Lecture 10: Transpor Layer
Principles of Reliable Data Transfer

COMP 332, Fall 2018

Victoria Manfredi

WESLEYAN
UNIVERSITY

Acknowledgements: materials adapted from Computer Networking: A Top Down Approach 7th edition: ©1996-2016, J.F Kurose and K.W. Ross, All Rights Reserved as well as from slides by Abraham Matta at Boston University, and some material from Computer Networks by Tannenbaum and Wetherall.

Today

1. Announcements
   - homework 4 due today at11:59p
   - homework 5 posted

2. Reliable data transport
   - principles
   - reliable channel
   - NAKs and ACKs
   - coping with garbled ACKs and NAKs
   - NAK-free protocol
   - channels with errors and loss

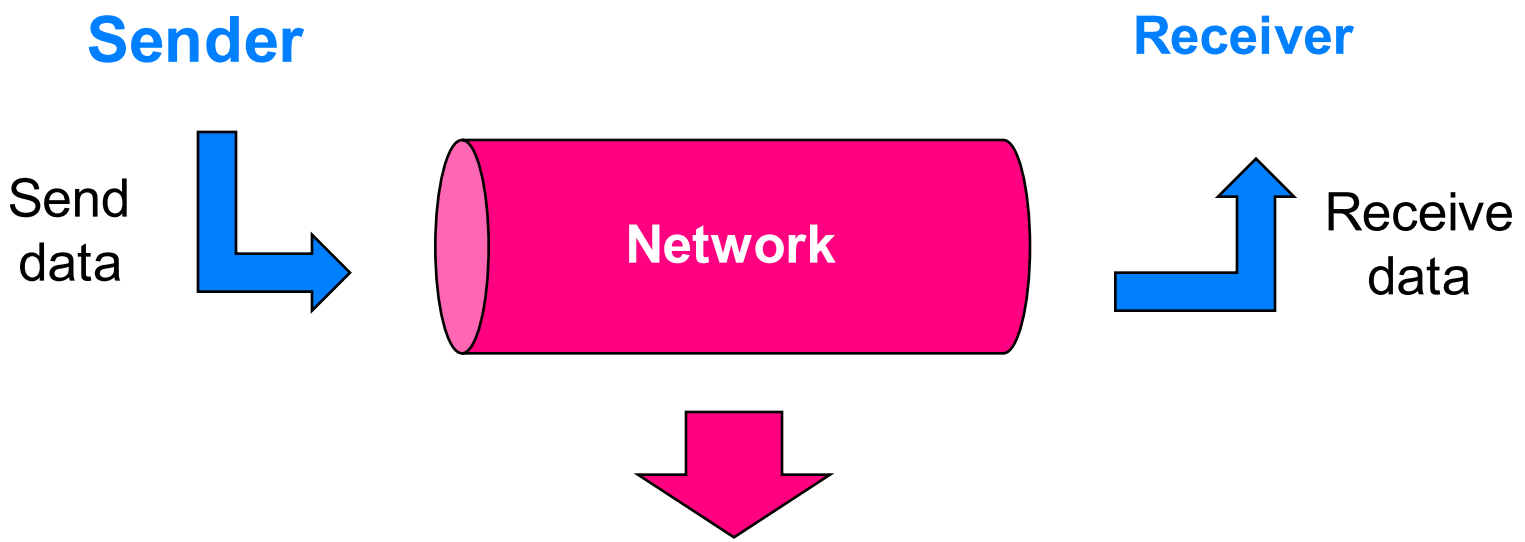
Why are we looking at?

To help you understand why TCP operates the way it does (we'll cover next week)

Reliable Data Transport

PRINCIPLES

Why can't we do the following?



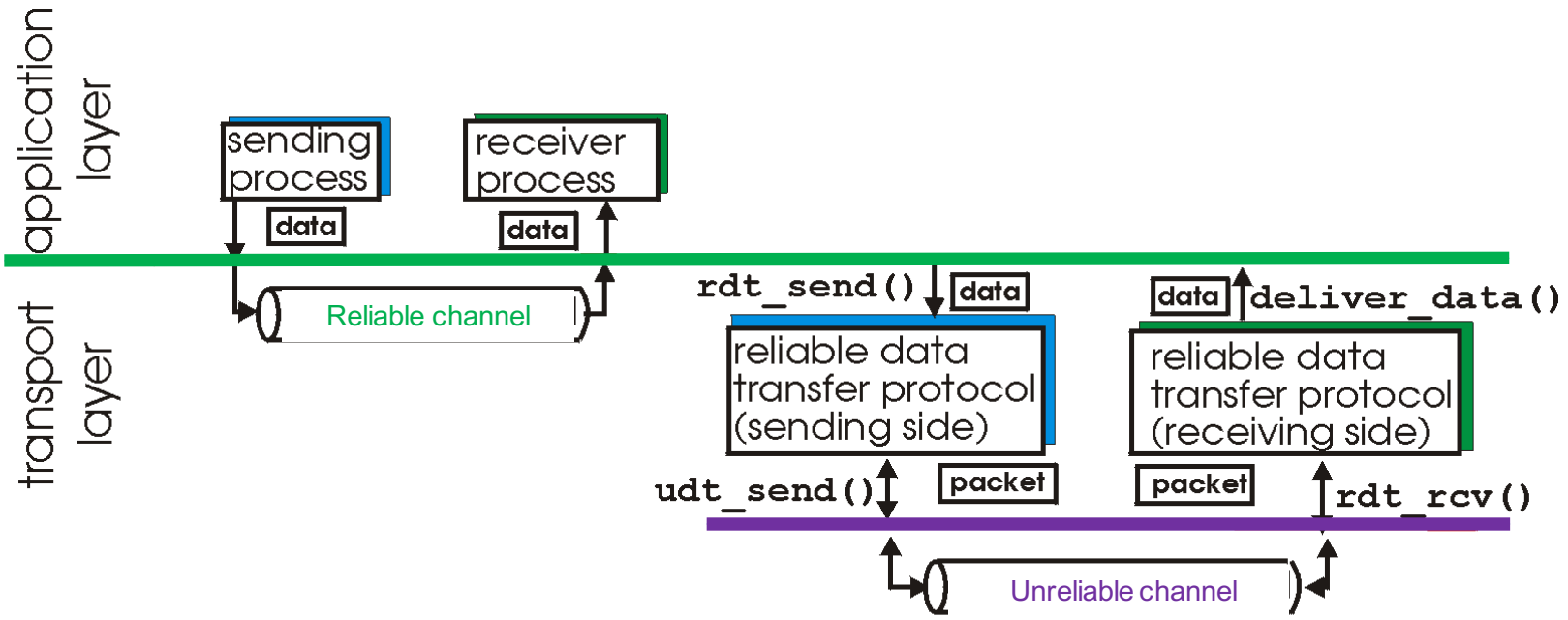
Because Internet is unreliable channel

Packets can be corrupted, duplicated, reordered, delayed, lost

Q: What can we do?

