The Transfer Layer

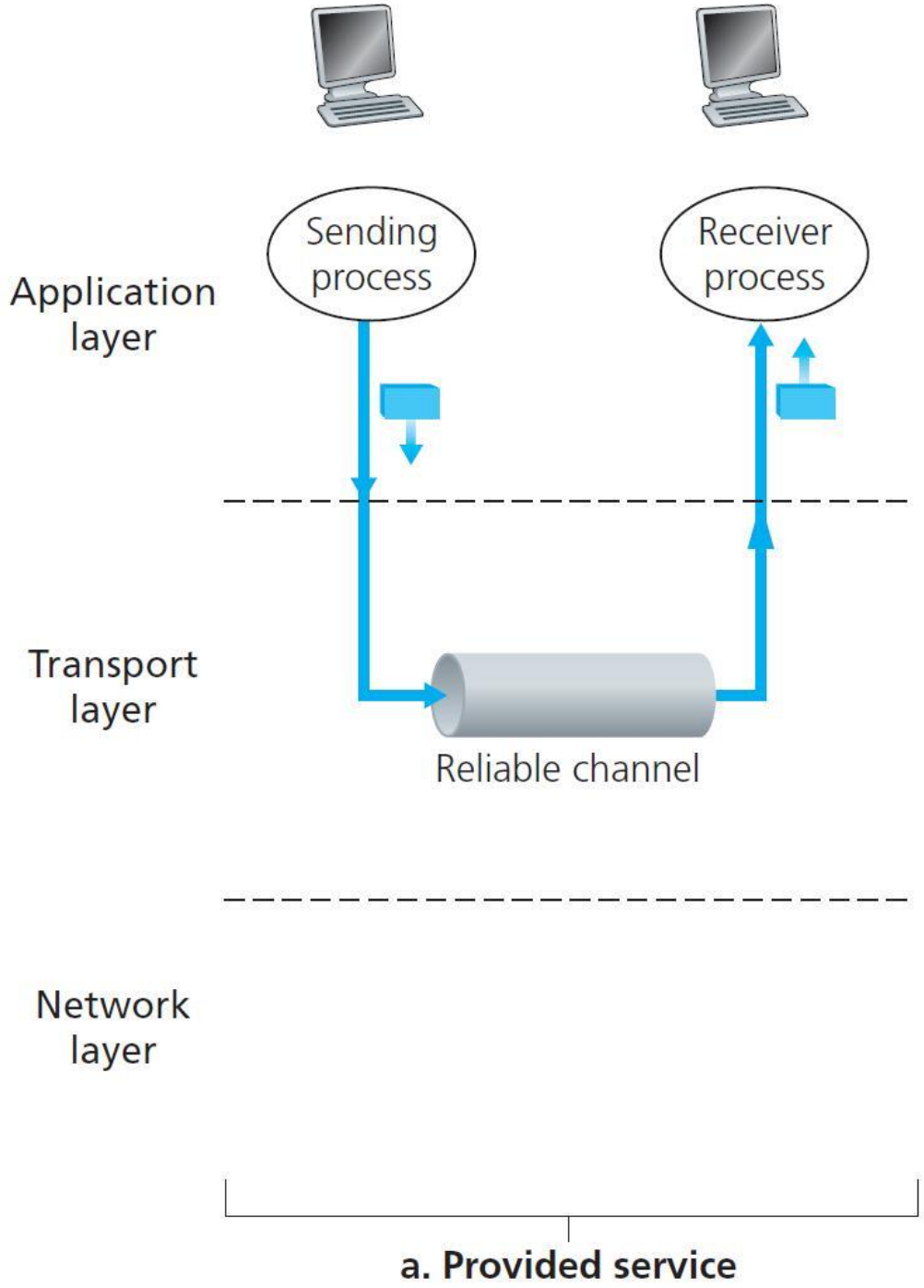
Reliable Data Transfer

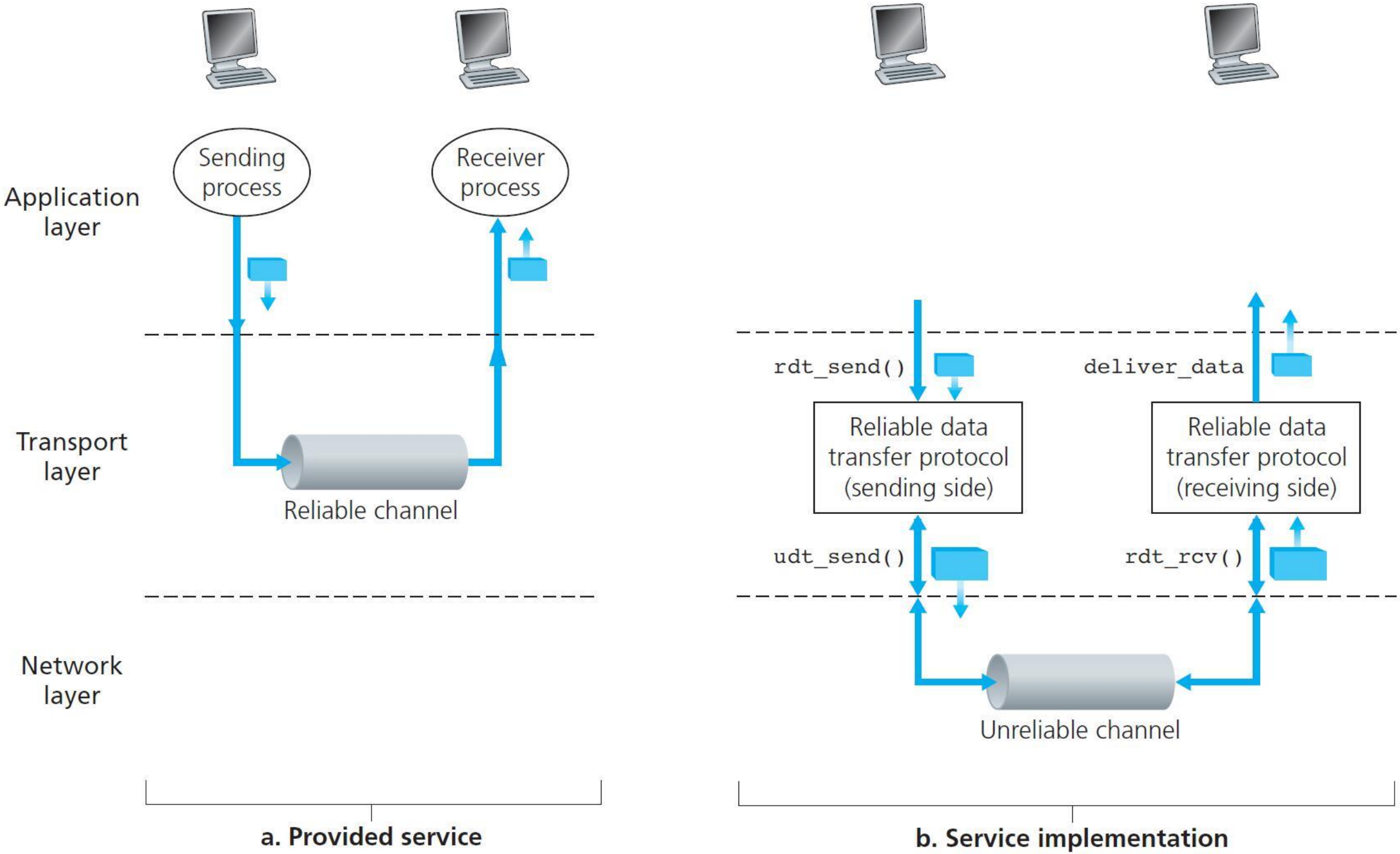
School of Software Engineering
South China University of Technology
Dr. Chunhua Chen
chunhuachen@scut.edu.cn
2016 Spring

The very simple model

- With a reliable channel, no transferred data bits are corrupted (flipped from 0 to 1, or vice versa) or lost, and all are delivered in the order in which they were sent.

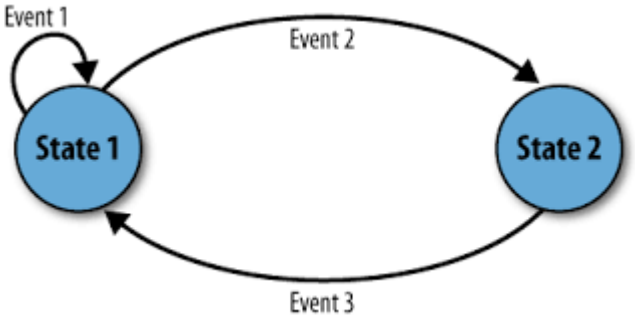
- How to build such a reliable channel?

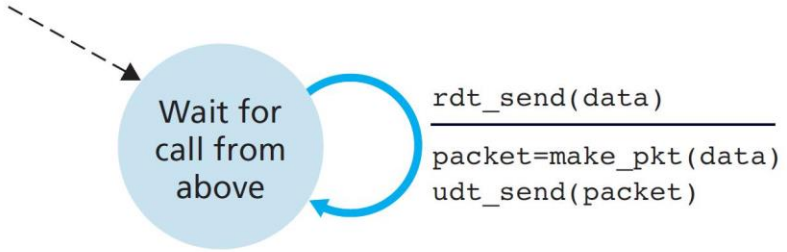



Application layer
Transport layer
Network layer
Sending process
Receiver process
Reliable channel
rdt_send()
deliver_data
Reliable data transfer protocol (sending side)
Reliable data transfer protocol (receiving side)
udt_send()
rdt_rcv()
Unreliable channel
a. Provided service
b. Service implementation

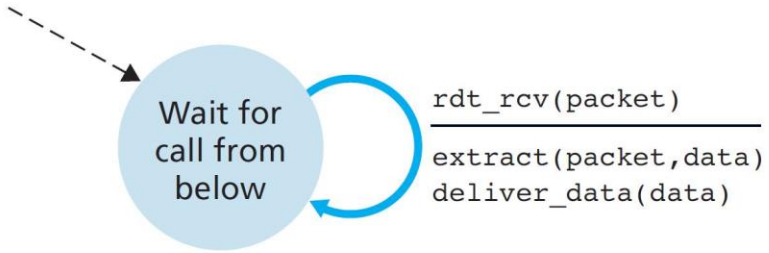
RDT over a Perfectly Reliable Channel: rdt1.0

