Computer Networks

Lisa Frye, Instructor
frye@kutztown.edu
Kutztown University

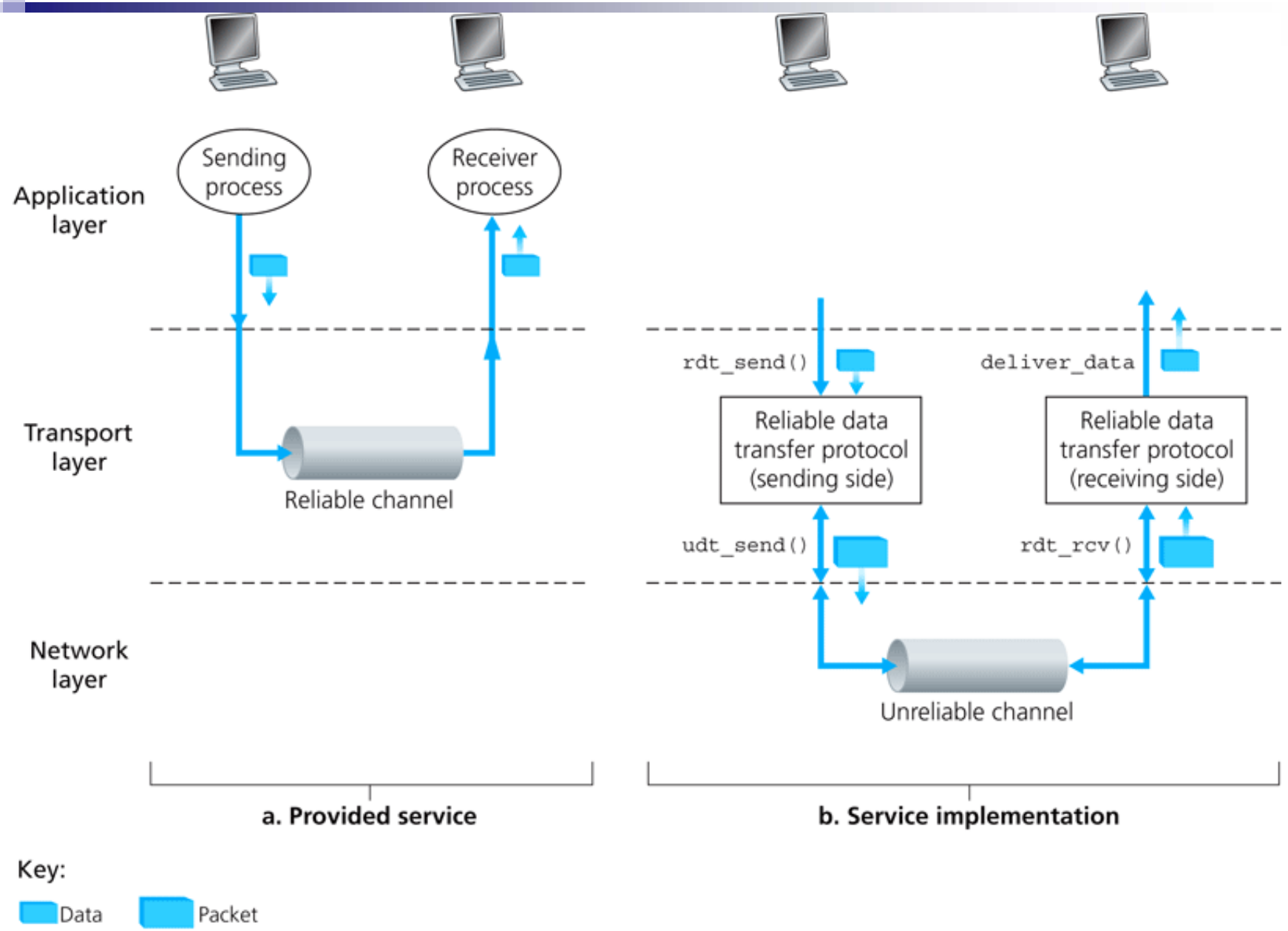
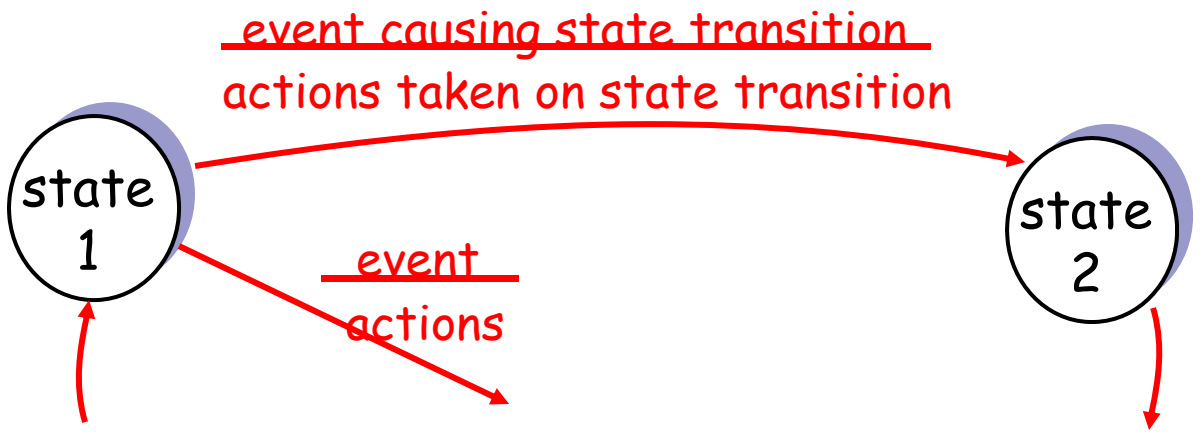


Figure 3.8 ♦ Reliable data transfer: Service model and service implementation

Reliable data transfer: getting started

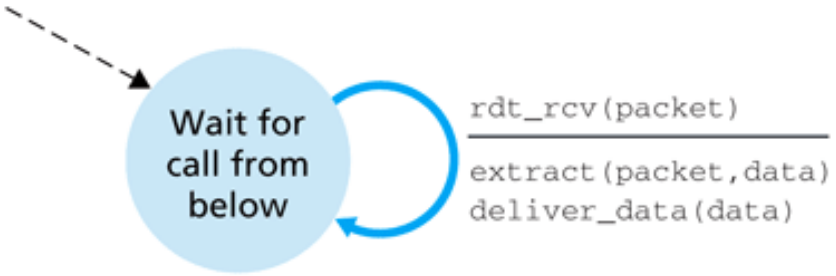
- incrementally develop sender, receiver sides of reliable data transfer protocol (rdt)
- consider only unidirectional data transfer
  - but control info will flow on both directions!
- use finite state machines (FSM) to specify sender, receiver



