Unit-3:
Transport Layer

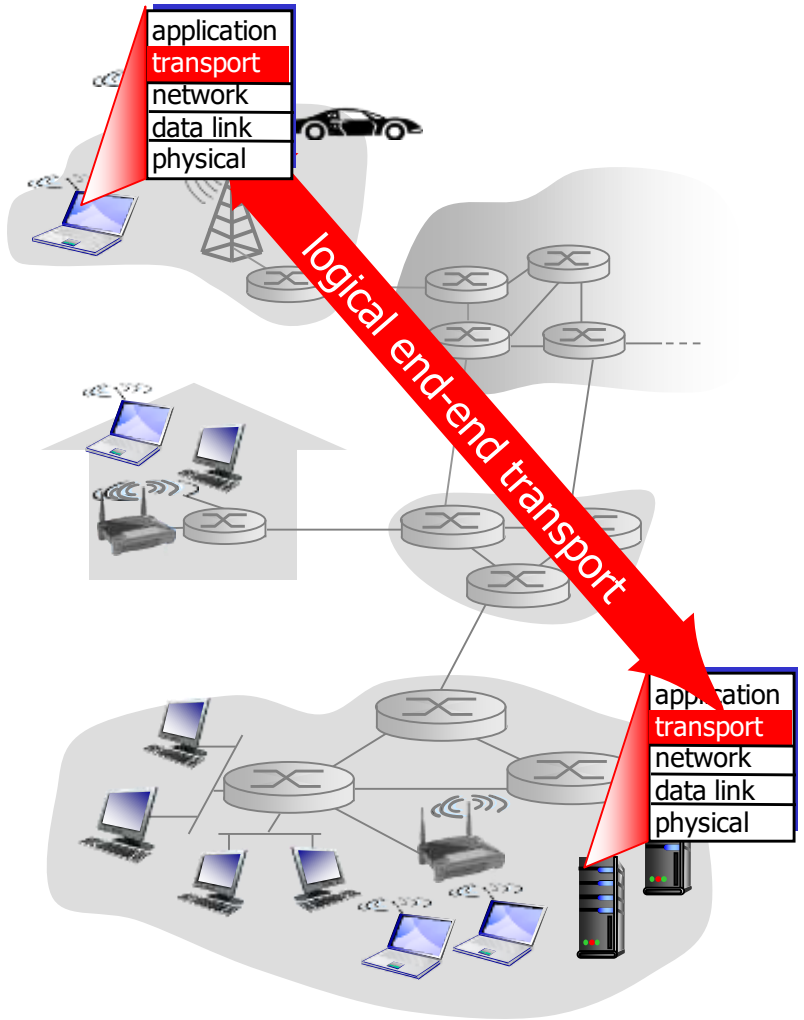
Outline

- Introduction and Transport Layer Services
- Multiplexing and Demultiplexing
- Connection less transport (UDP)
- Principles of Reliable Data Transfer
- Connection oriented transport (TCP)
- Congestion Control

Introduction and Transport Layer Services

Transport Layer Services and Protocols

- It provides logical communication between application processes running on different hosts.
- A transport protocols run in end systems.
- Sender side: It breaks application messages into segments, then passes to network layer.
- Receiver side: It reassembles segments into messages, then passes to application layer.
- Example: TCP and UDP



Transport Layer Services and Protocols – Example

West Side

Applications Messages = Letters in envelops

Network Layer Protocol = postal service
(including mail persons)

Hosts(End Systems) = Houses

Processes = Cousins

Transport Layer Protocol = Ann & Bill

East Side

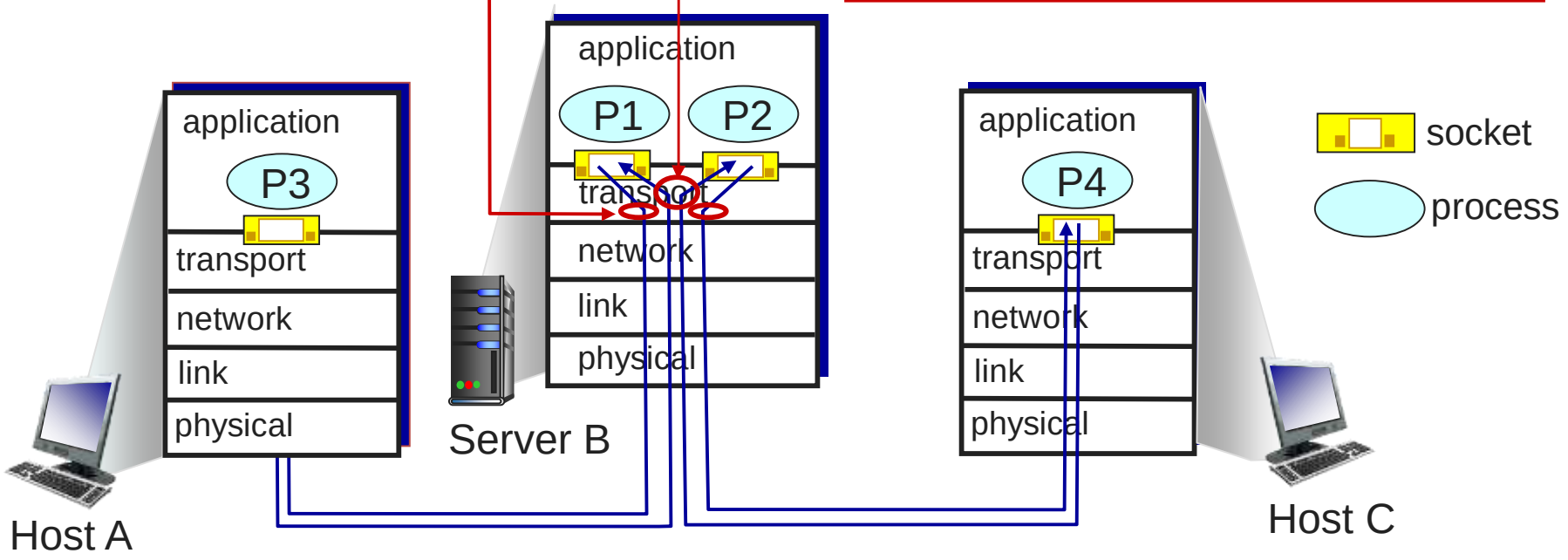
Multiplexing / Demultiplexing

Multiplexing at sender:

To handle data from multiple sockets, add transport header (later used for demultiplexing)

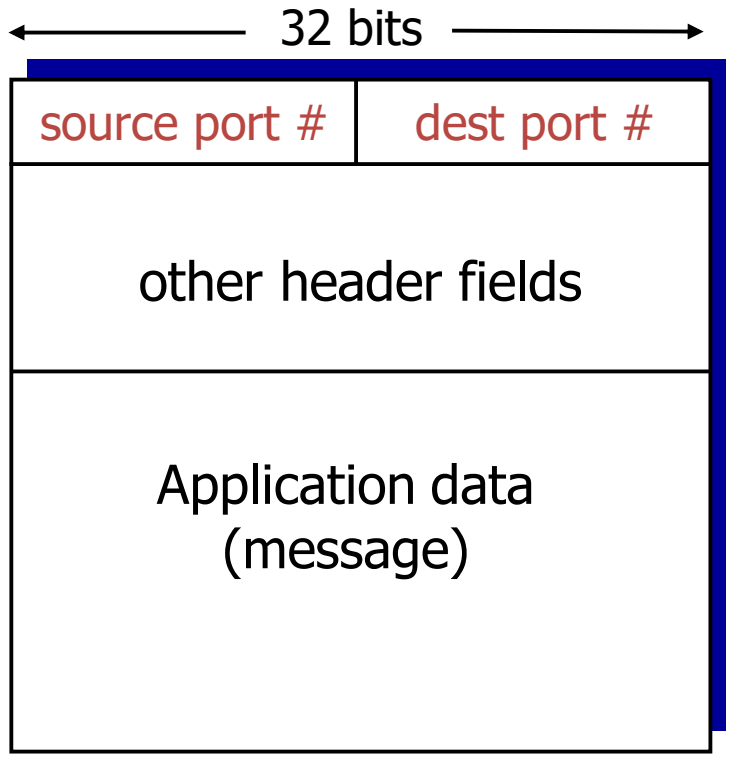
Demultiplexing at receiver:

Use header information to deliver received segments to correct socket



How de-mulptiplexing works?

- A host PC receives IP datagrams.
- Each datagram has source IP address and destination IP address.
- Each datagram carries one transport-layer segment.
- Each segment has source and destination port number.
- A host PC uses IP addresses & port numbers to direct segment to appropriate socket.