- Assumptions:
  - The underlying channel is completely reliable
- Describing a protocol using **finite-state machine (FSM)**





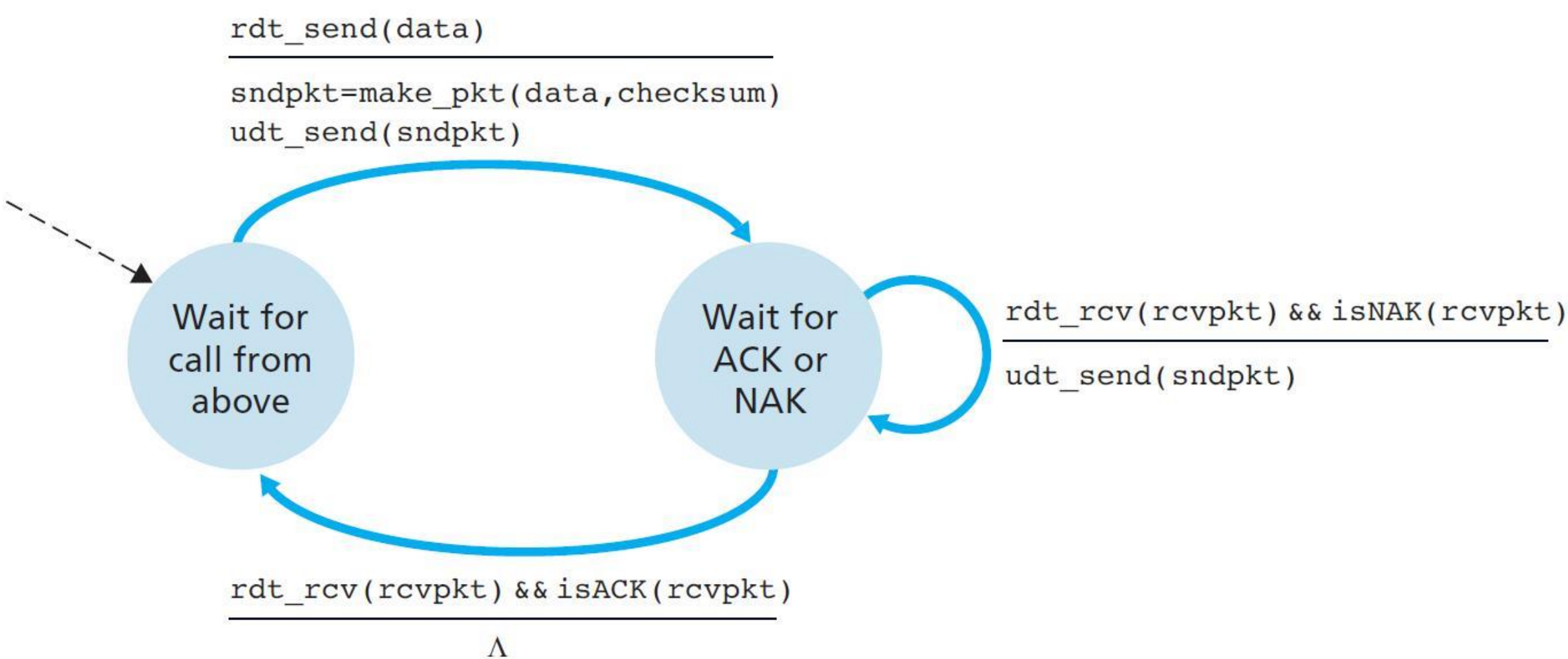
a. rdt1.0: sending side



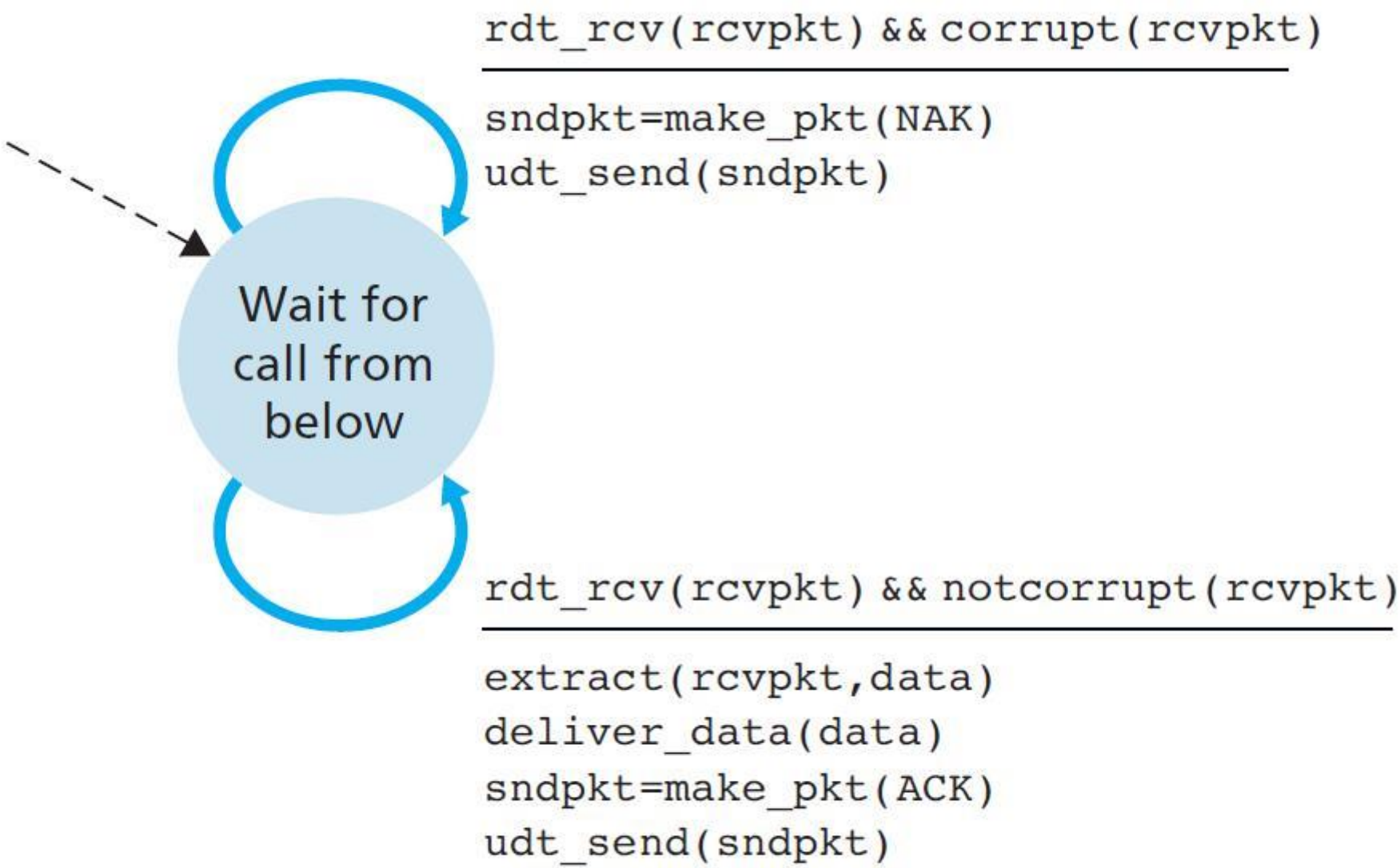
b. rdt1.0: receiving side

RDT over a Channel with Bit Errors: rdt2.0

- Assumptions:
  - Bits in a packet may be corrupted.
  - Such bit errors typically occur in the physical components of a network as a packet is transmitted, propagates, or is buffered.
- Mechanisms:
  - Error detection
    - Checksum, extra bits
  - Receiver feedback
    - the receiver to provide explicit feedback to the sender
      - **positive acknowledgments (ACK)**
      - **negative acknowledgments (NAK)**
  - Retransmission (Automatic Repeat request, ARQ)
    - A packet that is received in error at the receiver will be retransmitted by the sender.



a. rdt2.0: sending side



b. rdt2.0: receiving side

Analyze rdt2.0

- stop-and-wait at sender side
  - Wait for receiver's ACK before sending next segment
- Does not handler the cases of corrupted ACKs and NAKs.
  - the sender simply to resend the current data packet when it receives a garbled ACK or NAK packet?
    - duplicate packets at receiver side
    - But the receiver doesn't know whether it is duplicate or not!
- New problem: duplicate packets
- Solution: add a new field to the data packet and have the sender number its data packets by putting a **sequence number** into this field.

