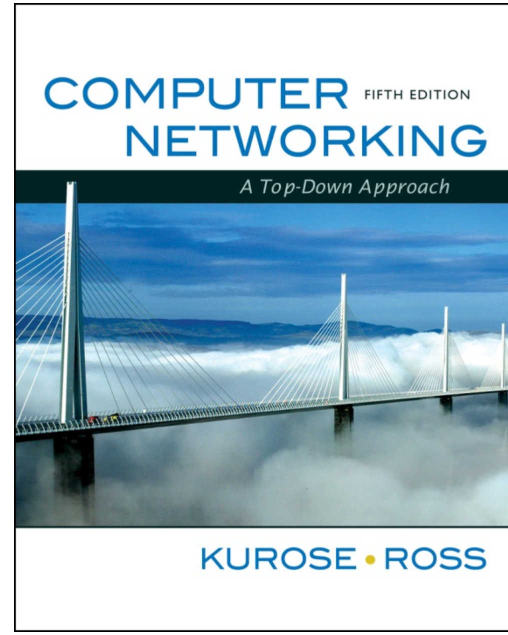
Tatiana Polishchuk, Postdoc

Email: tatiana.polishchuk@liu.se
Office: B:478

Transport Layer

TDTS04: Computer Networks & Distributed Systems
Instructor: Niklas Carlsson

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 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) in substantially unaltered form, that you mention their source (after all, we'd like people to use our book!)
- If you post any slides in substantially unaltered form 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

Computer Networking: A Top Down Approach
5th edition.
Jim Kurose, Keith Ross
Addison-Wesley, April 2009.

Transport Layer: Roadmap

3.1 Transport-layer services

3.2 Multiplexing and demultiplexing

3.3 Connectionless transport: UDP

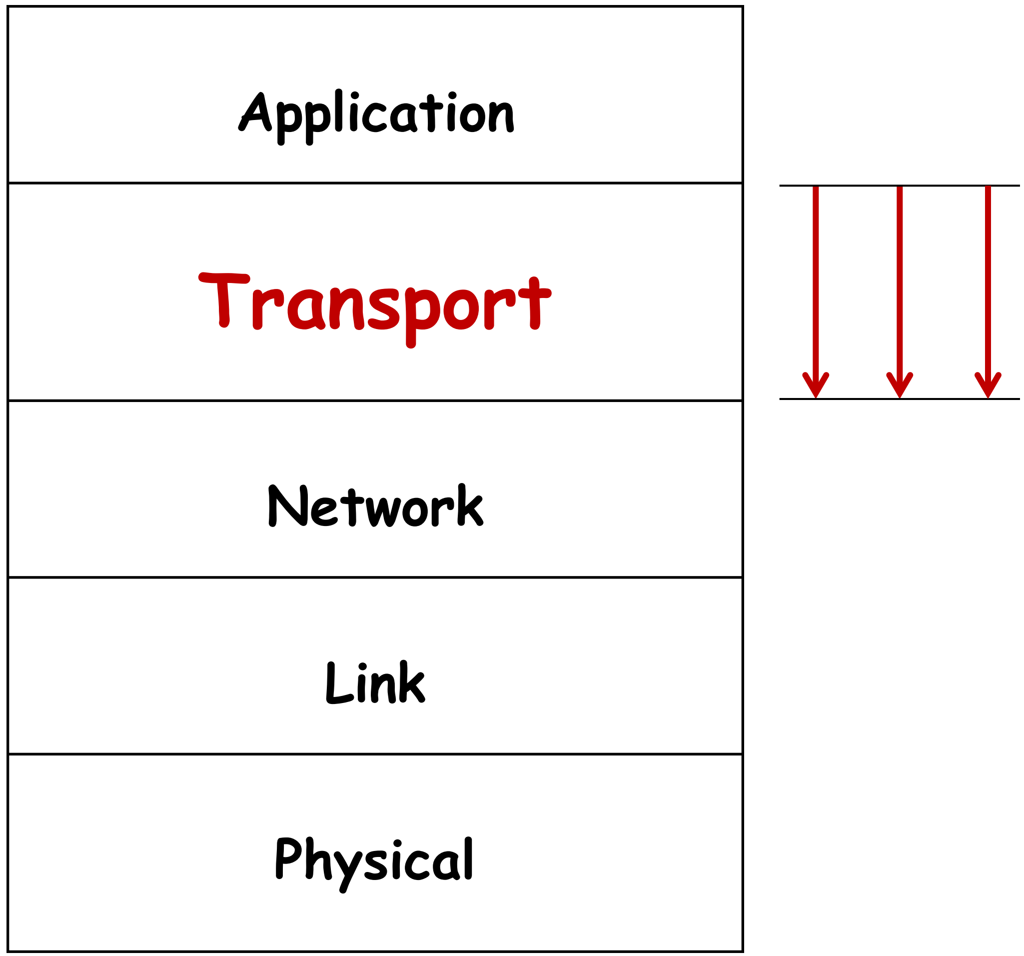
3.4 Principles of reliable data transfer

3.5 Connection-oriented transport: TCP
- segment structure
- reliable data transfer
- flow control
- connection management

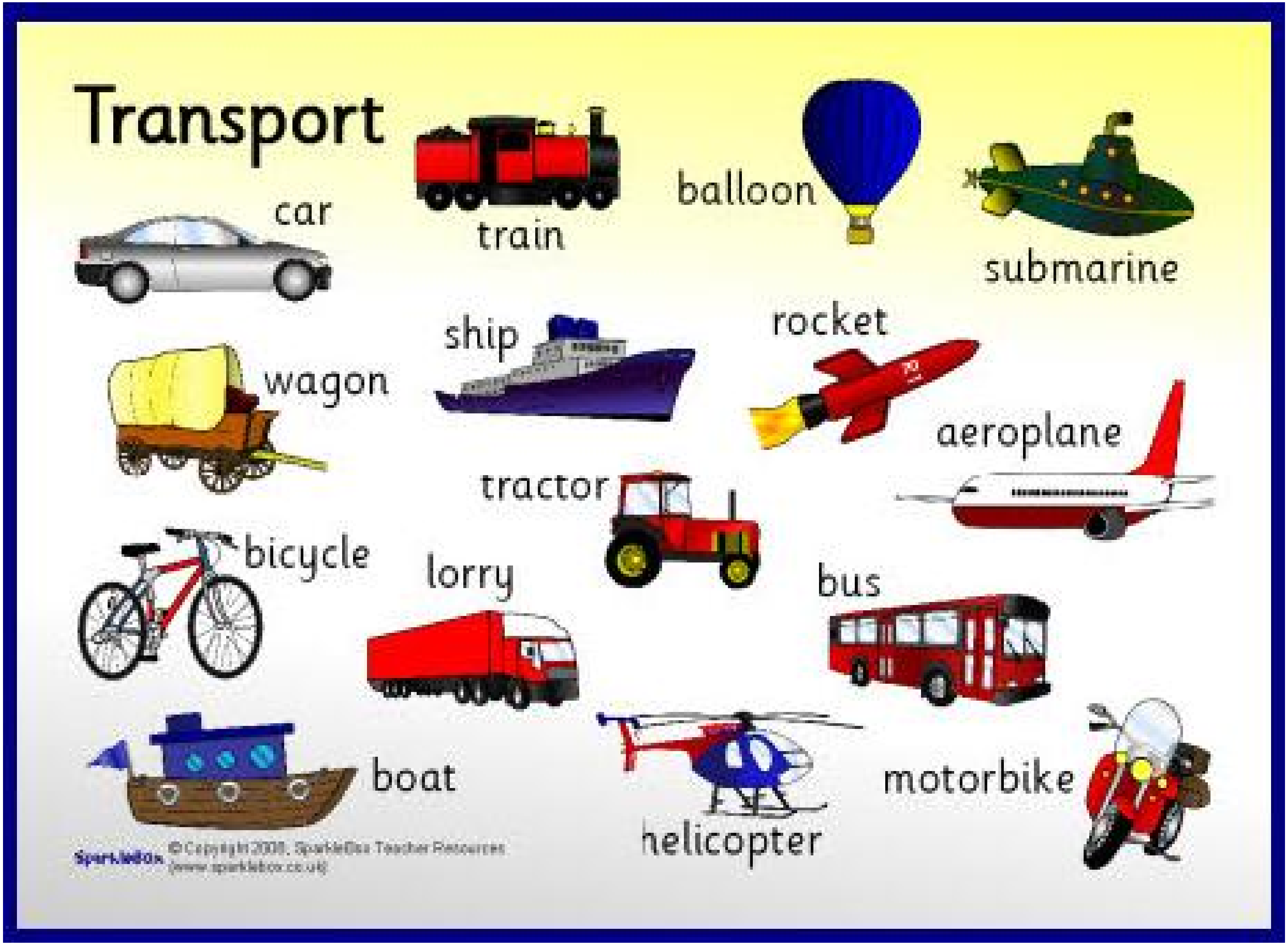
3.6 Principles of congestion control

3.7 TCP congestion control

TCP/IP protocol stack

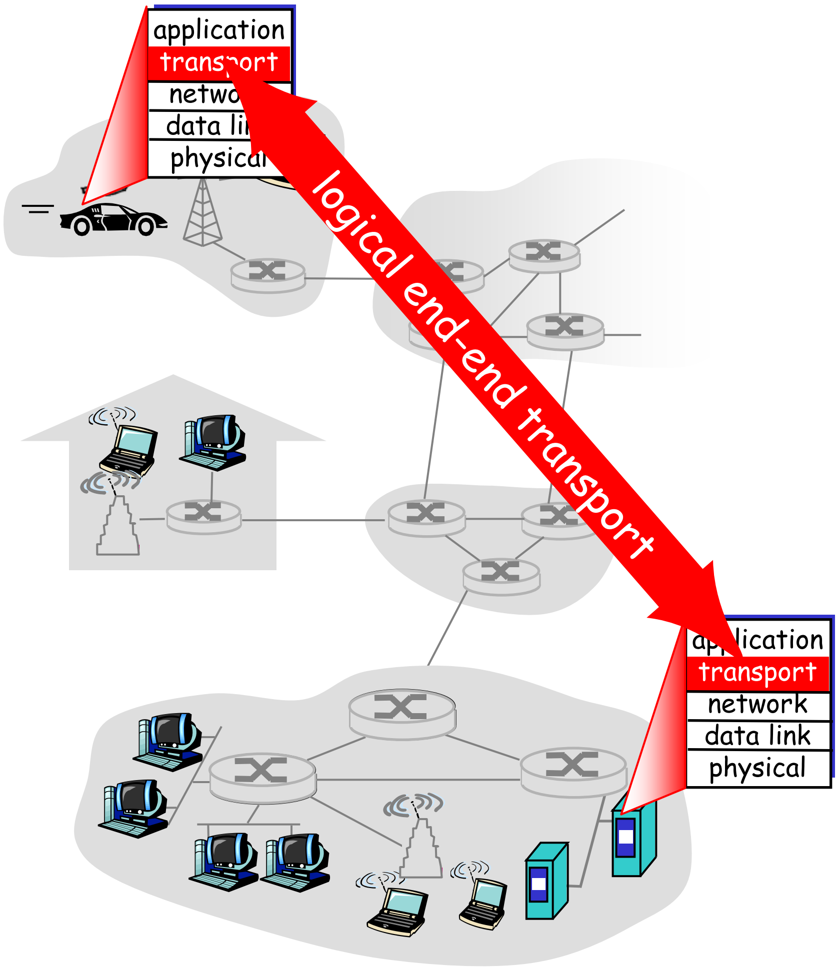


Transport?