Principles of reliable data transfer

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

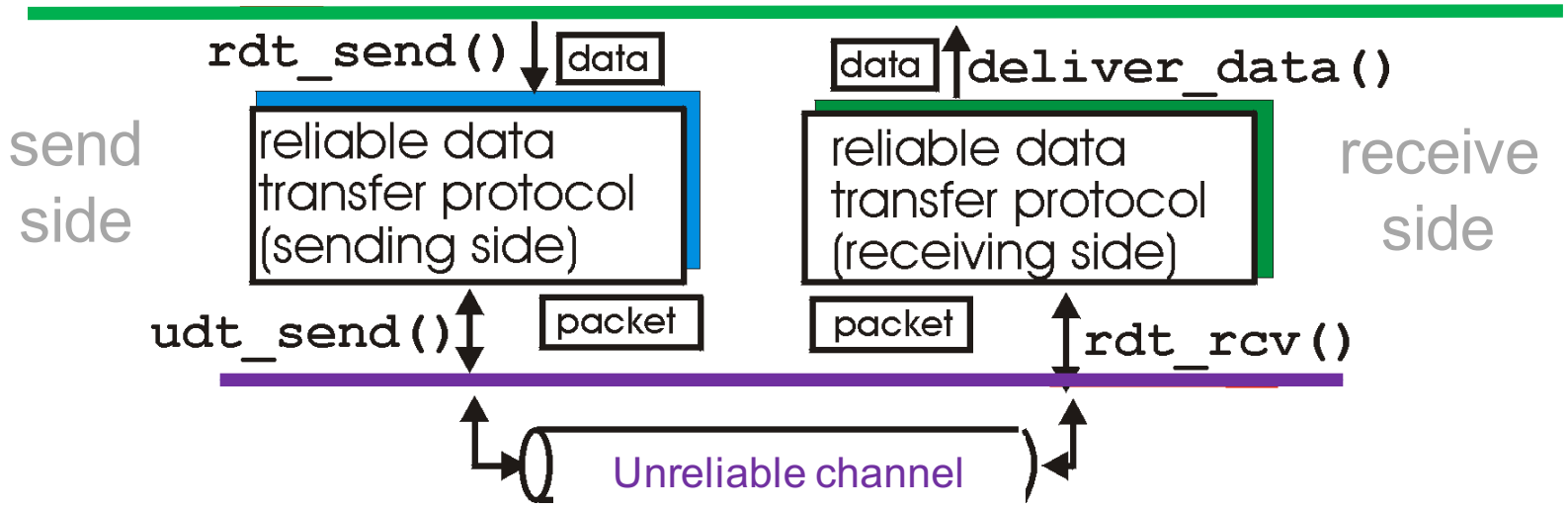


Provided service **Service implementation**

Abstraction

`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

Our plan

Incrementally develop

- sender, receiver sides of **r**eliable **d**ata **t**ransfer protocol (rdt)

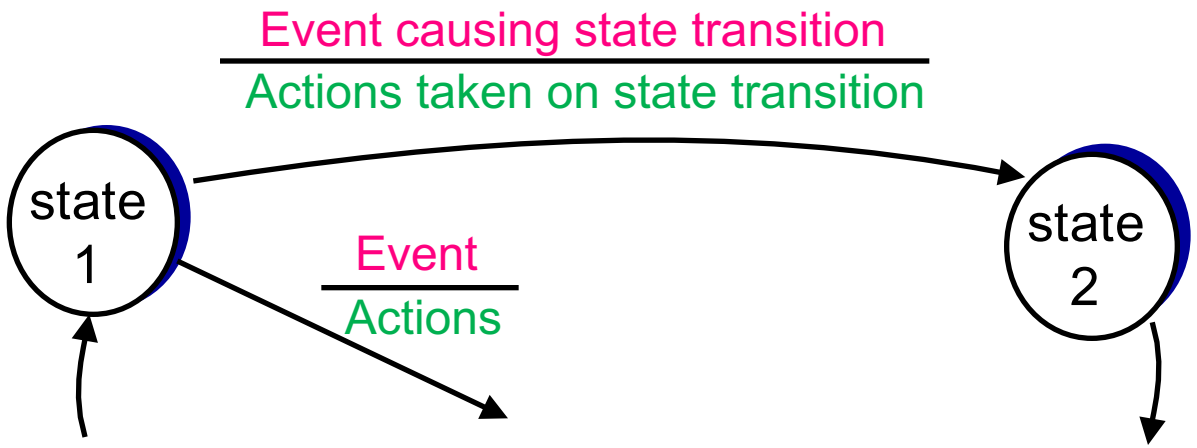
Consider only unidirectional data transfer

- but control info will flow in both directions!

Use finite state machines (FSM)

- to specify sender, receiver

State: when in this state, next state is uniquely determined by next event



Reliable Data Transport

RELIABLE CHANNEL

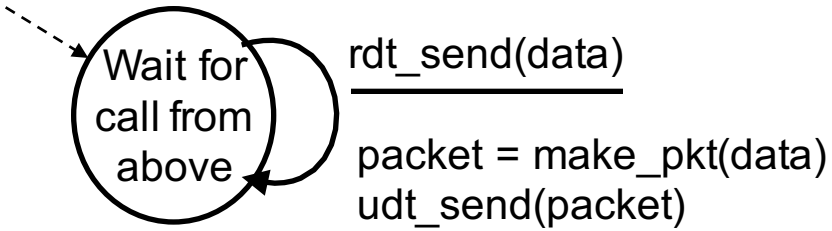
rdt1.0: reliable transfer over a reliable channel

Underlying channel perfectly reliable

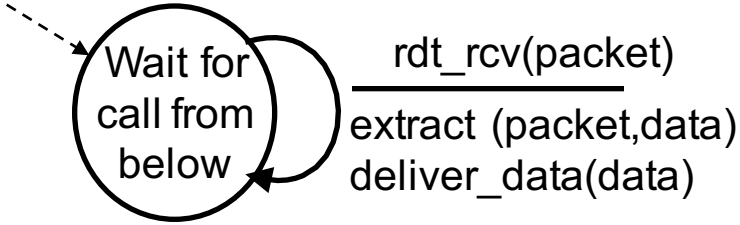
- no bit errors
- no loss of packets

Separate FSMs for sender, receiver:

- sender sends data into underlying channel
- receiver reads data from underlying channel



sender



receiver

Unreliable data transfer protocol would look the same

Reliable Data Transport

ACKS AND NAKS

rdt2.0: channel with bit errors

Problem: underlying channel may flip bits in packet

- how to detect and recover from errors?
- how do humans detect errors in conversation?

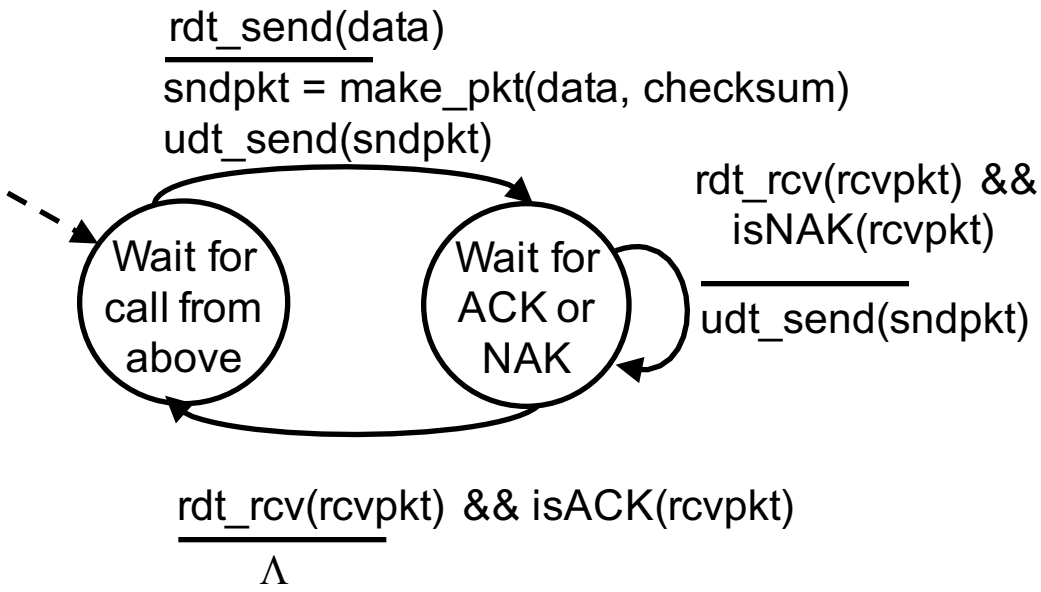
Solution

- Checksum
  - to detect bit 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)