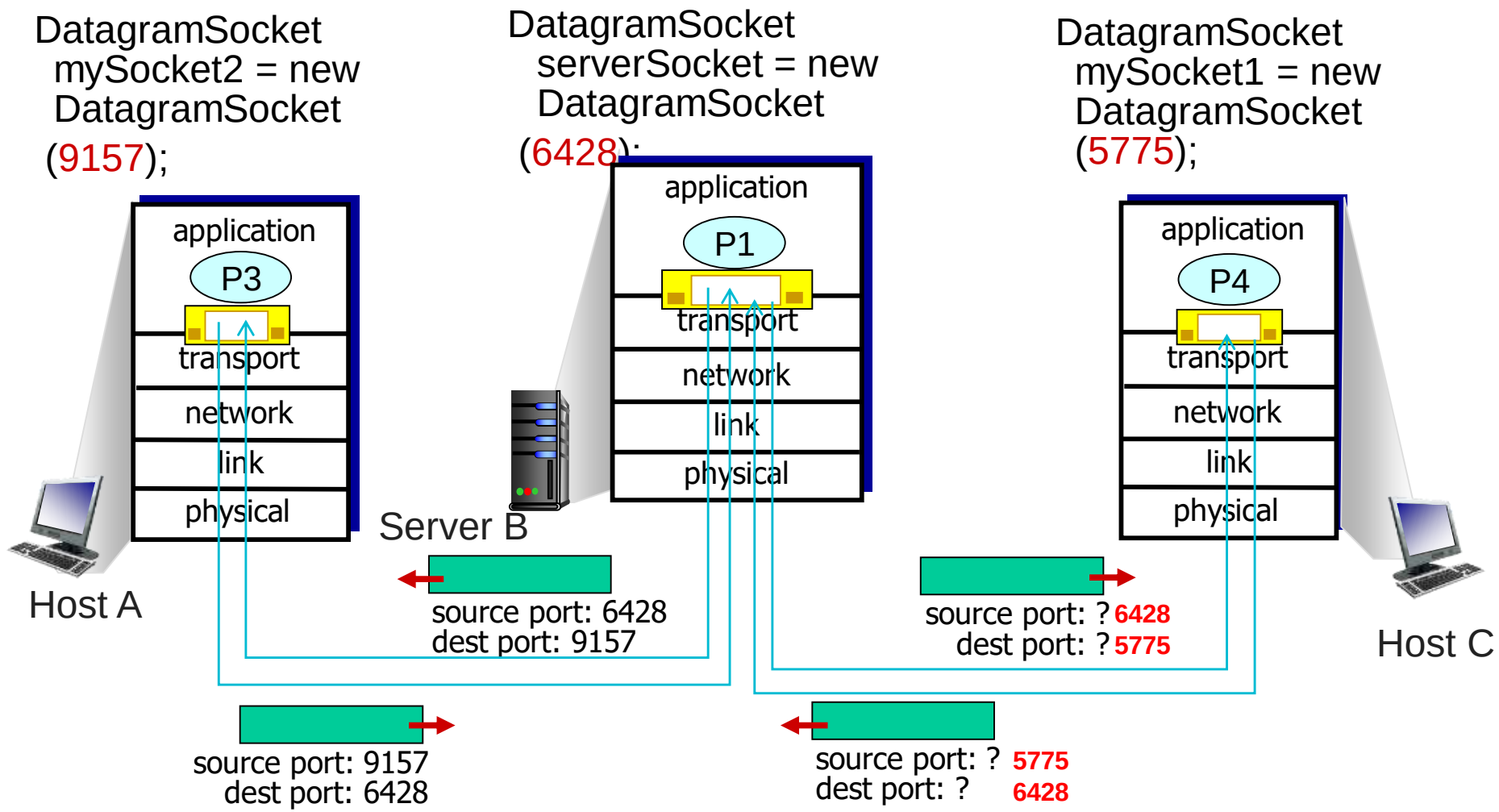


TCP/UDP segment format

Connectionless demultiplexing

- Create socket with host-local port #:
  - DatagramSocket mySocket1 = new DatagramSocket(12534);
- After creating datagram, must specify:
  - destination IP address
  - destination port #
- When host receives UDP segment:
  - It checks destination port # in segment and directs UDP segment to socket with that port #.
- IP datagrams with same destination port #, but different source IP addresses and/or source port numbers will be directed to same socket at destination.

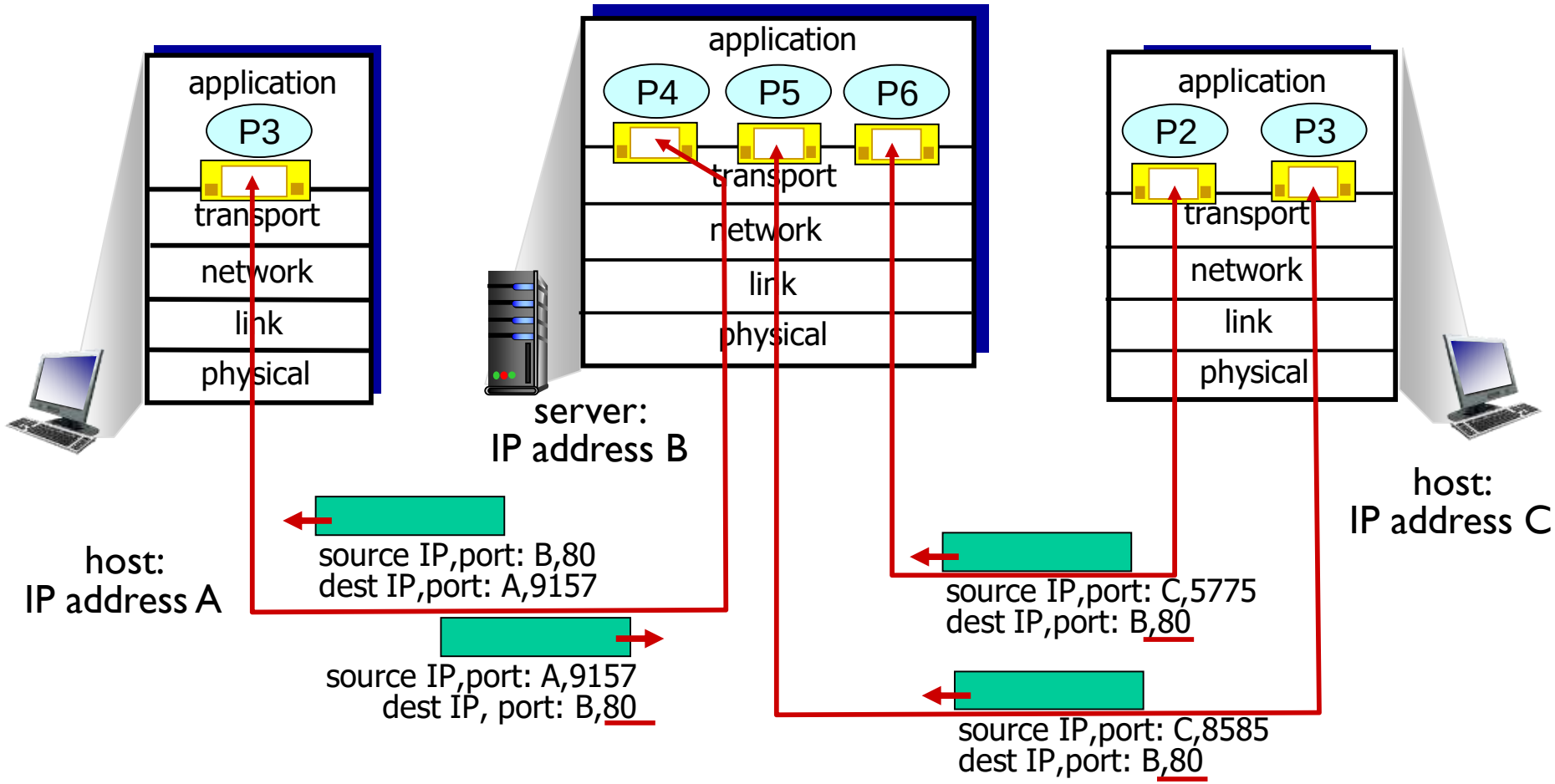
Example: Connectionless



Connection-oriented demultiplexing

- TCP socket identified by 4-tuple:
  - Source IP address
  - Source port number
  - Destination IP address
  - Destination port number
- When TCP segment arrives, receiver uses all four values to direct(demultiplex) segment to appropriate socket.
- Server host may support many simultaneous TCP sockets:
  - each socket identified by its own 4-tuple.
- Web servers have different sockets for each connecting client.
- Persistent HTTP will have same socket per each request.
- Non-persistent HTTP will have different socket for each request.

