Last time

- Mobility in Cellular networks
  - HLR, VLR, MSC
  - Handoff
- Transport Layer
  - Introduction
  - Multiplexing / demultiplexing
  - UDP

This time

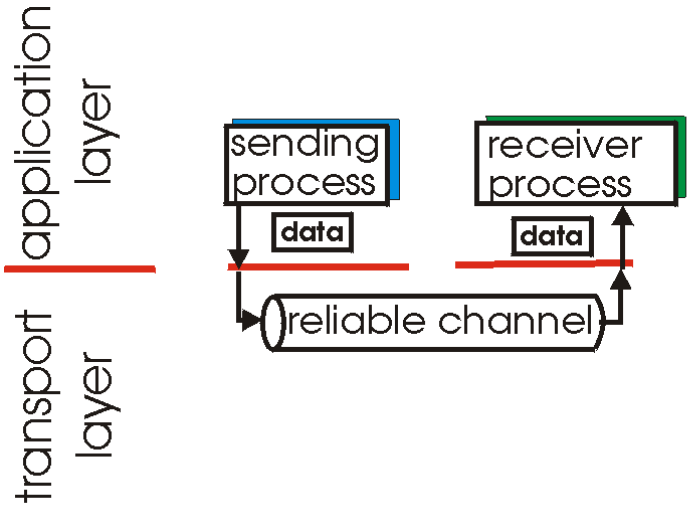
- Reliable Data Transfer
- Midterm review

Chapter 3 outline

- 3.1 Transport-layer services
- 3.2 Multiplexing and demultiplexing
- 3.3 Connectionless transport: UDP
- 3.4 Principles of reliable data transfer
- 3.5 Connection-oriented transport: TCP
  - segment structure
  - reliable data transfer
  - flow control
  - connection management
- 3.6 Principles of congestion control
- 3.7 TCP congestion control

Principles of Reliable data transfer

- Important in app., transport, link layers
- Top-10 list of important networking topics!

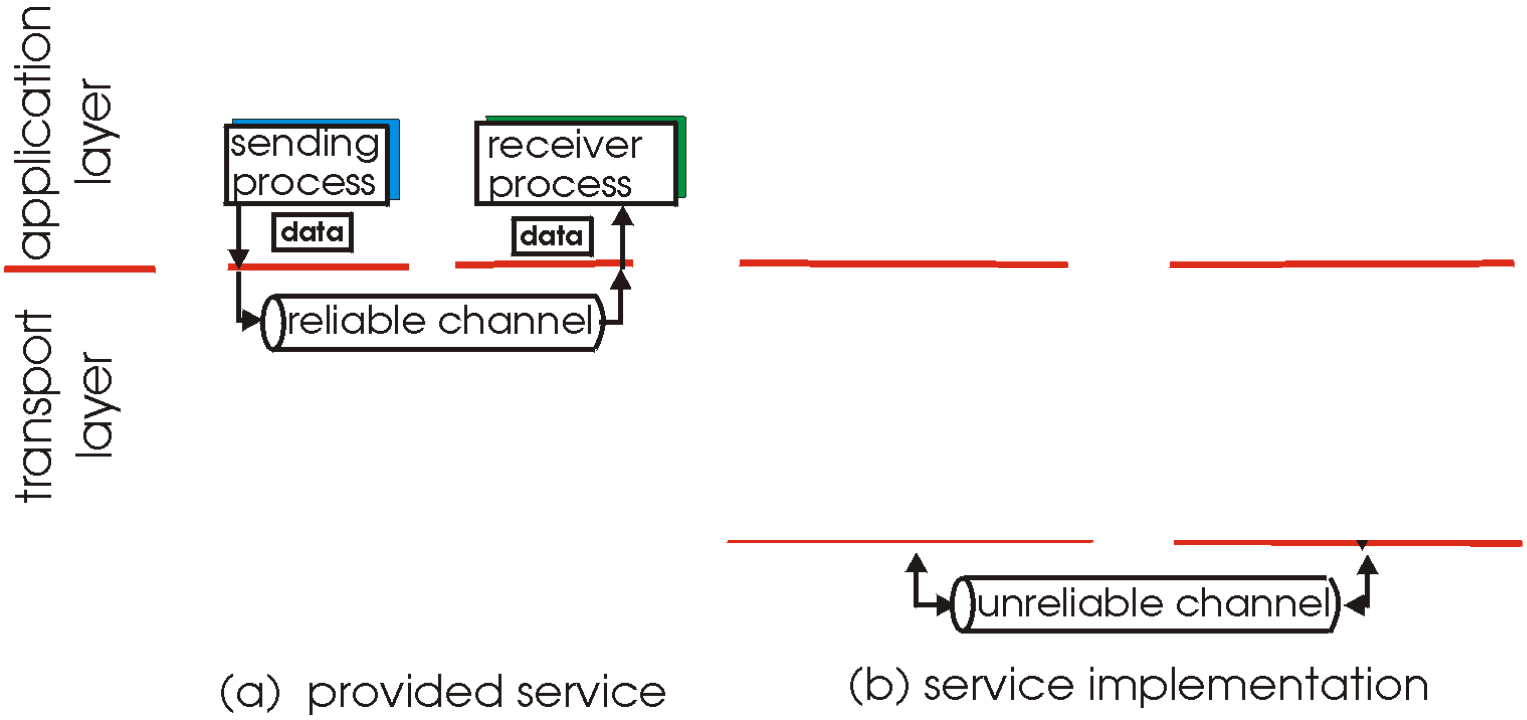


(a) provided service

- Characteristics of unreliable channel will determine complexity of reliable data transfer protocol (rdt)