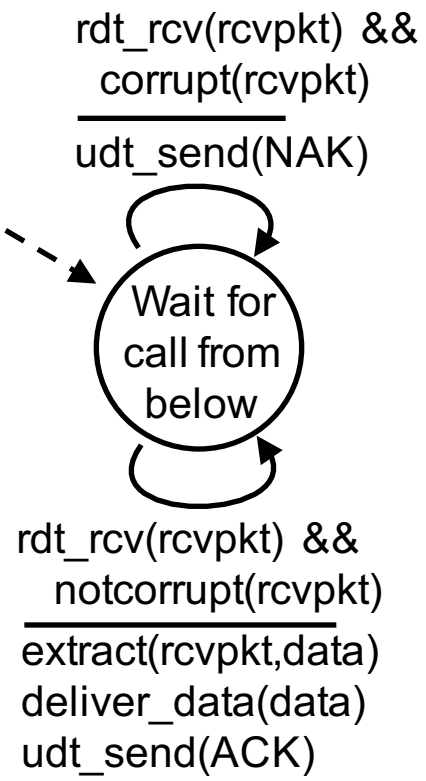
- error detection
- feedback: control msgs (ACK,NAK) from receiver to sender

rdt2.0: FSM specification



Sender

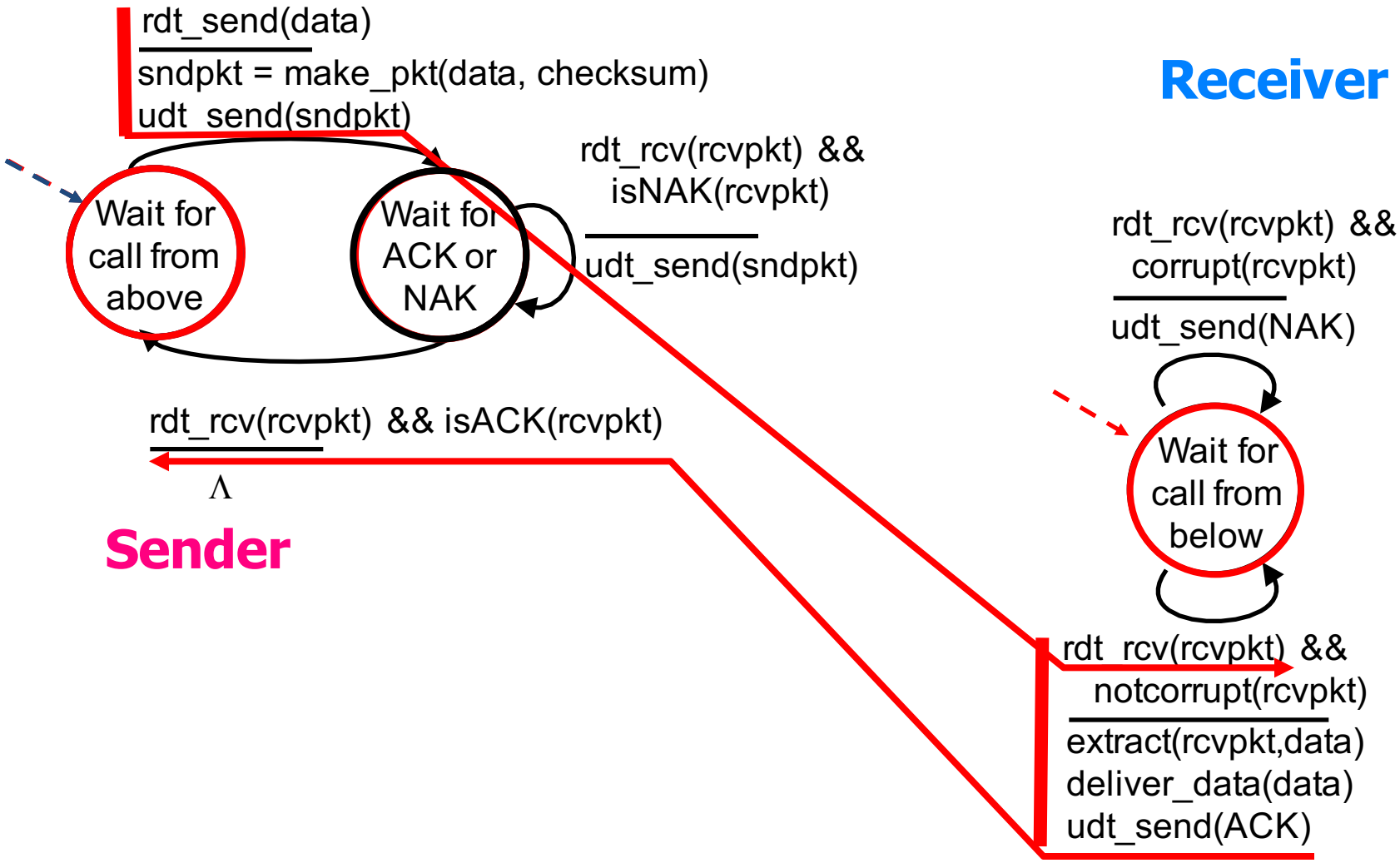
Receiver



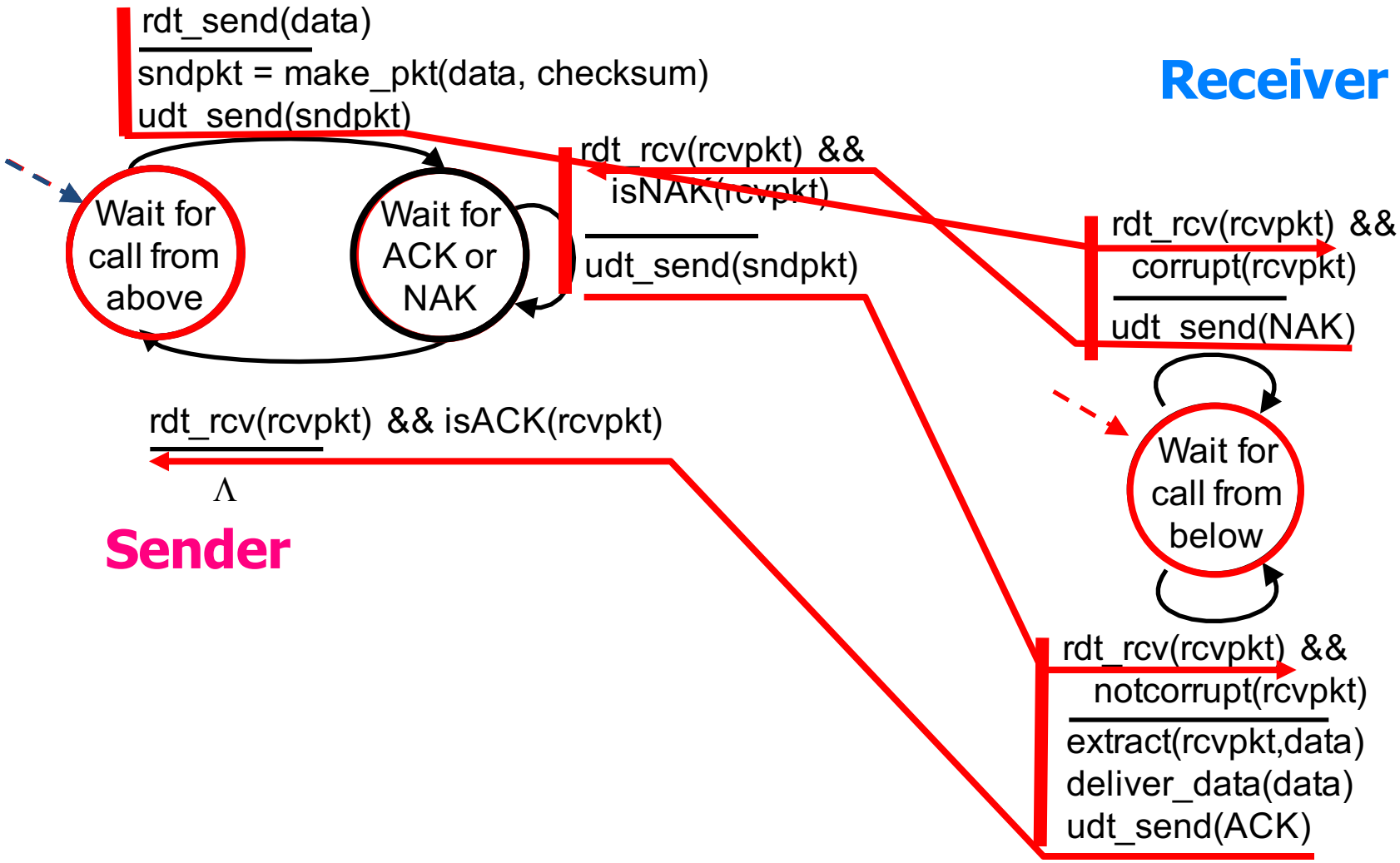
Stop and wait

Sender sends one packet, then waits for receiver response

rdt2.0: operation with no errors



rdt2.0: error scenario



rdt2.0 has a fatal flaw!

What if ACK/NAK corrupted?

NAK corrupted to ACK

- sender may not retransmit when actually needed
- **new packet seen as duplicate**

ACK corrupted to NAK

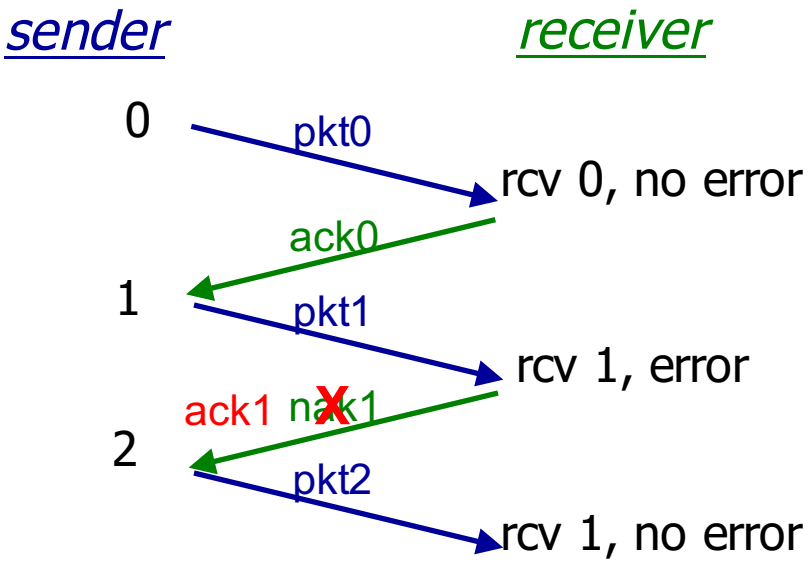
- sender may retransmit packet when not needed
- **duplicate seen as new packet**

ACK/NAK just corrupted

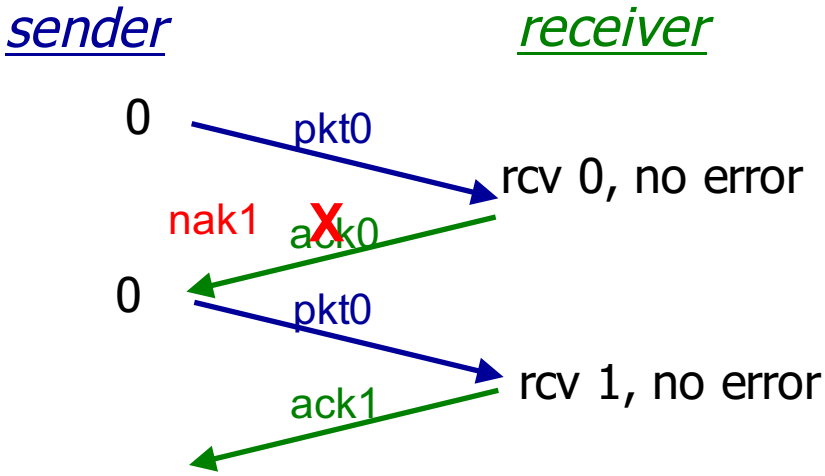
- what do you do?