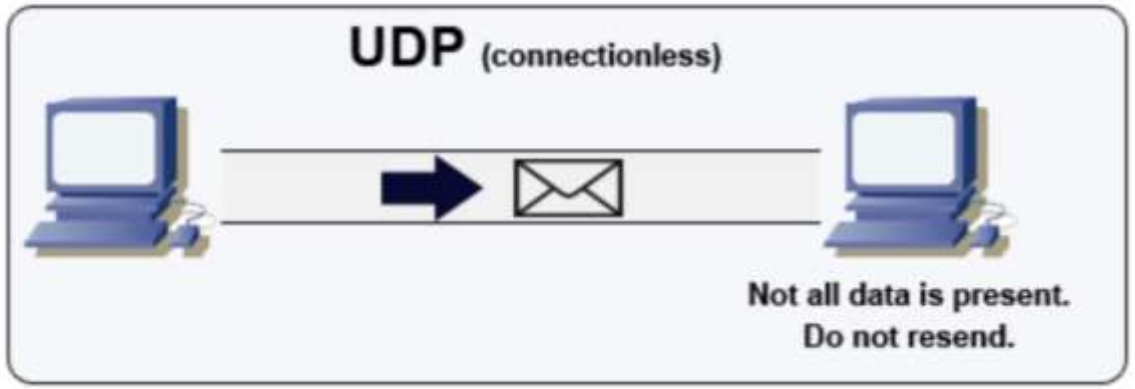
Example: Connection-oriented



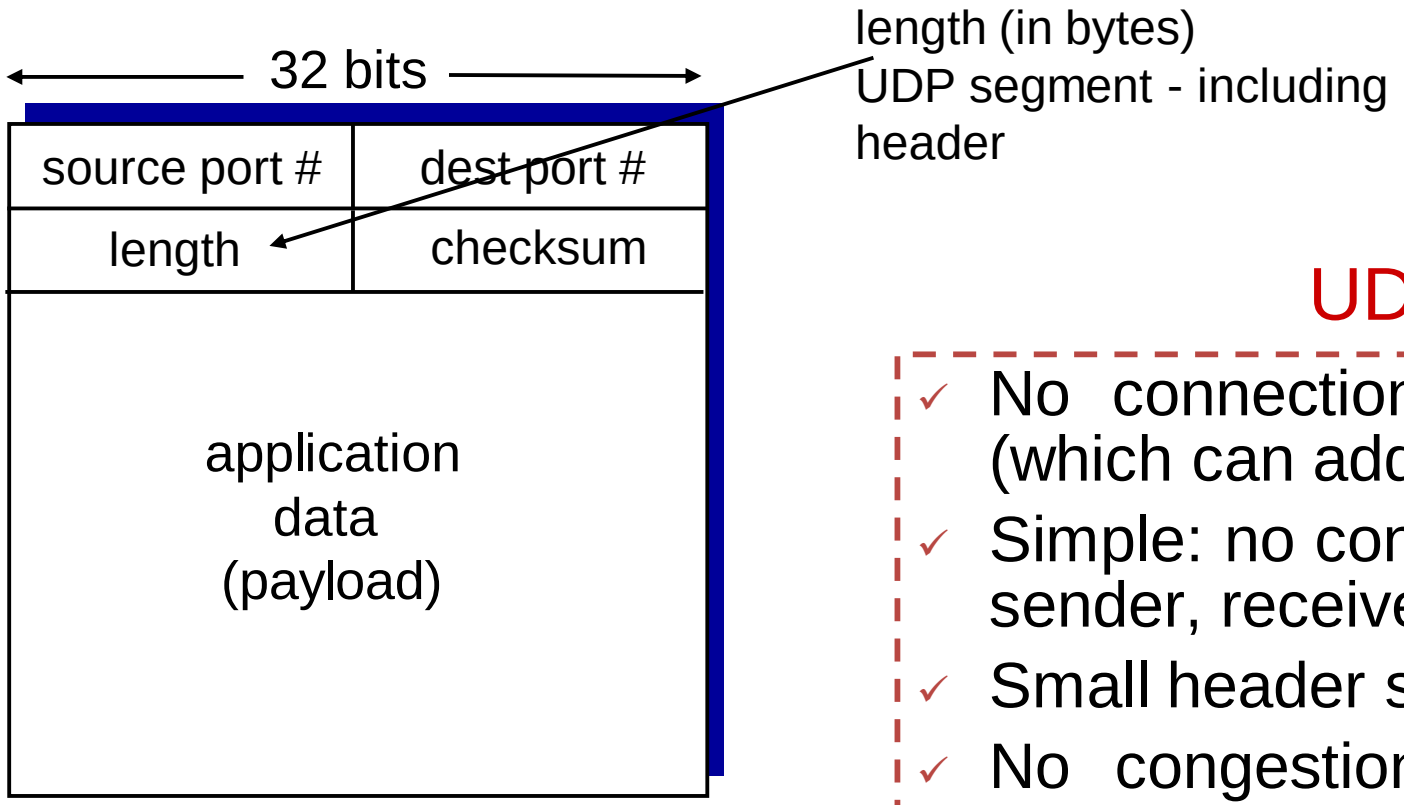
three segments, all destined to IP address: B,
dest port: 80 are demultiplexed to *different* sockets

Connectionless Transport - UDP

- User Datagram Protocol – Transport layer protocol.
- It takes message from application process, attach source & destination port# for Multiplexing/Demultiplexing then pass to network layer.
- Making best effort to deliver the segment.
- No handshaking between sender and receiver in transport layer.
- So, that UDP is connectionless protocol. E.g. DNS, SNMP, RIP
- UDP used in streaming multimedia apps (loss tolerant, rate sensitive).



UDP Segment - Header



UDP segment format

UDP

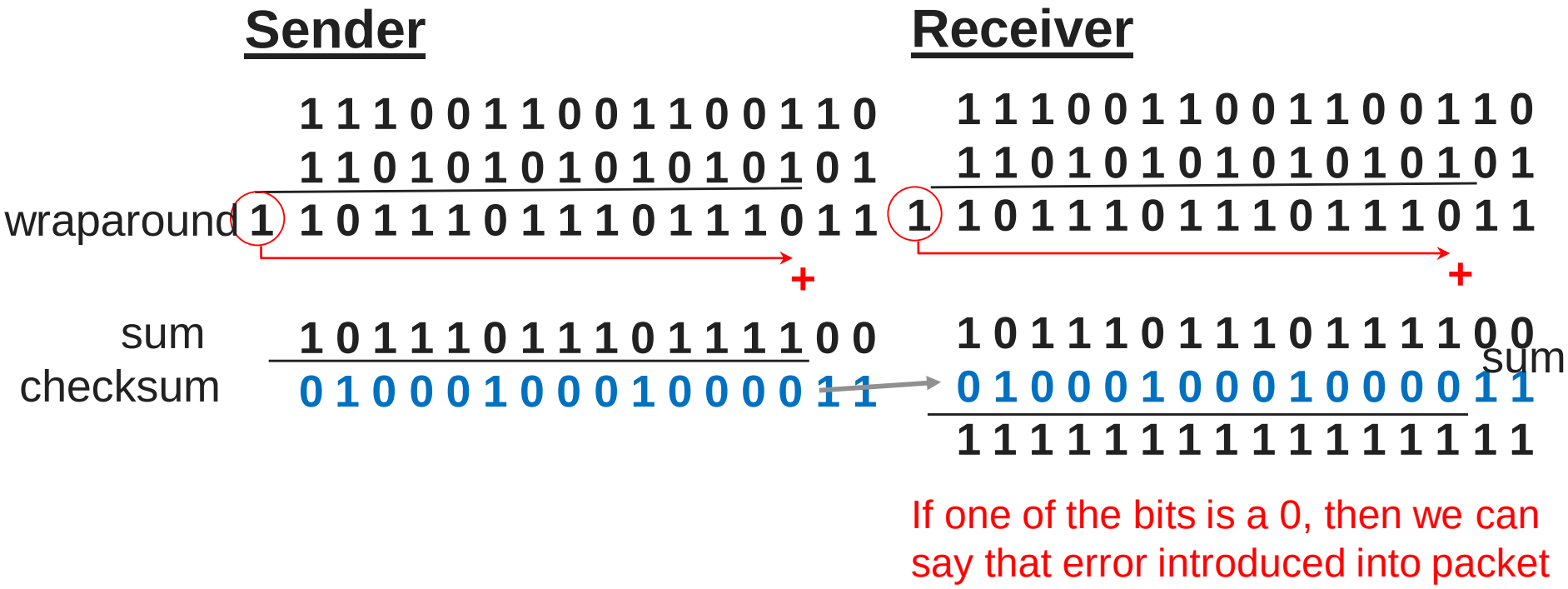
- ✓ No connection establishment (which can add delay)
- ✓ Simple: no connection state at sender, receiver
- ✓ Small header size
- ✓ No congestion control: UDP can blast away as fast as desired

UDP - Checksum

- Checksum is used to detect errors in transmitted segment.
- **Sender:**
  - Treat segment contents, including header fields, as sequence of 16-bit integers.
  - Checksum: addition (one'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

Checksum - Example

- Add two 16-bit integers word



Note *:* when adding numbers, a carryout from the most significant bit needs to be added to the result

Checksum - Practice

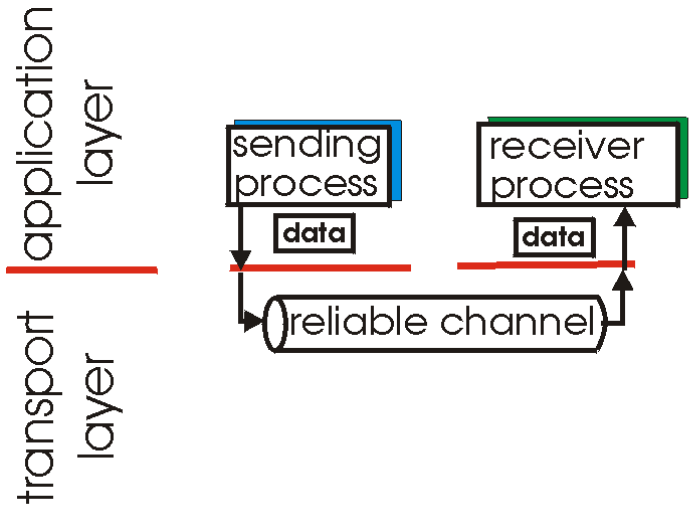
1. 10011001111000100010010010000100 (Use 8 bit word)
2. 1001001110010011100110000 1001101 (Use 16 bit word)

- Checksum Value:

1. 11011010
2. 1101010000011110

Principles of Reliable data Transfer

Principles of reliable data transfer



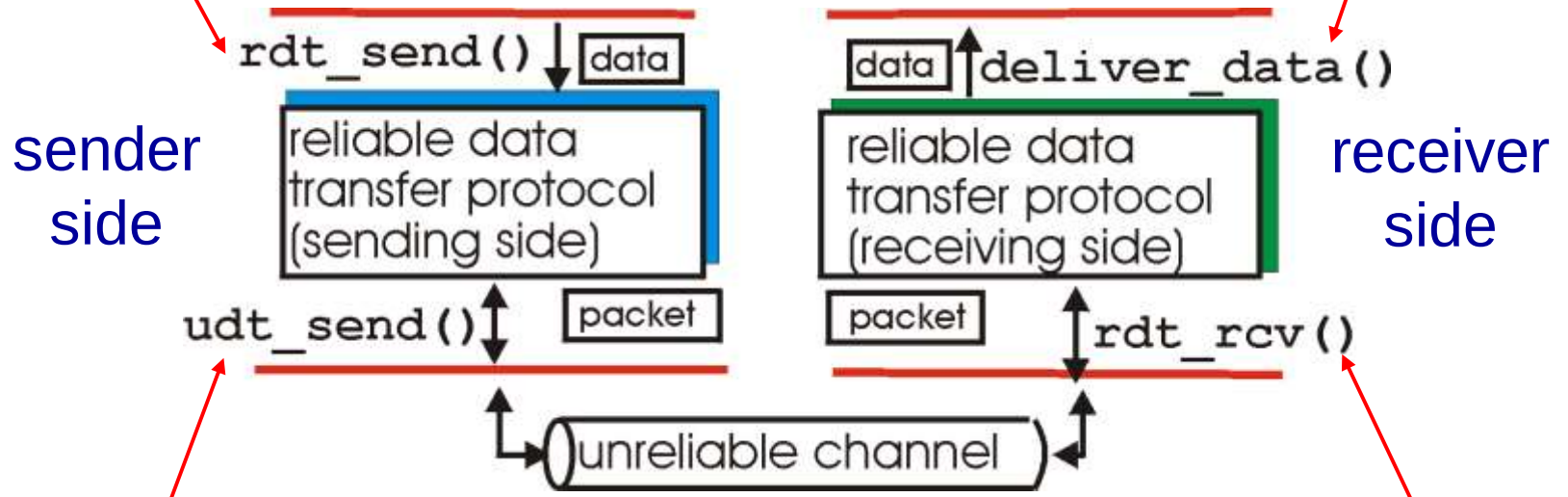
(a) provided service

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

Reliable Data Transfer(rdt)

rdt_send(): called from above layer, (e.g., by application layer). Passed data to receiver