Rdt1.0: reliable transfer over a reliable channel

- underlying channel perfectly reliable
  - no bit errors
  - no loss of packets

a. rdt1.0: sending side

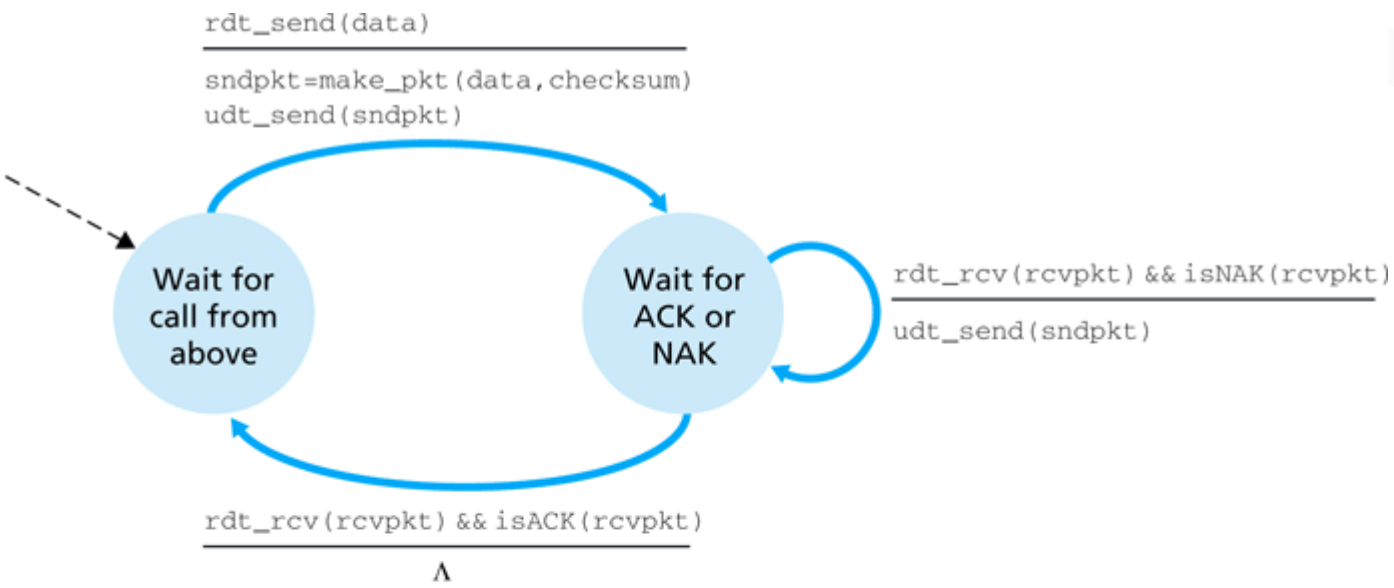


b. rdt1.0: receiving side

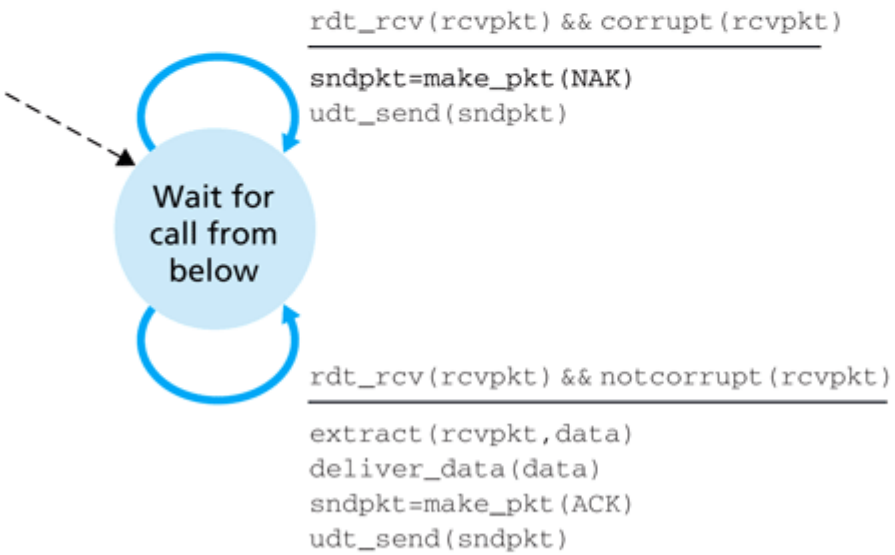
Figure 3.9 ♦ rdt1.0 – A protocol for a completely reliable channel

Rdt2.0: channel with bit errors

- underlying channel may flip bits in packet
  - recall: UDP checksum to detect bit errors
- *the* question: how to recover from errors:
  - *acknowledgements (ACKs):*
  - *negative acknowledgements (NAKs):*
- new mechanisms in **`rdt2.0`** (beyond **`rdt1.0`**):
  - error detection
  - receiver feedback: control msgs (ACK,NAK) rcvr->sender



