives, send ACK
last bit of 2nd packet arrives, send ACK
last bit of 3rd packet arrives, send ACK

$$U_{\text{sender}} = \frac{3 * L / R}{RTT + L / R} = \frac{.024}{30.008} = 0.0008$$

Increase utilization by a factor of 3!

Pipelining Protocols

Go-back-N:

- Sender can have up to **N** unacked packets in pipeline.
- Rcvr only sends cumulative acks
  - ➢ Doesn't ack packet if there's a gap
- Sender has timer for oldest unacked packet
  - ➢ If timer expires, retransmit all unacked packets

Selective Repeat:

- Sender can have up to **N** unacked packets in pipeline.
- Rcvr acks individual packets
- Sender maintains timer for each unacked packet
  - ➢ When timer expires, retransmit only unack packet

Go-back-N

- To improve the efficiency of transmission (to fill the pipe), multiple packets must be in transition while the sender is waiting for acknowledgment.
- The key to Go-back-*N* is that we can send several packets before receiving acknowledgments, but the receiver can only buffer one packet.
- We keep a copy of the sent packets until the acknowledgments arrive.
- In the Go-Back-N protocol, the acknowledgment number is cumulative and defines the sequence number of the next packet expected to arrive.

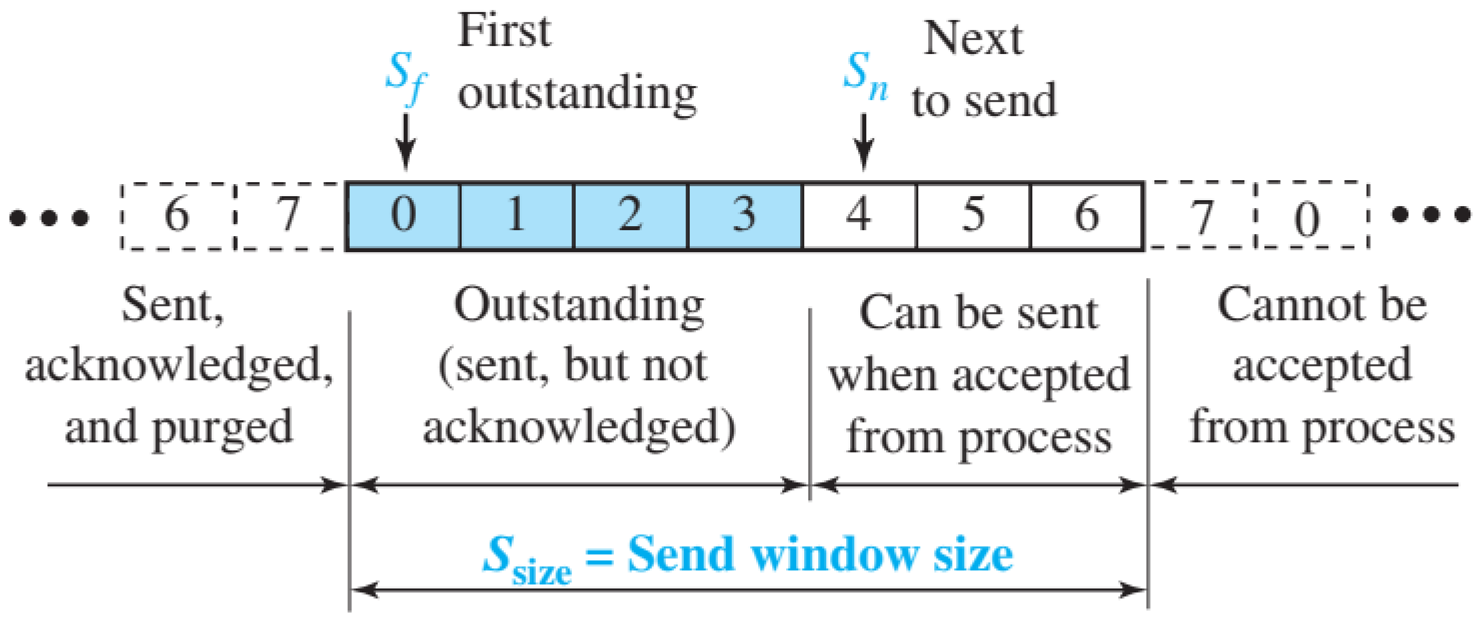
Go-back-N (Cont.)