Principles of Reliable data transfer

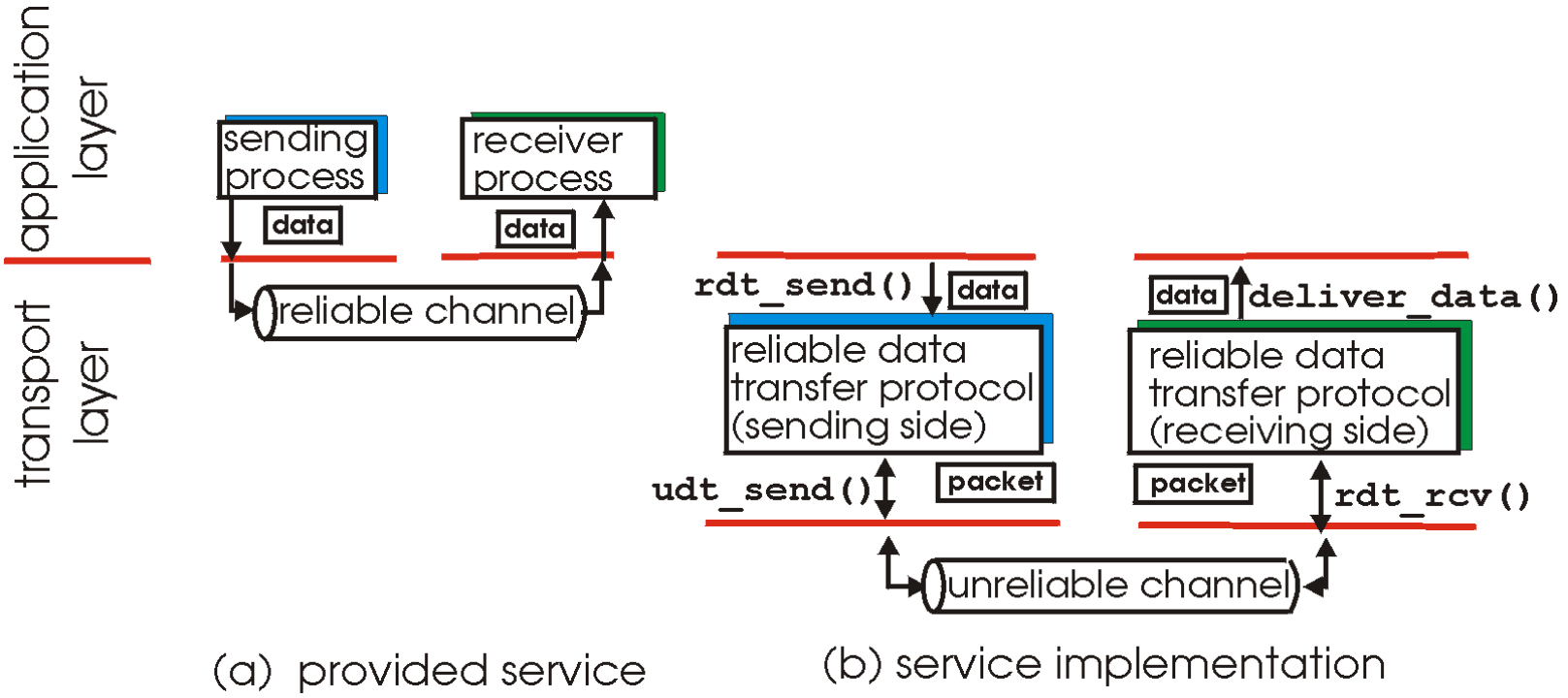
- Important in app., transport, link layers
- Top-10 list of important networking topics!



- Characteristics of unreliable channel will determine complexity of reliable data transfer protocol (rdt)

Principles of Reliable data transfer

- Important in app., transport, link layers
- Top-10 list of important networking topics!