Transport
car
train
balloon
submarine
ship
rocket
wagon
aeroplane
tractor
bicycle
lorry
bus
boat
helicopter
motorbike
SparkleBox
© Copyright 2008, SparkleBox Teacher Resources (www.sparklebox.co.uk)

Transport services

- provide *logical communication* between app processes running on different hosts
- transport protocols run in end systems
  - send side: breaks app messages into segments, passes to network layer
  - rcv side: reassembles segments into messages, passes to app layer



Internet transport-layer protocols

- unreliable, unordered delivery: (UDP)
  - no-frills extension of "best-effort" IP

- reliable, in-order delivery (TCP)
  - congestion control
  - flow control
  - connection setup
  - …

Applications?

Transport Layer: Roadmap

3.1 Transport-layer services

3.2 Multiplexing and demultiplexing

3.3 Connectionless transport: UDP

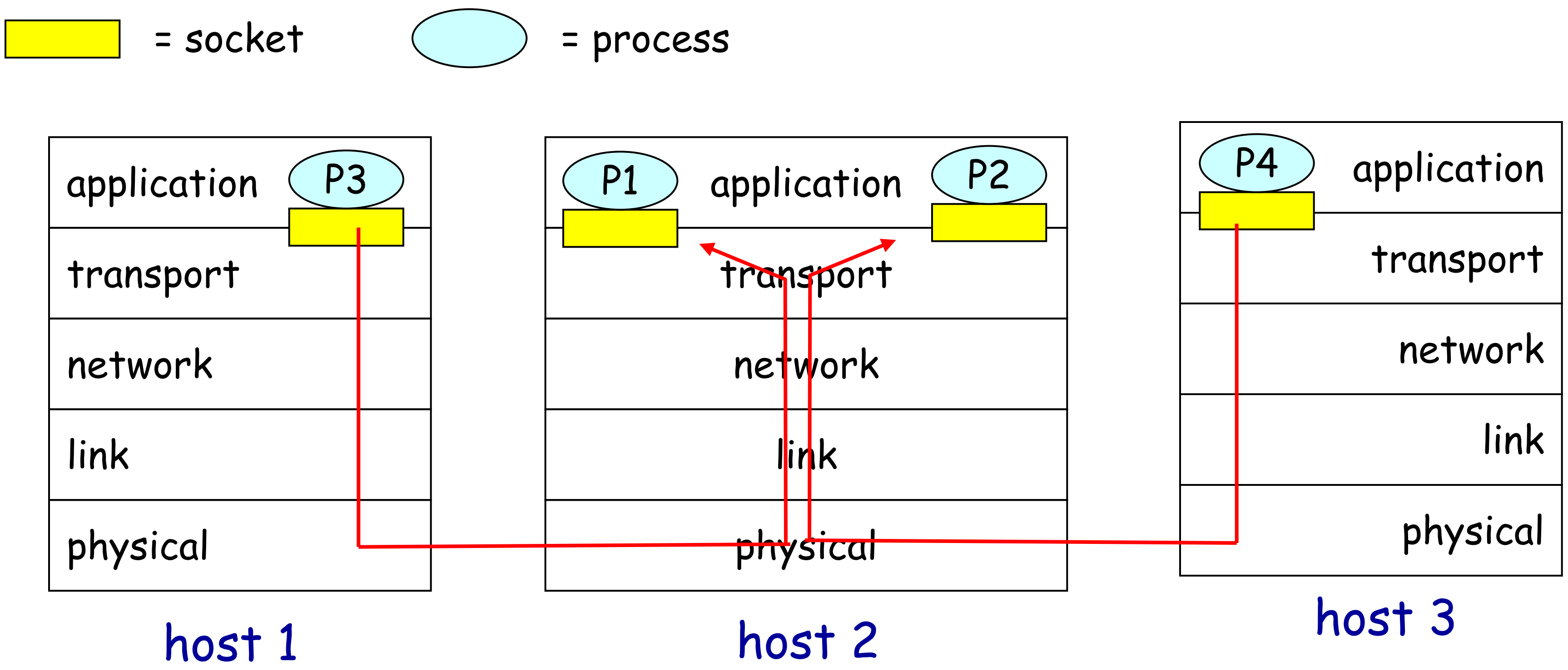
3.4 Principles of reliable data transfer

3.5 Connection-oriented transport: TCP
- segment structure
- reliable data transfer
- flow control
- connection management

3.6 Principles of congestion control

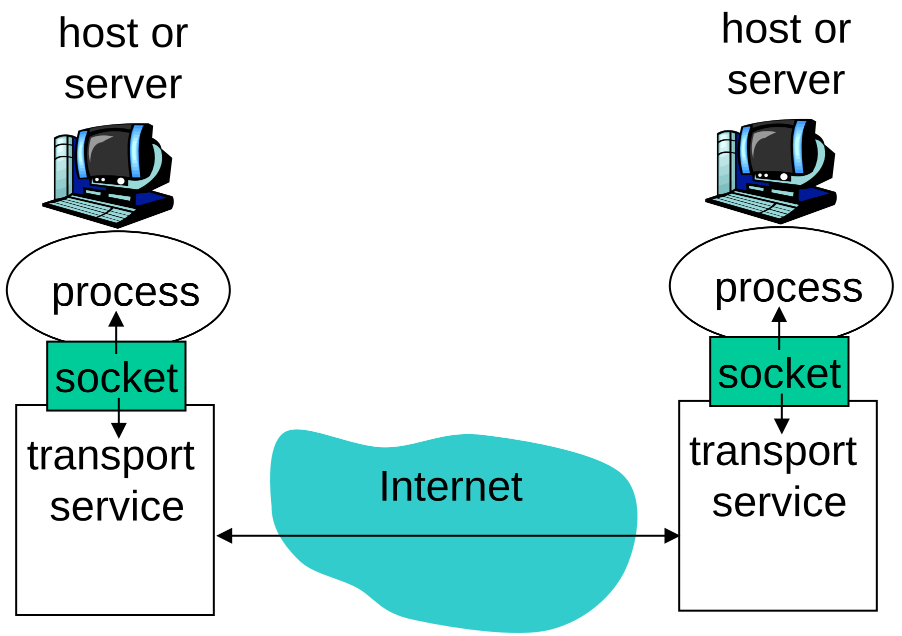
3.7 TCP congestion control

Multiplexing/demultiplexing



Recall: Sockets

- process sends/receives messages to/from its socket
- socket analogous to door
  - provides connection between the application and network



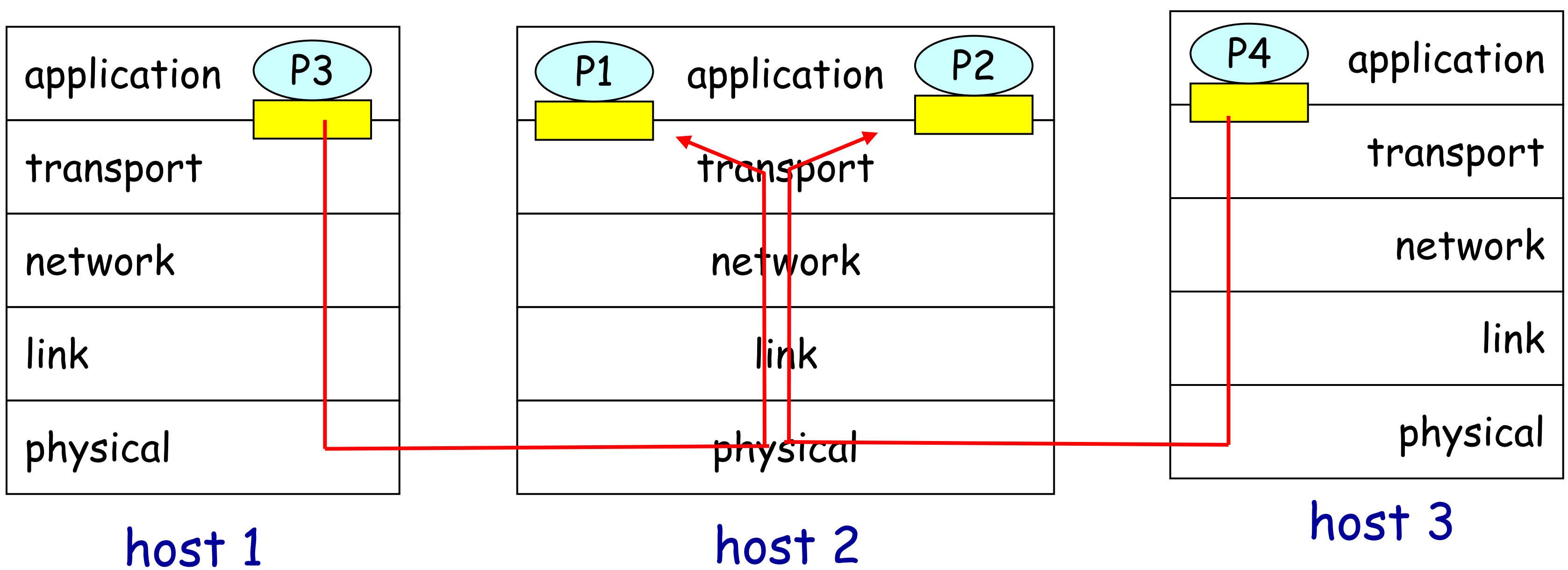
Multiplexing/demultiplexing

Demultiplexing at rcv host:

delivering received segments to correct socket

Multiplexing at send host:

gathering data from multiple sockets, enveloping data with header (later used for demultiplexing)



How demultiplexing works

- each datagram has source IP address, destination IP address
- host uses IP addresses & port numbers to direct segment to appropriate socket

Connectionless demultiplexing

- *UDP* :