Send Window

- The send window is an imaginary box covering the sequence numbers of the data packets that can be in transit or can be sent.
- In each window position, some of these sequence numbers define the packets that have been sent; others define those that can be sent.
- The maximum size of the window is **$2^m - 1$**

Go-back-N (Cont.)

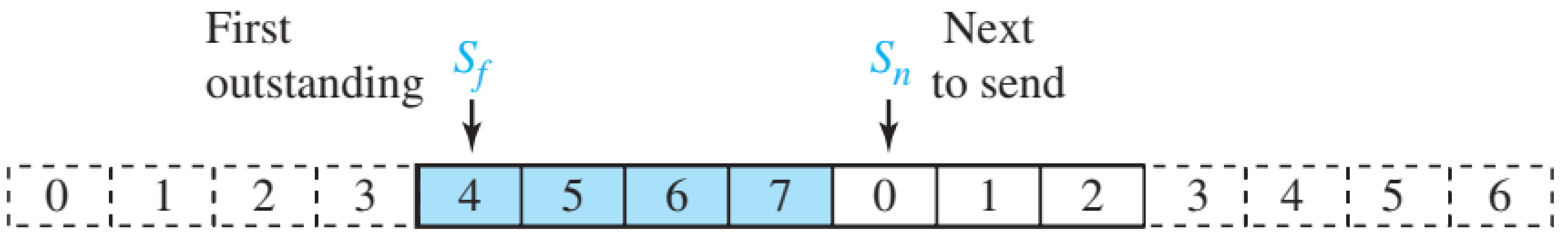
The Following Figure shows a sliding window of size **7** (***m*** **= 3**) for the **Go-Back-N** protocol.



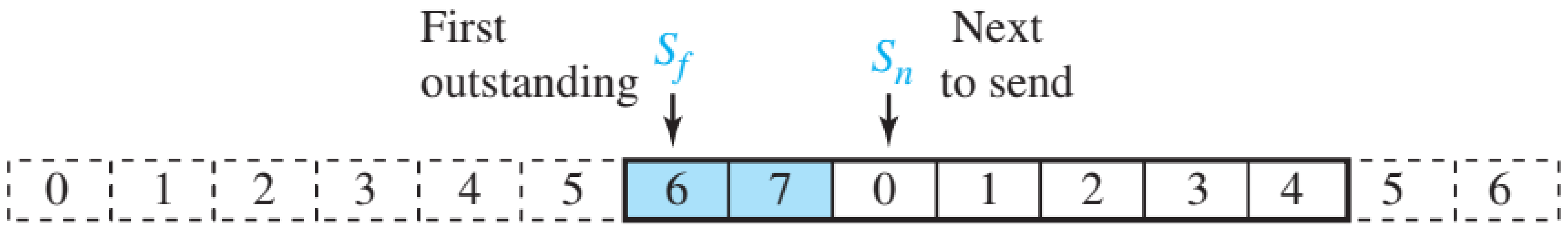
S_f
First outstanding
S_n
Next to send
6
7
0
1
2
3
4
5
6
7
0
Sent, acknowledged, and purged
Outstanding (sent, but not acknowledged)
Can be sent when accepted from process
Cannot be accepted from process
S_{size} = Send window size

Go-back-N (Cont.)

