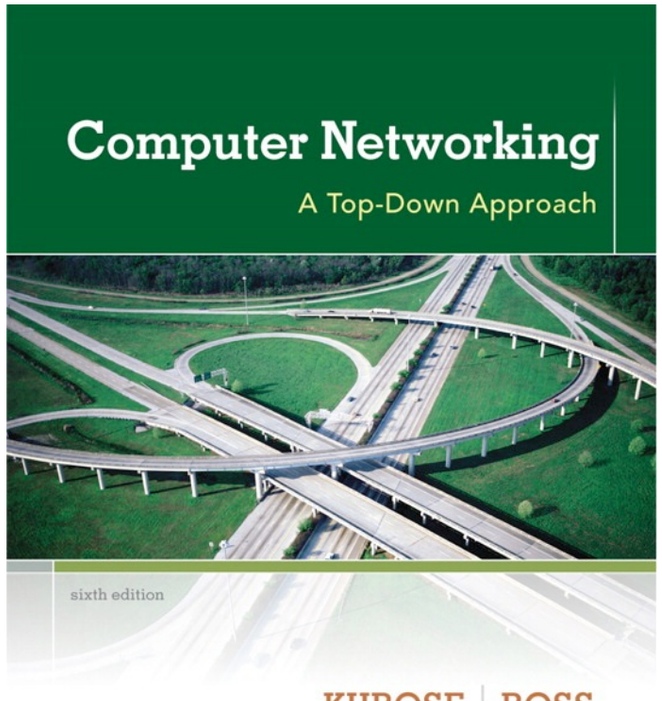
CN-Basic
L21

Reliable Data Transfer
Higher Version

Dr. Ram P Rustagi
rprustagi@ksit.edu.in
http://www.rprustagi.com
https://www.youtube.com/rprustagi

Chapter 3
Transport Layer



A note on the use of these ppt slides:

We're making these slides freely available to all (faculty, students, readers). They're in PowerPoint form so you see the animations; and can add, modify, and delete slides (including this one) and slide content to suit your needs. They obviously represent a *lot* of work on our part. In return for use, we only ask the following:

- If you use these slides (e.g., in a class) that you mention their source (after all, we'd like people to use our book!)
- If you post any slides on a www site, that you note that they are adapted from (or perhaps identical to) our slides, and note our copyright of this material.

Thanks and enjoy! JFK/KWR

All material copyright 1996-2012
J.F Kurose and K.W. Ross, All Rights Reserved

©

Computer Networking: A Top Down Approach
6th edition
Jim Kurose, Keith Ross
Addison-Wesley
March 2012

rdt2.0 has a Fatal Flaw!

- What happens if ACK/ NAK corrupted?
- Sender doesn't know what happened at receiver!
- Two possibilities to handle
  - Add more checksum bits to recover from error
  - Can we just retransmit: possible duplicate pkts?
- Handling duplicates:
- Sender retransmits current pkt if ACK/NAK corrupted
- Sender adds *sequence number* to each pkt
  - Is `1` bit seq num ok?
  - Receiver discards (doesn't deliver up) duplicate pkt
  - Does ACK/NAK require seq number?