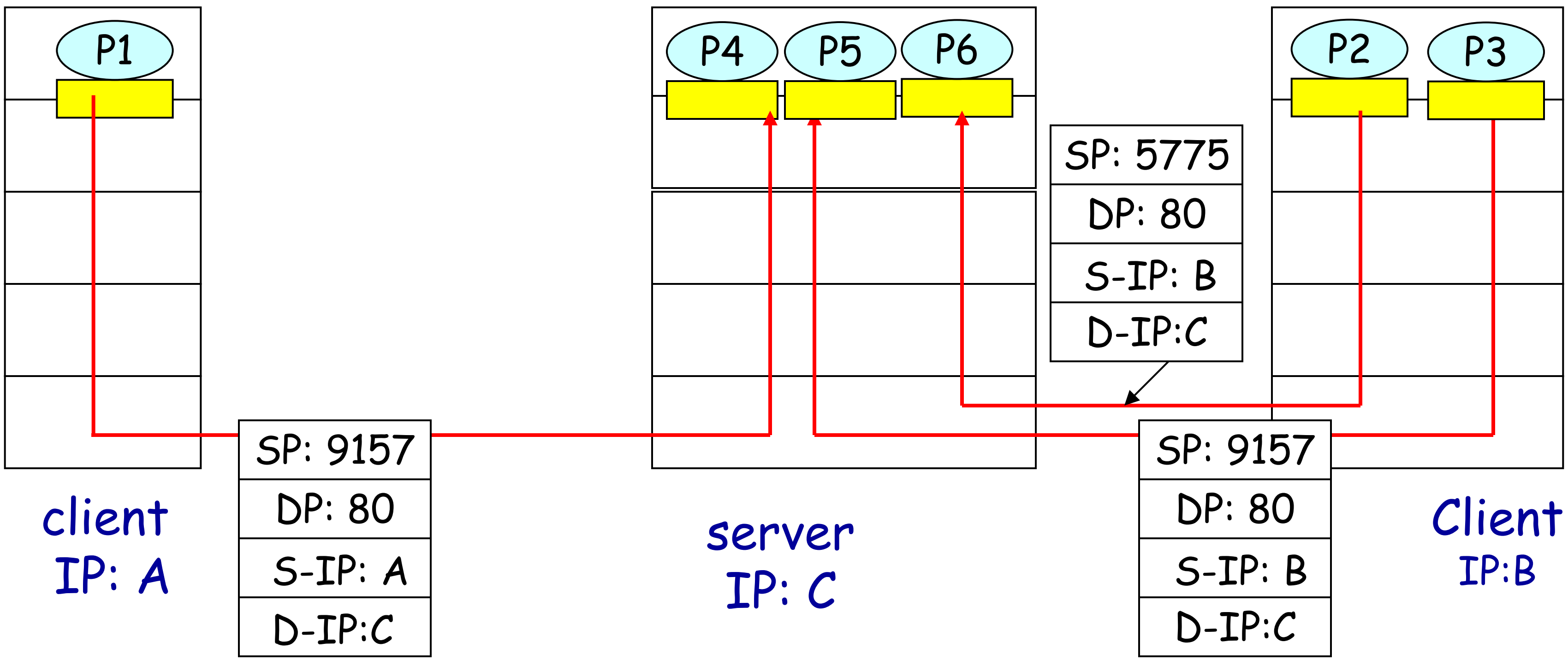
  when creating datagram to send into UDP socket, must specify the 2-tuple

  - dest IP address
  - dest port number

Connection-oriented demux

- TCP socket identified by 4-tuple:
  - source IP address
  - source port number
  - dest IP address
  - dest port number
- recv host uses all four values to direct segment to appropriate socket
- server host may support many simultaneous TCP sockets:
  - each socket identified by its own 4-tuple

Connection-oriented demux (cont)



Transport Layer: Roadmap

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

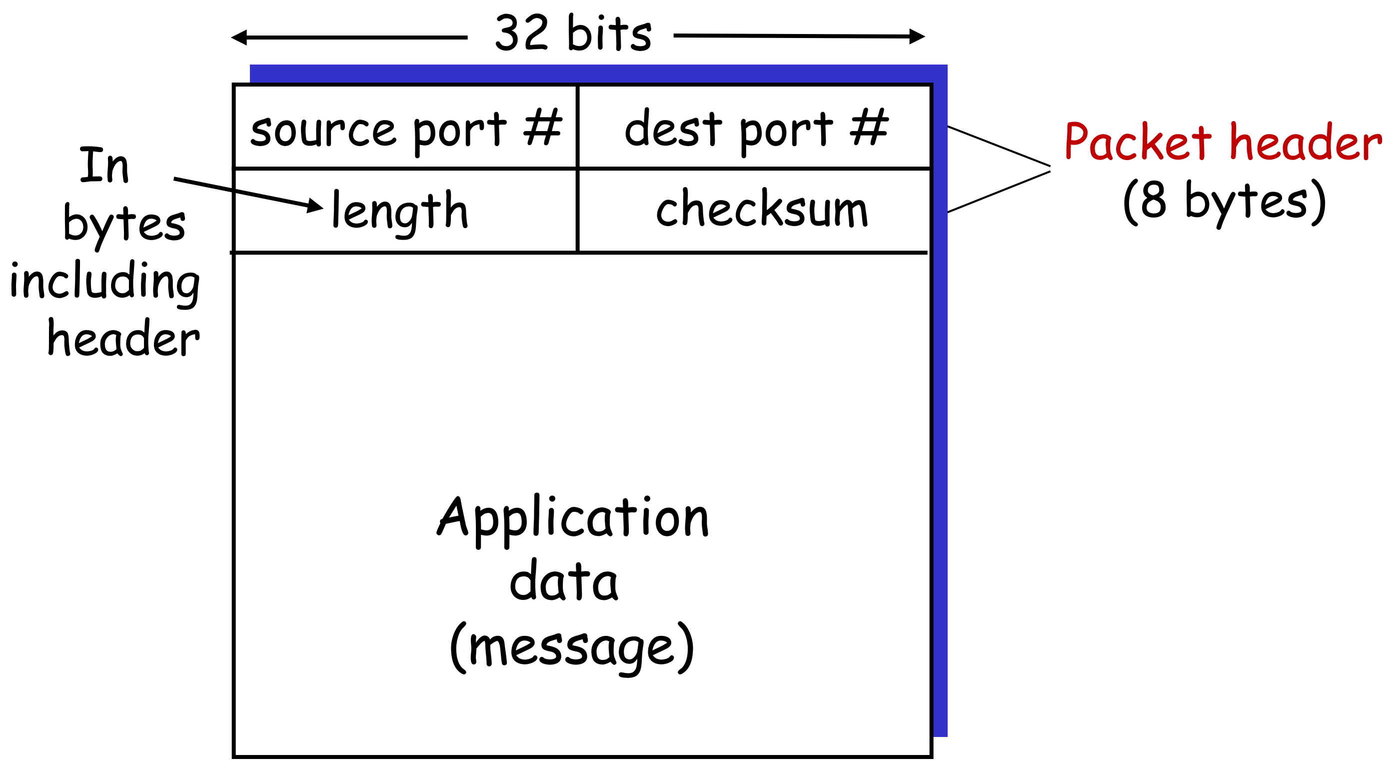
3.7 TCP congestion control

UDP: User Datagram Protocol [RFC 768]

- *connectionless:*
  - no handshaking between UDP sender and receiver

- *only 2 functions*
  - mult/demult
  - error detection (optional)

UDP segment format



UDP checksum

Goal: detect "errors" (e.g., flipped bits) in transmitted segment

Sender:

- treat segment contents as sequence of 16-bit integers
- checksum: addition (1's complement sum) of segment contents
- sender puts checksum value into UDP checksum field

Receiver:

- compute checksum of received segment
- check if computed checksum equals checksum field value:
  - NO - error detected
  - YES - no error detected.

Internet Checksum Example

- Note: when adding numbers, a carryout from the most significant bit needs to be added to the result
- Example: add two 16-bit integers

		1	1	1	0	0	1	1	0	0	1	1	0	0	1	1	0
		1	1	0	1	0	1	0	1	0	1	0	1	0	1	0	1
wraparound	1	1	0	1	1	1	0	1	1	1	0	1	1	1	0	1	1
sum		1	0	1	1	1	0	1	1	1	0	1	1	1	1	0	0
checksum		0	1	0	0	0	1	0	0	0	1	0	0	0	0	1	1

Transport Layer: Roadmap

3.1 Transport-layer services

3.2 Multiplexing and demultiplexing

3.3 Connectionless transport: UDP

3.4 Principles of reliable data transfer

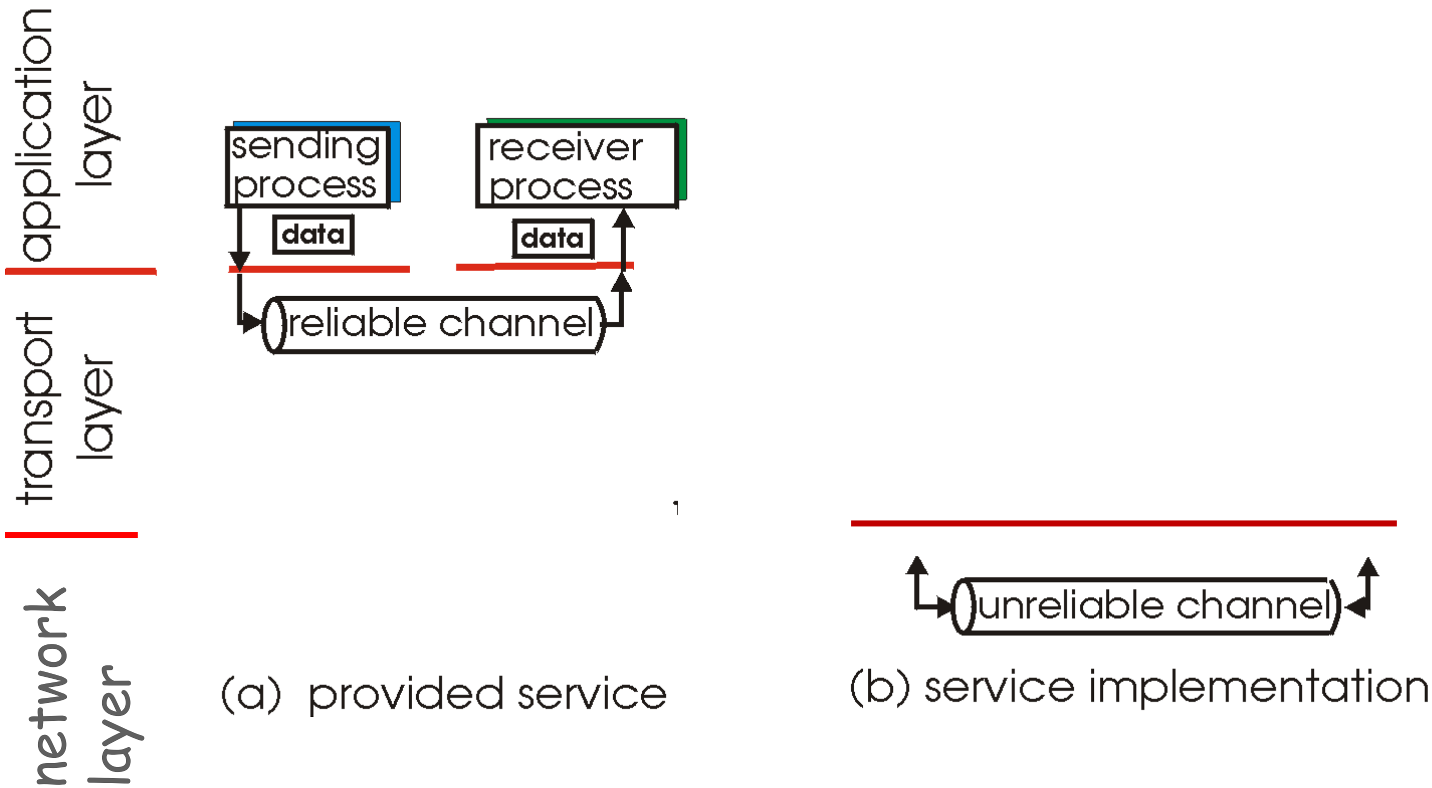
3.5 Connection-oriented transport: TCP
- segment structure
- reliable data transfer
- flow control
- connection management

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!