RDT over a Channel with Bit Errors: rdt2.1

Sender

pkt#0 or pkt#1

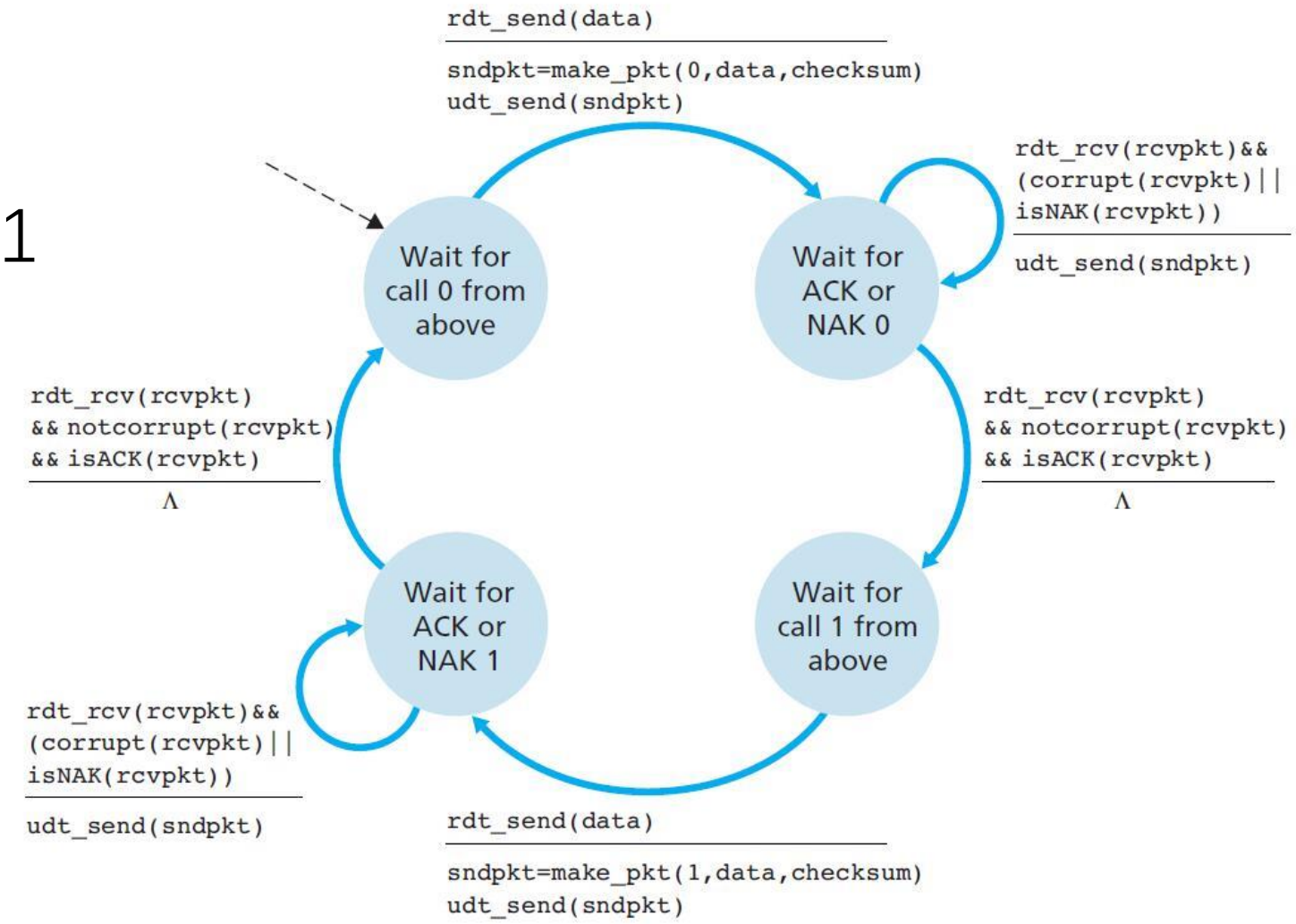


Figure 3.11 ♦ rdt2.1 sender

RDT over a Channel with Bit Errors: rdt2.1

Receiver

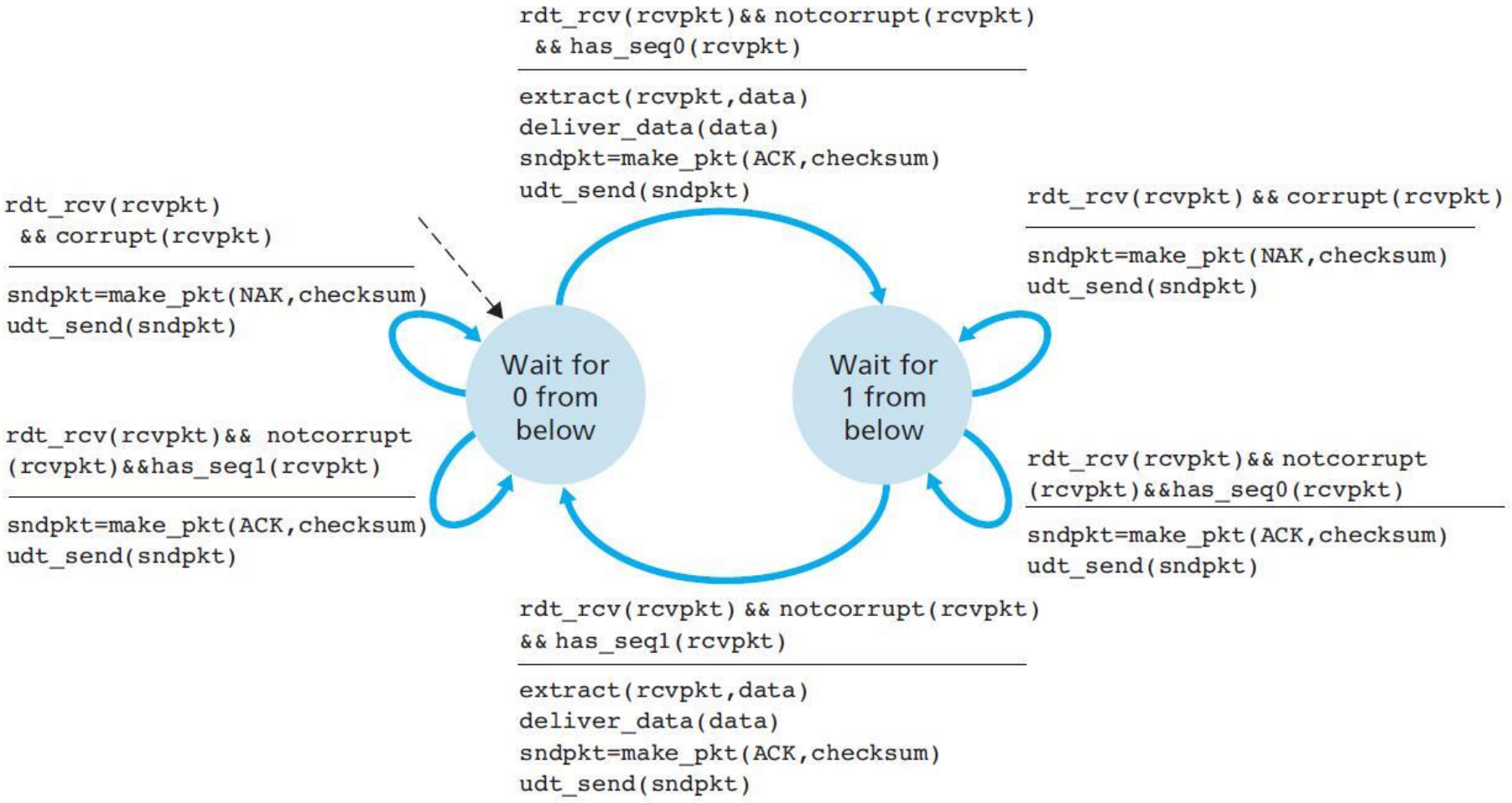


Figure 3.12 ♦ rdt2.1 receiver

Analyze rdt2.1

- More states
- Reflecting whether the packet currently being sent (by the sender) or expected (at the receiver) should have a sequence number of 0 or 1.

- Can we do NAK-free? rdt.2.2
  - Receiver: send an ACK for the last correctly received packet
  - Sender that receives two ACKs for the same packet (that is, receives duplicate ACKs) knows that the receiver did not correctly receive the packet following the packet that is being ACKed twice.