- just assume NAK and resend?

(no seq #s on pkt/ack just yet, using only for clarity)

NAK corrupted to ACK



ACK corrupted to NAK



Reliable Data Transport

GARBLED ACKS AND NAKS

rdt2.1: channel with bit errors, garbled N/ACKs

Problem: underlying channel may flip bits in packet

- packets, ACKs, NAKs may be garbled

Solution

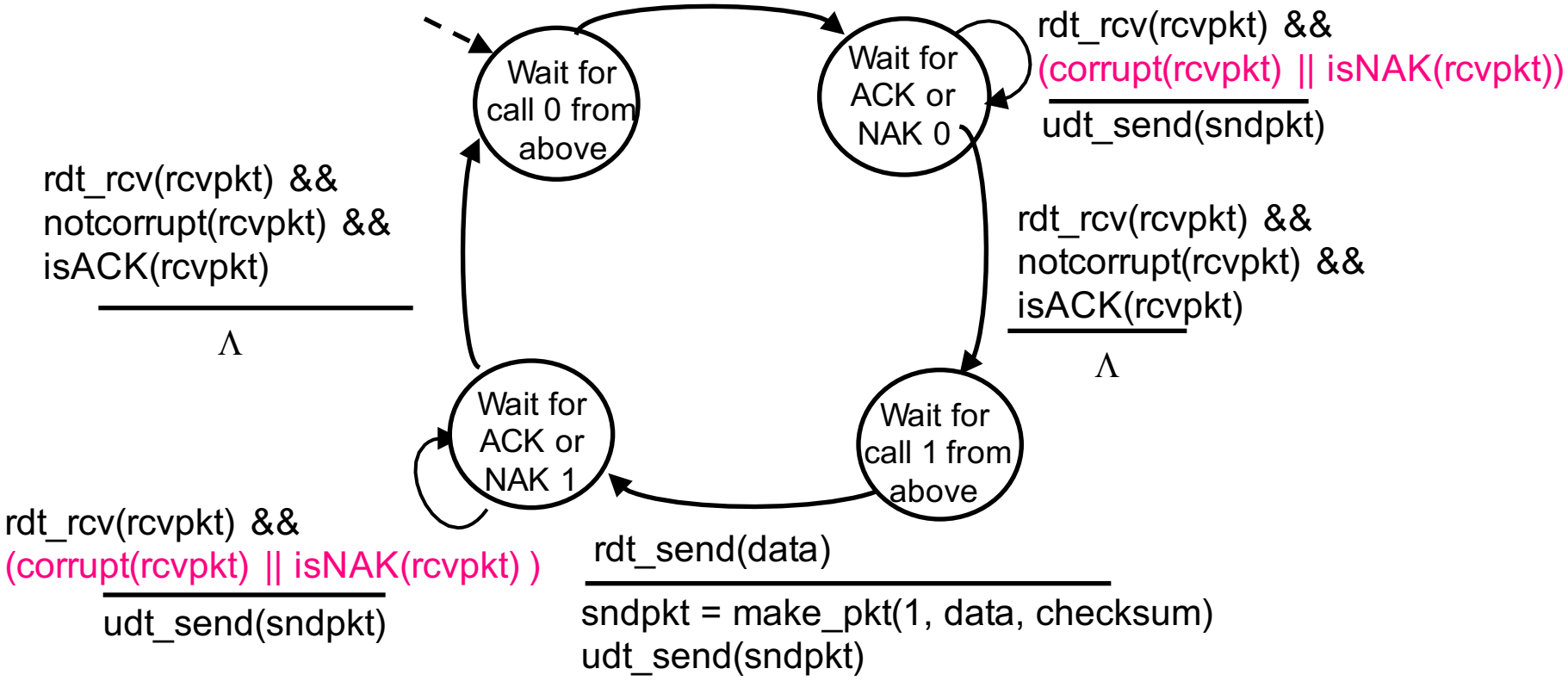
- Checksum for both packets and ACKs/NAKs
  - to detect bit errors
- Acknowledgements (ACKs)
  - receiver explicitly tells sender that pkt received OK
- Negative acknowledgements (NAKs)
  - receiver explicitly tells sender that pkt had errors
  - sender retransmits pkt on receipt of NAK
- Sender retransmits current pkt if ACK/NAK corrupted
  - adds sequence # to each pkt
  - receiver discards duplicate pkt