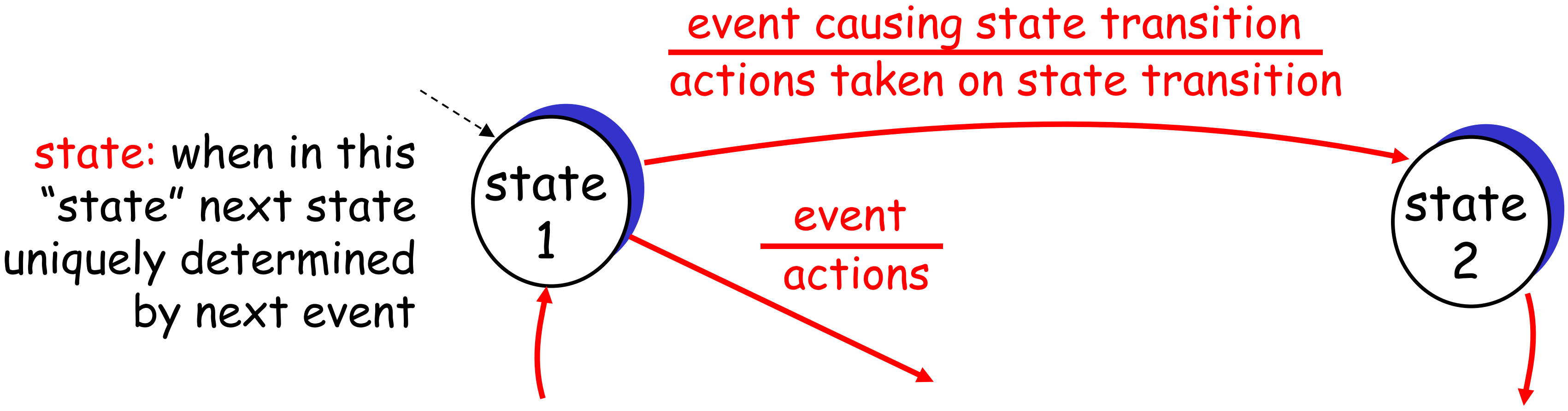


- characteristics of unreliable channel will determine complexity of reliable data transfer protocol (rdt)

Reliable data transfer: getting started

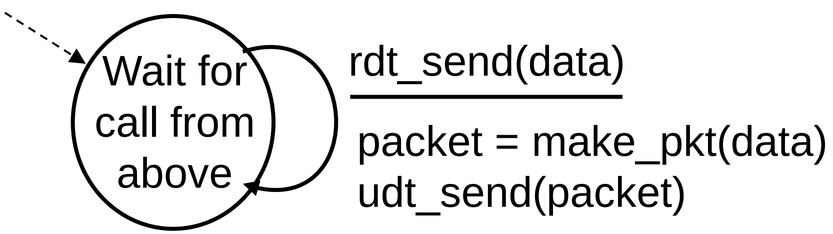
We'll:

- incrementally develop sender, receiver sides of reliable data transfer protocol (rdt)
- use finite state machines (FSM) to specify sender, receiver

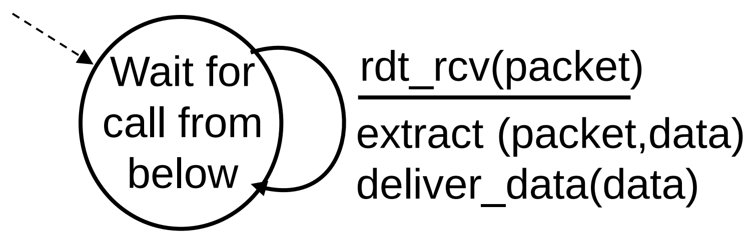


Rdt1.0: reliable transfer over a reliable channel

- underlying channel perfectly reliable
  - no bit errors
  - no loss of packets
- separate FSMs for sender, receiver:



sender



receiver

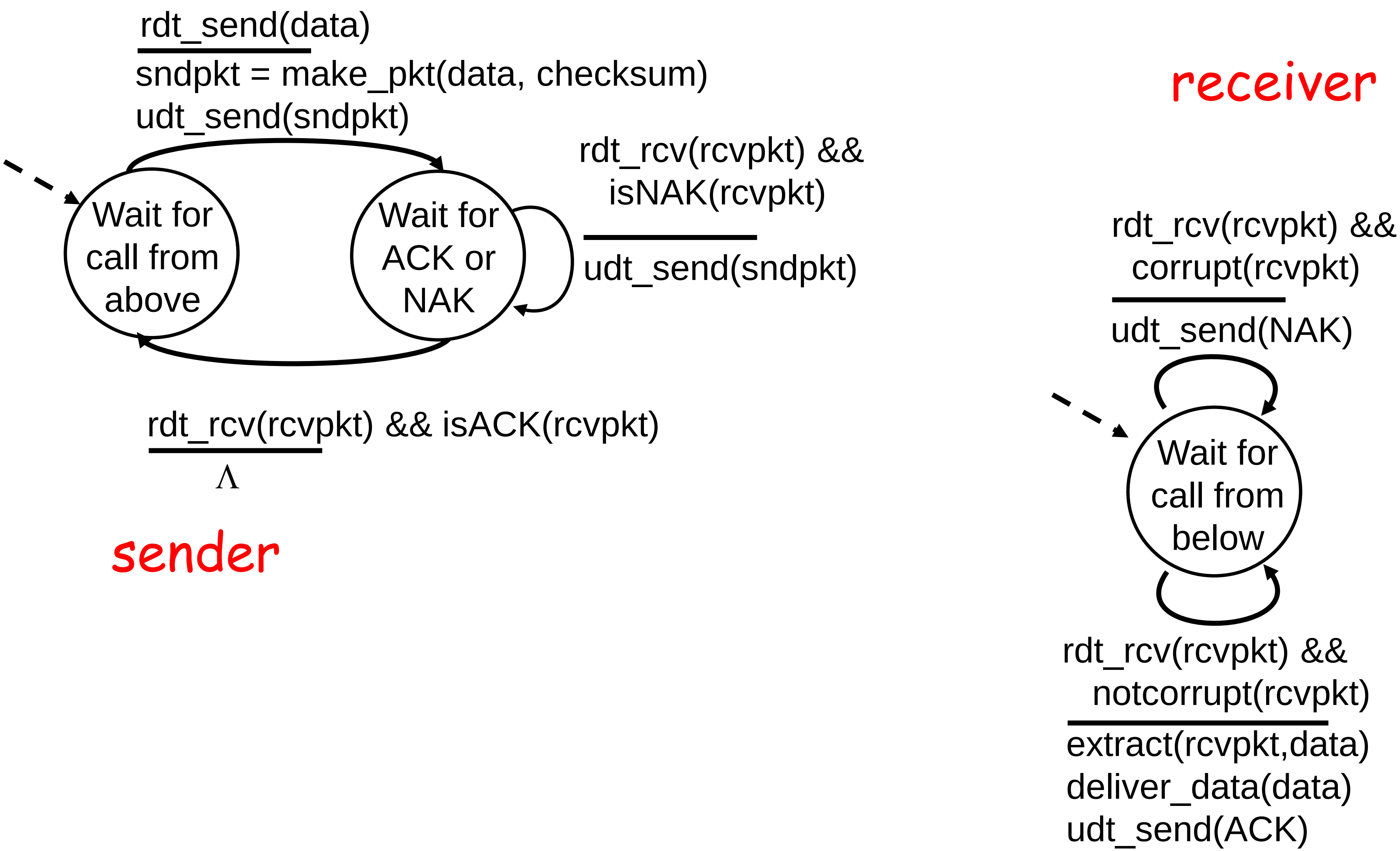
Rdt2.0: channel with bit errors

- underlying channel may flip bits in packet
- use checksum to detect bit errors
- how to recover from errors?

Rdt2.0: channel with bit errors

- Mechanisms for error recovery:
  - *acknowledgements (ACKs):* receiver explicitly tells sender that pkt received OK
  - *negative acknowledgements (NAKs):* receiver explicitly tells sender that pkt had errors
  - *retransmissions* - sender retransmits pkt on receipt of NAK

rdt2.0: FSM specification



rdt2.0 has a fatal flaw!