RDT over a Channel with Bit Errors: rdt2.2

Sender

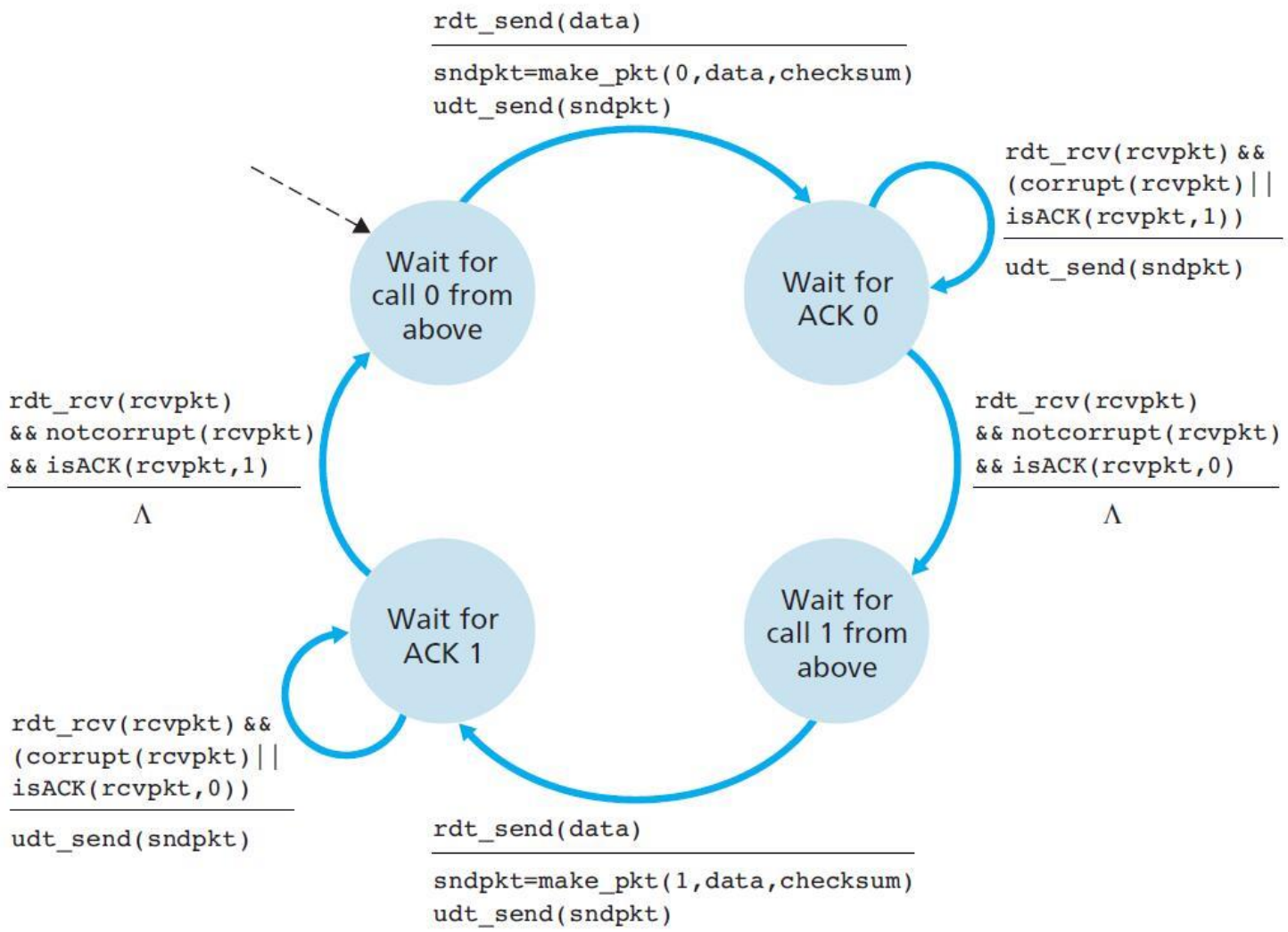


Figure 3.13 ♦ rdt2.2 sender

RDT over a Channel with Bit Errors: rdt2.2

Receiver

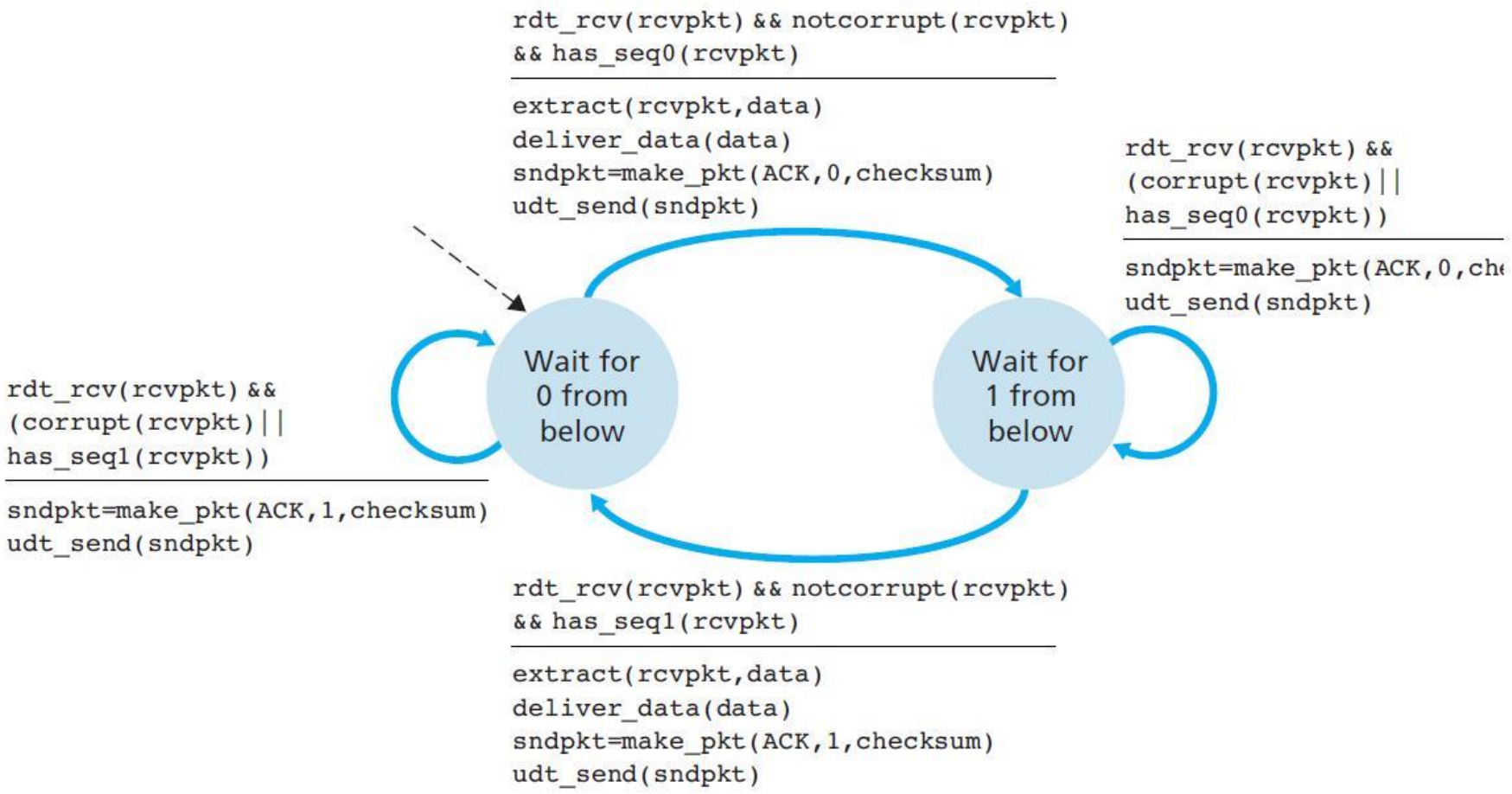


Figure 3.14 ♦ rdt2.2 receiver

RDT over a Lossy Channel with Bit Errors: rdt3.0

Sender

how to detect packet loss
what to do when packet loss occurs.

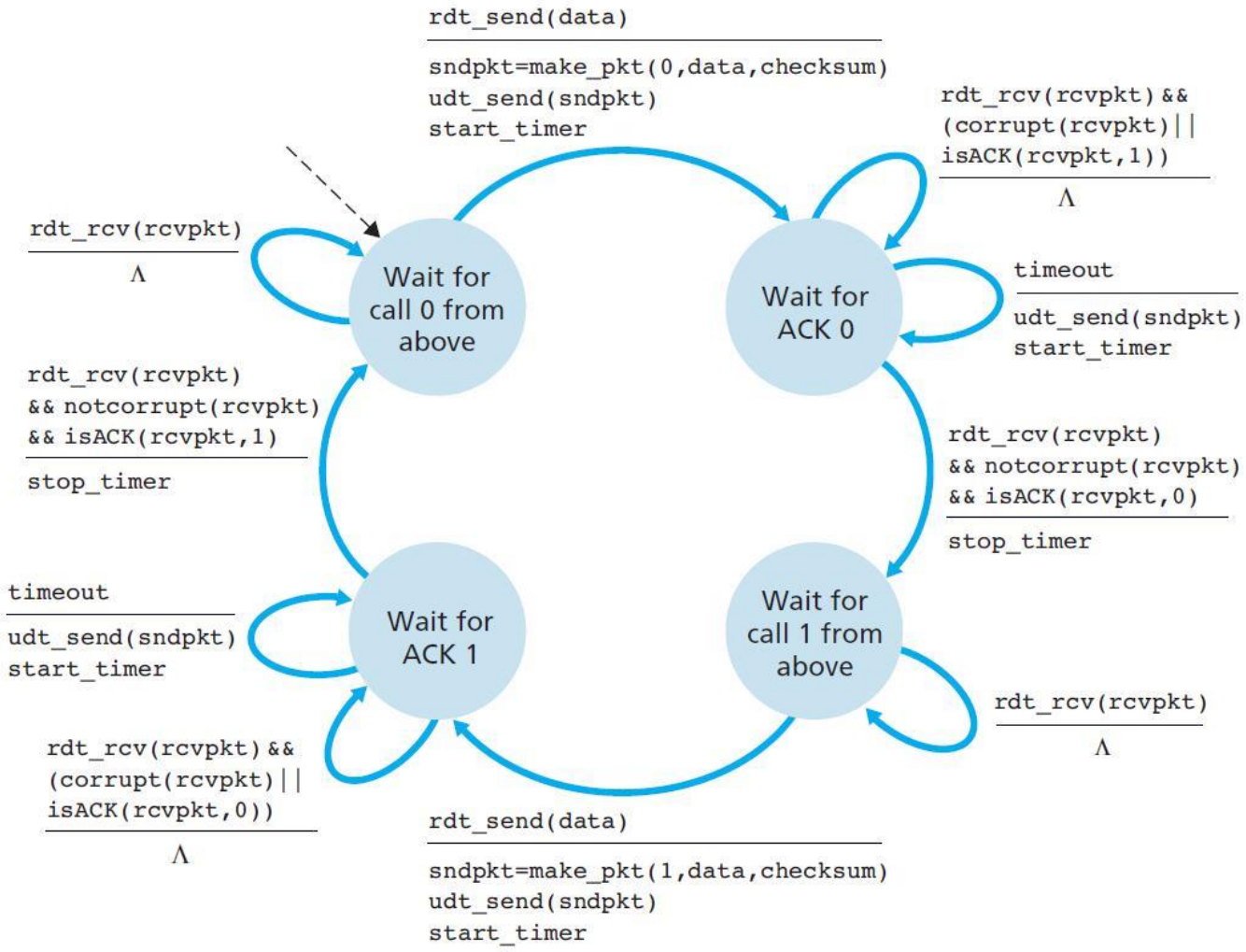
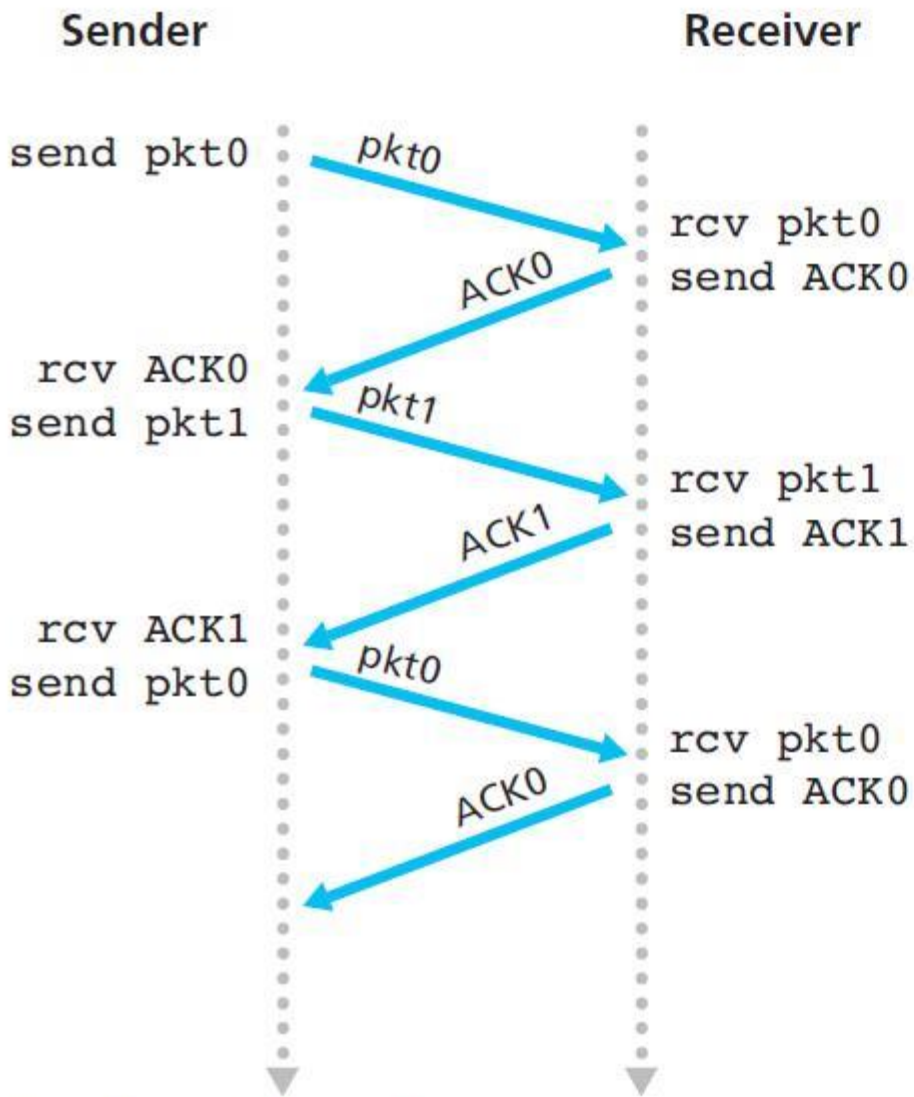
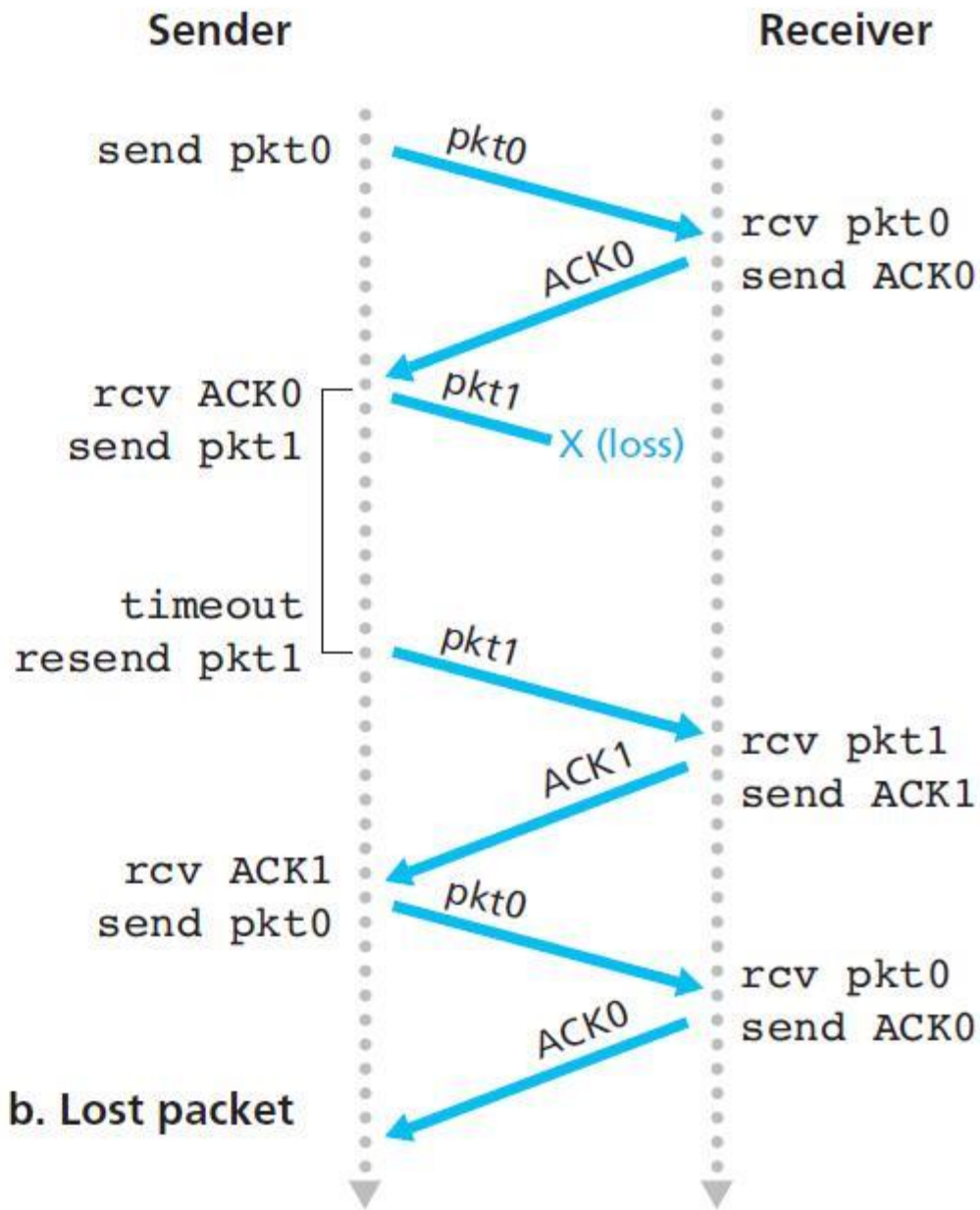