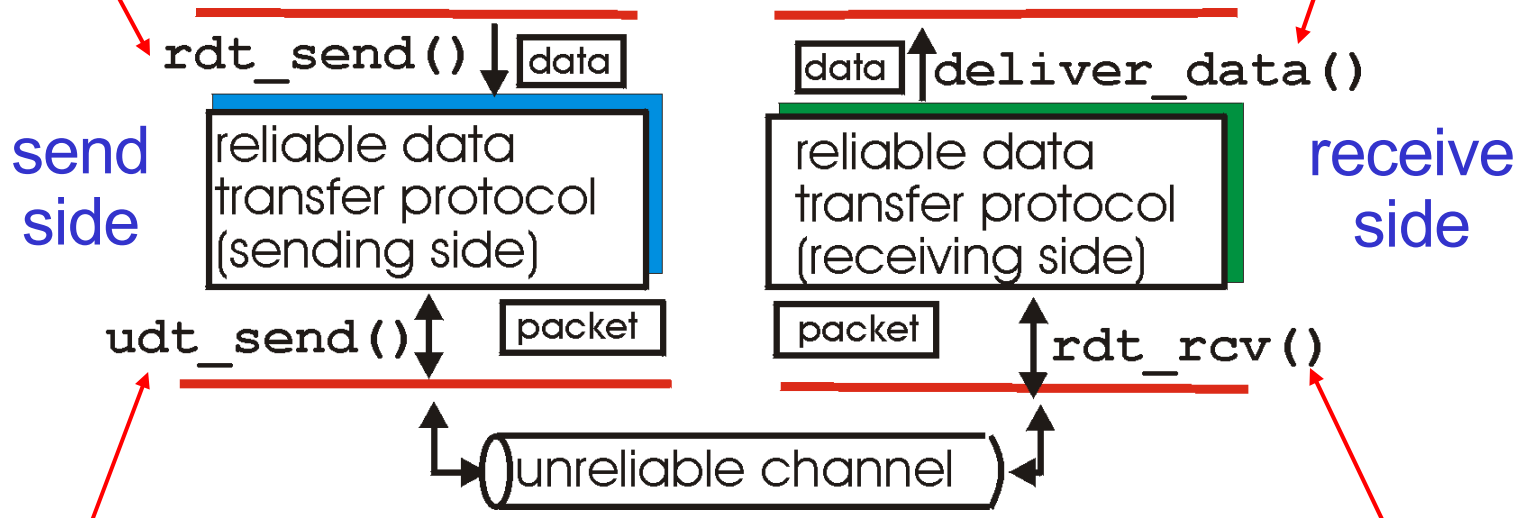


- Characteristics of unreliable channel will determine complexity of reliable data transfer protocol (rdt)

Reliable data transfer: getting started

`rdt_send()`: called from above, (e.g., by app.). Passed data to deliver to receiver upper layer

`deliver_data()`: called by **`rdt`** to deliver data to upper



`udt_send()`: called by rdt, to transfer packet over unreliable channel to receiver

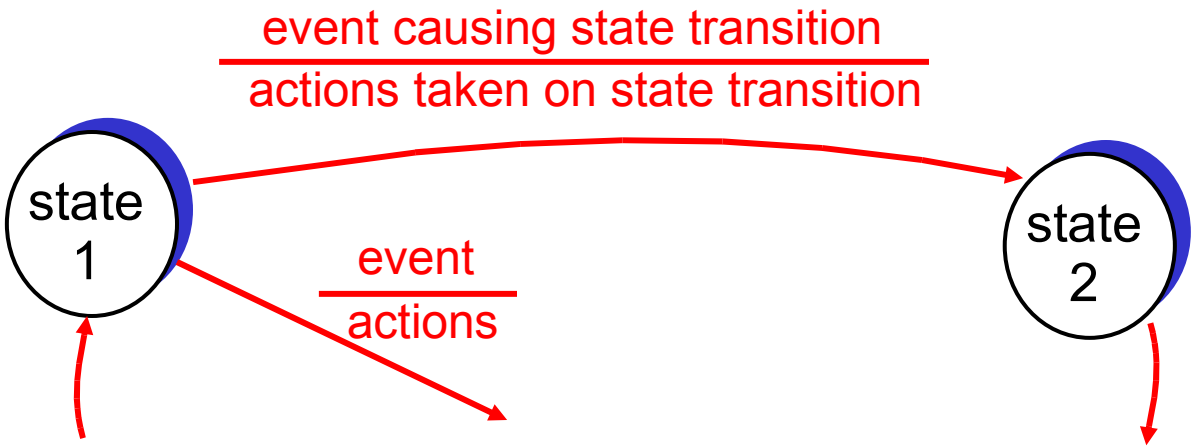
`rdt_rcv()`: called when packet arrives on rcv-side of channel

Reliable data transfer: getting started

We'll:

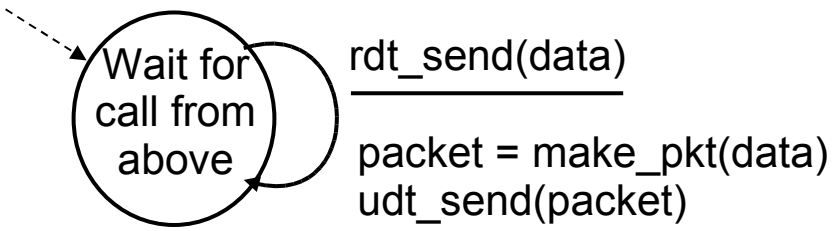
- Incrementally develop sender, receiver sides of reliable data transfer protocol (rdt)
- Consider only unidirectional data transfer
  - but control info will flow on both directions!
- Use finite state machines (FSM) to specify sender, receiver