Stop and Wait

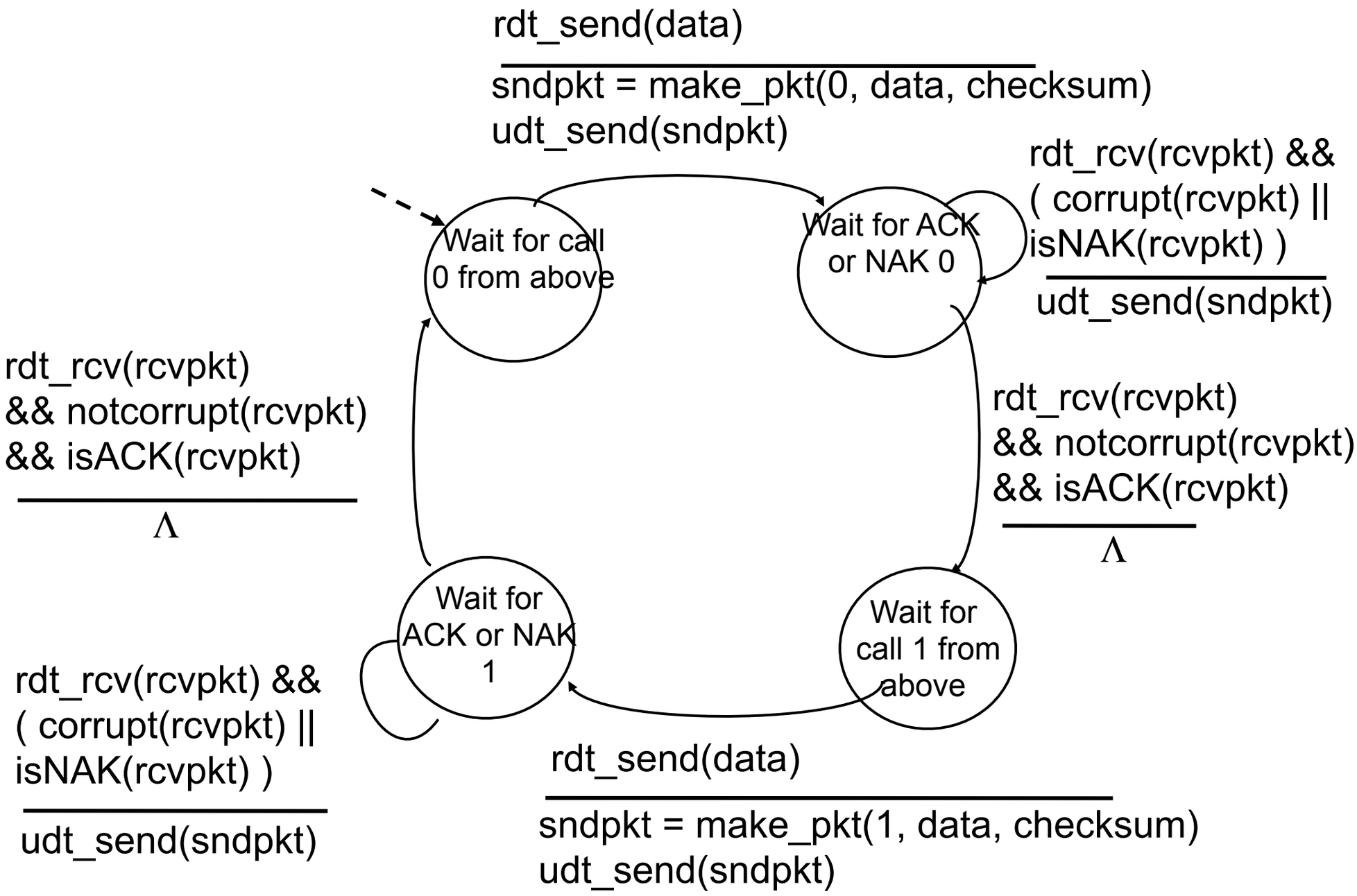
Sender sends one packet,
then waits for receiver response