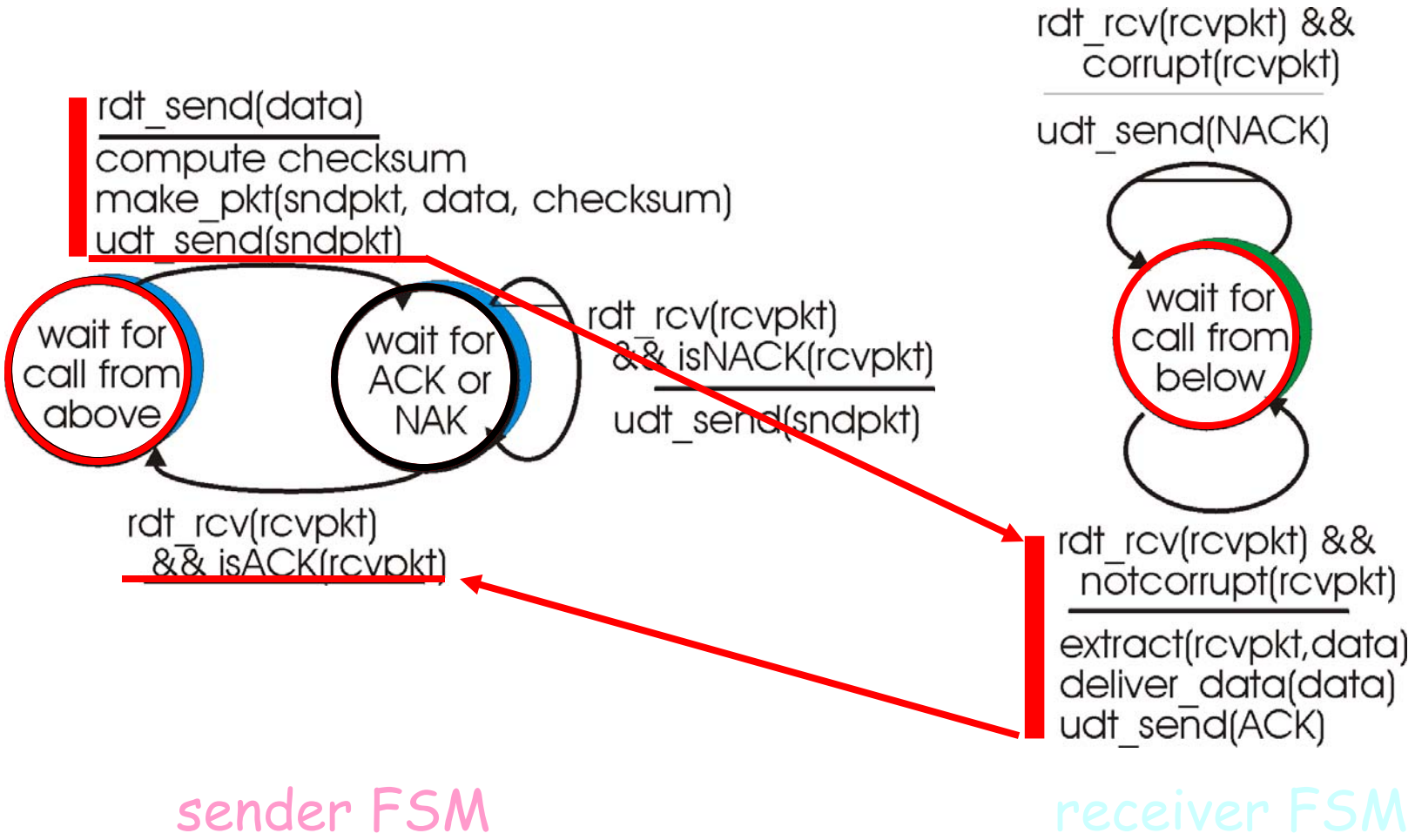
a. rdt2.0: sending side



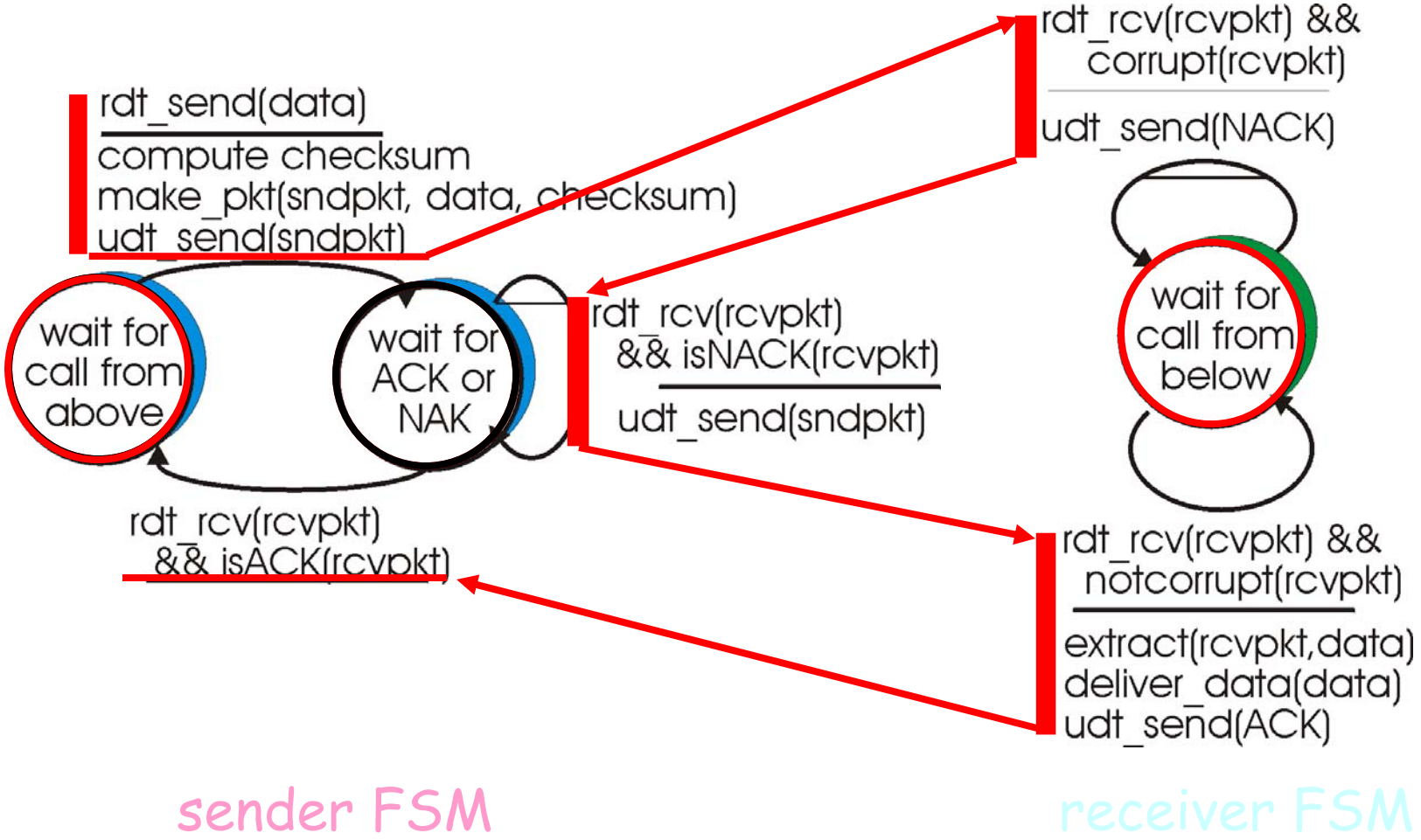
b. rdt2.0: receiving side

Figure 3.10 ♦ rdt2.0–A protocol for a channel with bit errors

rdt2.0: in action (no errors)



rdt2.0: in action (error scenario)



Problems with rdt 2.0

- What if ACK or NAK packet is corrupt?
- How should the protocol recover from errors in ACK or NAK packets???