state: when in this "state" next state uniquely determined by next event

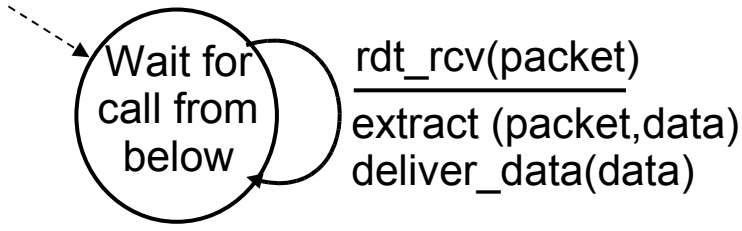


Rdt1.0: reliable transfer over a reliable channel

- Underlying channel perfectly reliable
  - no bit errors
  - no loss of packets
- Separate FSMs for sender, receiver:
  - sender sends data into underlying channel
  - receiver read data from underlying channel



sender

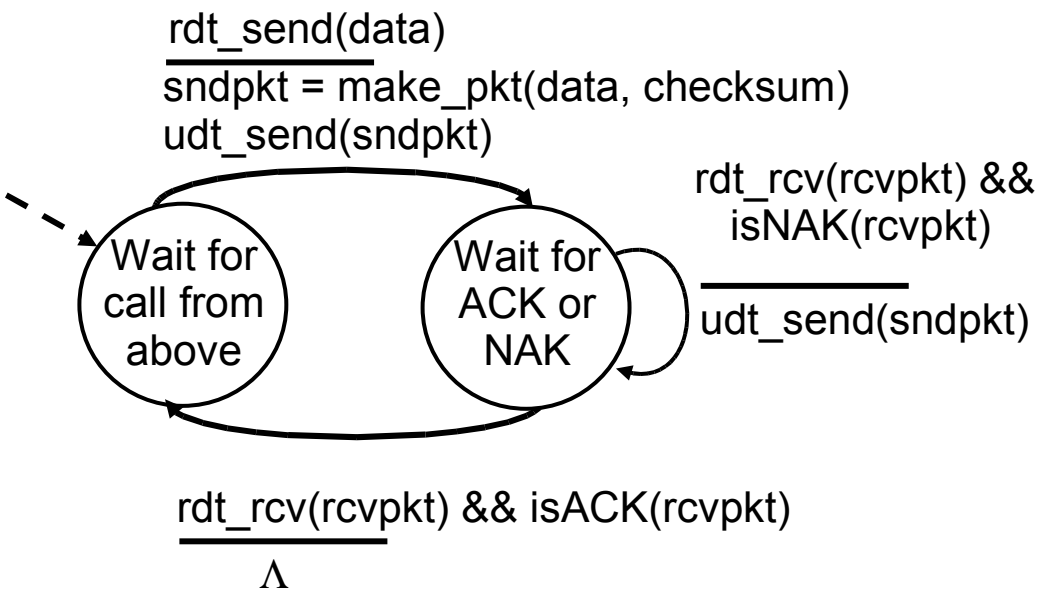


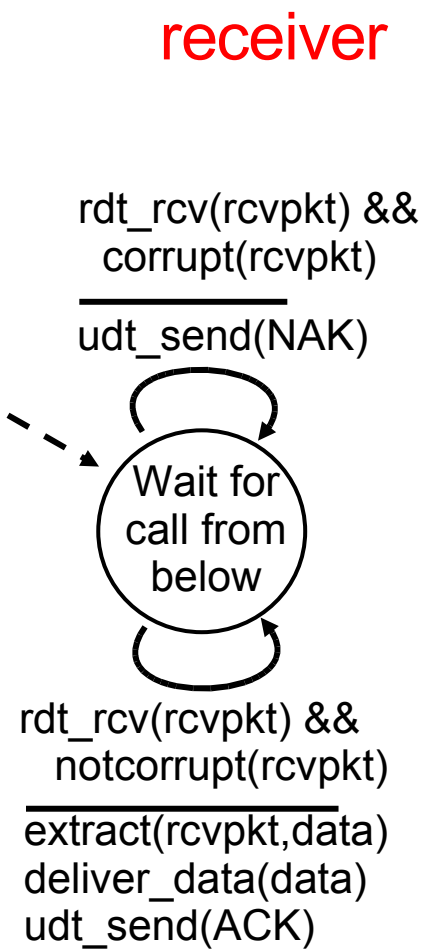