rdt2.1: discussion

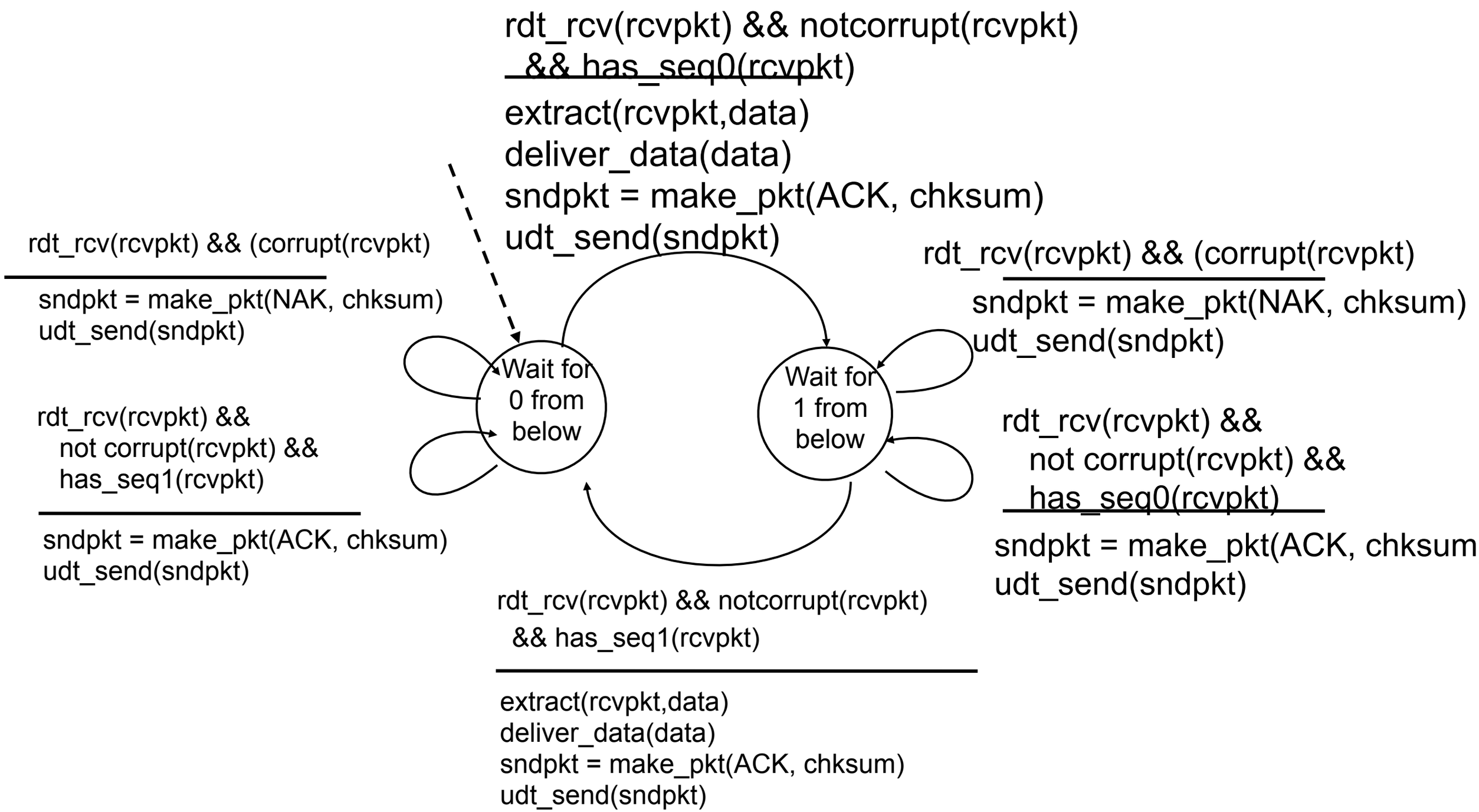
- sender:
- Seq # added to pkt
- Two seq. #'s (`0`, `1`) will suffice. Why?
- Must check if received ACK/NAK corrupted
- Number of states?
  - State must "remember" whether "expected" pkt should have seq # either of `0` or `1`
  - Should it remain 2 or increase to 4?

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

rdt2.1: sender, handles garbled ACK/NAKs



rdt2.1: receiver, handles garbled ACK/NAKs



rdt2.1: discussion

- Receiver
  - No of states doubled (two) from one(rdt 2.0)
- Sender
  - No of states doubled (four) from two (rdt 2.0)
- Additional two states are mirror images of first two
    - It differs in use of seq number `1` instead of `0`

rdt2.1: discussion

- Can we have a NAK free protocol?
  - One less message type to deal with
  - Instead of NAK on receipt of bad packet
    - Send ACK for last received correct packet
    - This ACK may become duplicate
    - Do we need to differentiate between Ack `0` & Ack `1`?
      - ACK must contain sequence number
- Duplicate ACKs
  - Implies to sender
    - Receiver didn't receive newer packets correctly

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

rdt_send(data)
sndpkt = make_pkt(0, data, checksum)
udt_send(sndpkt)

Wait for call 0 from above

Wait for ACK 0

rdt_rcv(rcvpkt) &&
(corrupt(rcvpkt) ||
isACK(rcvpkt,1))
udt_send(sndpkt)

rdt_rcv(rcvpkt)
&& notcorrupt(rcvpkt)
&& **isACK(rcvpkt,0)**
Λ

sender FSM fragment

rdt_rcv(rcvpkt) &&
(corrupt(rcvpkt) ||
has_seq1(rcvpkt))
udt_send(sndpkt)

Wait for 0 from below

receiver FSM fragment

rdt_rcv(rcvpkt) && notcorrupt(rcvpkt)
&& has_seq1(rcvpkt)
extract(rcvpkt,data)
deliver_data(data)
sndpkt = make_pkt(ACK1, chksum)
udt_send(sndpkt)

rdt3.0: channels with errors *and* loss

New assumption:

Underlying channel can also lose packets (data, ACKs)