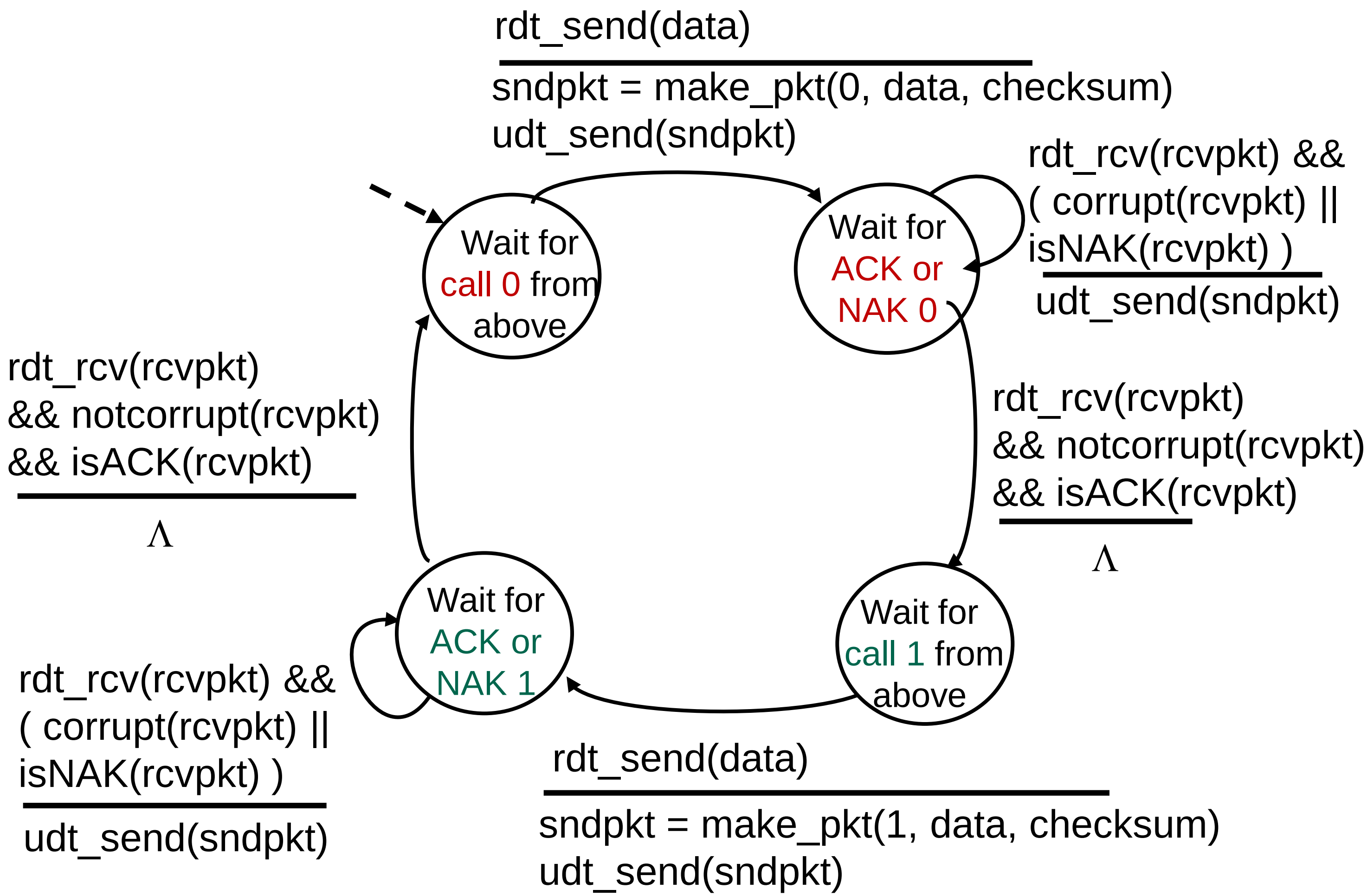
What happens if ACK/NAK corrupted?

- sender doesn't know what happened at receiver!
- can't just retransmit: possible duplicate

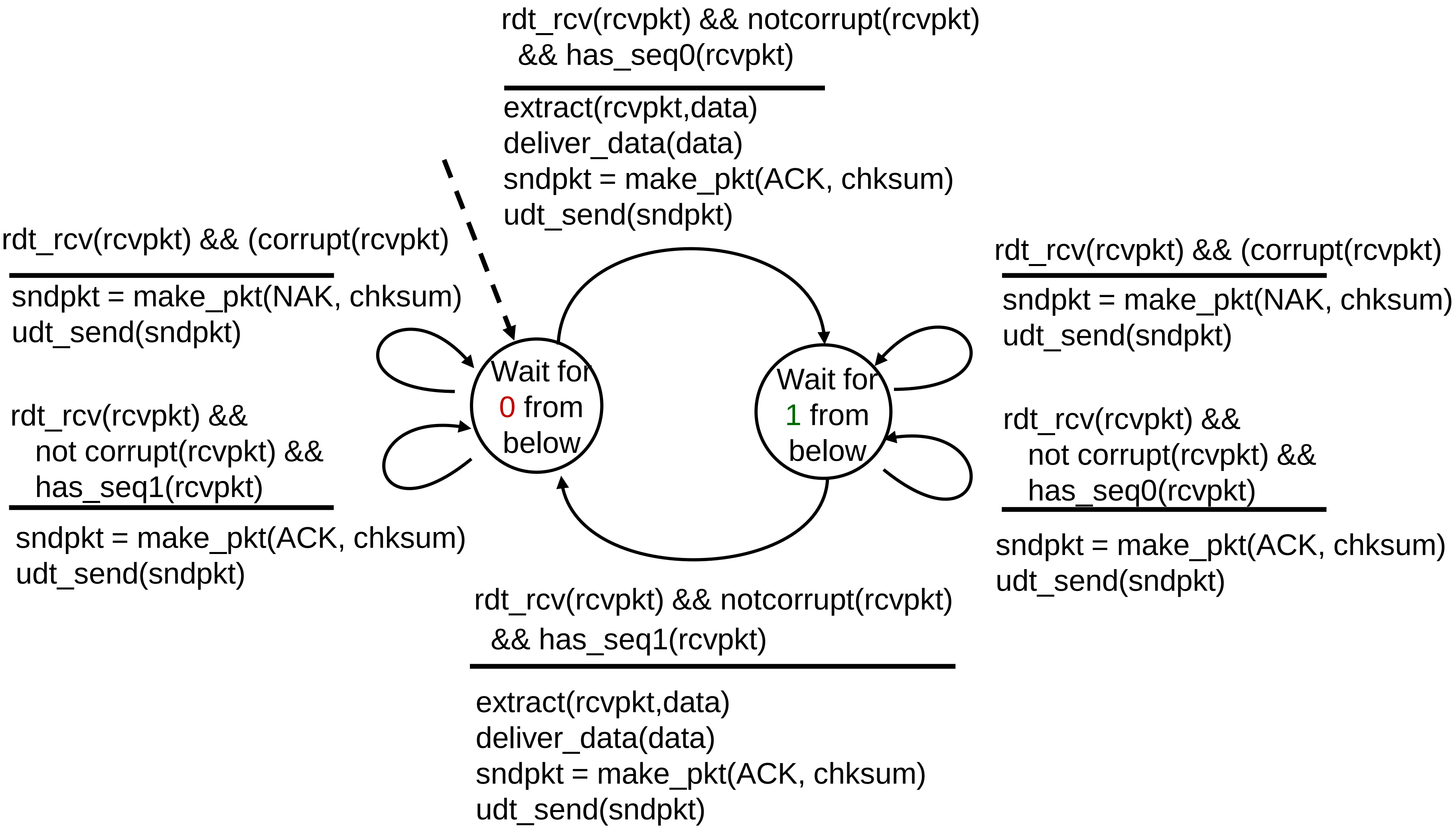
Handling duplicates:

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

rdt2.1: sender, handles garbled ACK/NAKs



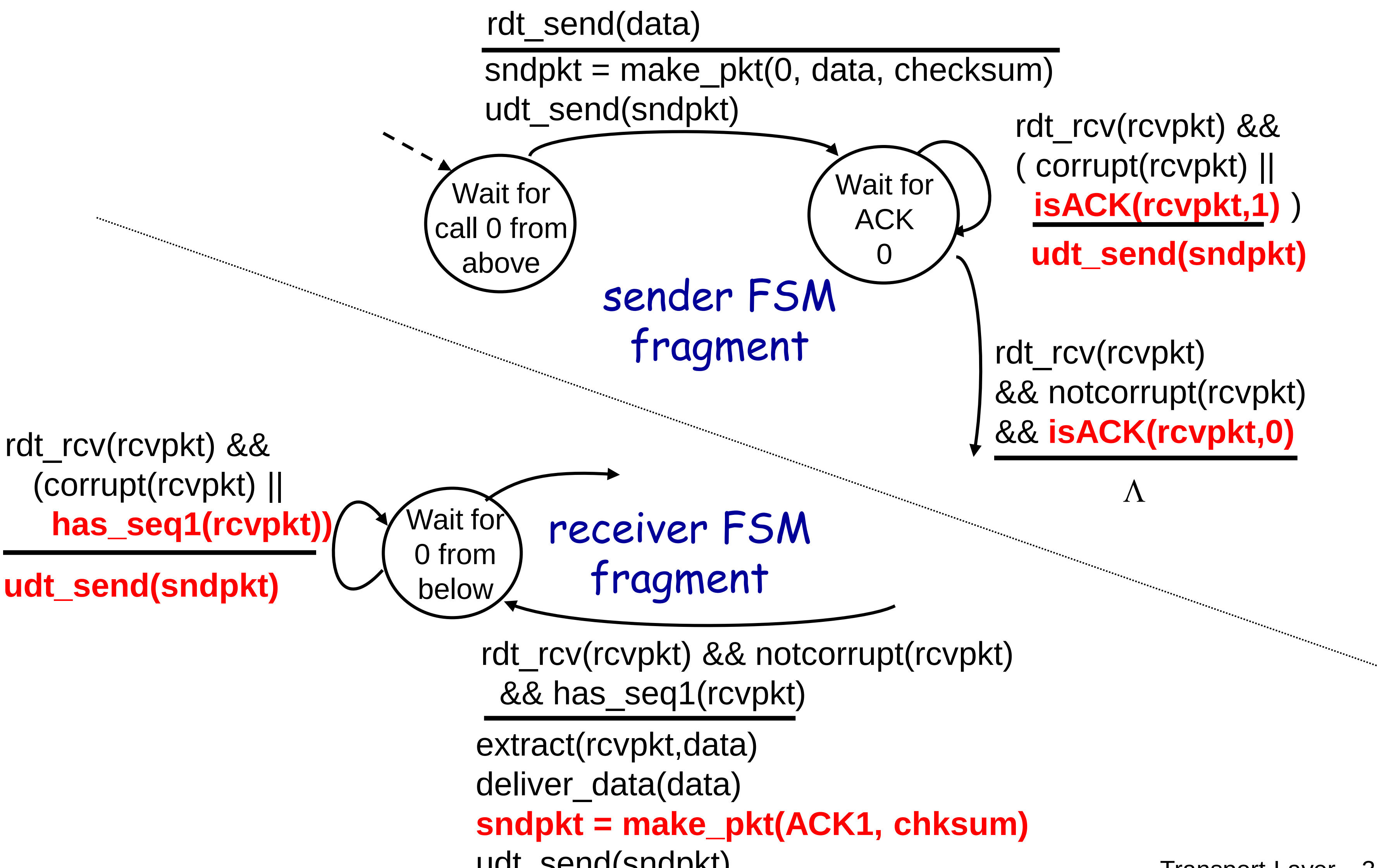
rdt2.1: receiver, handles garbled ACK/NAKs