receiver

Rdt2.0: channel with bit errors

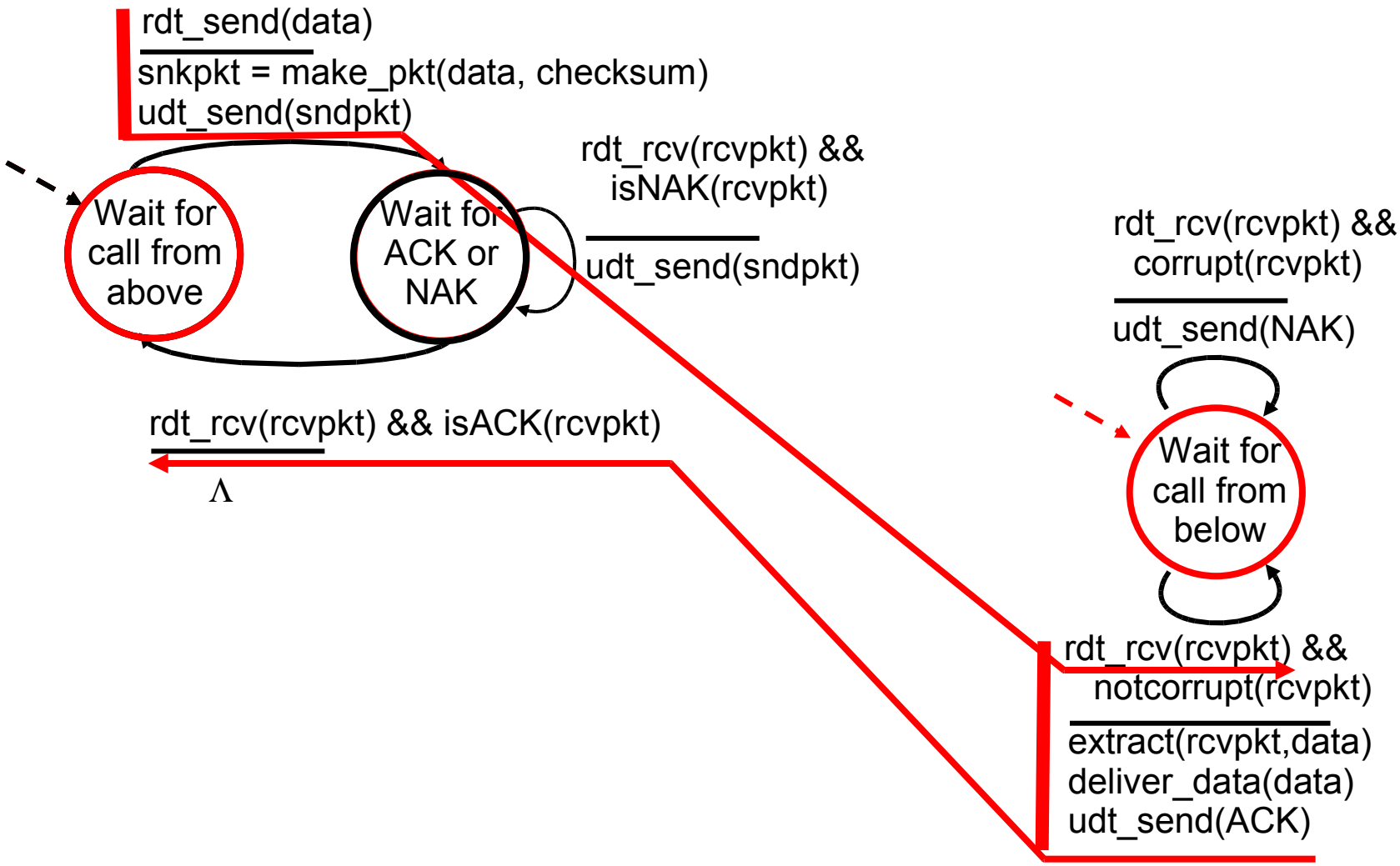
- Underlying channel may flip bits in packet
  - checksum to detect bit errors
- *The* question: how to recover from errors:
  - *acknowledgements (ACKs):* receiver explicitly tells sender that pkt received OK
  - *negative acknowledgements (NAKs):* receiver explicitly tells sender that pkt had errors
  - 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: FSM specification