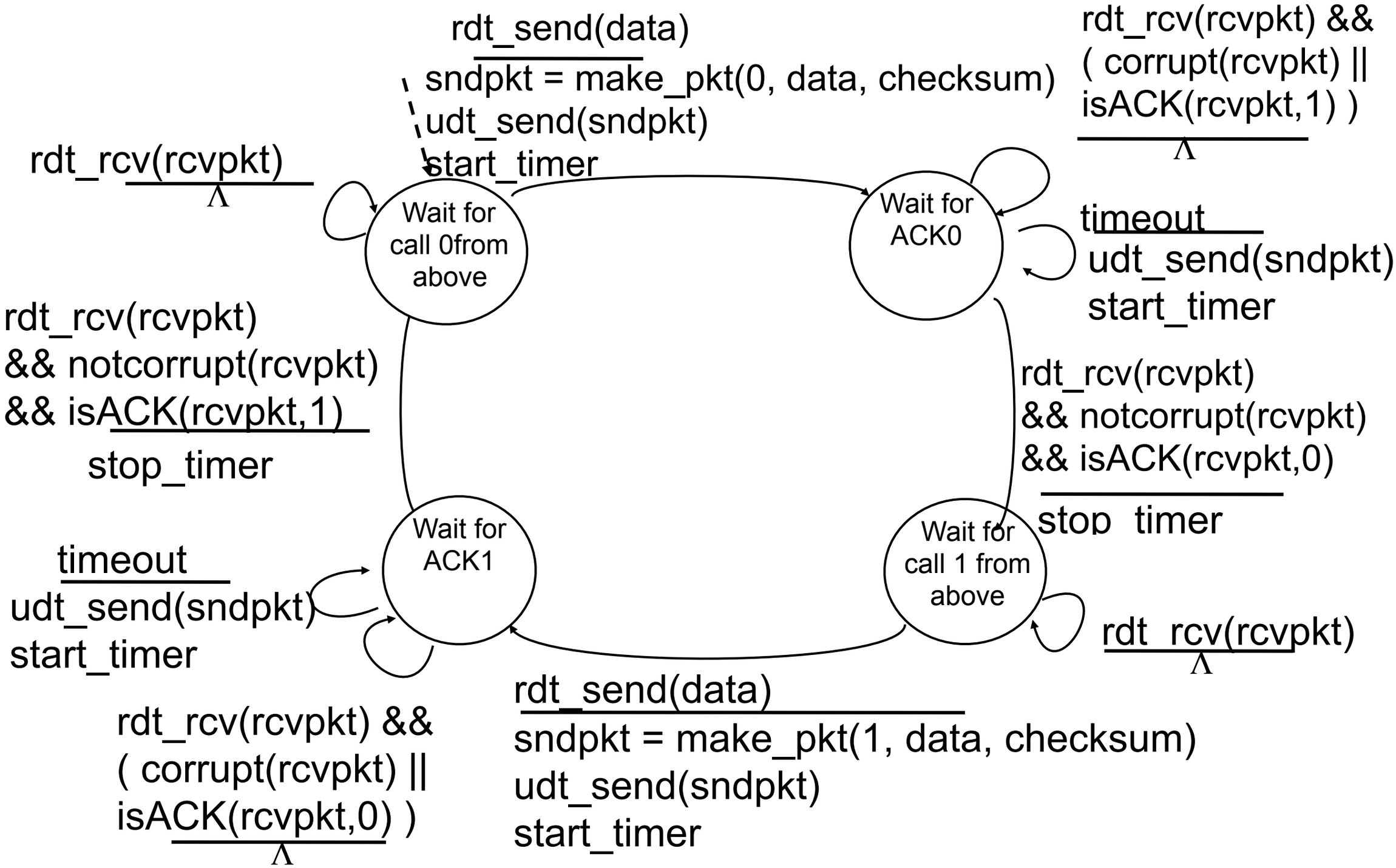
- Checksum, seq. #, ACKs, retransmissions will be of help … but not enough

- 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 seq. #'s already handles this
  - Receiver must specify seq # of pkt being ACKed
  - Requires countdown timer