Sliding the send window



a. Window before sliding

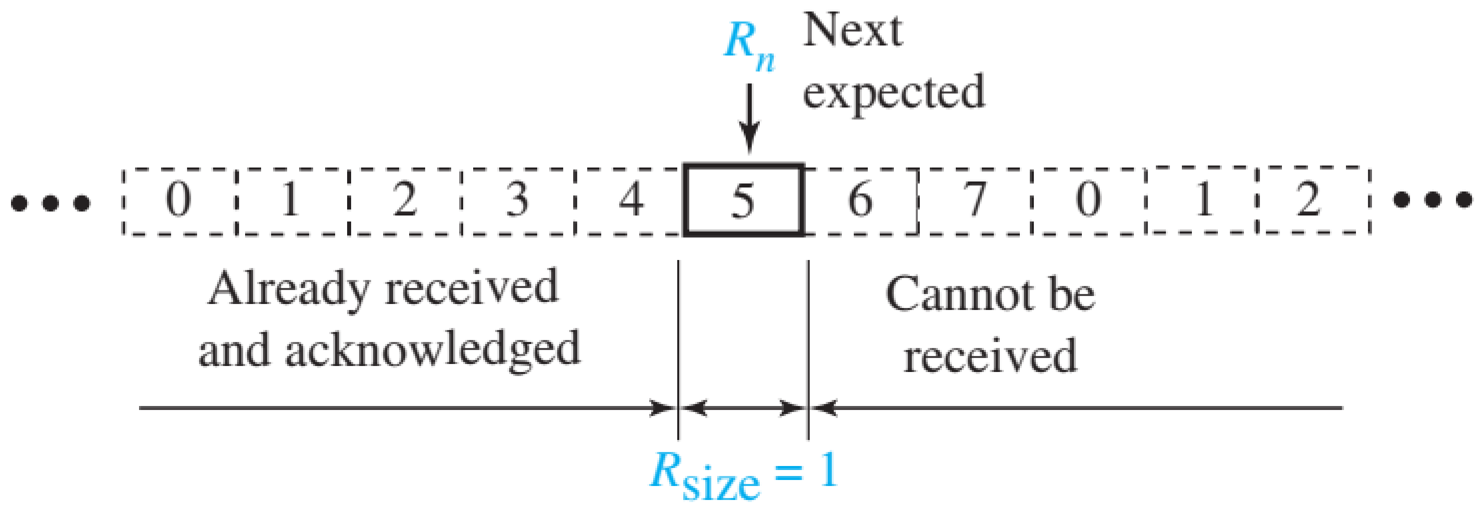


b. Window after sliding (an ACK with ackNo = 6 has arrived)

Go-back-N (Cont.)

Receive Window

- the size of the receive window is always 1.
- The receiver is always looking for the arrival of a specific packet.
- Any packet arriving out of order is discarded and needs to be resent.



Go-back-N (Cont.)

Timers

- Although there can be a timer for each packet that is sent, in our protocol we use **only one**.
- The reason is that the timer for **the first outstanding packet always expires first**.
- We resend all outstanding packets when this timer expires.

Go-back-N (Cont.)

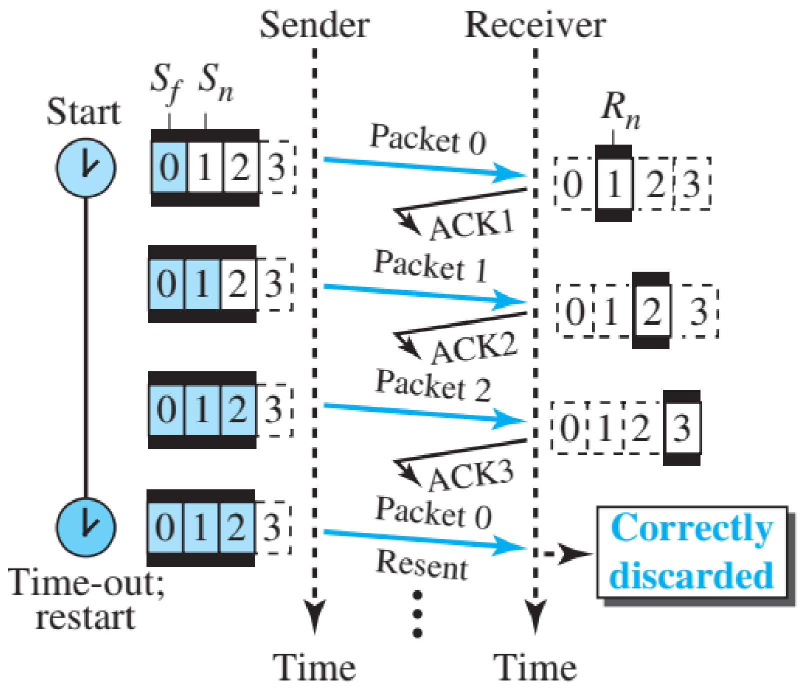
Resending packets

- When the timer expires, the sender resends all outstanding packets.
- For example, suppose the sender has already sent packet 6 ($S_n = 7$), but the only timer expires.
- If $S_f = 3$, this means that packets 3, 4, 5, and 6 have not been acknowledged; the sender goes back and resends packets 3, 4, 5, and 6.
- That is why the protocol is called **Go-Back-N**.
- On a time-out, the machine goes back **N** locations and resends all packets