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

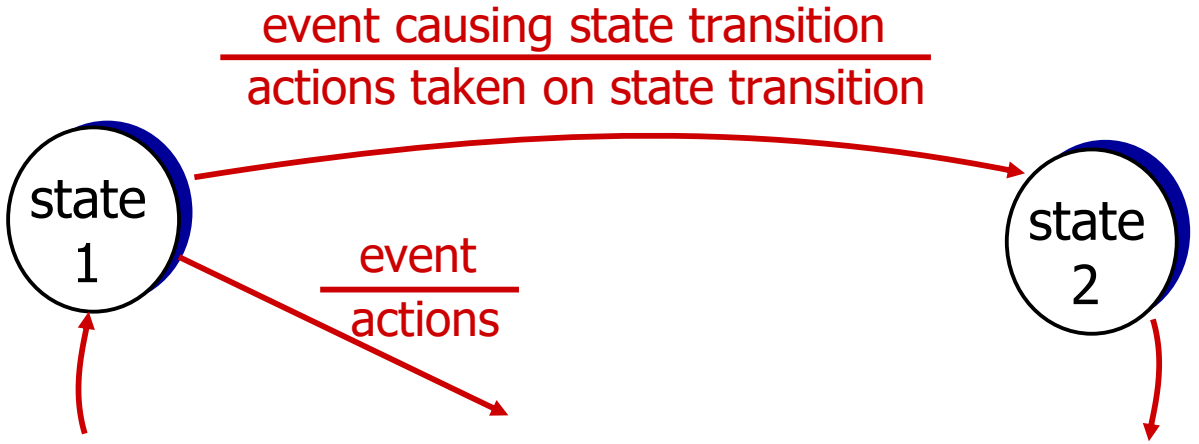


udt_send(): called by rdt, to transfer packet over unreliable channel to receiver

rdt_rcv(): called when packet arrives on receiver side of channel

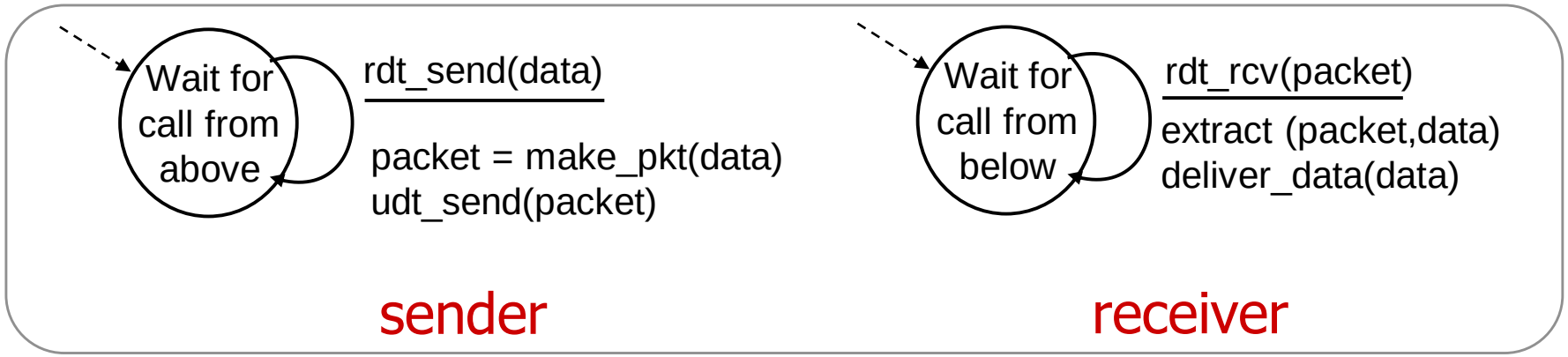
Reliable Data Transfer – Cont…

- Consider only unidirectional data transfer.
  - A control information will flow on both directions.
- Use finite state machines (FSM) to specify sender and receiver.
- When in this "state" next state uniquely determined by next event.
- Arrow indicates the transition of protocol from one state to another.



rdt 1.0

- Reliable transfer over a reliable channel
- Underlying channel perfectly reliable channel
  - no bit errors
  - no loss of packets
- Separate FSMs for sender & receiver:
  - Sender sends data into underlying channel
  - Receiver reads data from underlying channel