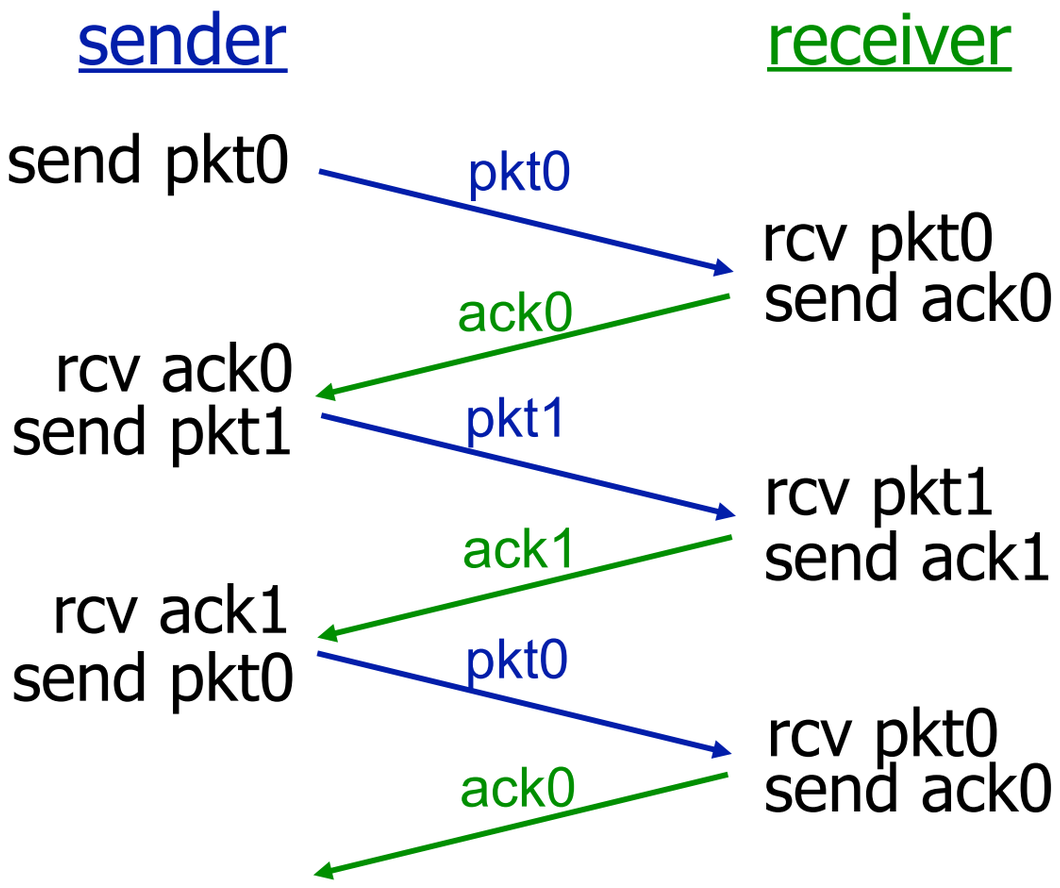
rdt 3.0

- Countdown timer
  - Sender starts timer each time a packet is sent
    - Either first time or retransmits
  - Sender responds to timer interrupt
    - retransmits the packet
- **Alternating bit protocol**
  - Another name for rdt 3.0
  - As pkt sequence number alternates
    - between `0` and `1`