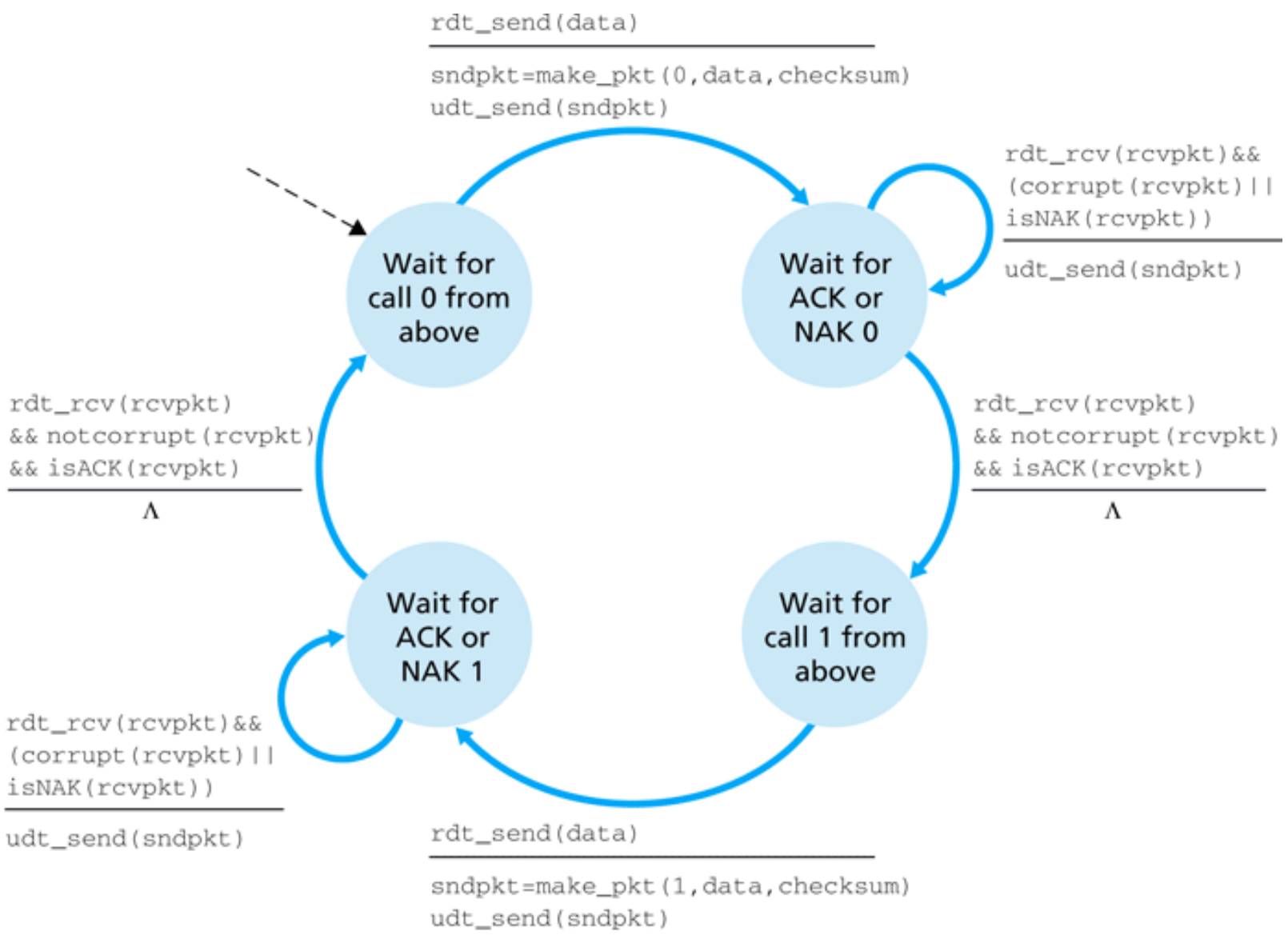


Figure 3.11 ♦ rdt2.1 sender

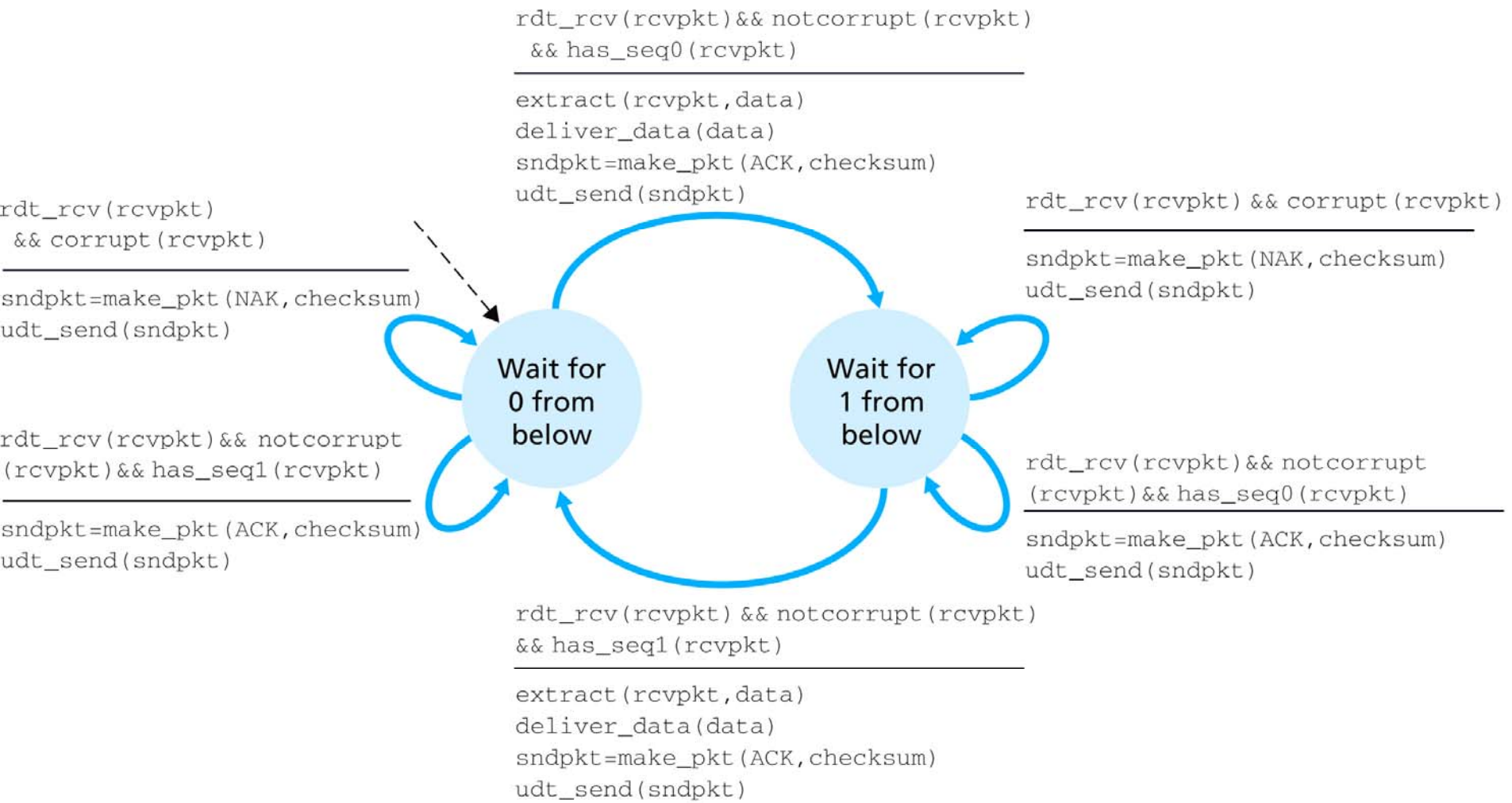


Figure 3.12 ♦ rdt2.1 receiver

- In rdt 2.1 is it possible for the sender and receiver to enter into a deadlock state?
  - Yes – true
  - No - false

rdt2.2: a NAK-free protocol

- Instead of NAK, receiver sends ACK for last packet received OK
  - Receiver must *explicitly* include seq # of pkt being ACKed
- Duplicate ACK at sender results in same action as NAK: *retransmit current packet*