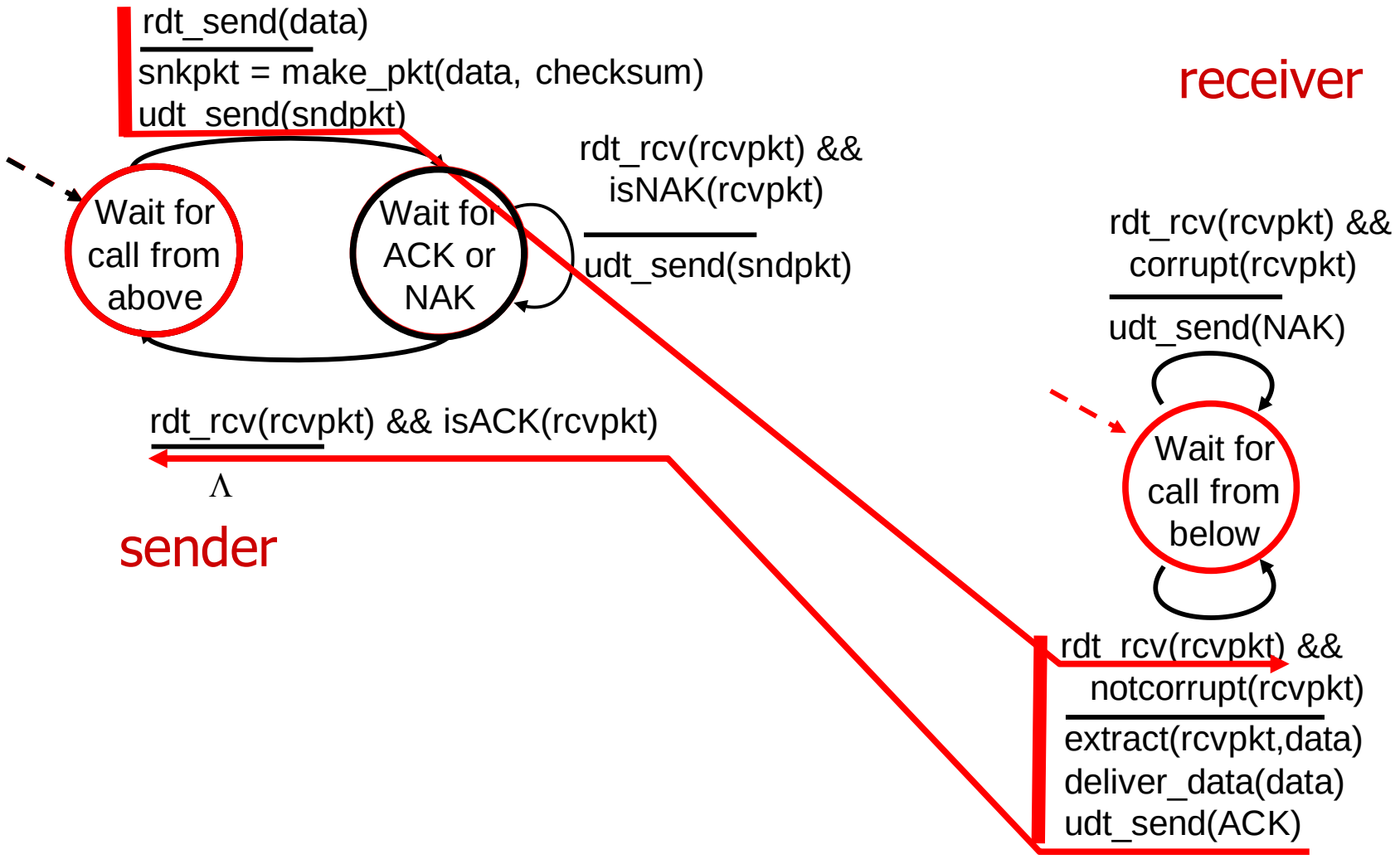


rdt 2.0 – Stop & Wait Protocol

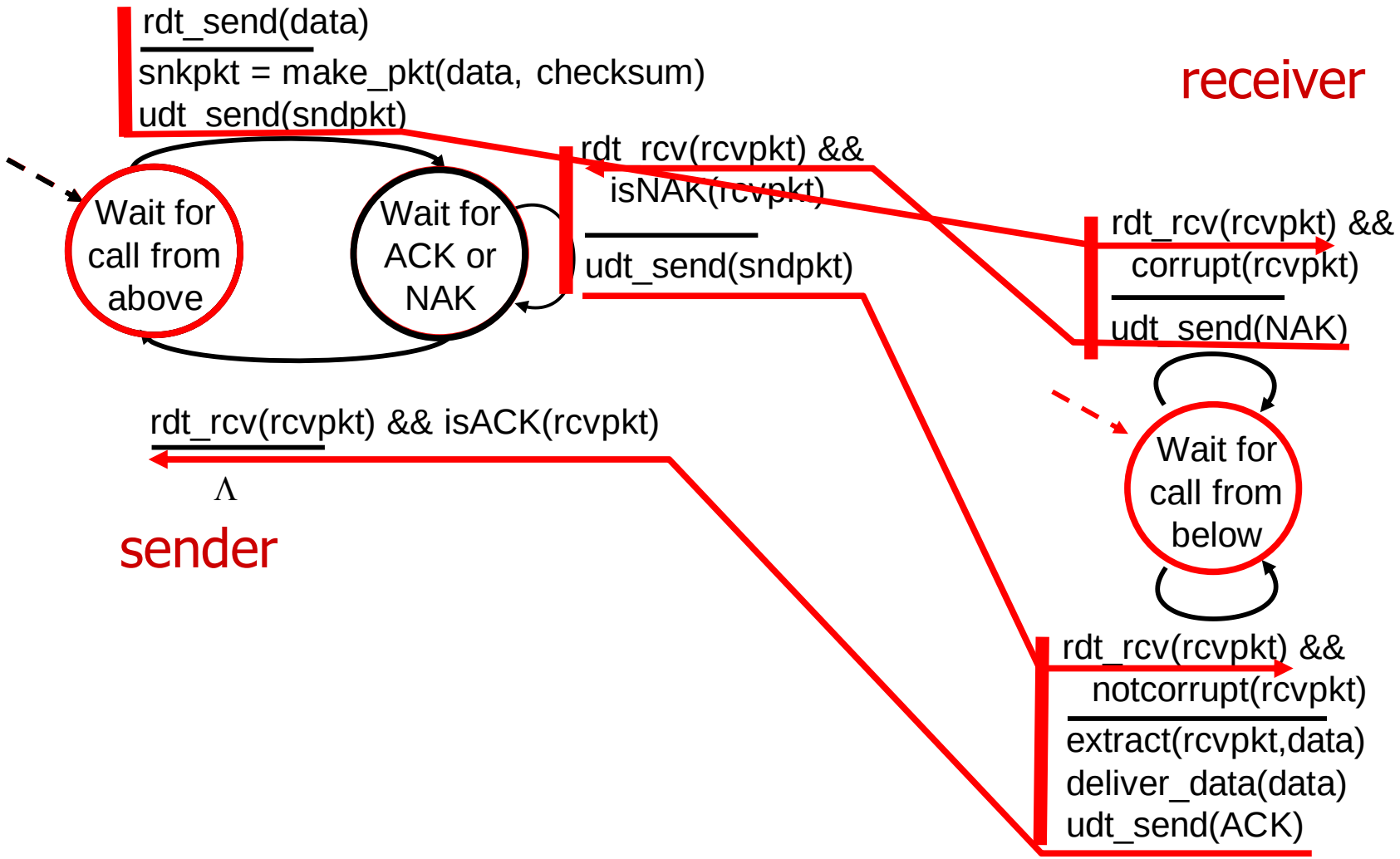
rdt 2.0 – channel with bit errors

- Underlying channel may flip bits in packet.
  - checksum to detect bit errors.
- To recover from errors:
  - **acknowledgements** (ACKs): receiver explicitly tells sender that packet received OK.
  - **negative acknowledgements** (NAKs): receiver explicitly tells sender that packet had errors.
  - sender **retransmits** packet on receipt of NAK.
- New mechanisms in rdt 2.0 (beyond rdt 1.0):
  - **Error detection**
  - **Feedback**: control messages(ACK,NAK) from receiver to sender
  - **Retransmission**: Error in received packet, retransmitted by the sender
- It is known as stop-and-wait protocol.

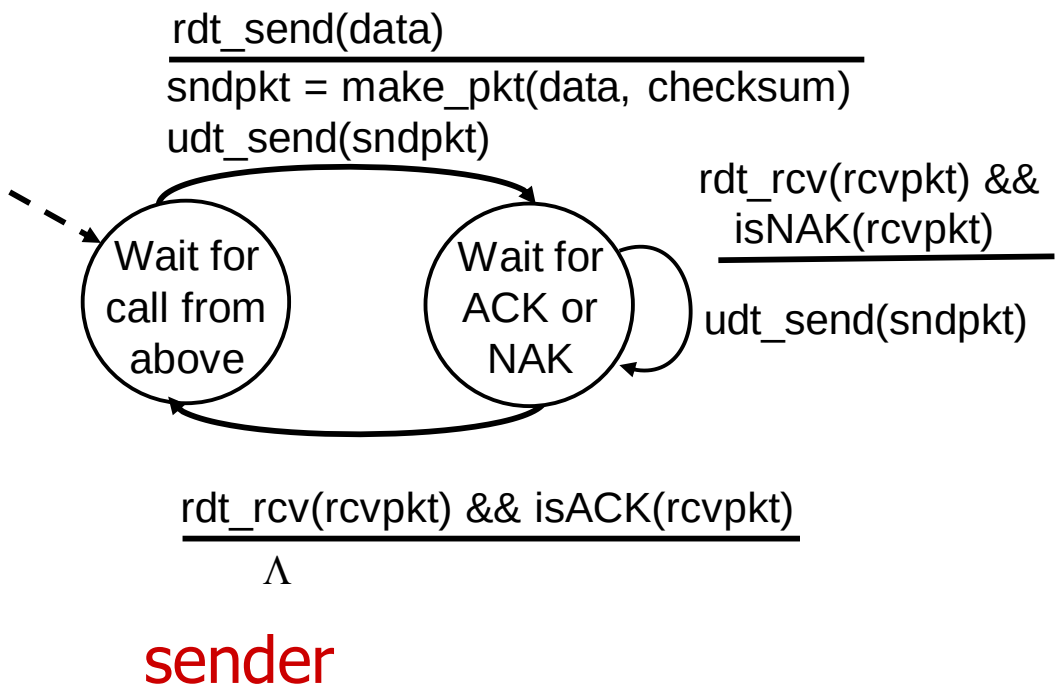
rdt 2.0 – with no error

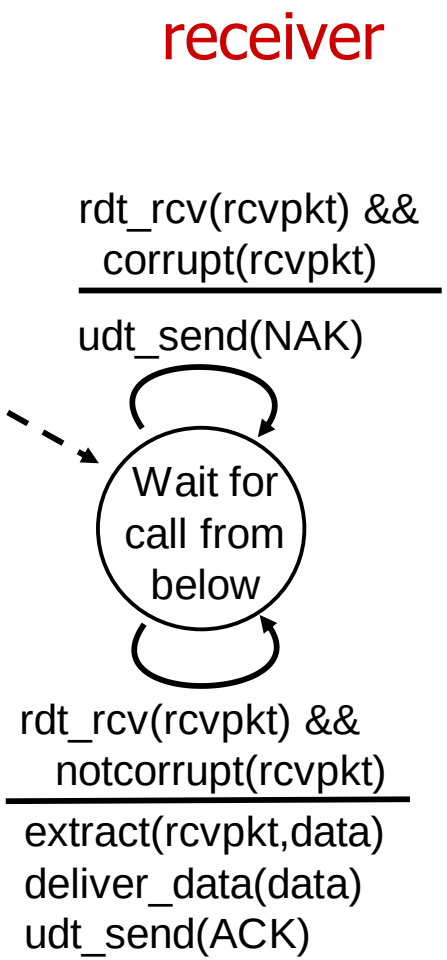


rdt 2.0 – with error



rdt 2.0 – FSM Specification





rdt 2.1 - handled ACKs/NAKs

rdt 2.1 - handled ACKs/NAKs

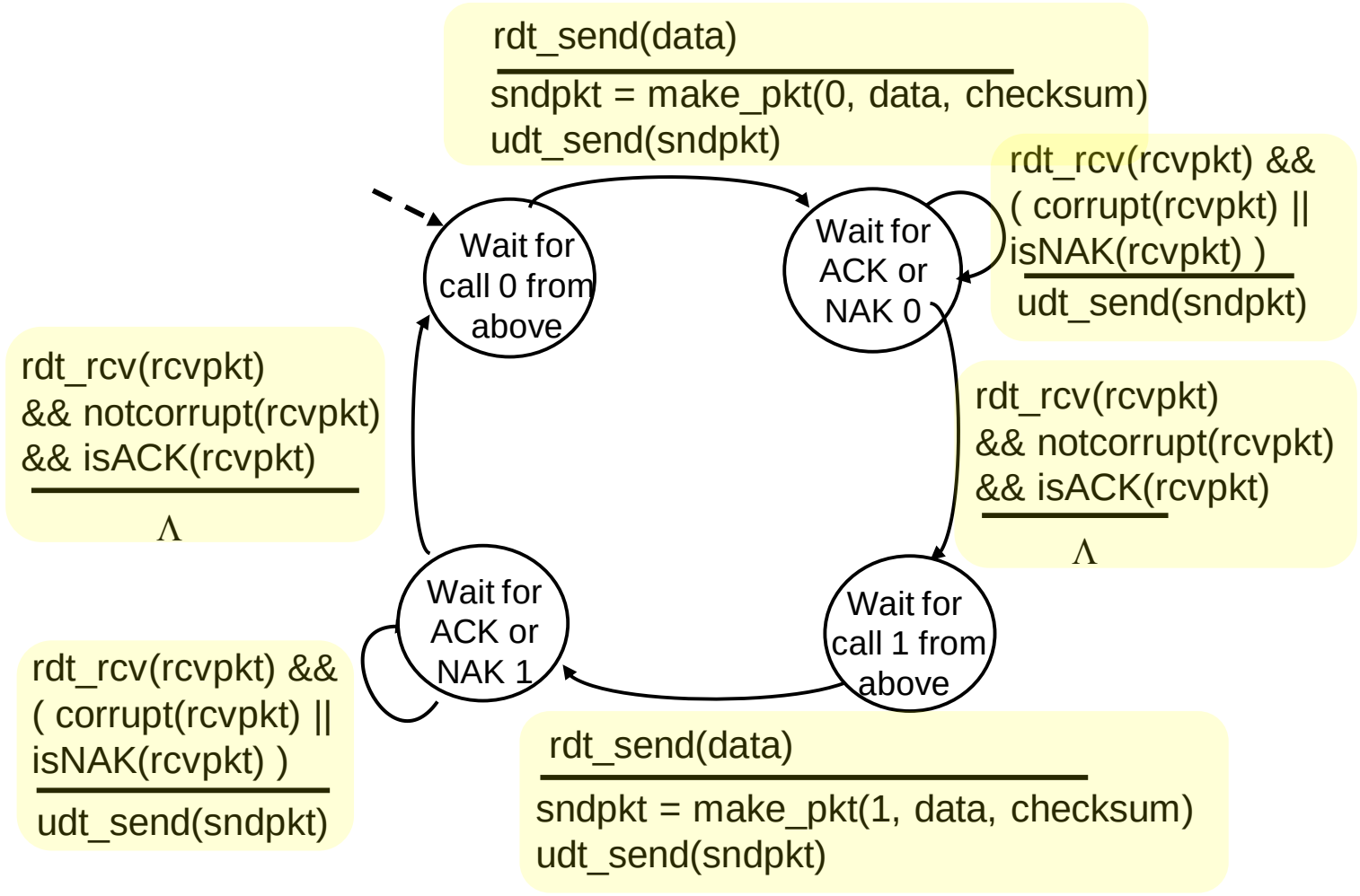
Sender:

- Sequence # added to packet
- Two sequence #'s (0,1) will suffix
- It must check if received ACK/NAK corrupted or not
- It can be twice as many states
  - State must "remember" whether "expected" packet should have sequence # of 0 or 1

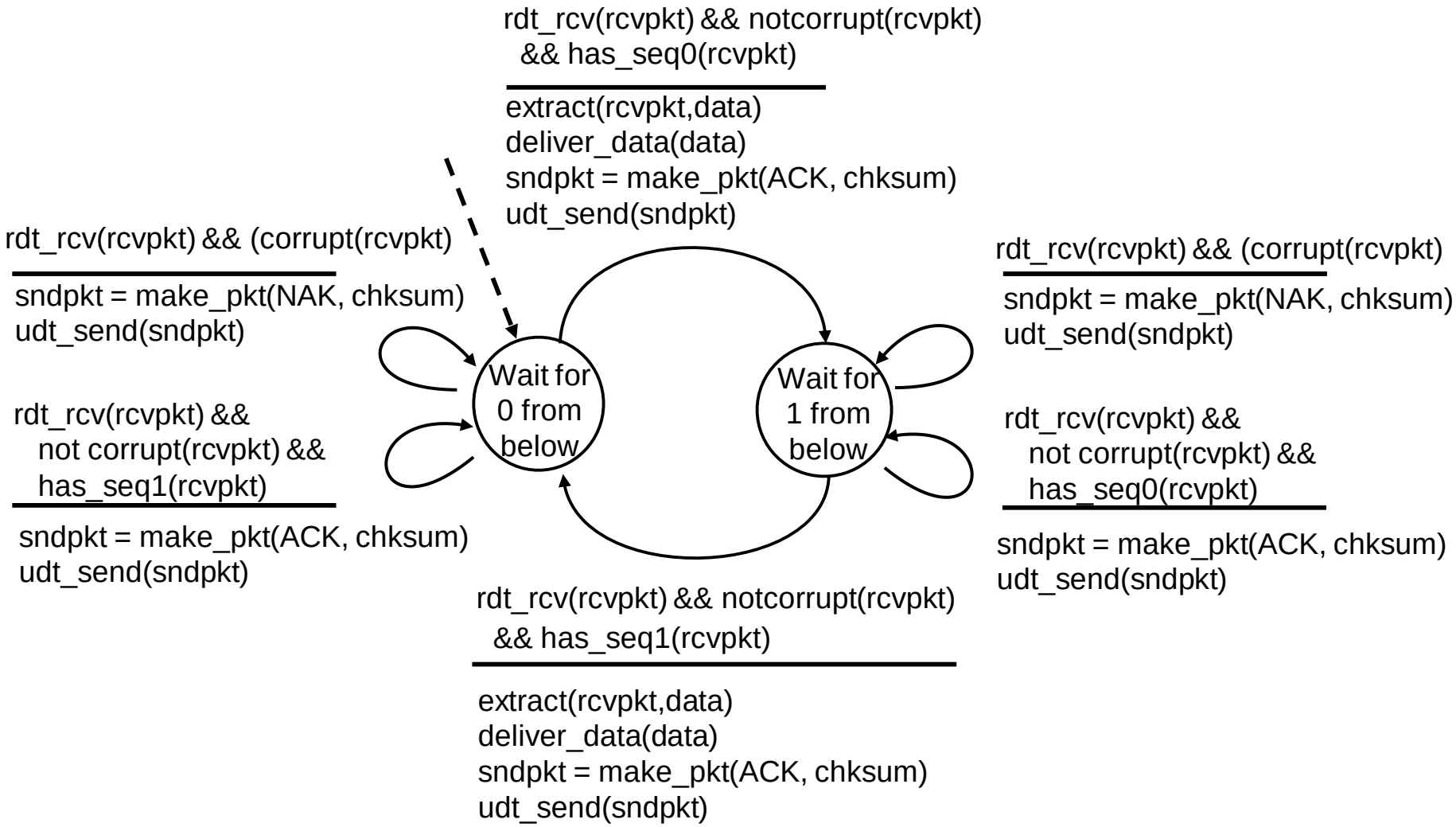
Receiver:

- It must check if received packet is duplicate or not
- State indicates whether 0 or 1 is expected packet sequence #
- Receiver can not know if its last ACK/NAK received OK at sender