Figure 3.15 ♦ rdt3.0 sender



a. Operation with no loss



b. Lost packet

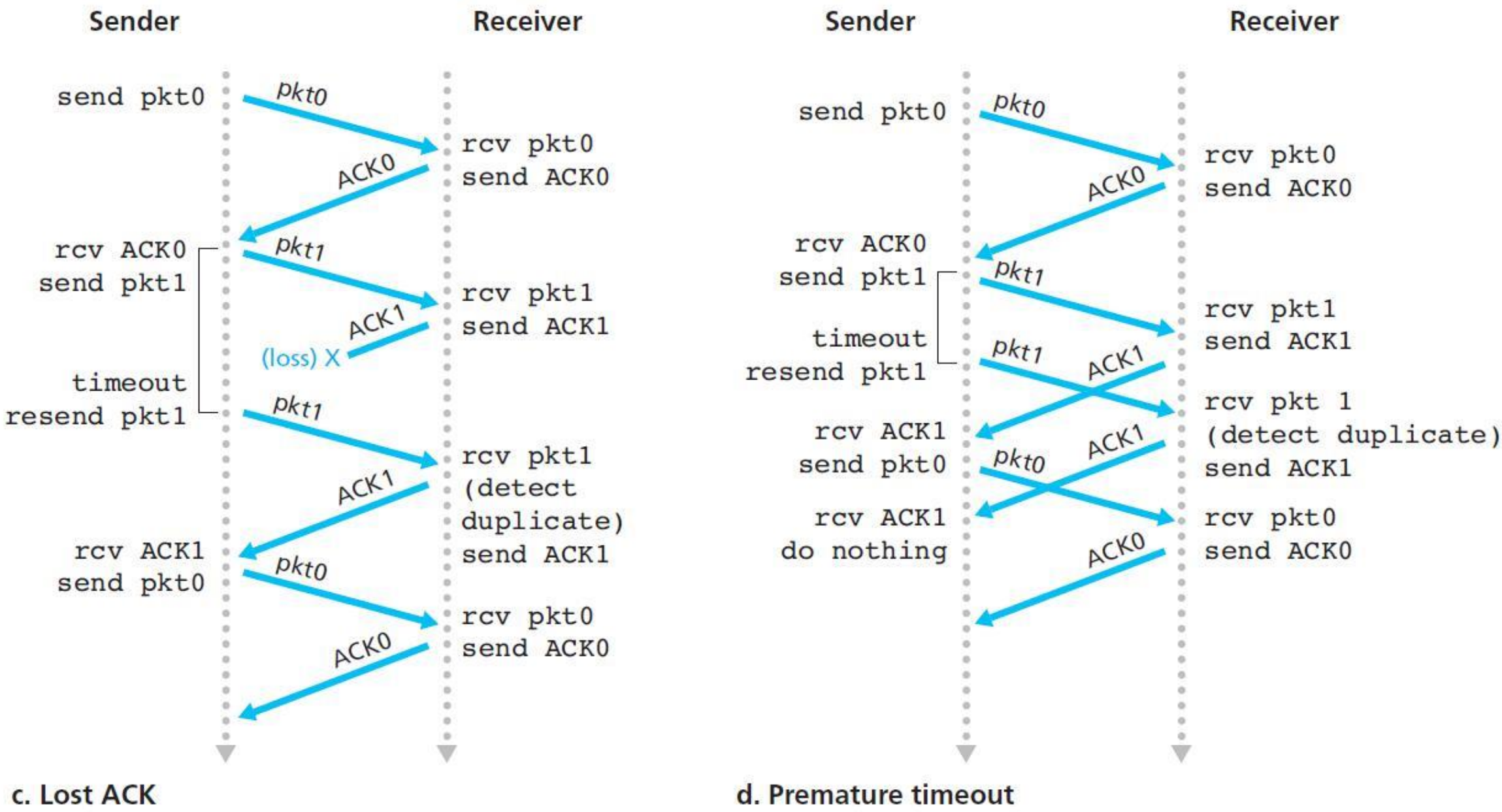
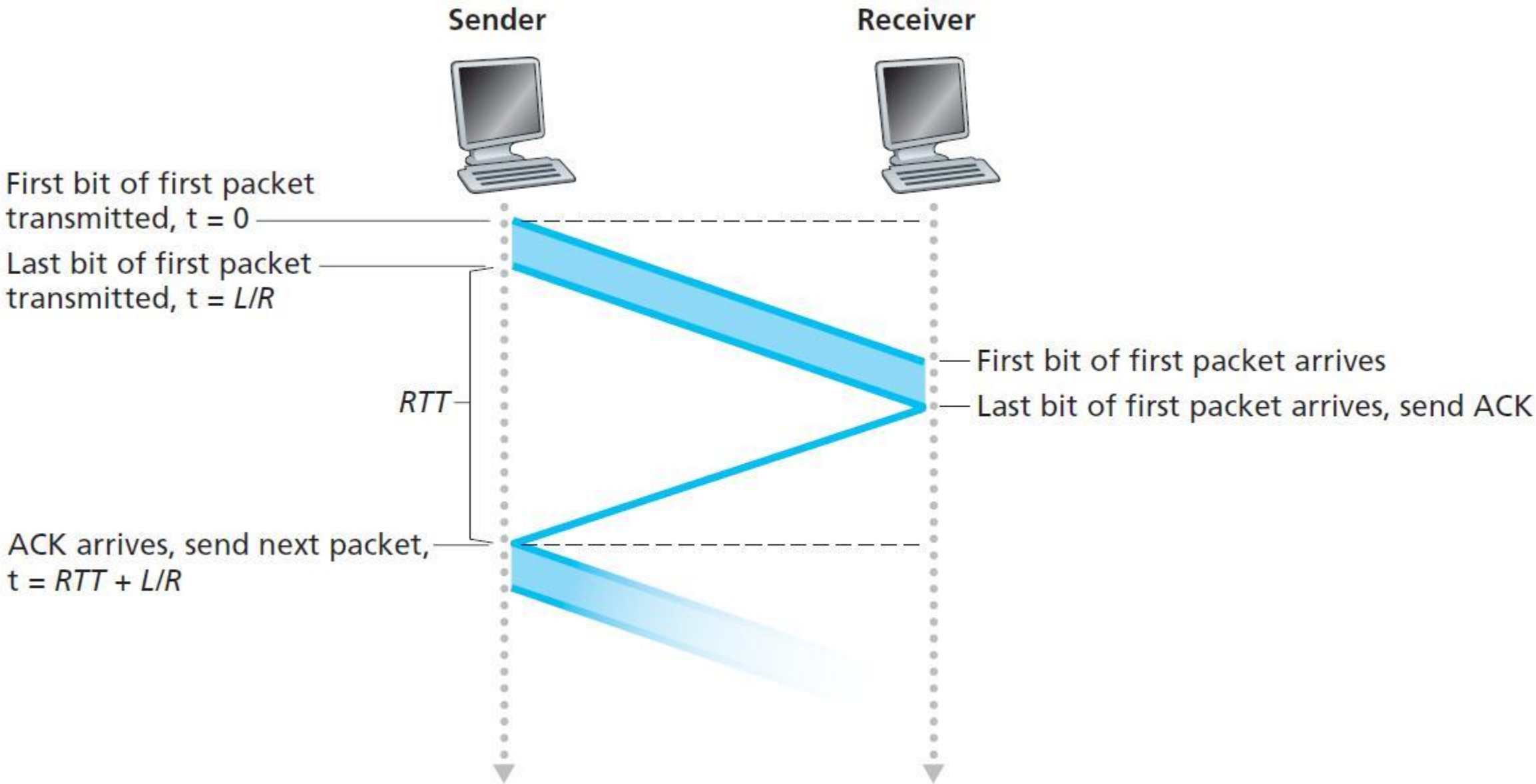