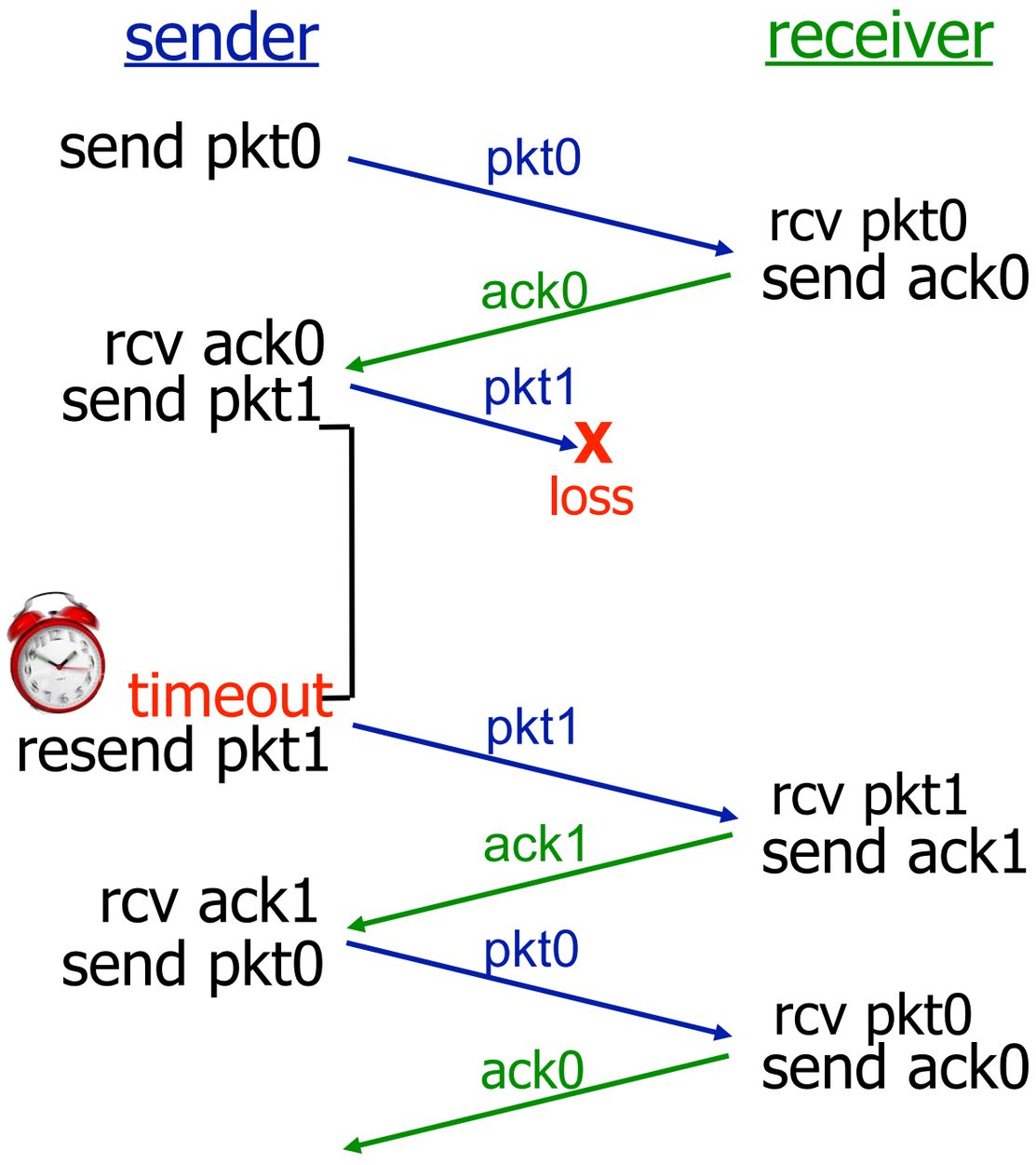
rdt3.0 sender



rdt3.0 in action

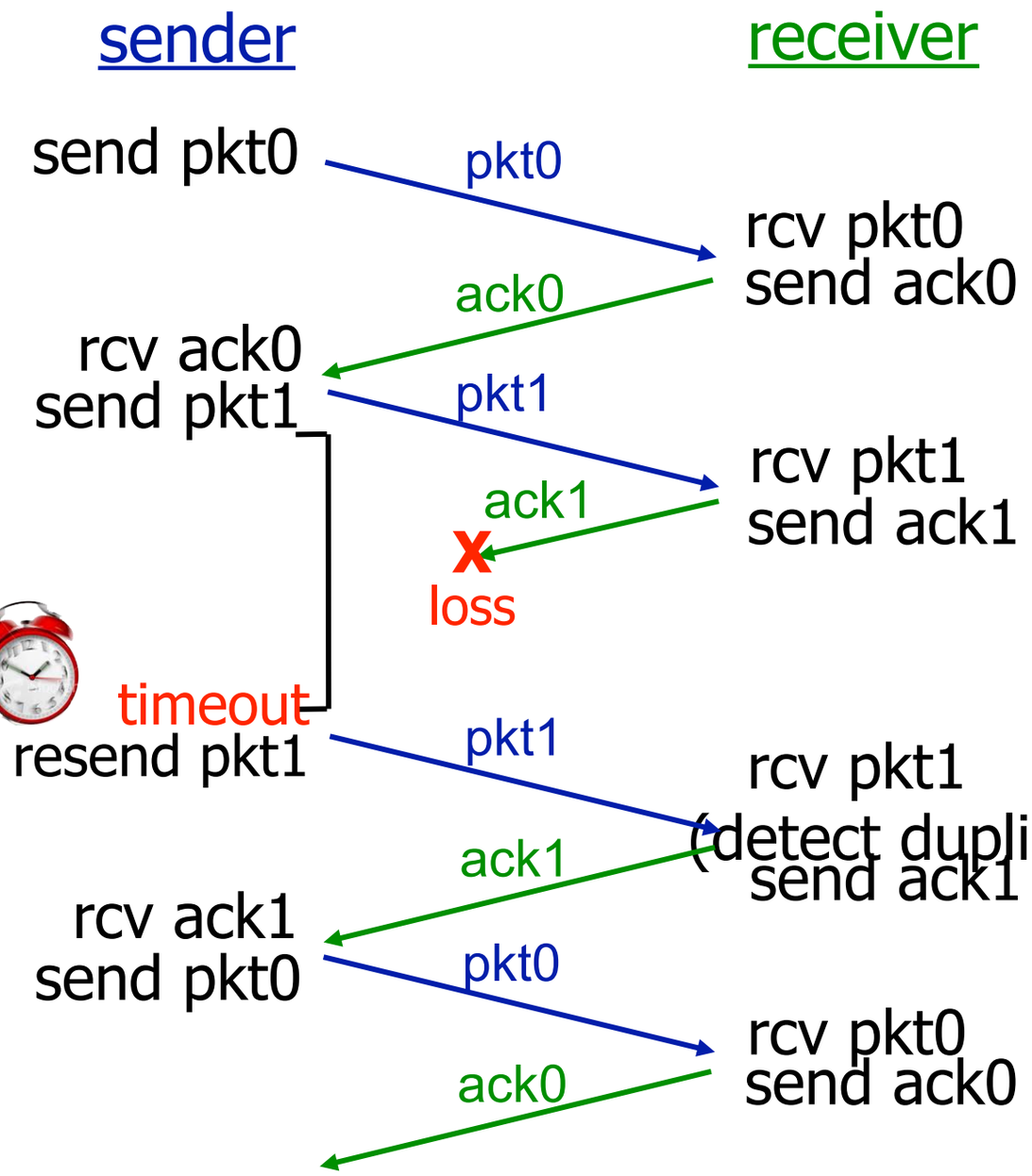


(a) no loss

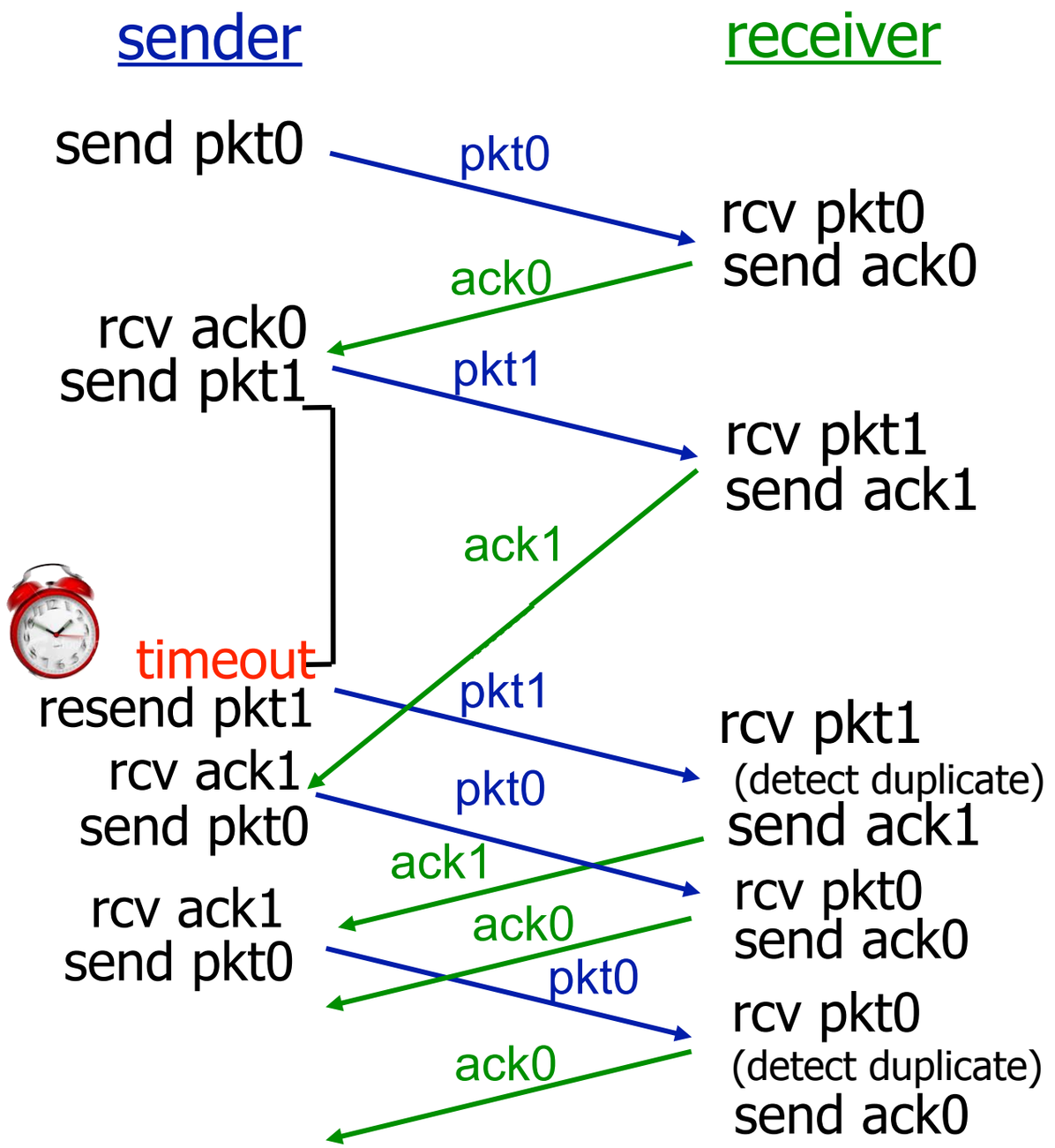


(b) packet loss

rdt3.0 in action



(c) ACK loss



(d) premature timeout/ delayed ACK

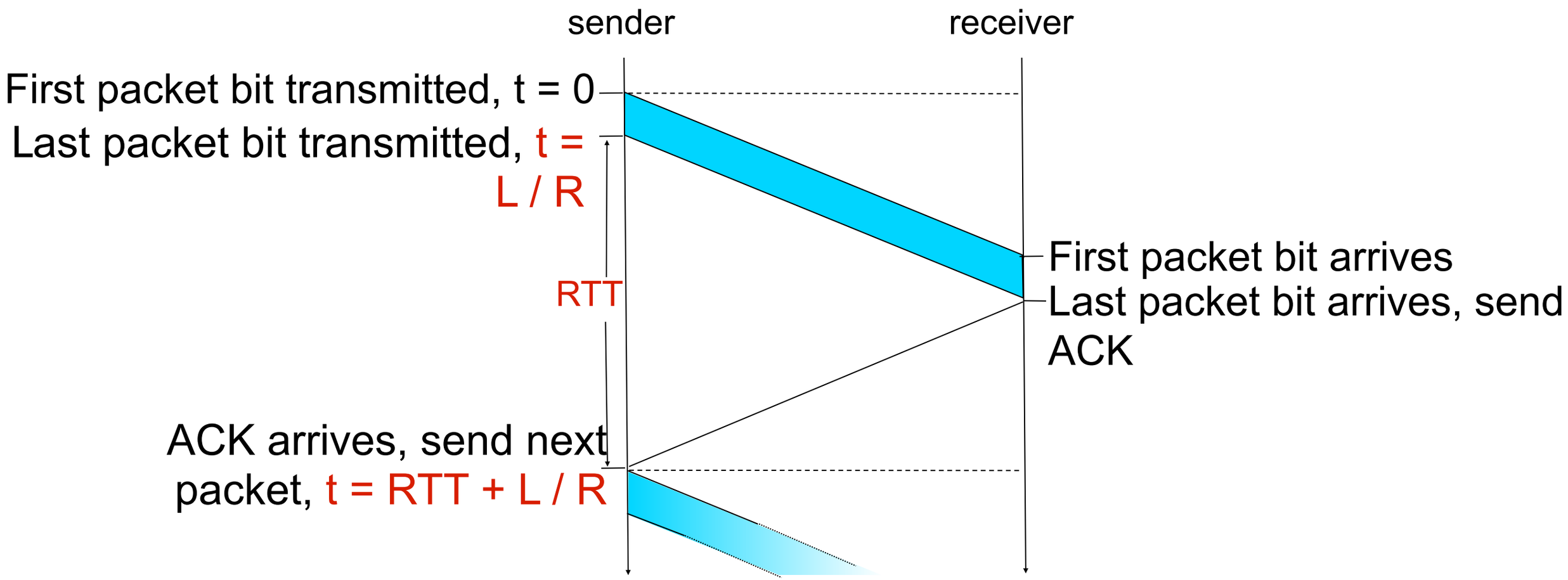
Performance of rdt3.0

- **rdt3.0** is correct, but performance stinks
- e.g.: 1Gbps link, 15 ms prop. delay, 8000 bit packet:

$$D_{trans} = \frac{L}{R} = \frac{8000\ bits}{10^9\ bits/sec} = 8\ microsecs$$

- If RTT=30 msec, 1KB pkt every 30 msec: 33kB/sec thruput over 1 Gbps link
- Network protocol limits use of physical resources!
- Solution: pipelining
-

rdt3.0: stop-and-wait operation



- U_{sender}: *utilization* – fraction of time sender busy sending

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

Exercises : Timeline Diagrams

- Ex01: Work out the Timeline sequence diagram for all possible cases of RDT 2.1

- Ex02: Workout the timeline sequence diagram for for all possible cses of RDT 2.2

Exercises:

- Ex 03: Consider the case where network channel can lose the packet but does not corrupt the packet. Let us call this protocol as RDT 2.1b.
  - Design the timeline sequence diagram which deals with packet loss.
  - Design the state transition diagram.

Exercises:

- Ex 4: Consider the case where network channel can loses every second packet in each direction, but doesn't corrupt the packet. Let us call this protocol as RDT 2.1c.
  - Design the timeline sequence diagram which deals with packet loss.
  - Design the state transition diagram.

Summary

- RDT 2.1
- RDT 2.2
- RDT 3.0