rdt 2.1 - Sender, handled ACKs/NAKs



rdt 2.1 - Receiver, handled ACKs/NAKs

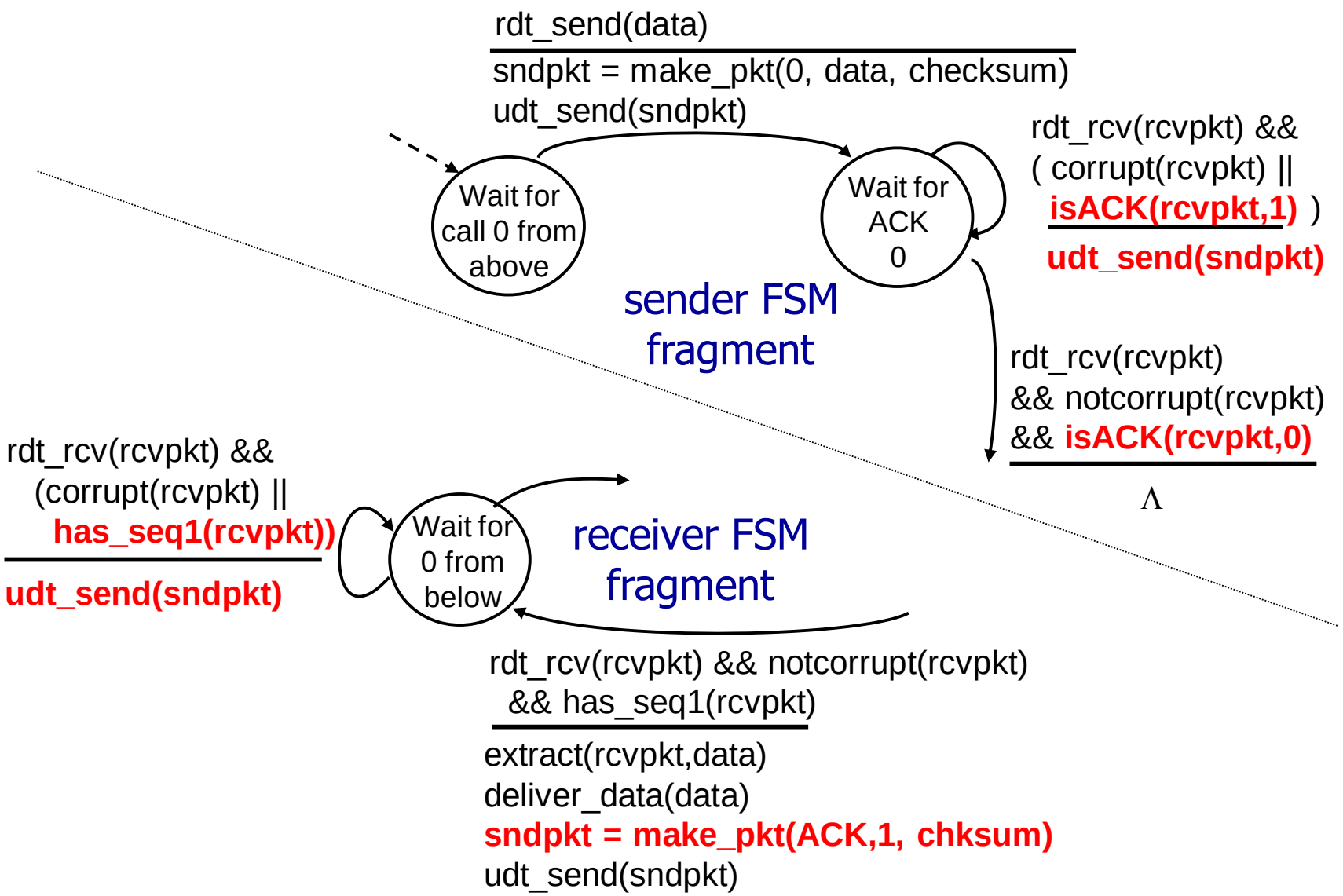


rdt 2.2 & RDT 3.0

rdt 2.2

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

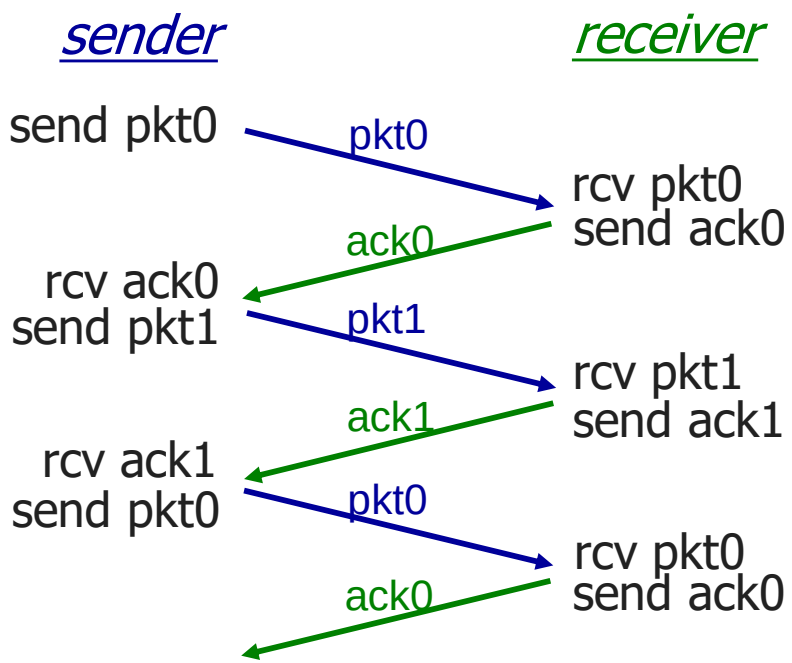
rdt 2.2 – Sender and Receiver



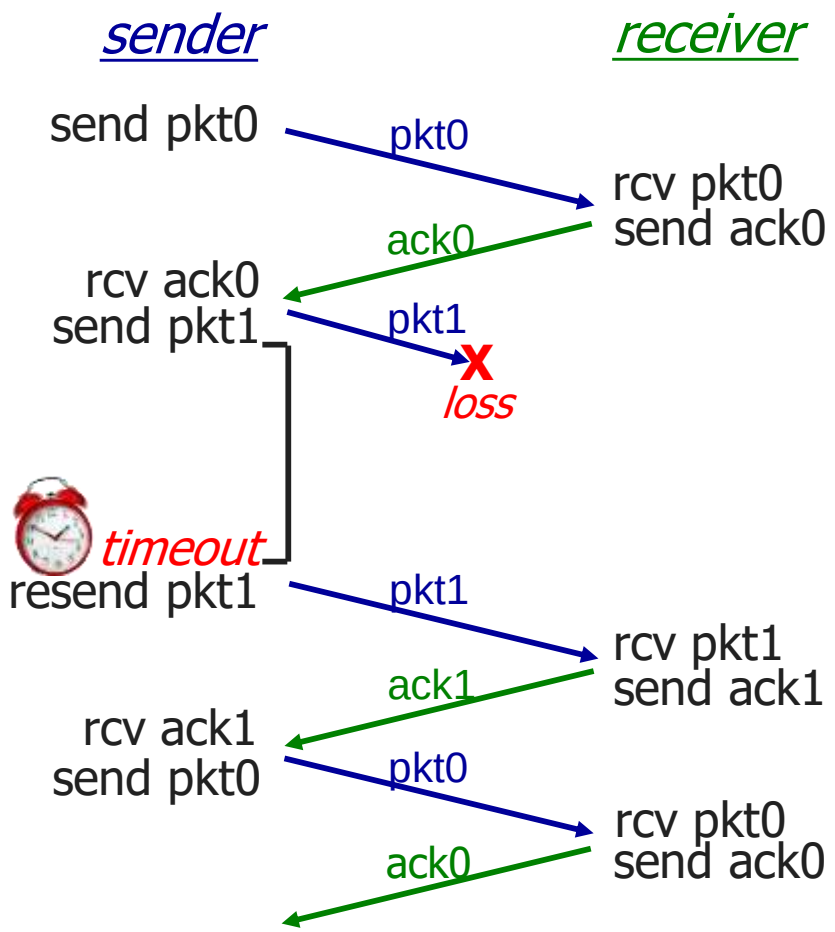
rdt 3.0: channel with error and loss

- Underlying channel can also lose packets (data, ACKs).
  - Even checksum, sequence #, ACKs, retransmissions will not enough help.
- Sender waits “reasonable” amount of time for ACK.
- It retransmits if no ACK received in this time.
- If packet(or ACK) just delayed (not lost):
  - Retransmission will be duplicate, but sequence #’s already handled it.
- Receiver must specify sequence # of packet being ACKed.
- It requires countdown timer.

Rdt 3.0: Alternating-bit protocol

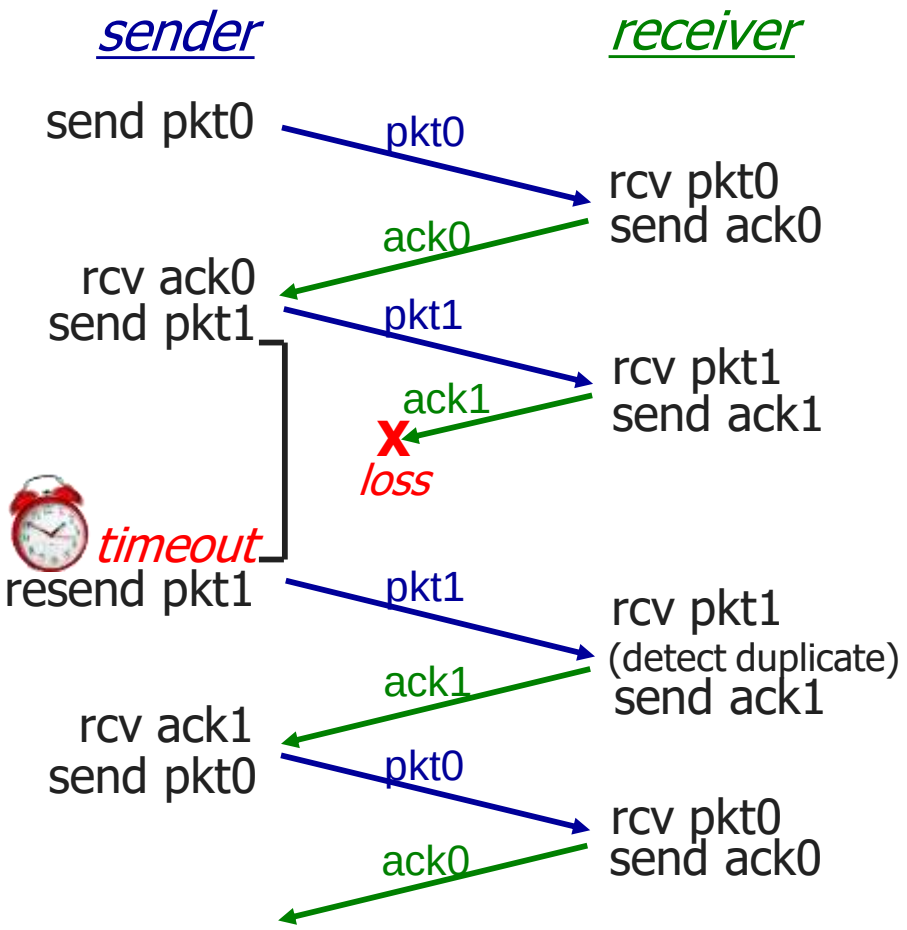


(a) no loss

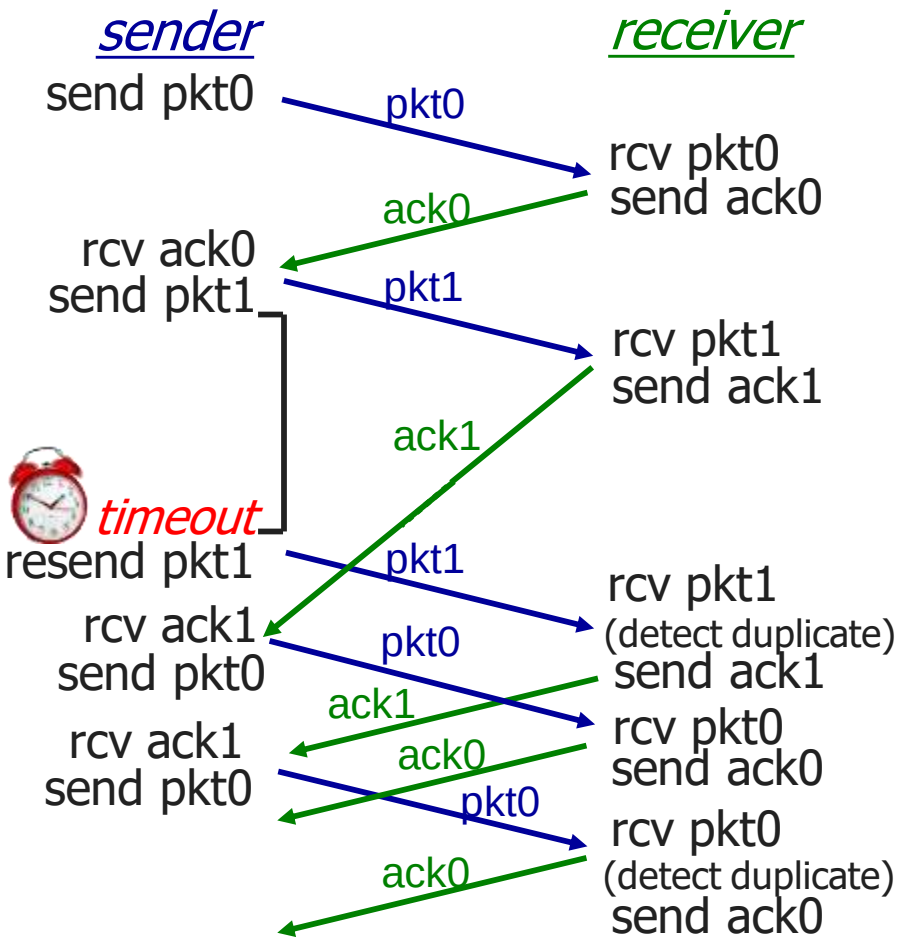


(b) packet loss

Rdt 3.0 – Cont…



(c) ACK loss



(d) premature timeout/ delayed ACK

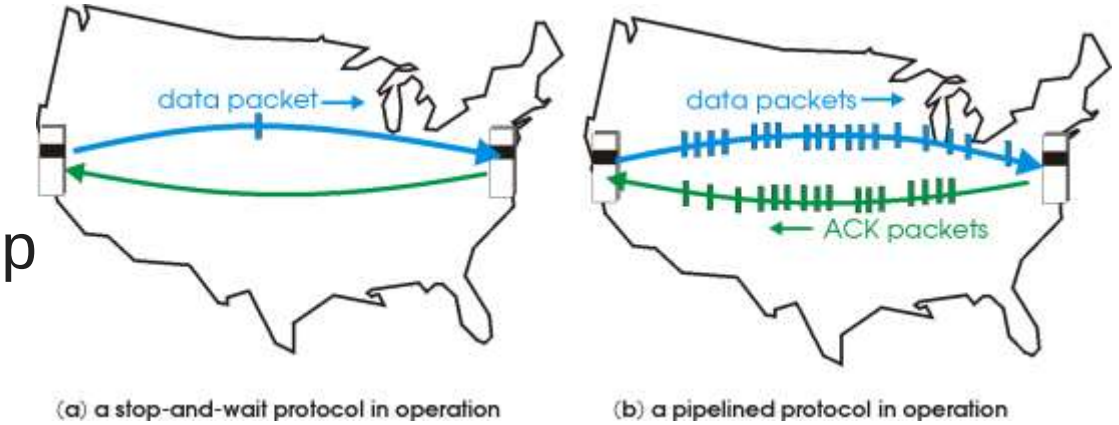
Pipelined Protocol

- Go-back-N Protocol
- Selective Repeat Protocol

Pipelined Protocol

- Its a technique in which multiple requests are written out to a single socket without waiting for the corresponding responses (acknowledged).
  - No. of Packets(request) must be increased.
  - Data or Packet should be buffered at sender and/or receiver.