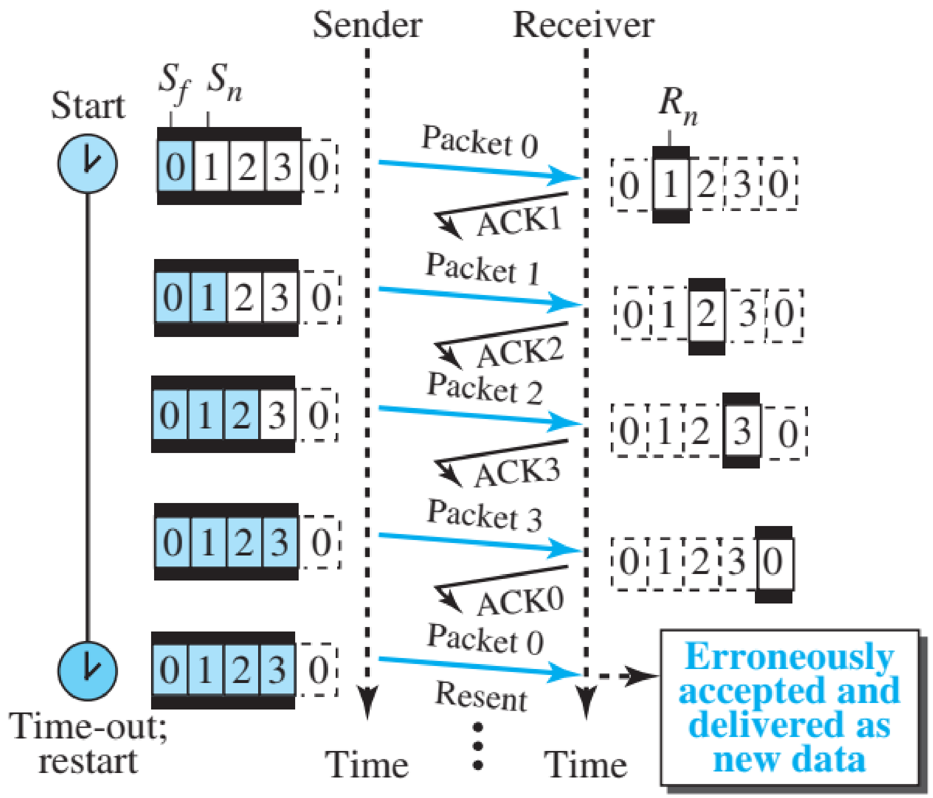
Go-back-N (Cont.)

Send Window Size

In the Go-Back-*N* protocol, the size of the send window must be less than 2^m; the size of the receive window is always 1.



a. Send window of size $< 2^m$



b. Send window of size $= 2^m$

Go-back-N Issues

- The **Go-Back-N** protocol simplifies the process at the receiver. The receiver keeps track of only one packet, and there is no need to buffer out-of-order packets; they are simply discarded.
- However, this protocol is inefficient if the underlying network protocol loses a lot of packets. Each time a single packet is lost or corrupted, the sender resends all outstanding packets, even though some of these packets may have been received safe and sound but out of order.

Selective-Repeat Protocol (SR)

- ❑Another protocol, called the **Selective-Repeat (SR) protocol,** has been devised, which, as the name implies, resends only selective packets, those that are actually lost.
- ❑The Selective-Repeat protocol also uses two windows: a send window and a receive window.
- ❑The maximum size of the send window is much smaller than GBN; it is $\mathbf{2^{m-1}}$.
- ❑The receive window is the same size as the send window ($\mathbf{2^{m-1}}$).

Selective-Repeat Protocol (Cont.)

Timer

- The Selective-Repeat uses one timer for each outstanding packet.
- When a timer expires, only the corresponding packet is resent.
- In other words, GBN treats outstanding packets as a group; SR treats them individually.
- Most transport-layer protocols that implement Selective Repeat use only one timer because the timeout for the **oldest unacknowledged packet** is sufficient to detect loss and trigger retransmission, while minimizing complexity and resource usage.

Selective-Repeat Protocol (Cont.)