rdt2.1: sender, handles garbled ACK/NAKs

Only 2 seq #s: 0, 1

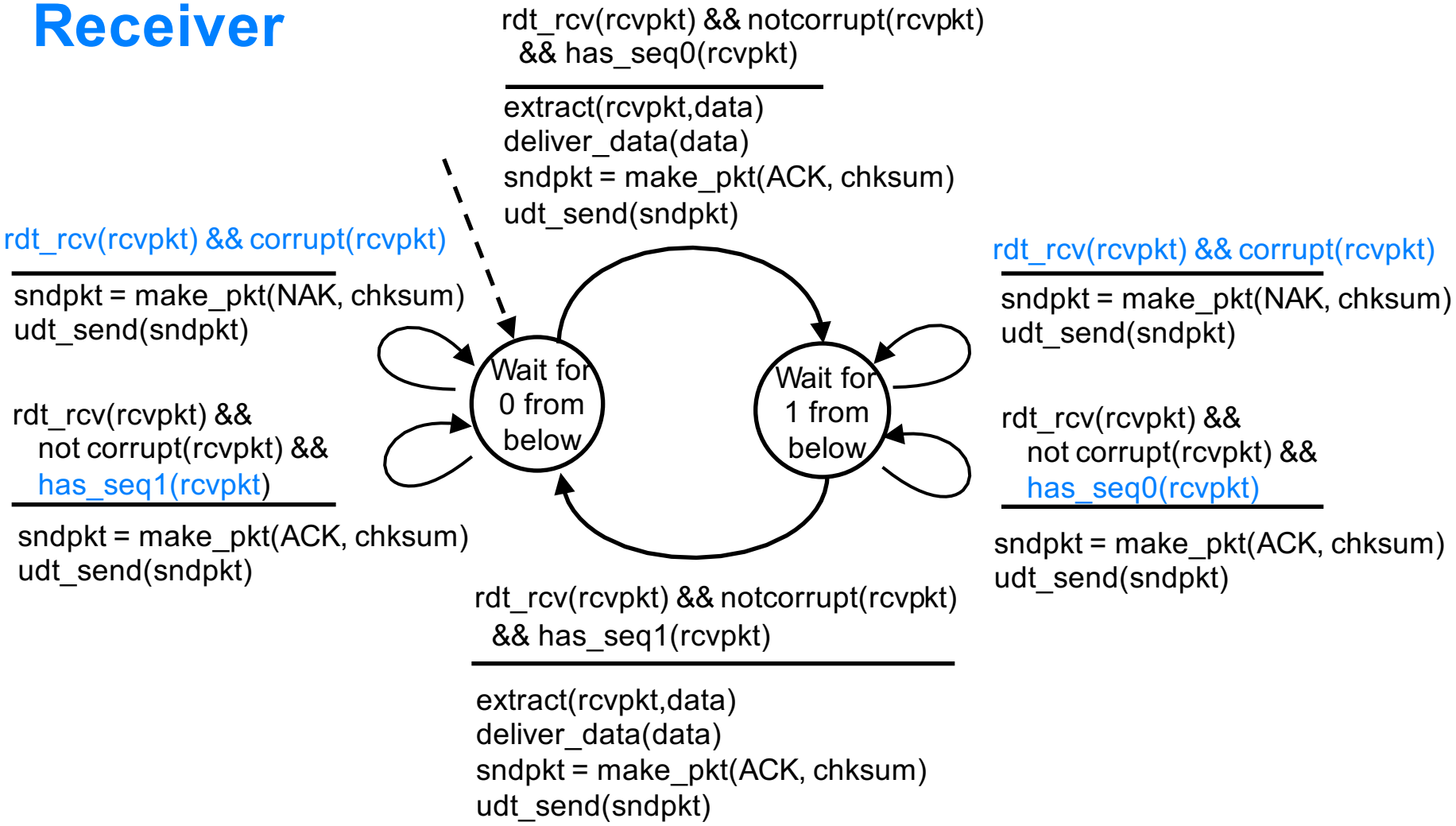
No channel reordering of pkts

Sender



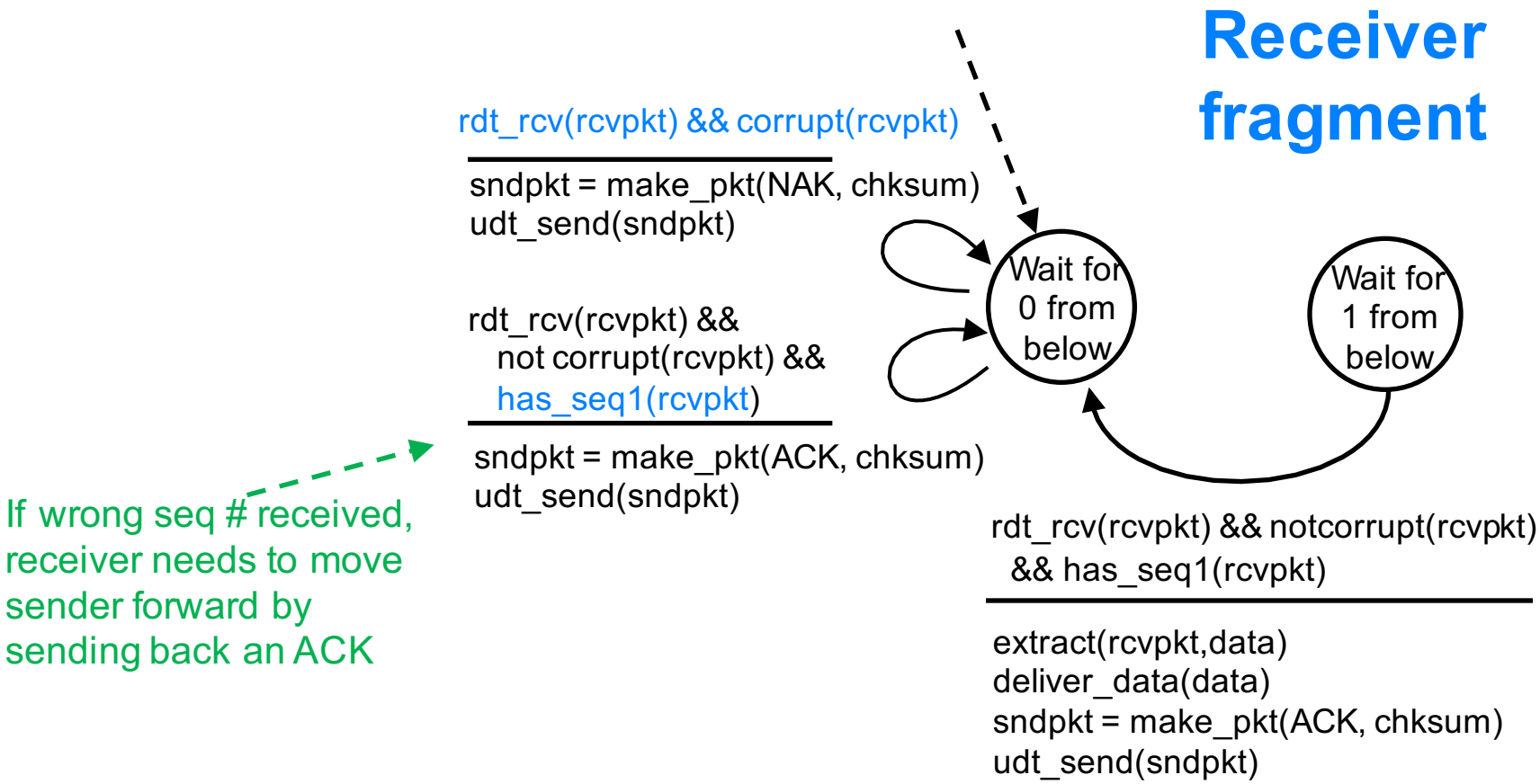
rdt2.1: receiver, handles garbled ACK/NAKs