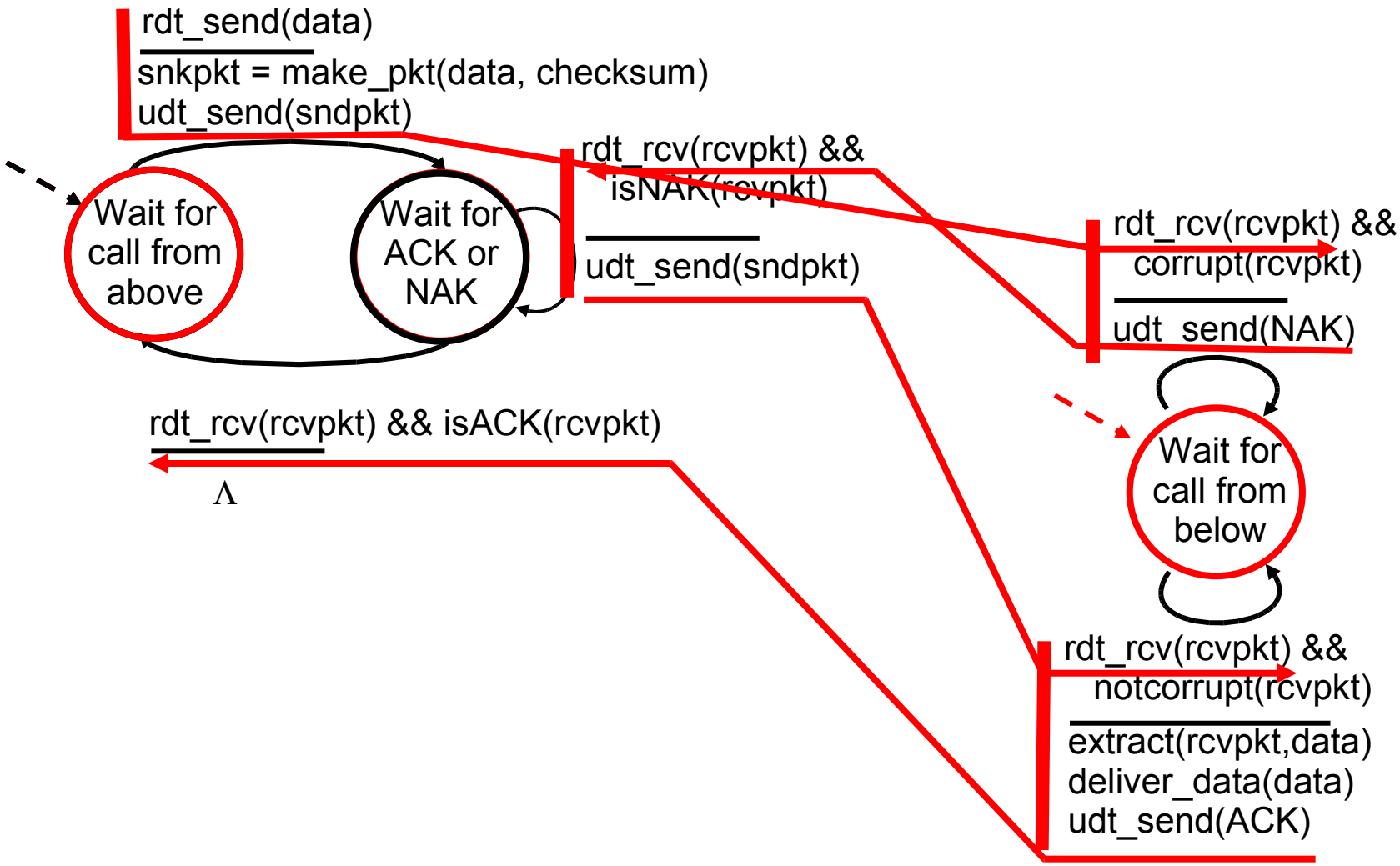




rdt2.0: operation with no errors



rdt2.0: error scenario



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

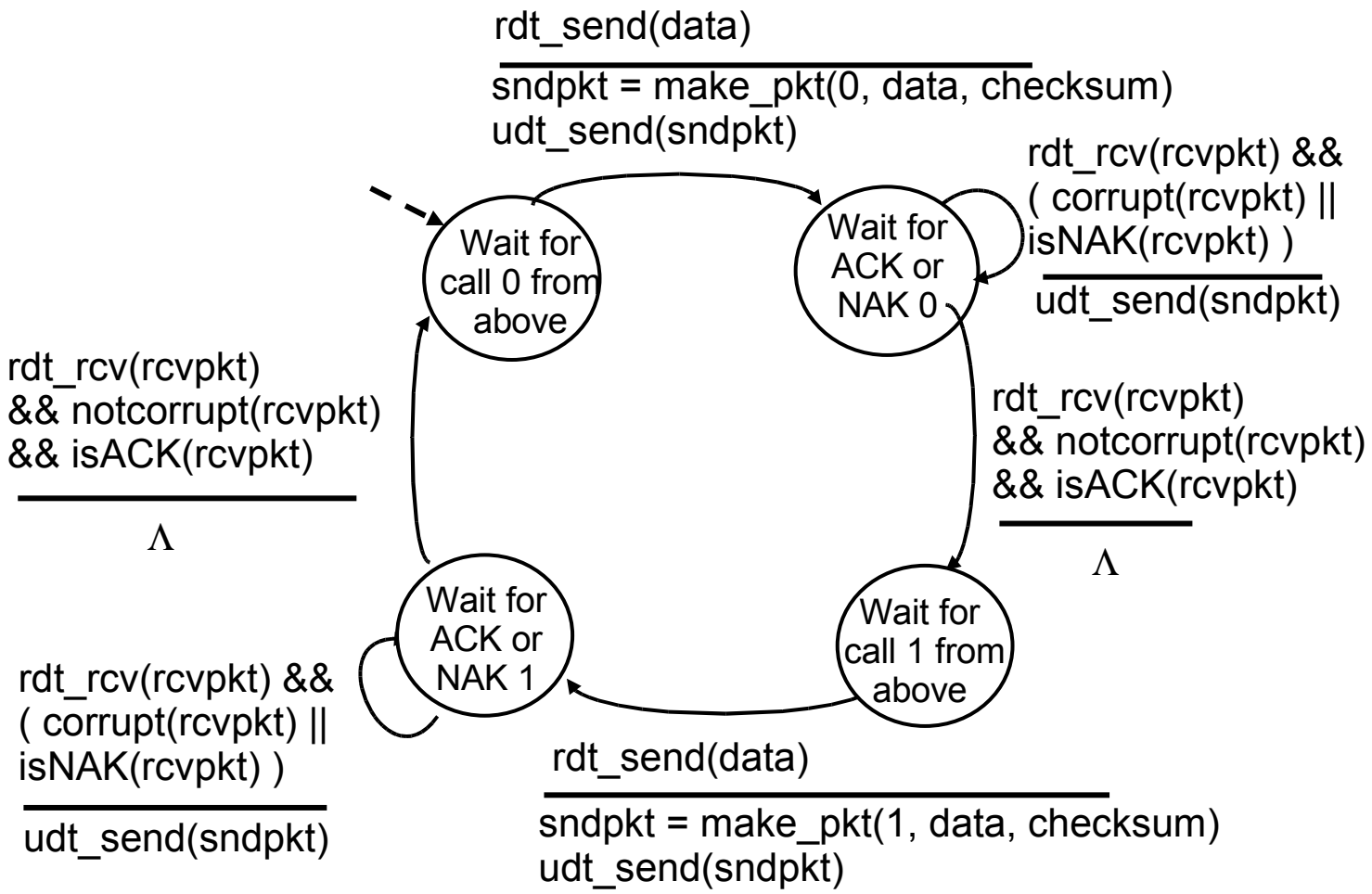
Handling duplicates:

- Sender retransmits current pkt if ACK/NAK garbled
- Sender adds *sequence number* to each pkt
- Receiver discards (doesn't deliver up) duplicate pkt

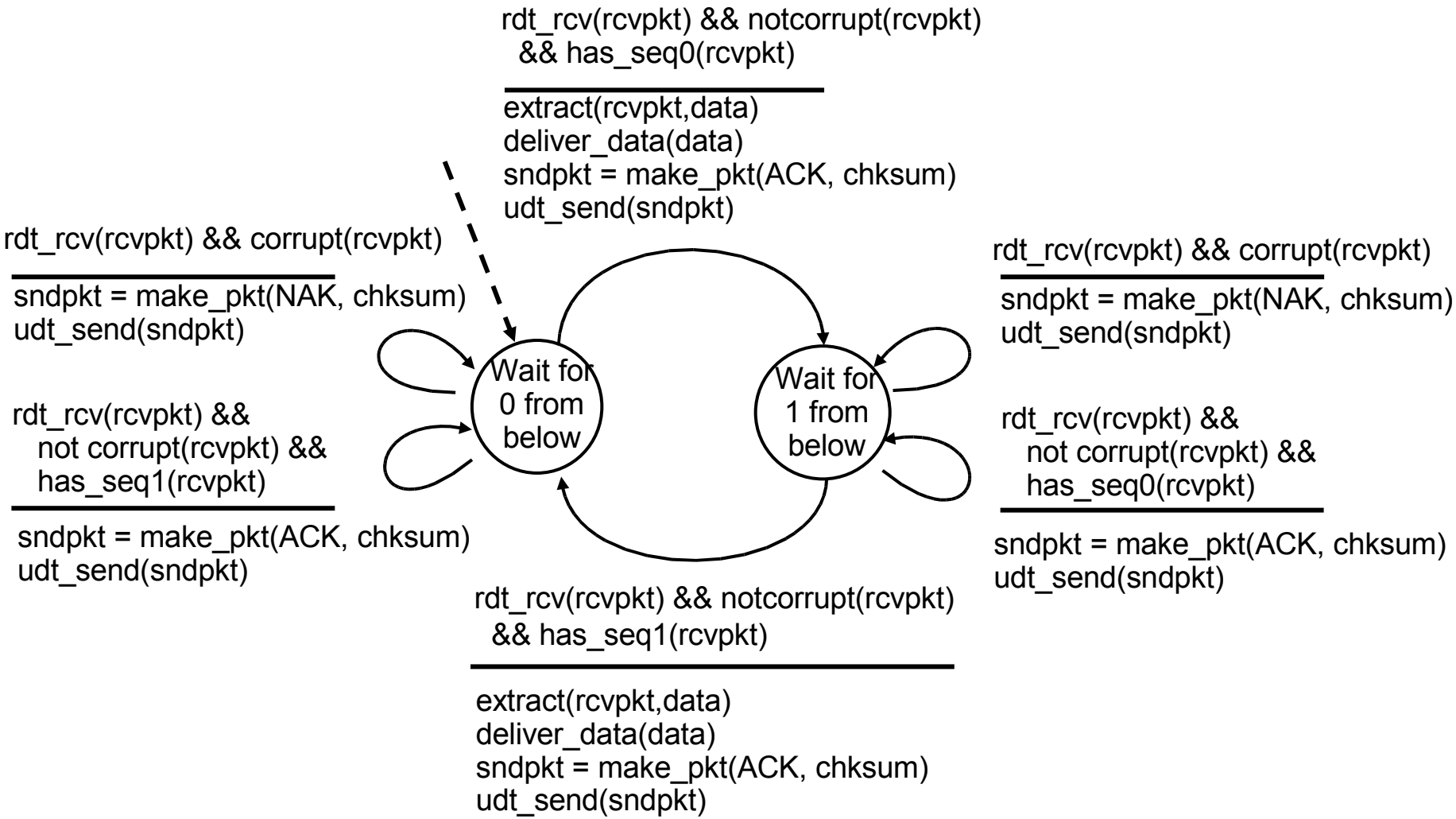
stop and wait

Sender sends one packet, then waits for receiver response

rdt2.1: sender, handles garbled ACK/NAKs



rdt2.1: receiver, handles garbled ACK/NAKs



rdt2.1: discussion

Sender:

- Seq # added to pkt
- Two seq. #'s (0,1) will suffice. Why?
- Must check if received ACK/NAK corrupted
- Twice as many states
  - state must "remember" whether "current" pkt has 0 or 1 seq. #

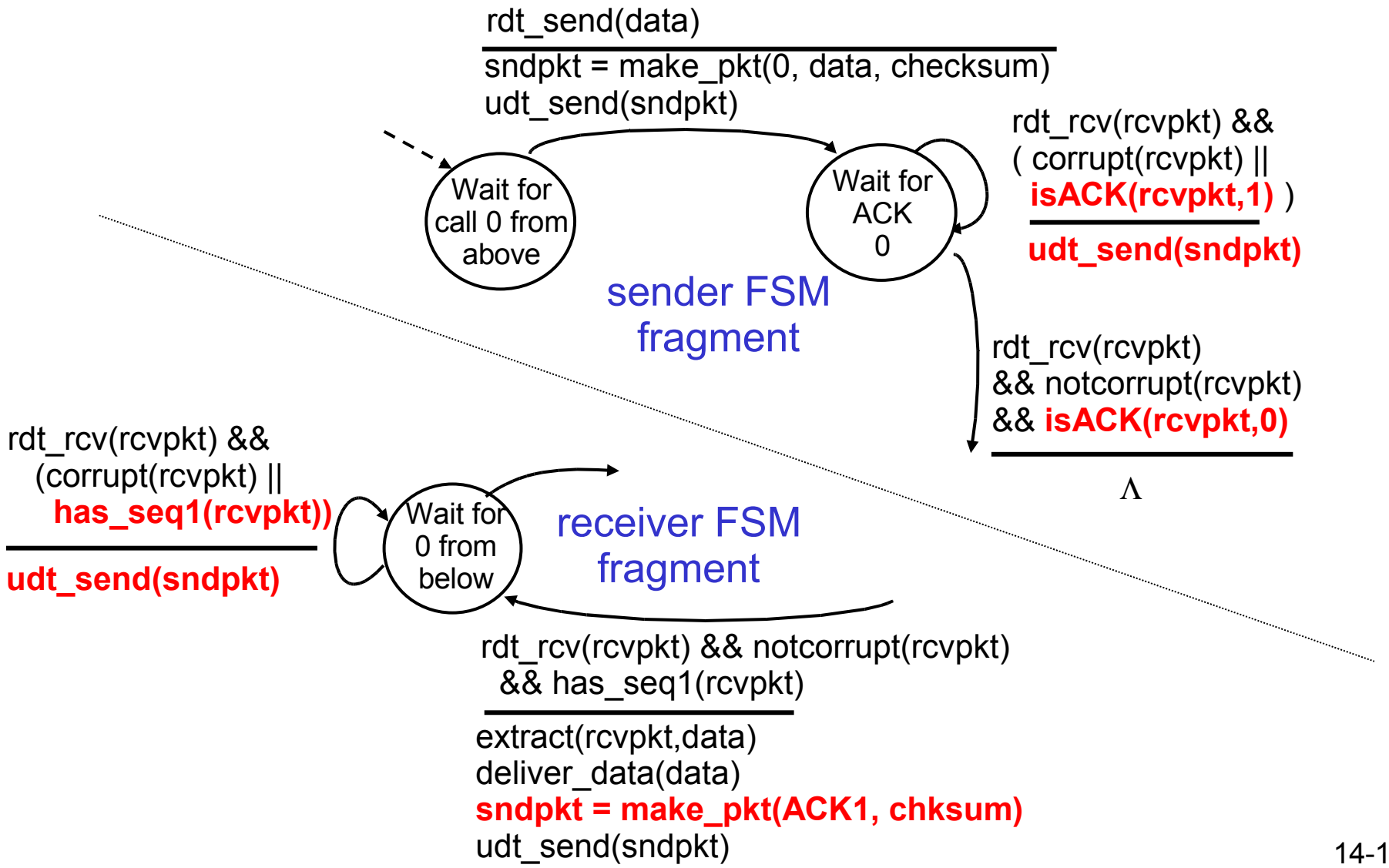
Receiver:

- Must check if received packet is duplicate
  - state indicates whether 0 or 1 is expected pkt seq #
- Note: receiver can *not* know if its last ACK/NAK received OK at sender

rdt2.2: a NAK-free protocol

- Same functionality as rdt2.1, using ACKs only
- Instead of NAK, receiver sends ACK for last pkt received OK
  - receiver must *explicitly* include seq # of pkt being ACKed
- Duplicate ACK at sender results in same action as NAK: *retransmit current pkt*

rdt2.2: sender, receiver fragments



Recap

- Reliable Data Transfer
  - Provide rdt over unreliable network layer
  - FSM model
  - rdt 1.0: rdt over reliable channels
  - rdt 2.0: rdt over channels with bit errors
  - rdt 2.1: handle garbled ACKs/NAKs
  - rdt 2.2: remove need for NAKs
- Midterm review

Next time

- Reliable Data Transfer with packet loss
- Pipelining