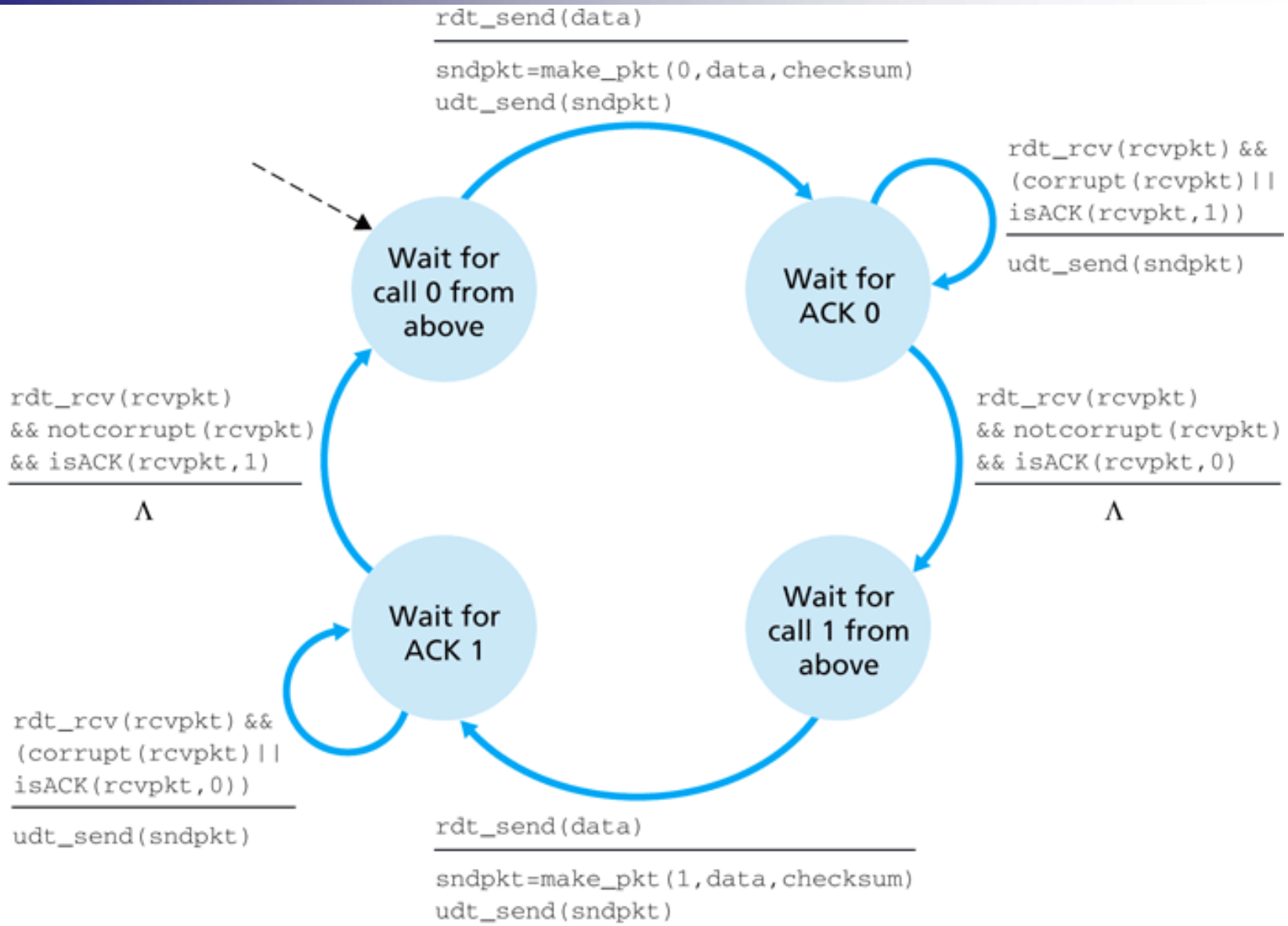


Figure 3.13 ♦ rdt2.2 sender

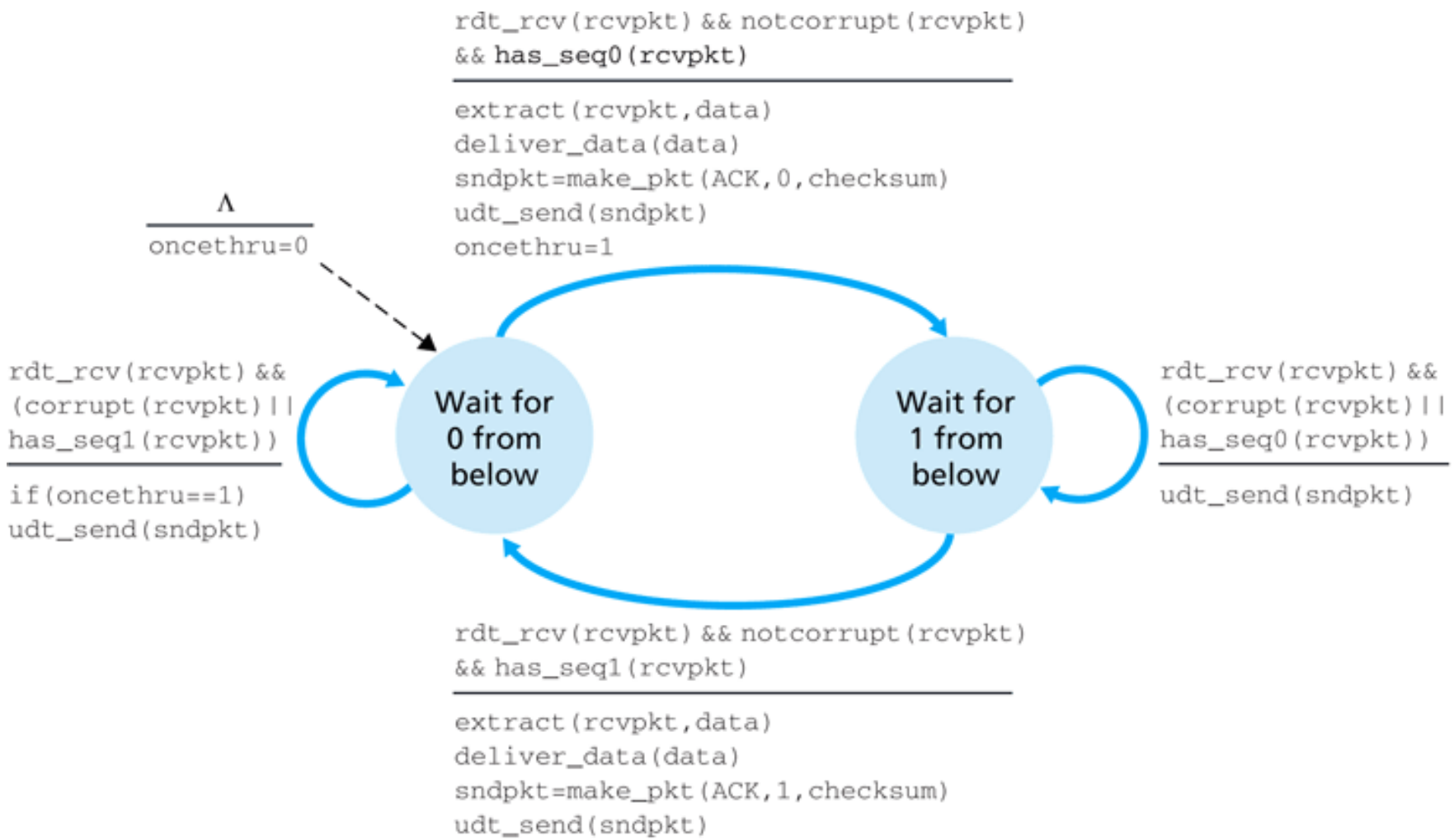


Figure 3.14 ♦ rdt2.2 receiver