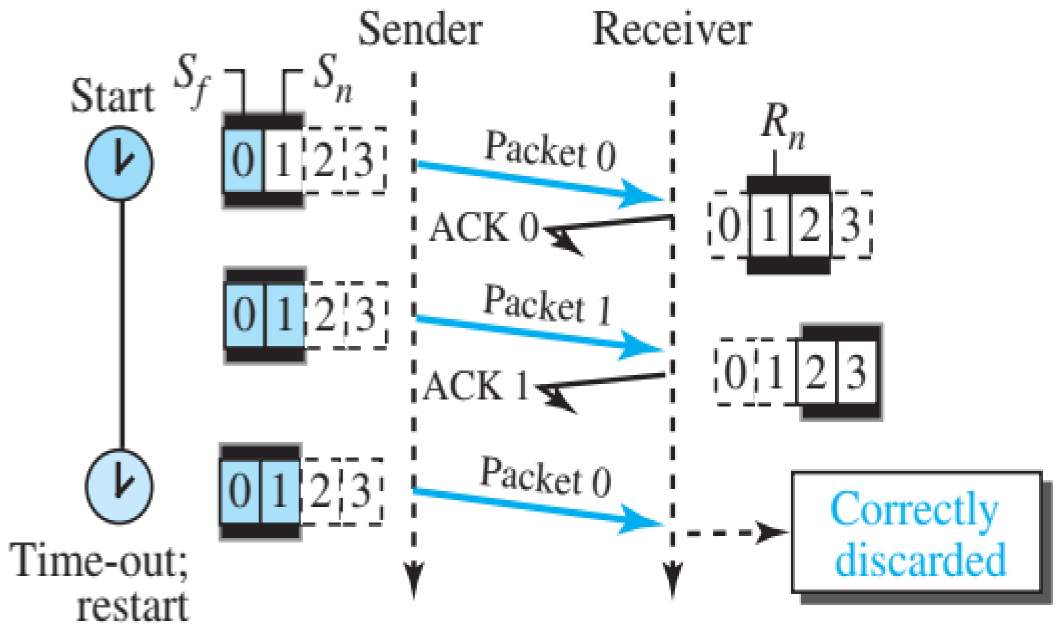
Acknowledgments

- In GBN an ackNo is cumulative; it defines the sequence number of the next packet expected, confirming that all previous packets have been received safe and sound.
- In SR, an ackNo defines the sequence number of a single packet that is received safe and sound; there is no feedback for any other.

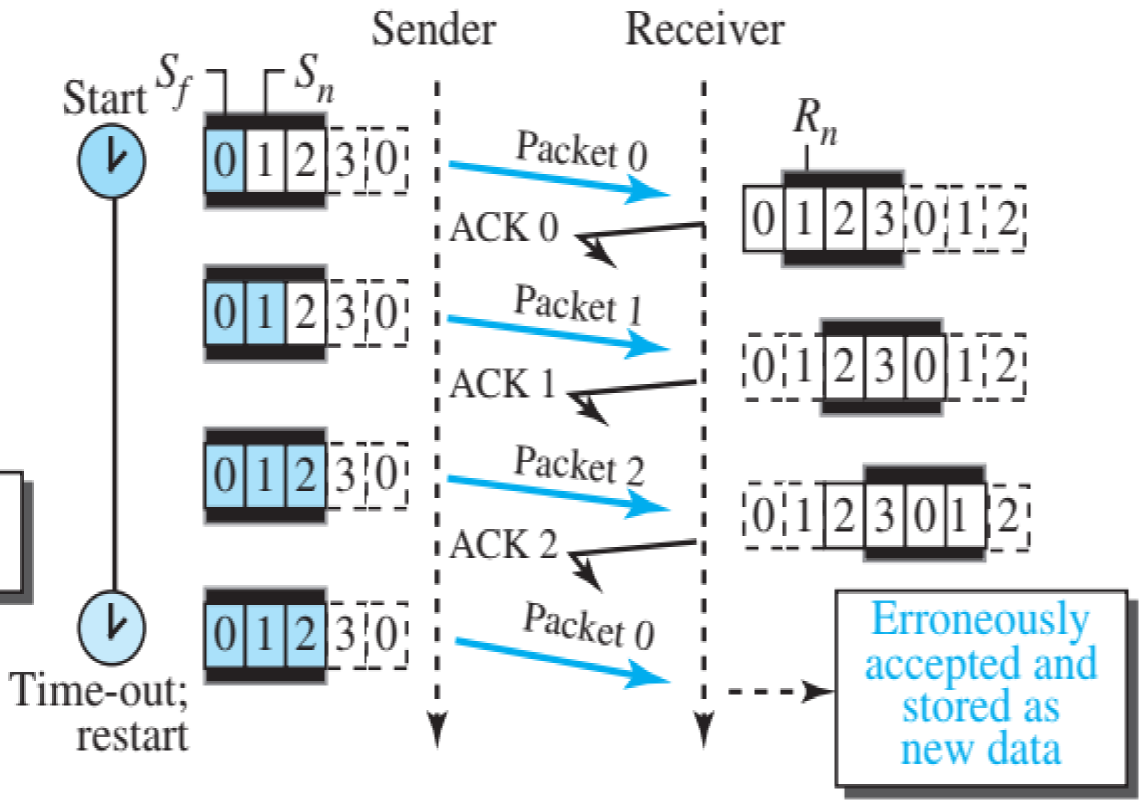
In the Selective-Repeat protocol, an acknowledgment number defines the sequence number of the error-free packet received.