Figure 3.16 ♦ Operation of `rdt3.0`, the alternating-bit protocol

Analyze rdt3.0

- Sender
  - Wait until time out!
  - #countdown timer
- **alternating-bit**
  - sequence numbers alternate between 0 and 1
- **Stop and wait**



a. Stop-and-wait operation

rdt3.0 stop-and-wait

- Rdt 3.0, very bad performance
  - Packet size L: 1000 bytes
  - Link rate R: 1Gbps (10^9 bits per second).
  - RTT: 30 milliseconds.

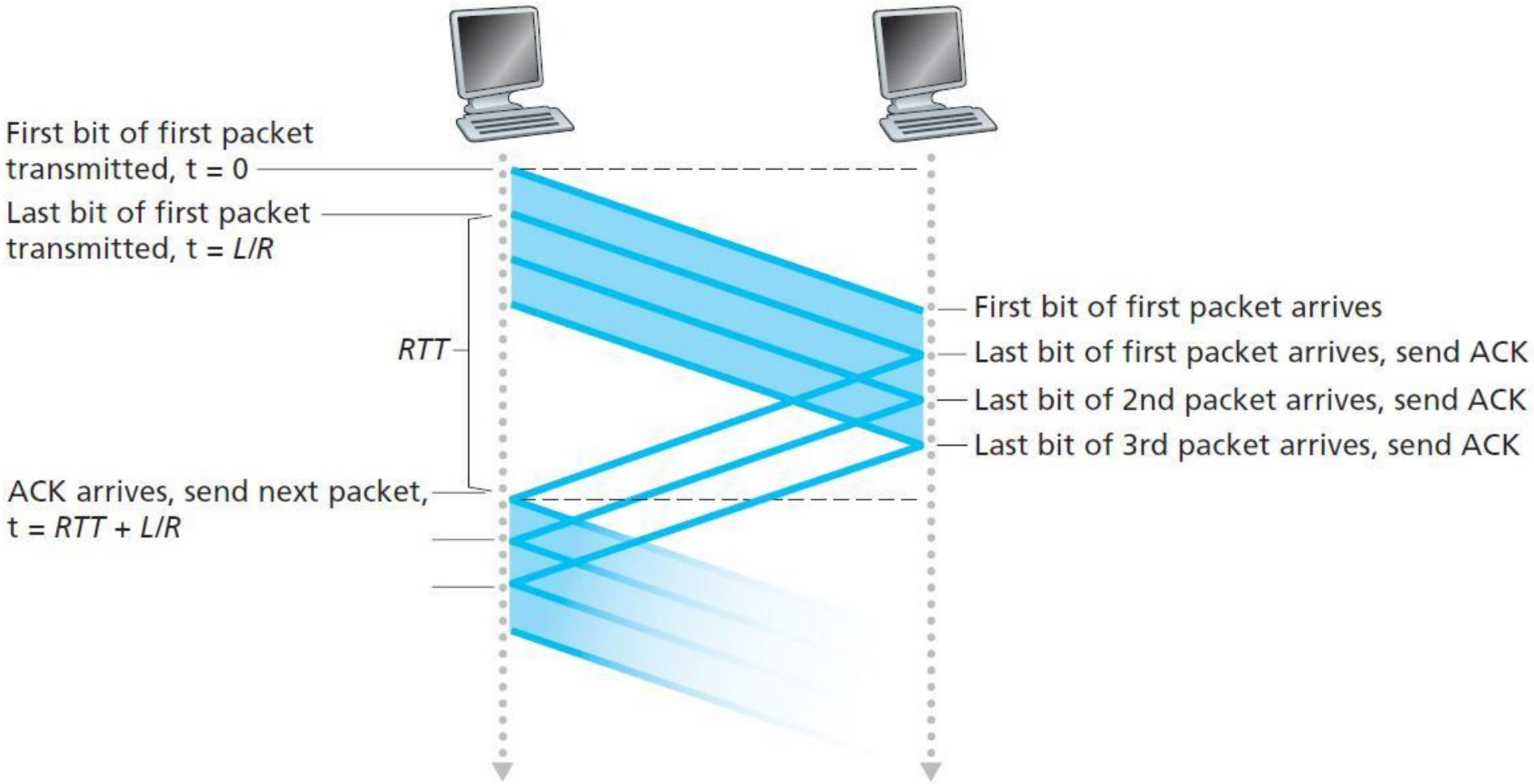
267 kbps

1,000 bytes in 30.008 ms

$$d_{trans} = \frac{L}{R} = \frac{8000 \text{ bits/packet}}{10^9 \text{ bits/sec}} = 8 \text{ microseconds}$$

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

Pipelined



b. Pipelined operation

Figure 3.18 ♦ Stop-and-wait and pipelined sending

Pipelined: Go-Back-N (GBN)

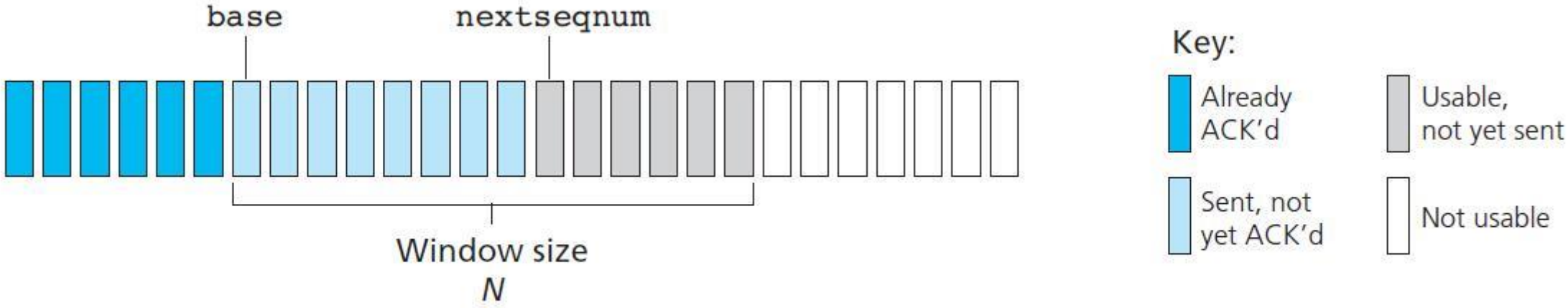


Figure 3.19 ♦ Sender's view of sequence numbers in Go-Back-N

sliding-window protocol.