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

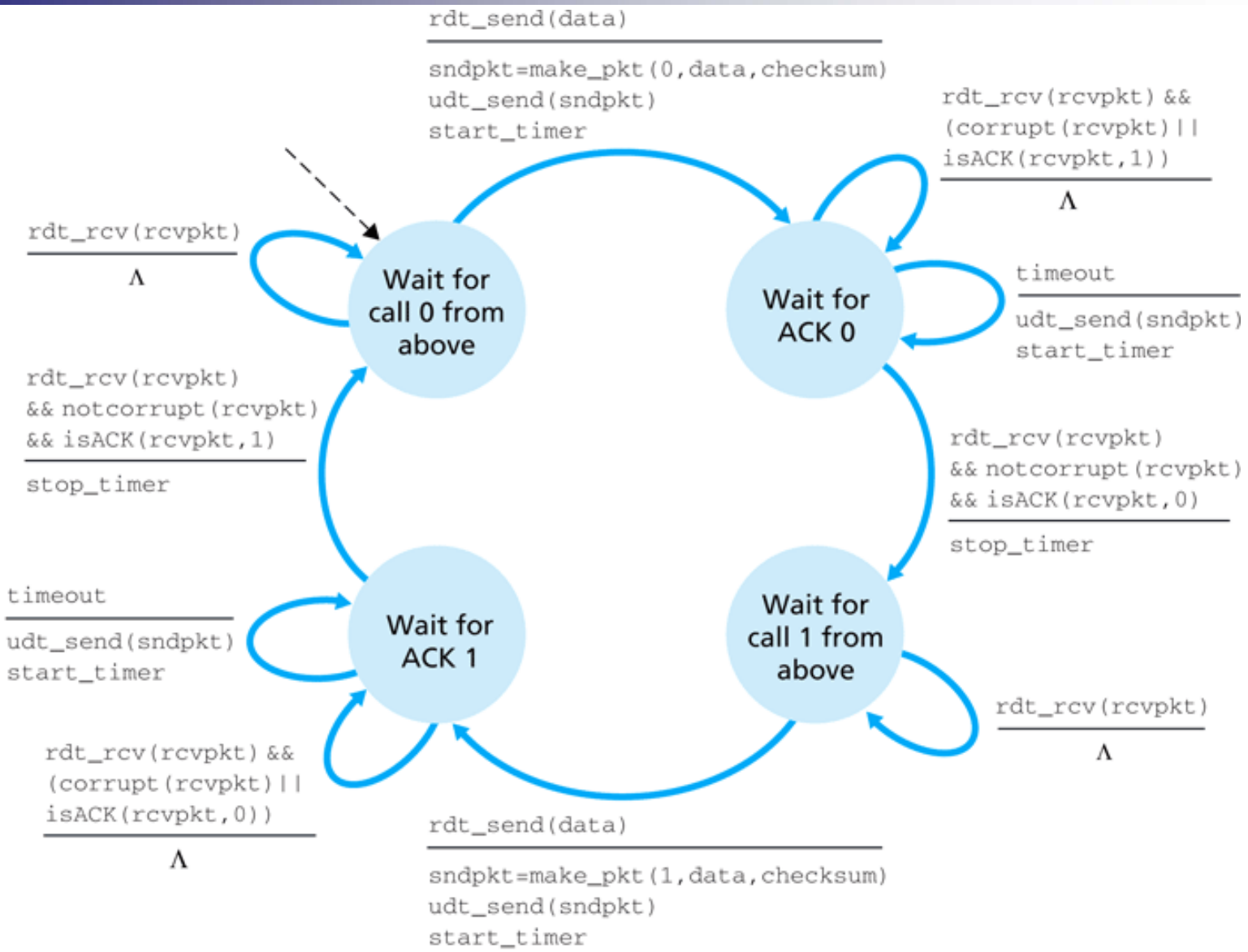
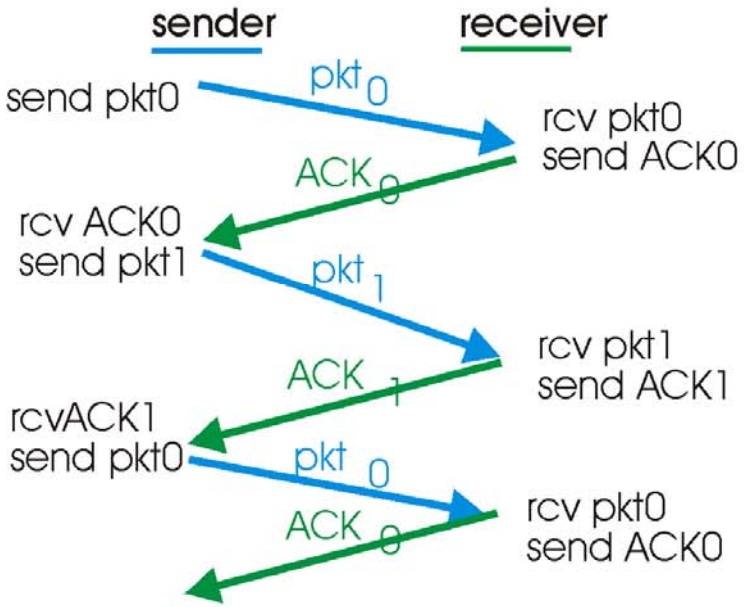
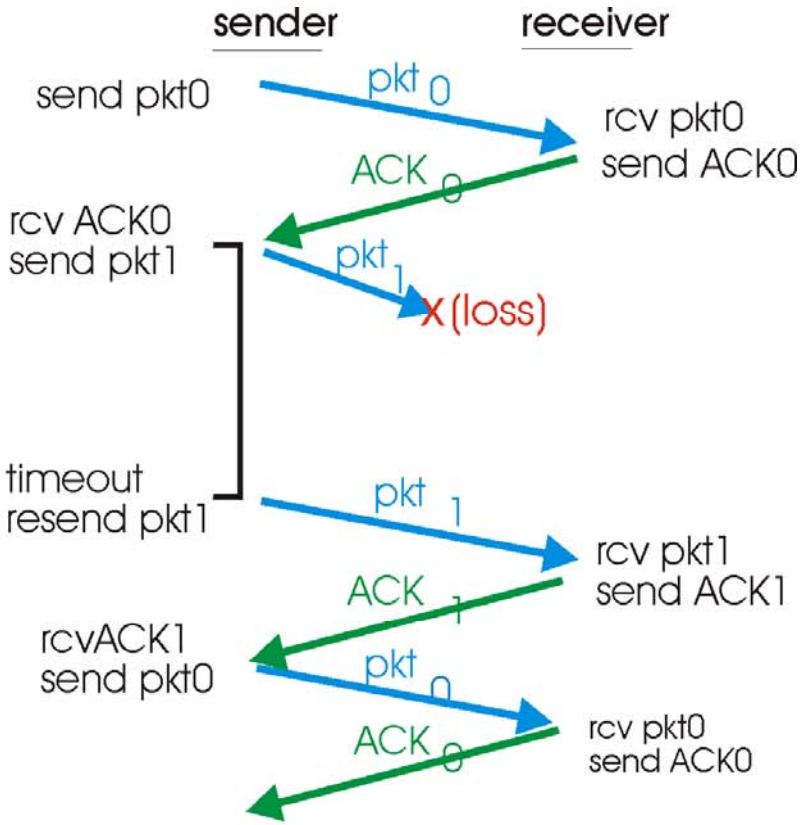