Receiver



rdt2.1: discussion

ACKs and NAKs don't need sequence numbers!



rdt2.1: discussion

Seq # added to pkt

- 2 seq #'s (0,1) suffice
- Q: Why?
  - Stop-and-wait protocol

Twice as many states

- state remembers whether expected pkt should have seq # 0 or 1

Sender checks

- if received ACK/NAK corrupted

Receiver checks

- if received packet is corrupted or duplicate
- state indicates whether 0 or 1 is expected pkt seq #

Reliable Data Transport

A NAK-FREE PROTOCOL

rdt2.2: a NAK-free protocol

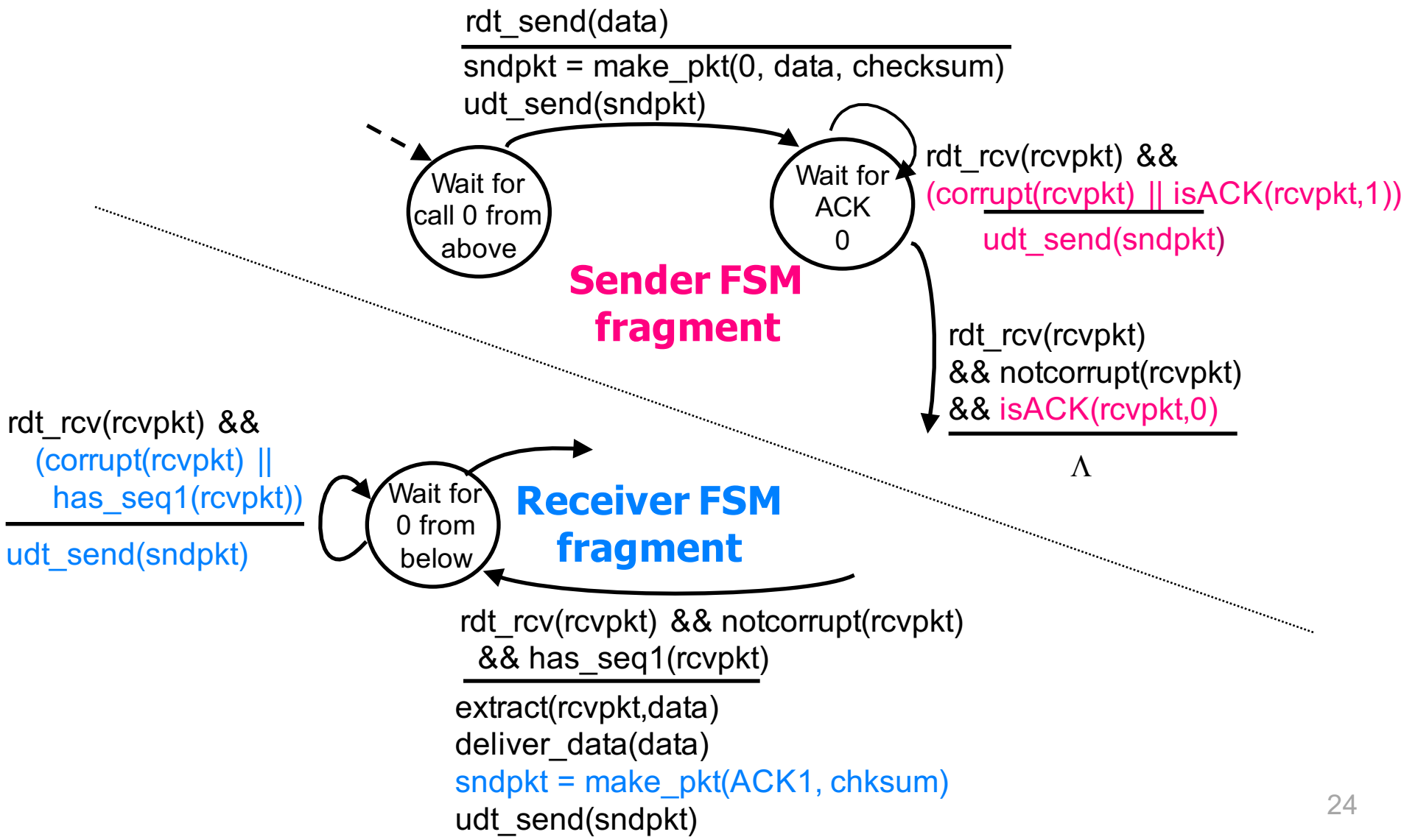
Same functionality as rdt2.1, using ACKs only

- 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



Reliable Data Transport

CHANNELS WITH ERROR AND LOSS

rdt3.0: channels with errors and loss

Problems

- underlying channel may flip bits in packet
  - both data and ACKs may be garbled
- underlying channel can also lose packets
  - both data and ACKs
- checksum, seq. #, ACKs, retransmissions will be of help
  - … but not enough