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



rdt3.0: channels with errors *and* loss

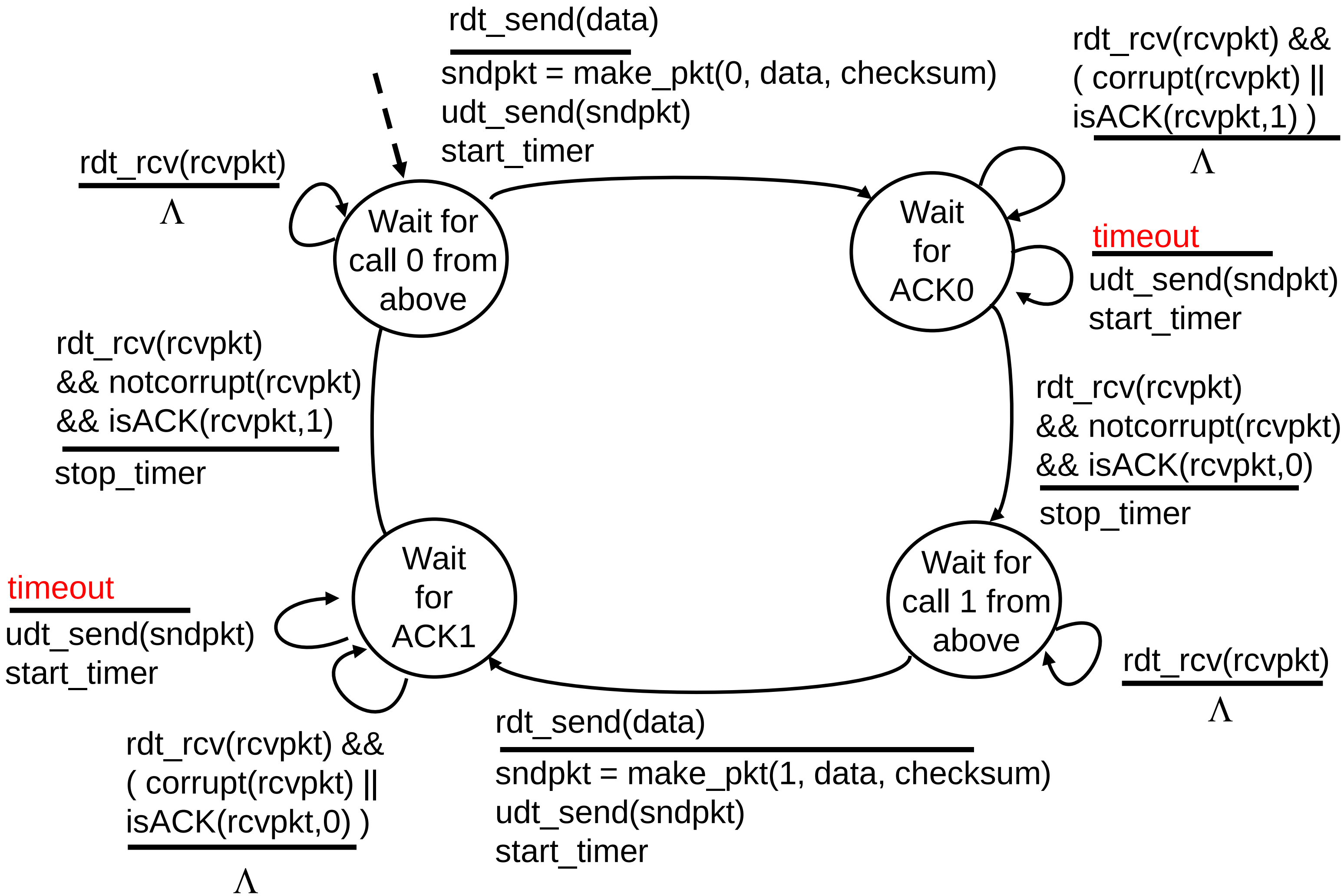
New assumption: underlying channel can lose packets (data or ACKs)

- checksum
- seq. #
- ACKs
- retransmissions
- ?

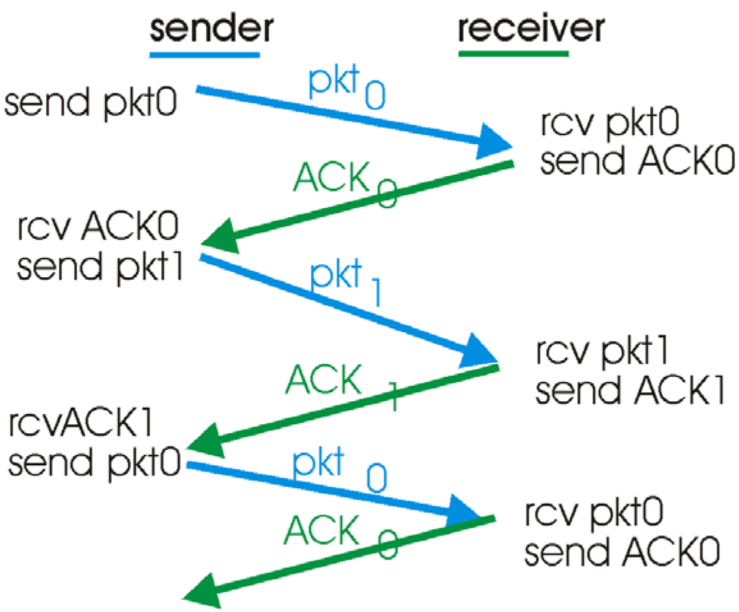
Approach: sender waits "reasonable" amount of time for ACK

- retransmits if no ACK received in this time
- if pkt (or ACK) just delayed (not lost):
  - retransmission will be duplicate, but use of seq. #'s already handles this
  - receiver must specify seq # of pkt being ACKed
- requires **countdown timer**