Figure 3.15 ♦ rdt3.0 sender

rdt3.0 in action

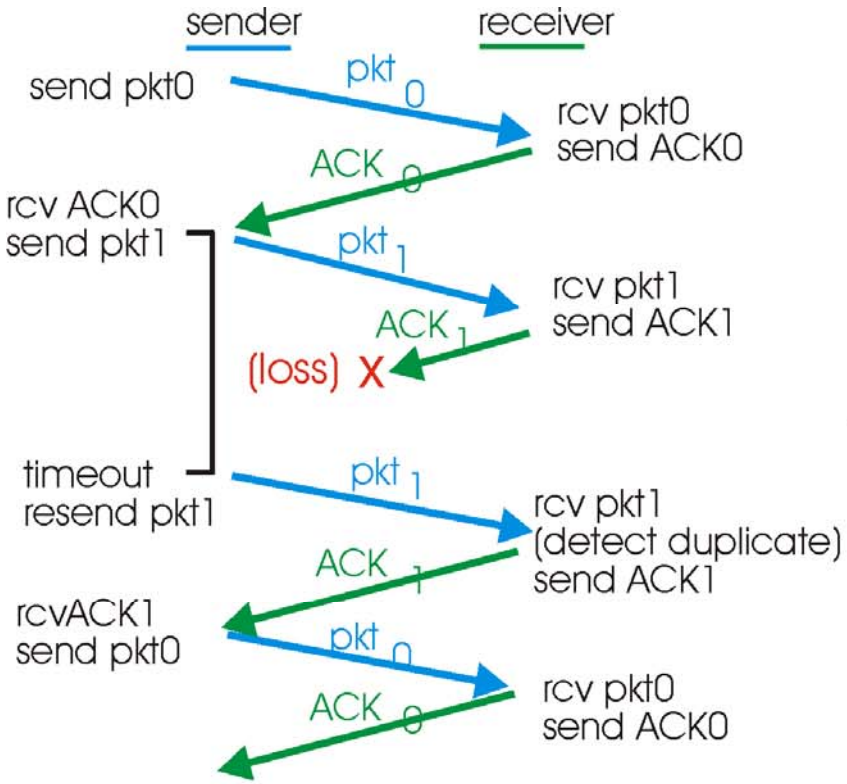


(a) operation with no loss

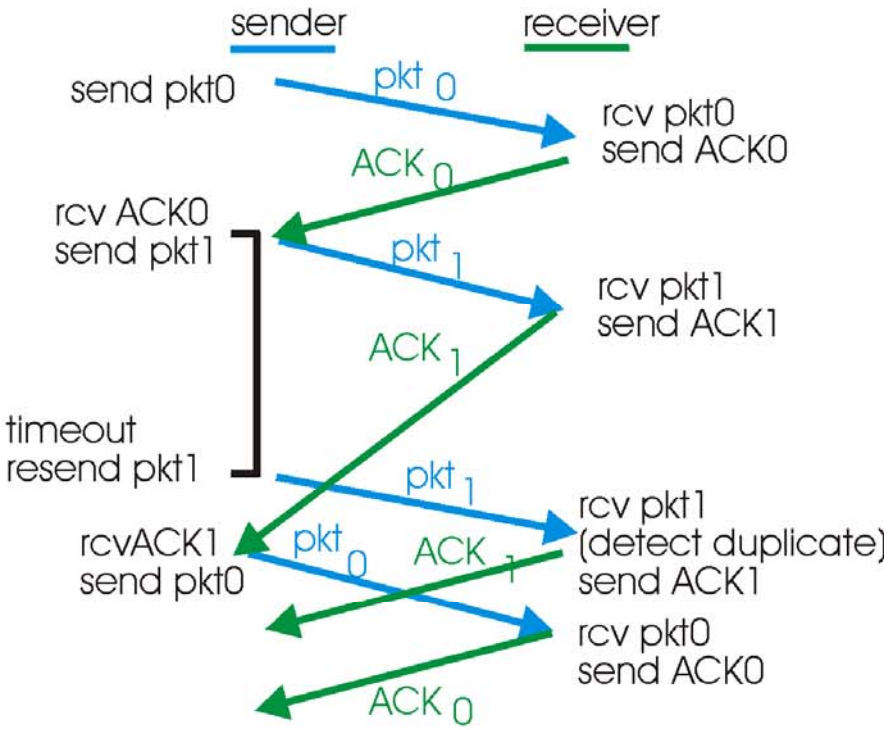


(b) lost packet

rdt3.0 in action



(c) lost ACK



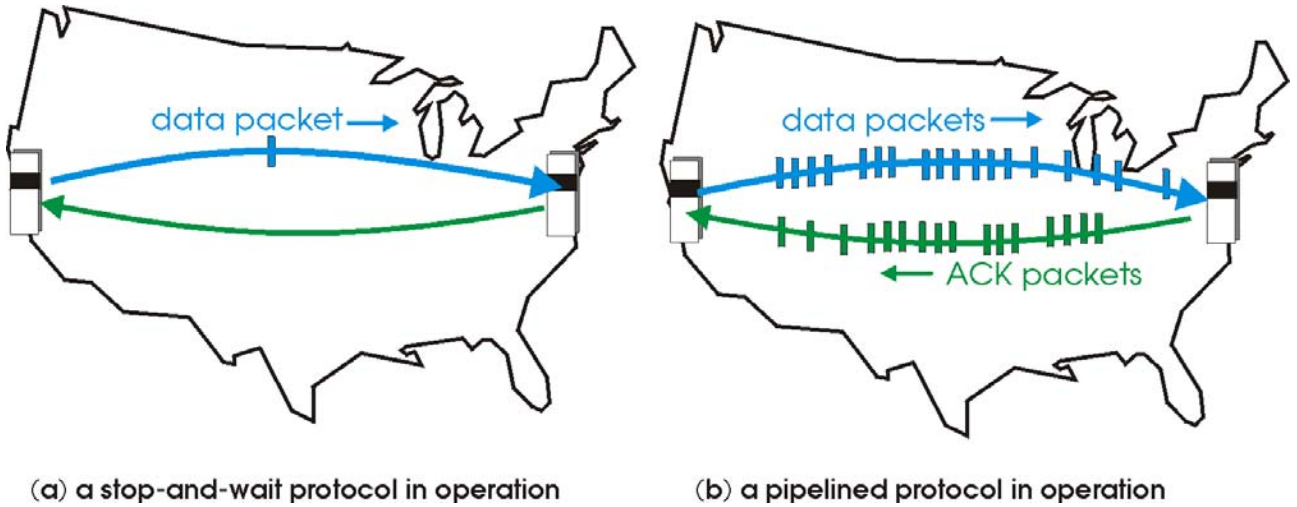
(d) premature timeout

- In protocol rdt 3.0, the ACK packets flowing from the receiver to the sender do not have sequence numbers (although they do have an ACK field that contains the sequence number of the packet they are acknowledging). Why is it that our ACK packets do not require sequence numbers?
  1. No need to know if an ACK is a duplicate or not