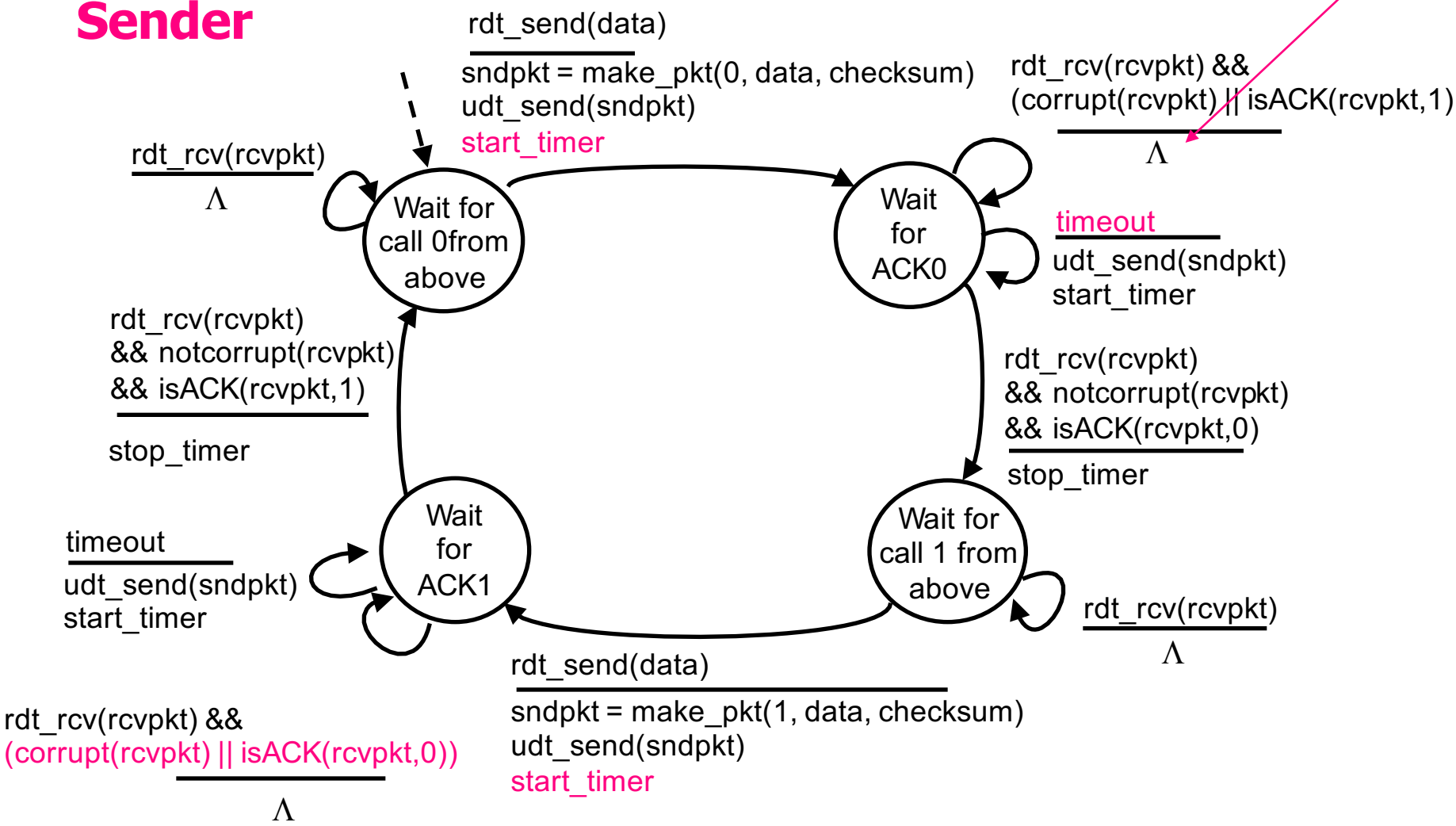
Solution: add countdown timer

- 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 seq #'s already handles this
- receiver must specify seq # of pkt being ACKed

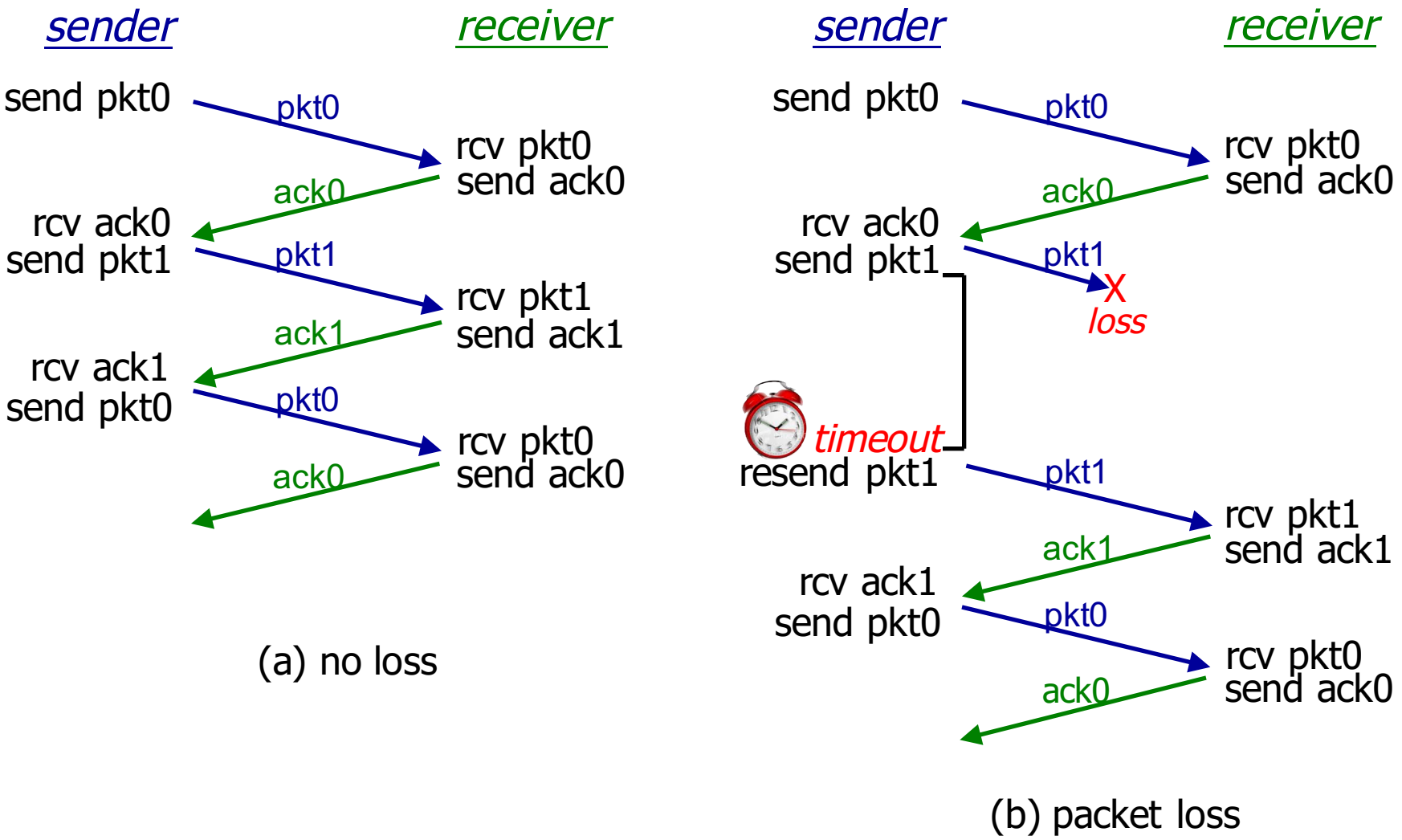
rdt3.0 sender

Why do nothing ? Why not resend pkt0? Because sender doesn't know whether ack1 means pkt 0 garbled or pkt 1 duplicate received By not resending pkt 0, sender doesn't introduce potentially unnecessary (even if valid) traffic: saves bandwidth