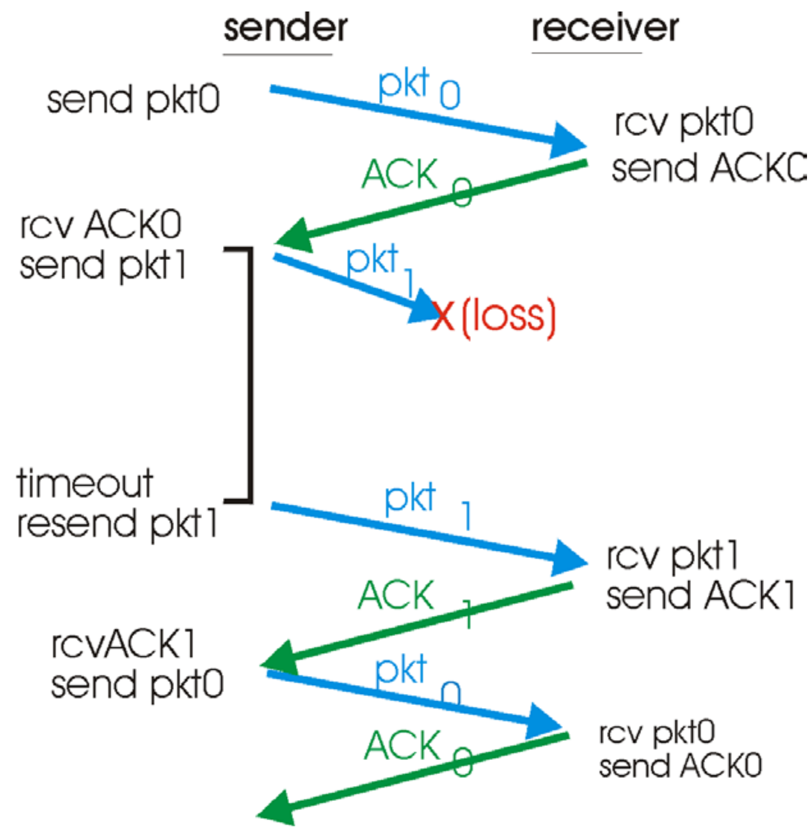
rdt3.0 sender



rdt3.0 in action

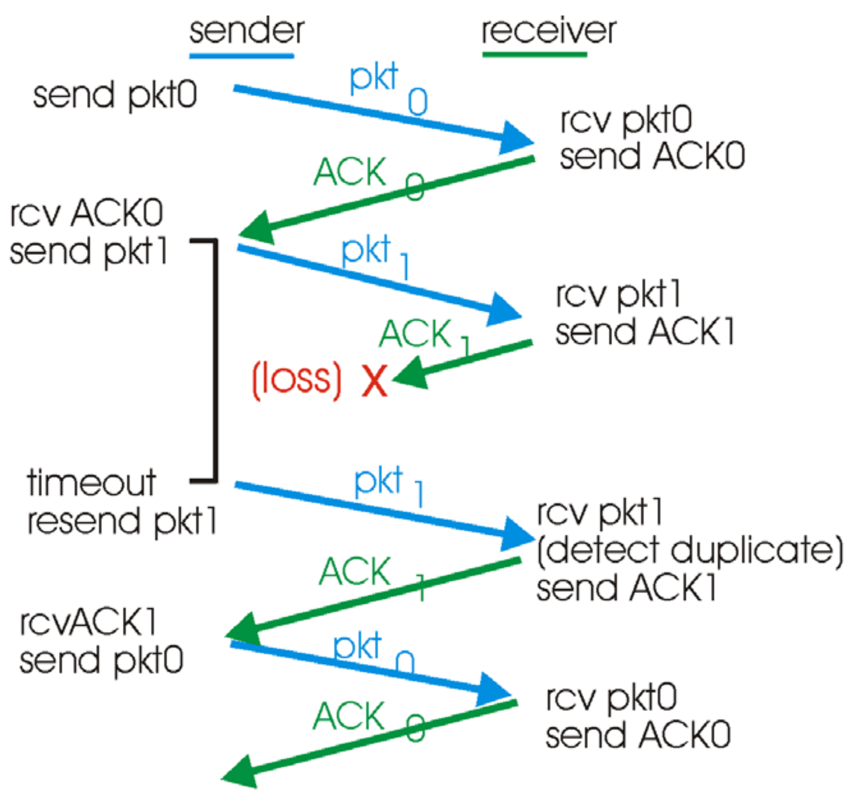


(a) operation with no loss

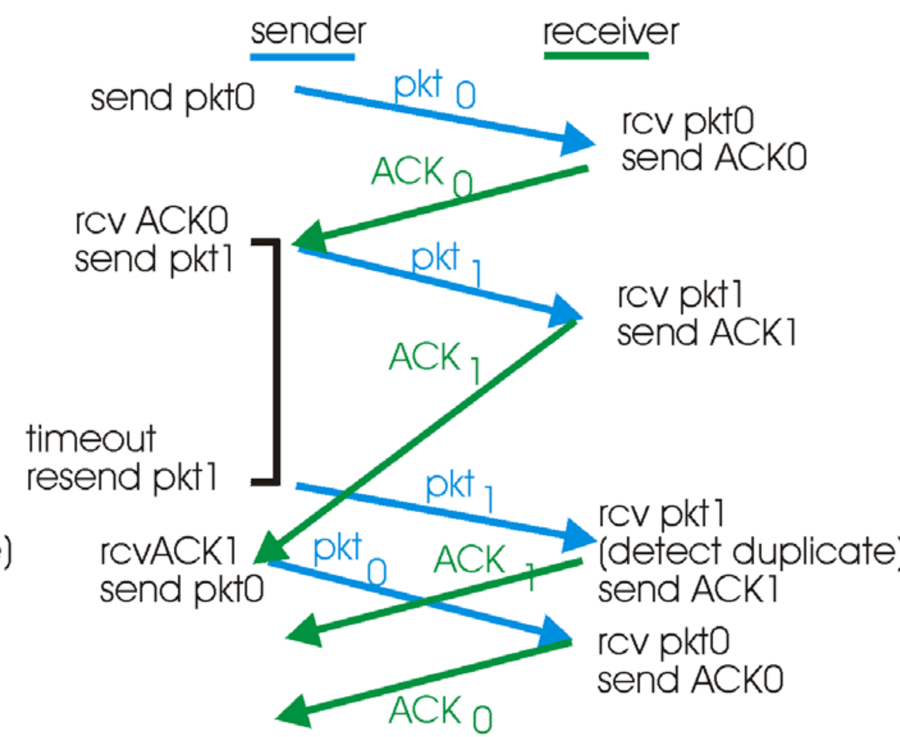


(b) lost packet

rdt3.0 in action

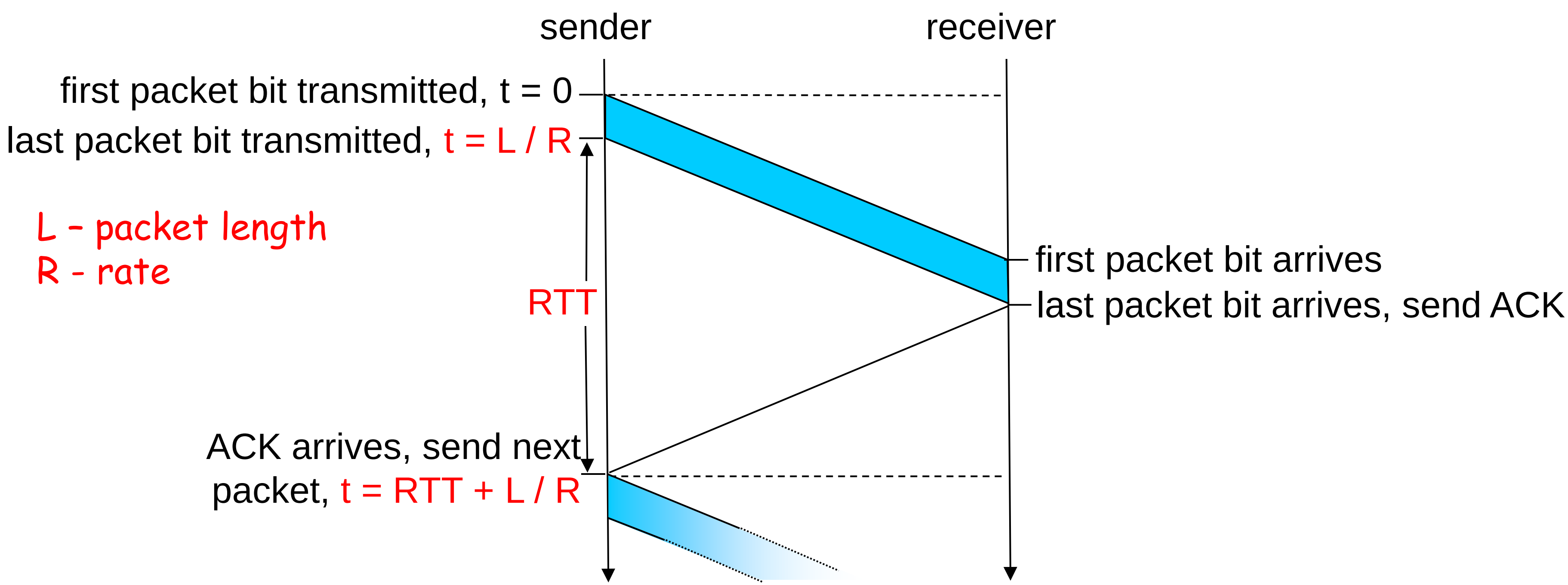


(c) lost ACK



(d) premature timeout

rdt3.0: stop-and-wait operation

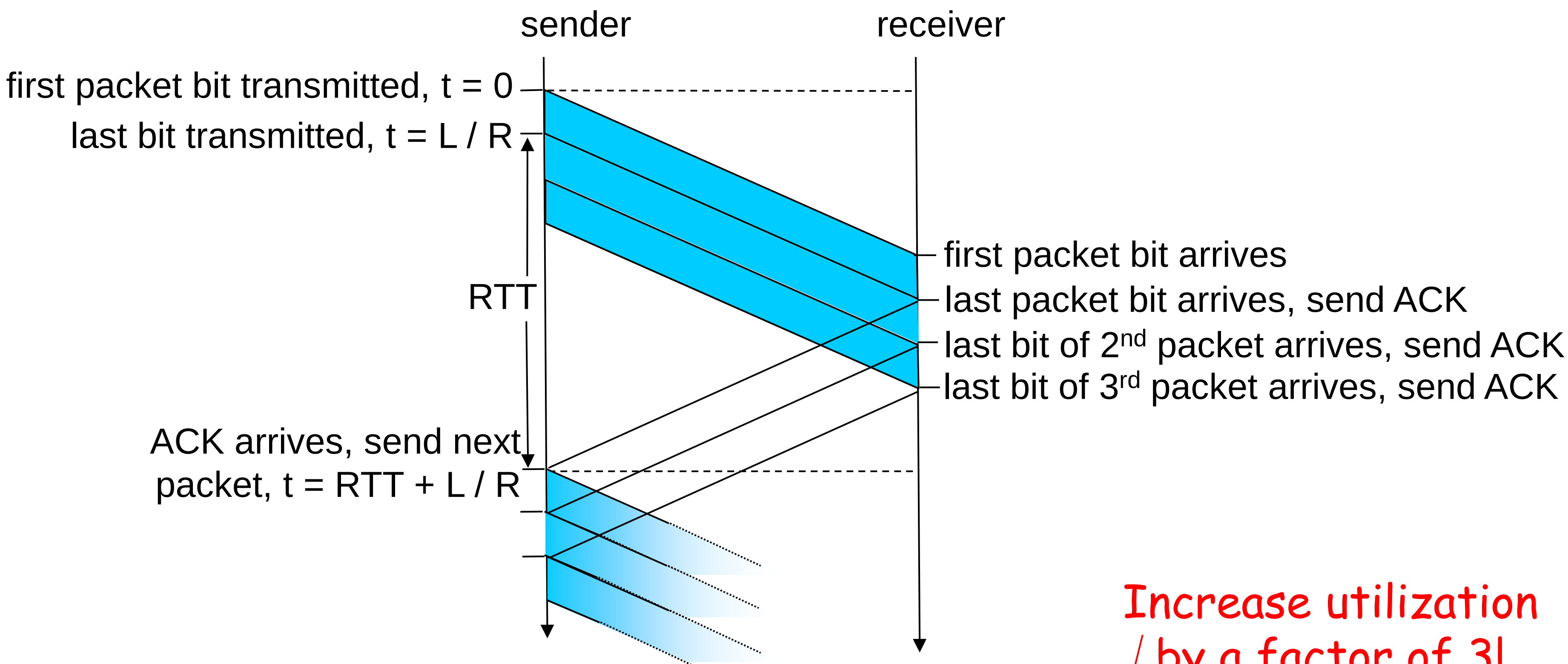


L - packet length
R - rate

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

Utilization

Pipelining: increased utilization



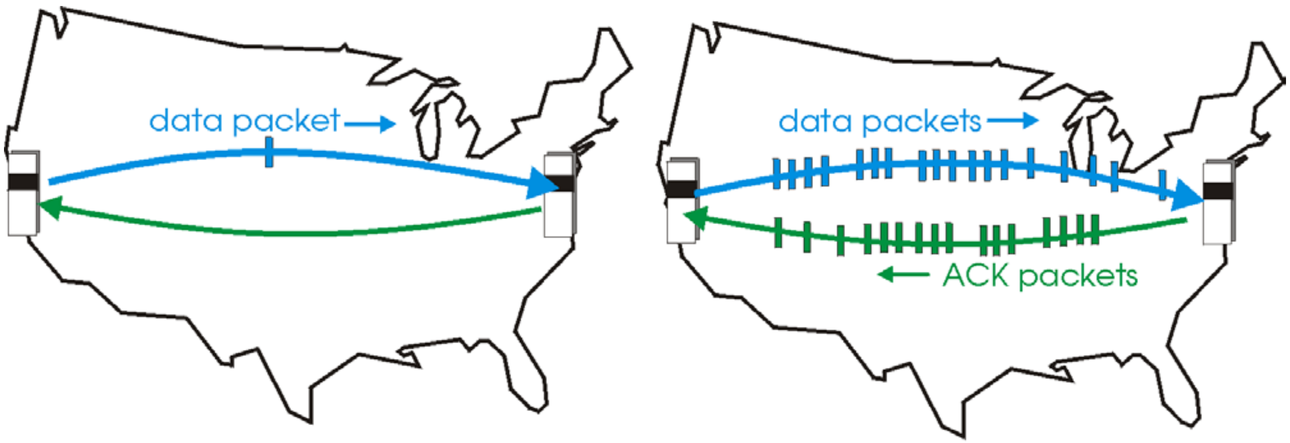
Increase utilization by a factor of 3!