  2. ACKS do not have to be received in order
  3. No need to know the size of the ACK packet

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

- Pipelined protocols -> sliding window protocols

Go-Back-N Protocol

- A sliding-window protocol

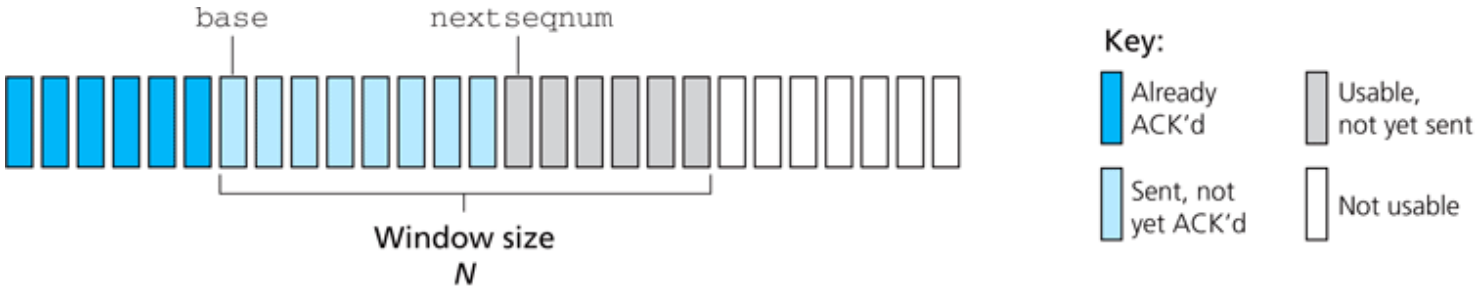


Figure 3.19 ♦ Sender's view of sequence numbers in Go-Back-N

- Timeouts
- Cumulative acknowledgement
- Performance
- Selective-Repeat Protocols

rdt 3.0 Receiver

- Draw the FSM for the receiver side of the protocol rdt 3.0

- NOTE: Table 3.1 useful for rdt concepts