Selective-Repeat Protocol (Cont.)

Window Size



a. Send and receive windows of size = 2^{m-1}



b. Send and receive windows of size > 2^{m-1}

Selective-Repeat Protocol (Cont.)

- We can now show why the size of the sender and receiver windows can be at most one-half of **2^m**. For an example, we choose **m = 2**, which means the size of the window is **$2^m/2$** or **$2^{(m-1)}$** = **2**.
- If the size of the window is **2** and all acknowledgments are lost, the timer for packet 0 expires and packet 0 is resent. However, the window of the receiver is now expecting packet 2, not packet 0, so this duplicate packet is correctly discarded (the sequence number 0 is not in the window).
- When the size of the window is **3** and all acknowledgments are lost, the sender sends a duplicate of packet 0. However, this time, the window of the receiver expects to receive packet 0 (0 is part of the window), so it accepts packet 0, not as a duplicate, but as a packet in the next cycle. This is clearly an error.

Selective-Repeat Protocol (Cont.)

Example

Assume a sender sends **6** packets: packets 0, 1, 2, 3, 4, and 5. The sender receives an ACK with AckNo = **3**. What is the interpretation if the system is using GBN or SR?

Solution

- If the system is using **GBN**, it means that packets 0, 1, and 2 have been received uncorrupted and the receiver is expecting packet 3.
- If the system is using **SR**, it means that packet 3 has been received uncorrupted; the ACK does not say anything about other packets.

Selective-Repeat Protocol (Cont.)

Feature / Aspect	Go-Back-N (GBN)	Selective Repeat (SR)
ACK Type	**Cumulative ACKs** – one ACK can confirm multiple packets.	**Individual ACKs** – each correctly received packet is acknowledged separately.
Receiver Behavior	Receiver discards out-of-order packets.	Receiver stores out-of-order packets until missing ones arrive.
Retransmission	Lost packet causes retransmission of that packet and all subsequent packets in the window.	Only the lost or corrupted packet is retransmitted.
Efficiency / Bandwidth Usage	Lower efficiency when errors occur (many unnecessary retransmissions).	Higher efficiency, as only erroneous packets are resent.
Sender Window Size (N)	Up to N unacknowledged packets can be sent.	Up to N unacknowledged packets can be sent, but the receiver window must also be of size N.
Receiver Window Size	Always 1 (since out-of-order packets are discarded).	Equal to sender window size N (to buffer out-of-order packets).
Complexity (Implementation)	Simple – easier to implement.	Complex – requires buffering and managing multiple ACKs.
Timer Usage	Single timer for the oldest unacknowledged packet.	Ideally multiple timers (one per packet), but most transport protocols use only one for efficiency.
Duplicate Frames Handling	Can occur if ACK is lost – sender retransmits all later frames.	Reduced duplicates because only lost frames are resent.
Throughput	Lower (wastes bandwidth when errors occur).	Higher (better utilization of link capacity).
Typical Use	Often used in simpler link-layer protocols (e.g., older ARQ systems).	Conceptually forms the basis for TCP (transport layer reliability).

Bidirectional Protocols: Piggybacking

- The three protocols we discussed earlier in this section are all **unidirectional**: data packets flow in only one direction and acknowledgments travel in the other direction.
- In real life, data packets are normally flowing in both directions: from client to server and from server to client. This means that acknowledgments also need to flow in both directions.
- A technique called **piggybacking** is used to improve the efficiency of the bidirectional protocols.
- When a packet is carrying data from A to B, it can also carry acknowledgment feedback about arrived packets from B; when a packet is carrying data from B to A, it can also carry acknowledgment feedback about the arrived packets from A.

The End