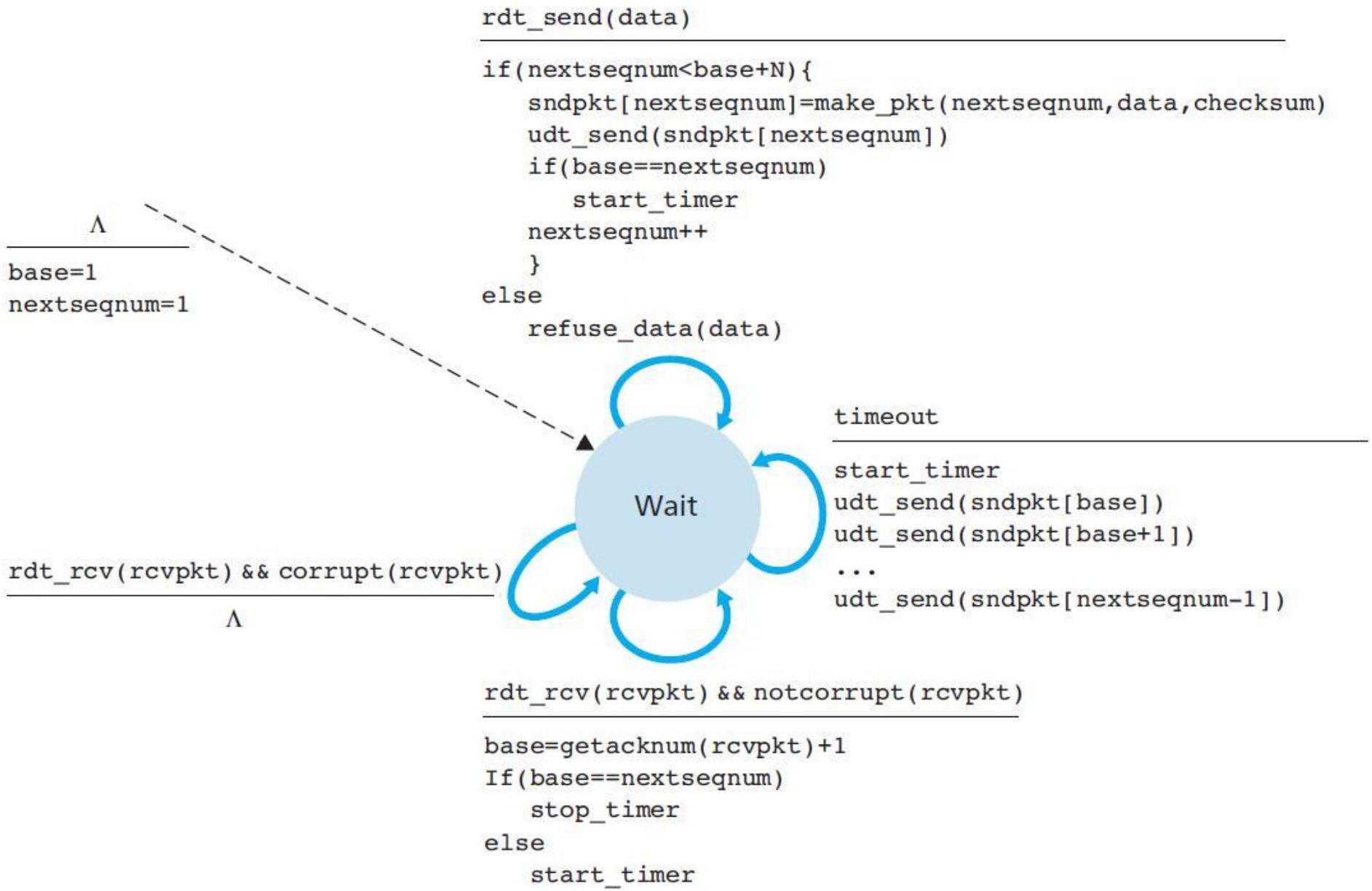


Figure 3.20 ♦ Extended FSM description of GBN sender

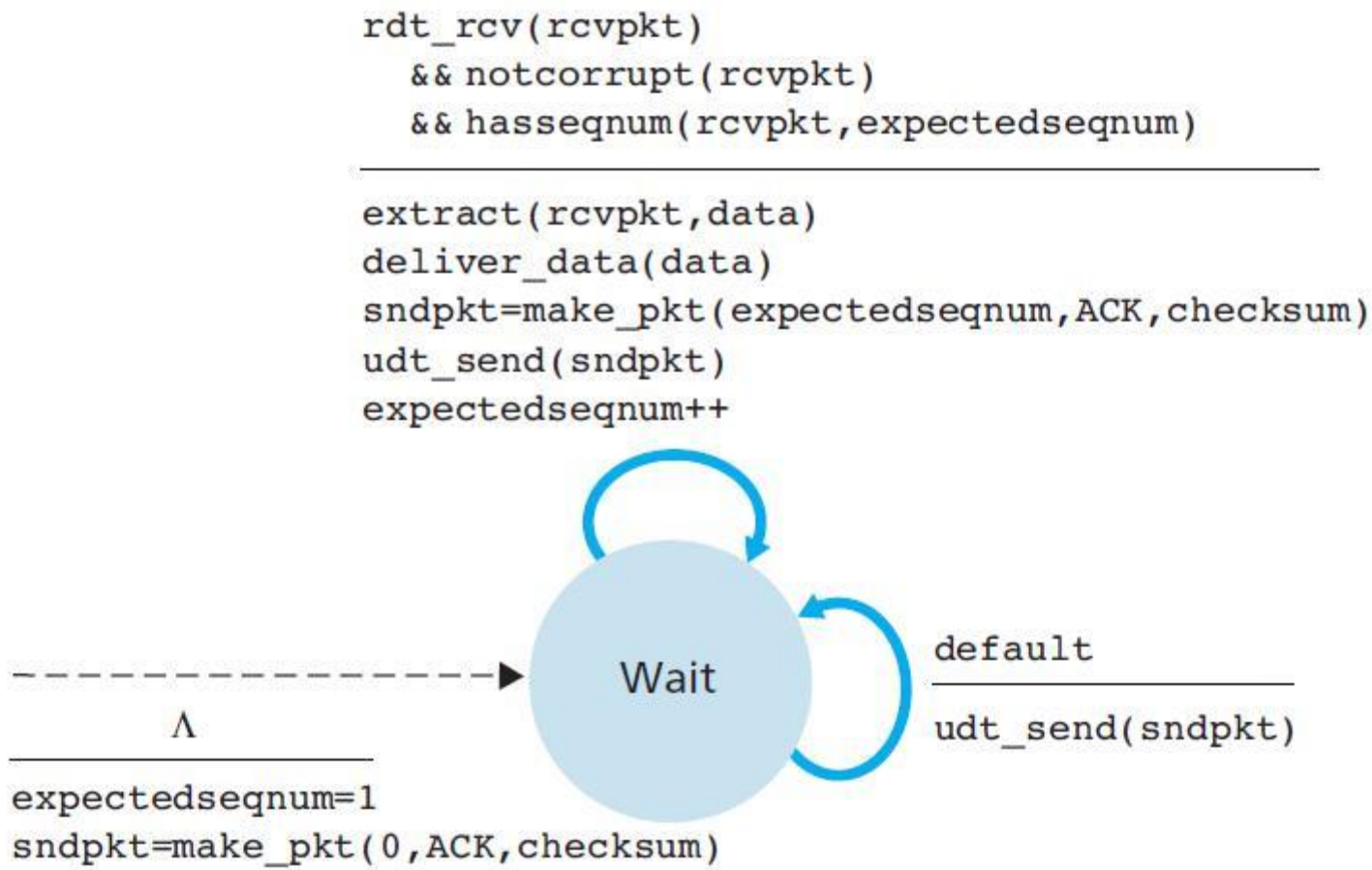


Figure 3.21 ♦ Extended FSM description of GBN receiver

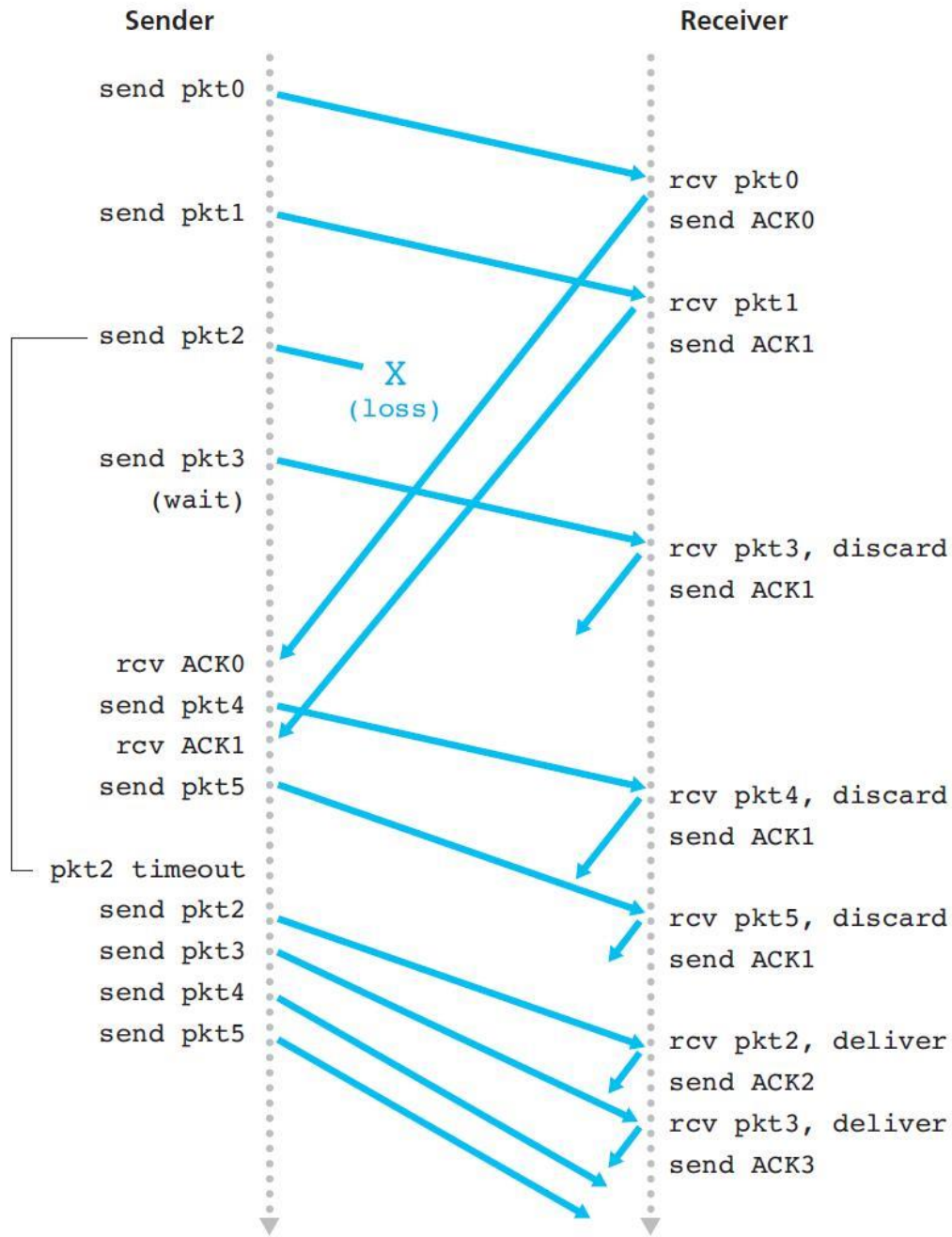


Figure 3.22 ♦ Go-Back-N in operation

Analyze GBN

- Sender
  - check if the window is full
  - **cumulative acknowledgment!!**
  - Timeout: resends all packets that have been previously sent but that have not yet been acknowledged.
- Receiver:
  - discards out-of-order packets