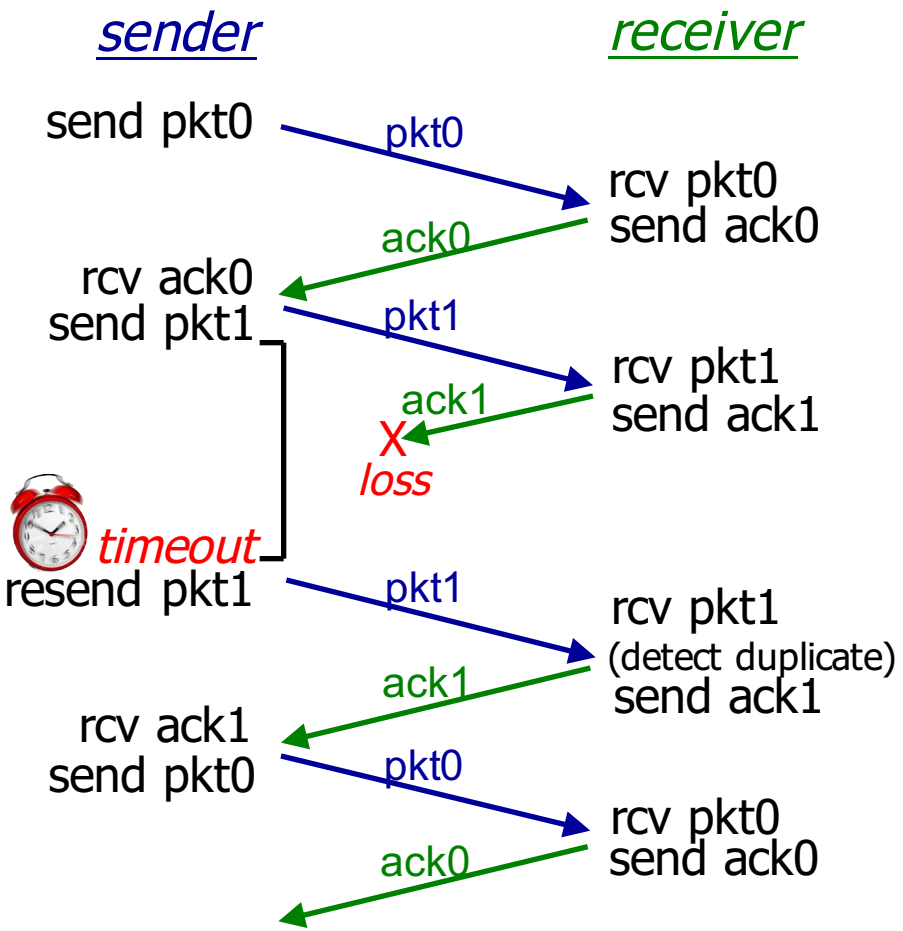
Sender



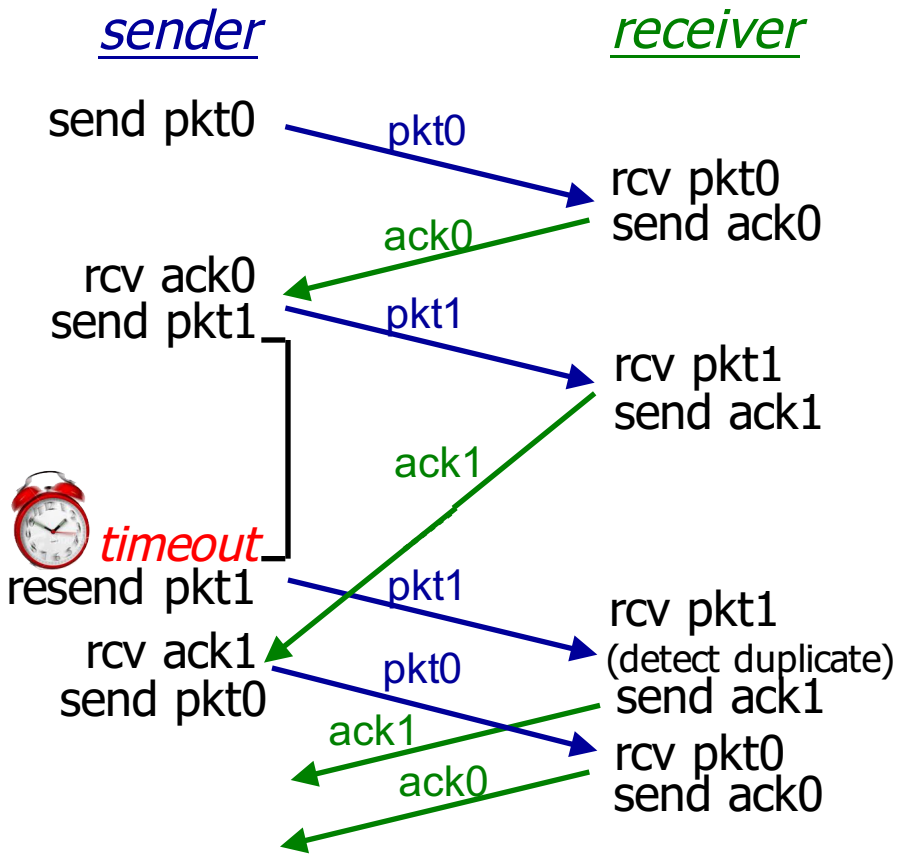
rdt3.0 in action



rdt3.0 in action

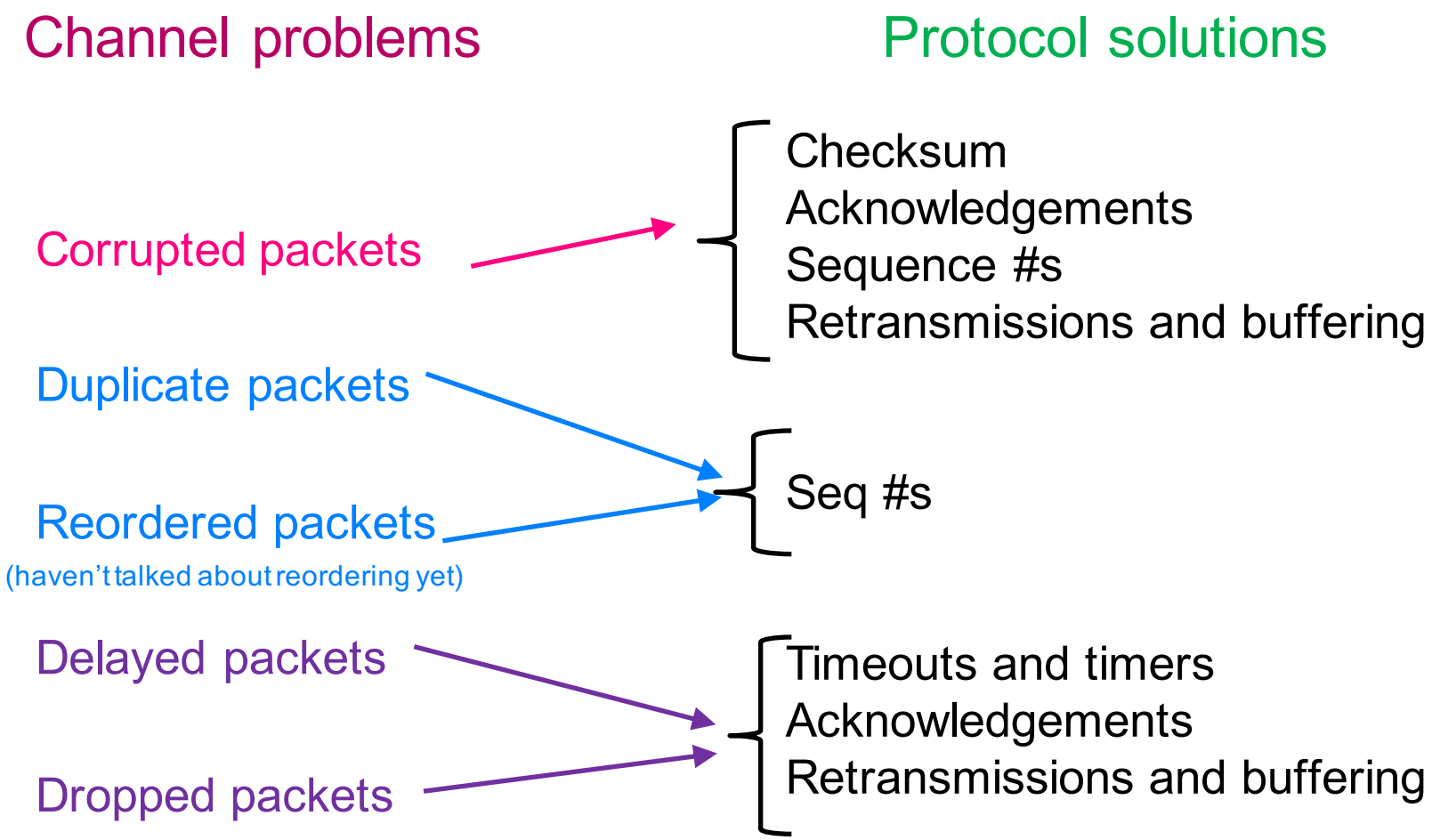


(c) ACK loss



(d) premature timeout/ delayed ACK

Summary of techniques and uses



# of seq #s must be > 2x window size if reordering