$$U_{sender} = \frac{3 * L / R}{RTT + L / R} = \frac{.024}{30.008} = 0.0008$$

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

Mechanisms for reliable data transfer

Stop-and-wait

- ✓ checksum
- ✓ seq. #
- ✓ ACKs
- ✓ retransmissions
- ✓ timeouts

Pipelined

- seq. # range increased
- windows
- Cumulative ACKs
- Send/recv buffers

Pipelined Protocols

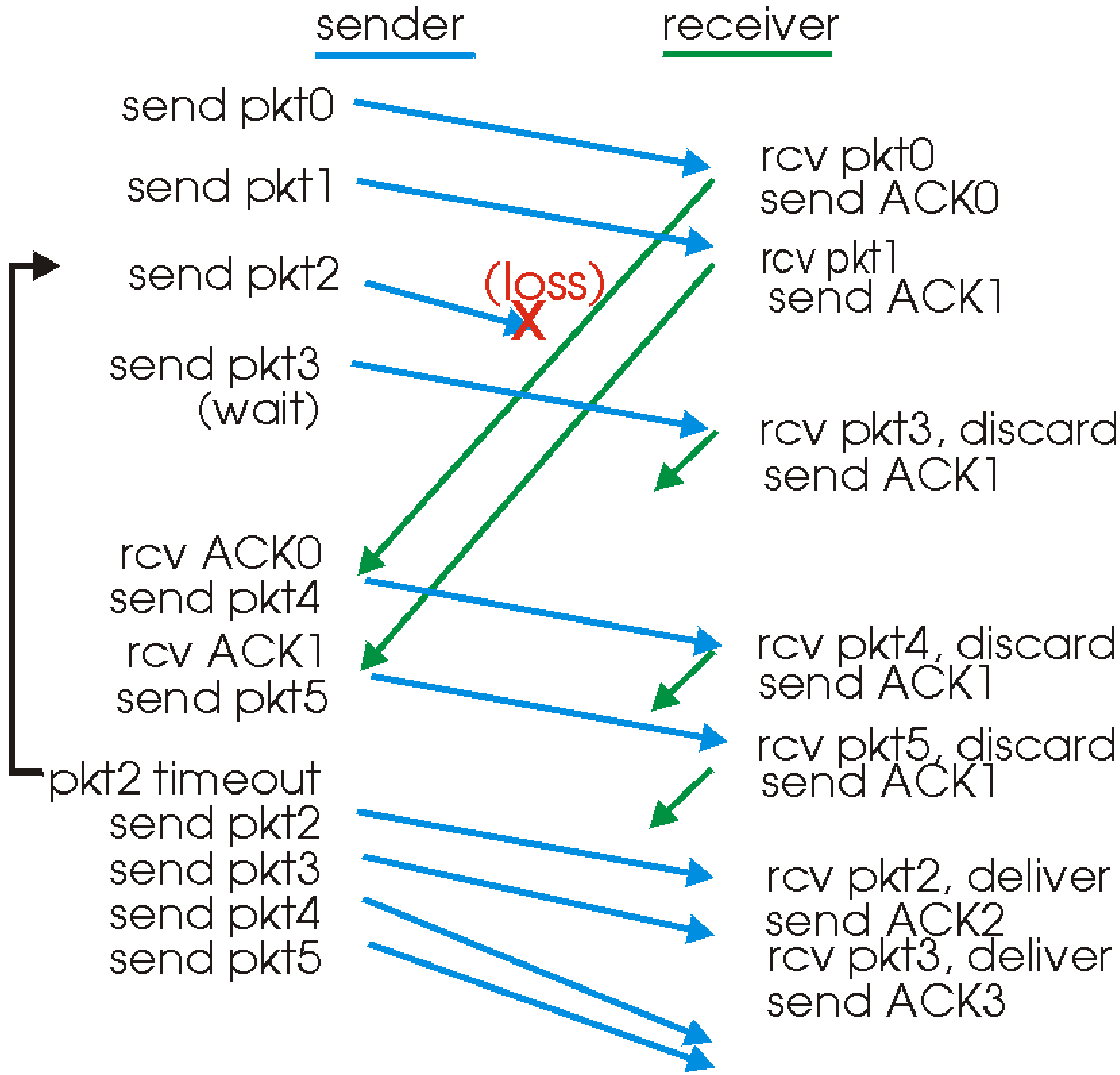
Go-back-N:

- N unacked packets in pipeline – Window
- *cumulative* acks
  - doesn't ack packet if there's a gap
- sender has timer for **oldest** unacked packet
  - if timer expires, retransmit **all** unack'ed packets

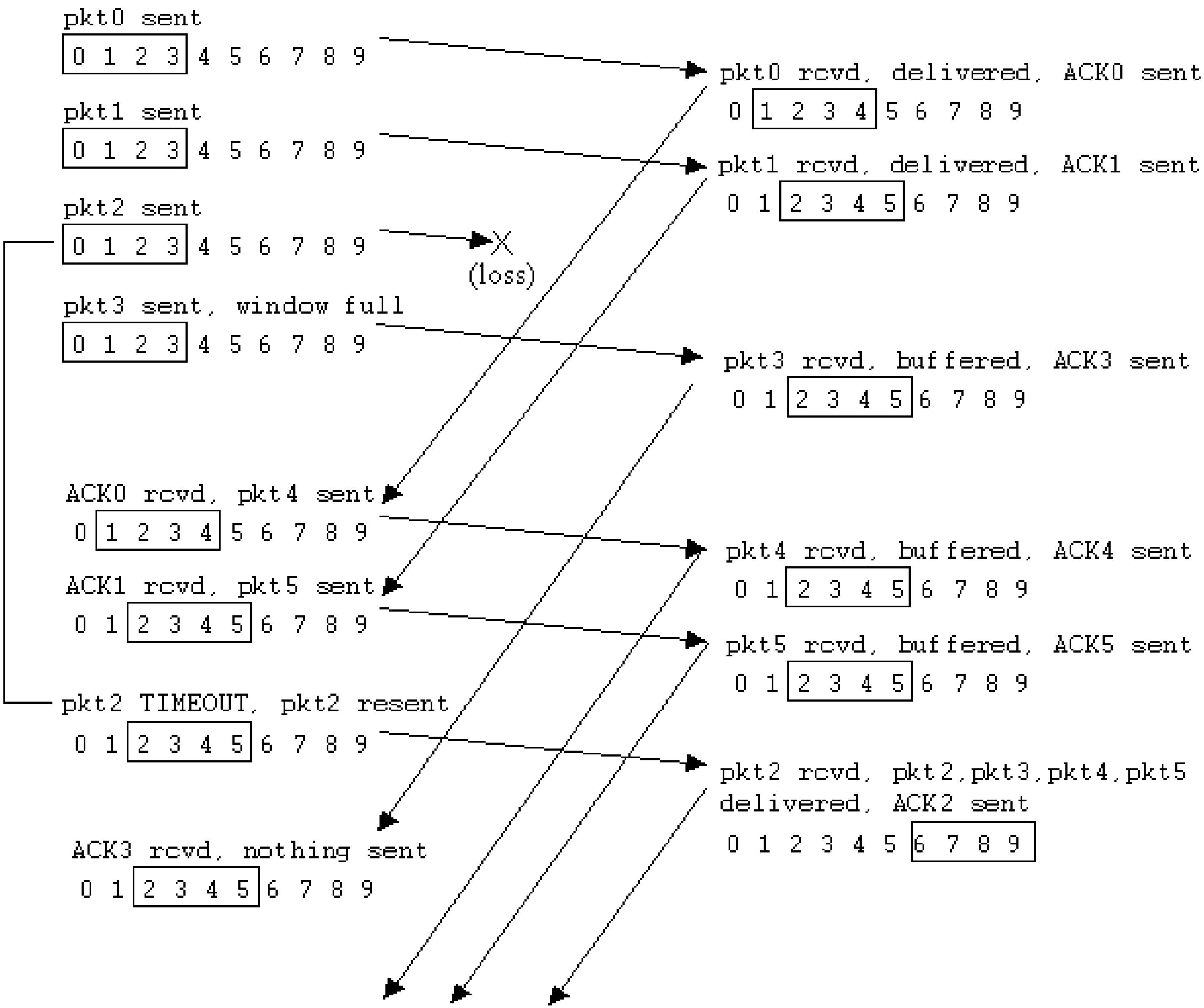
Selective Repeat:

- sender can have up to N unack'ed packets in pipeline
- rcvr sends *individual ack* for each packet
- sender maintains timer for each unacked packet
  - when timer expires, retransmit only unack'ed packet

GBN in action



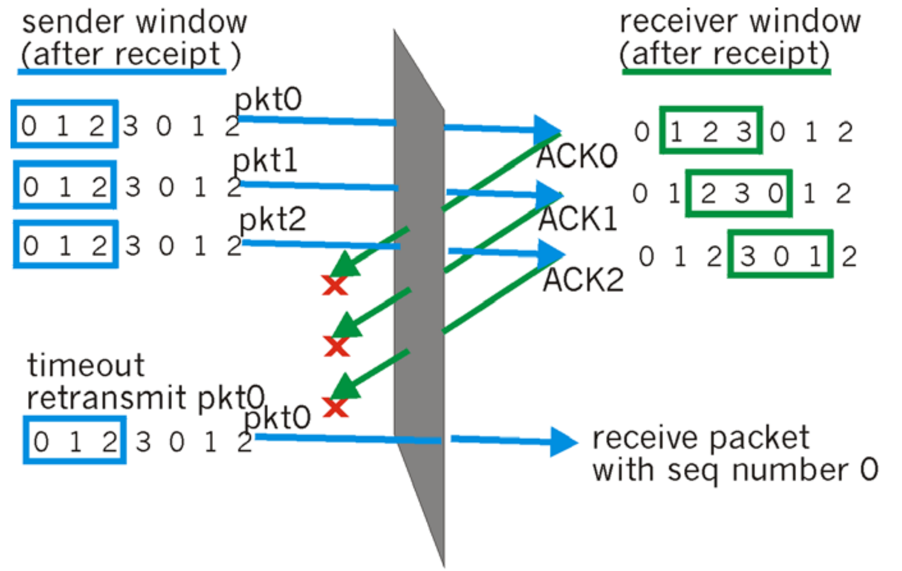
Selective repeat in action



Selective repeat: dilemma

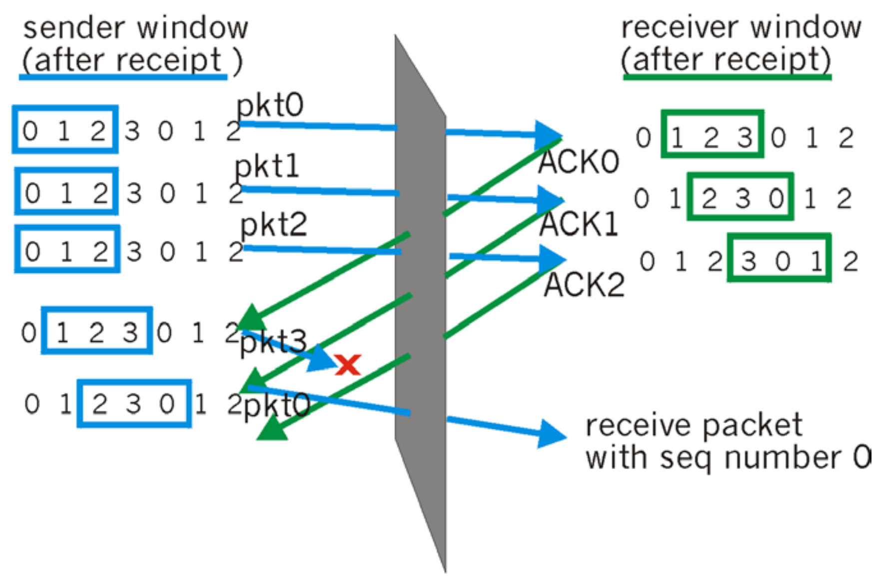
Example:

- seq #'s: 0, 1, 2, 3
- window size=3



(a)

- receiver sees no difference in two scenarios!
- incorrectly passes duplicate data as new in (a)



(b)

Q: what relationship between seq # size and window size?

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