Pipelined: Selective Repeat (SR)

- Sender: Avoid unnecessary retransmissions
- Receiver *individually* acknowledge correctly received packets

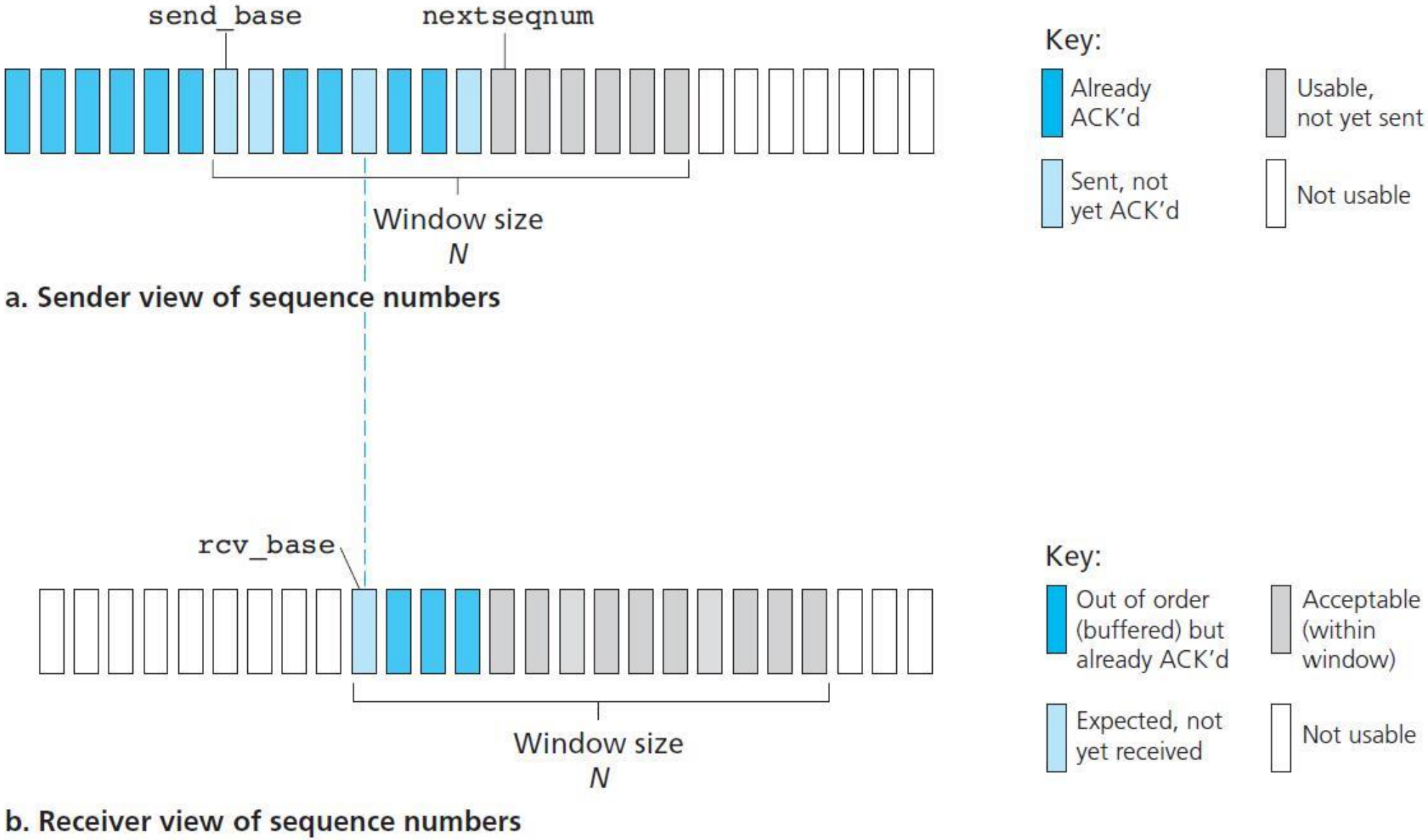


Figure 3.23 ♦ Selective-repeat (SR) sender and receiver views of sequence-number space

1. *Data received from above.* When data is received from above, the SR sender checks the next available sequence number for the packet. If the sequence number is within the sender's window, the data is packetized and sent; otherwise it is either buffered or returned to the upper layer for later transmission, as in GBN.
2. *Timeout.* Timers are again used to protect against lost packets. However, each packet must now have its own logical timer, since only a single packet will be transmitted on timeout. A single hardware timer can be used to mimic the operation of multiple logical timers [Varghese 1997].
3. *ACK received.* If an ACK is received, the SR sender marks that packet as having been received, provided it is in the window. If the packet's sequence number is equal to `send_base`, the window base is moved forward to the unacknowledged packet with the smallest sequence number. If the window moves and there are untransmitted packets with sequence numbers that now fall within the window, these packets are transmitted.

Figure 3.24 ♦ SR sender events and actions

1. *Packet with sequence number in* `[rcv_base, rcv_base+N-1]` *is correctly received.* In this case, the received packet falls within the receiver's window and a selective ACK packet is returned to the sender. If the packet was not previously received, it is buffered. If this packet has a sequence number equal to the base of the receive window (`rcv_base` in Figure 3.22), then this packet, and any previously buffered and consecutively numbered (beginning with `rcv_base`) packets are delivered to the upper layer. The receive window is then moved forward by the number of packets delivered to the upper layer. As an example, consider Figure 3.26. When a packet with a sequence number of `rcv_base=2` is received, it and packets 3, 4, and 5 can be delivered to the upper layer.
2. *Packet with sequence number in* `[rcv_base-N, rcv_base-1]` *is correctly received.* In this case, an ACK must be generated, even though this is a packet that the receiver has previously acknowledged.
3. *Otherwise.* Ignore the packet.

Figure 3.25 ♦ SR receiver events and actions

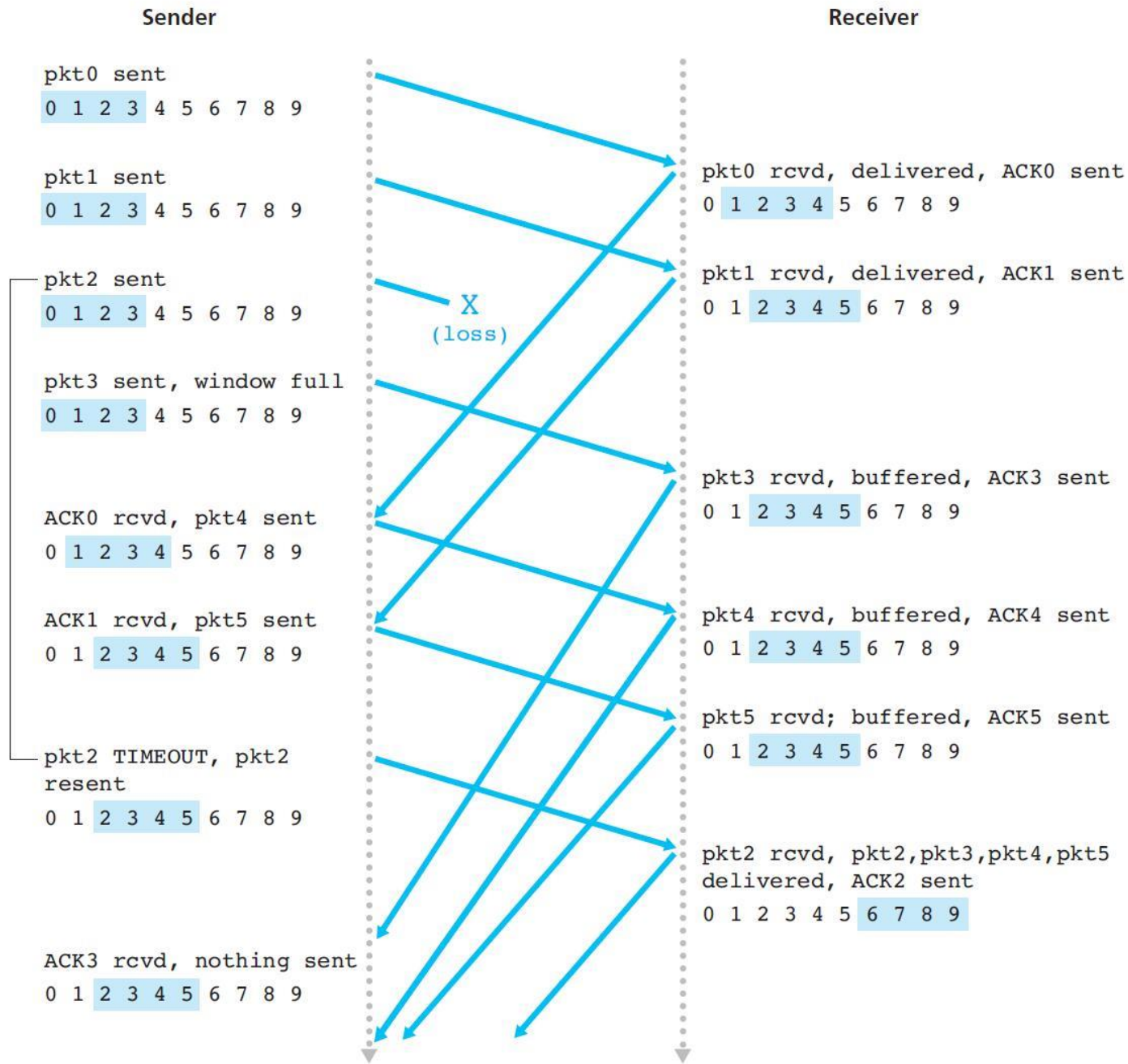


Figure 3.26 ♦ SR operation