- Two generic forms of pip
  1. *Go-back-N*
  2. *Selective Repeat*



(a) a stop-and-wait protocol in operation (b) a pipelined protocol in operation

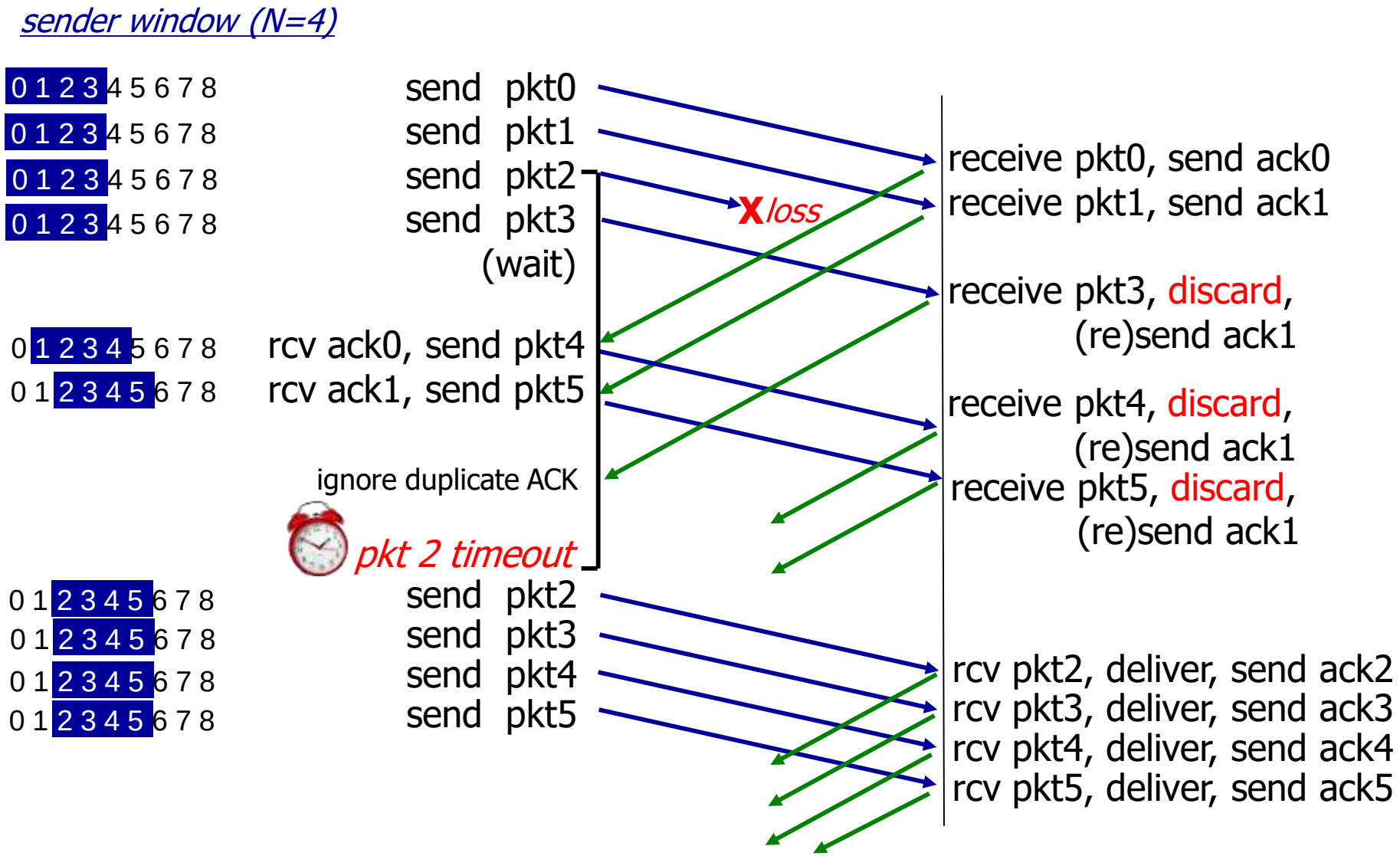
Go-back-N Protocol

- Sender can have up to N unacked packets in pipeline.
- Receiver only sends cumulative ACK. It doesn’t ACK packet if there’s a gap.
- Sender has timer for oldest unacked packet.
- When timer expires, retransmit all unacked packets.
- Sender send a number of frames specified by a window size even without receiving an ACK packet from the receiver.

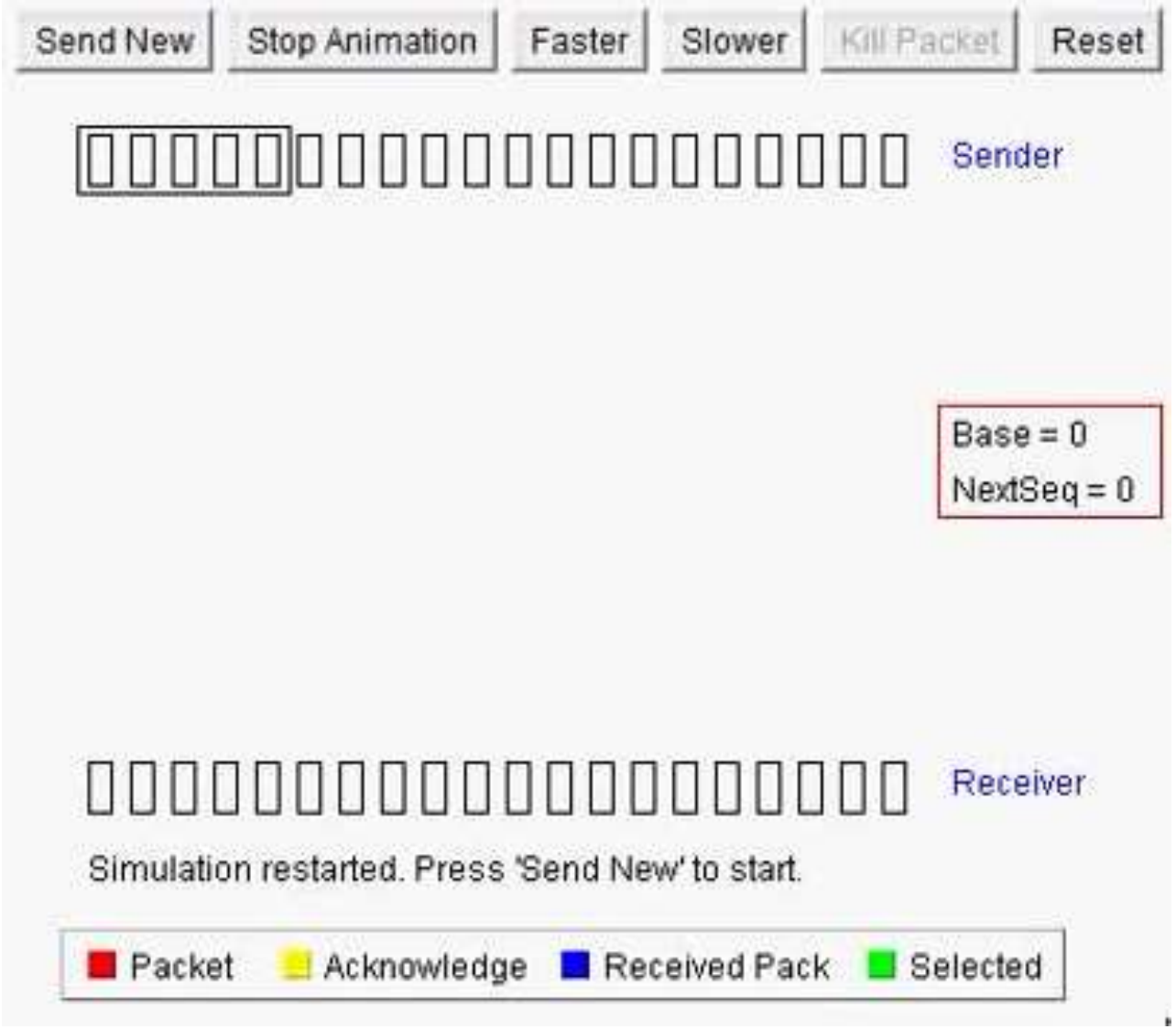
Go-back-N Protocol Cont…

- Receiver keeps track of the sequence no. of the next frame it expects to receive, and sends that number with every ACK it sends.
- Receiver will discard any frame that does not have the exact sequence number it expects.
  - Either a duplicate frame it already ACKed **OR**
  - An out-of-order frame it expects to receive later
- Receiver will resend an ACK for the last correct in-order frame.
- Once the sender has sent all of the frames in its window. It will detect that all of the frames since the first lost frame are outstanding.
- Then go back to the sequence number of the last ACK it received from receiver.
- Go-back-N protocol also known as sliding window protocol.

Go-back-N Protocol works



Go-back-N Visualization



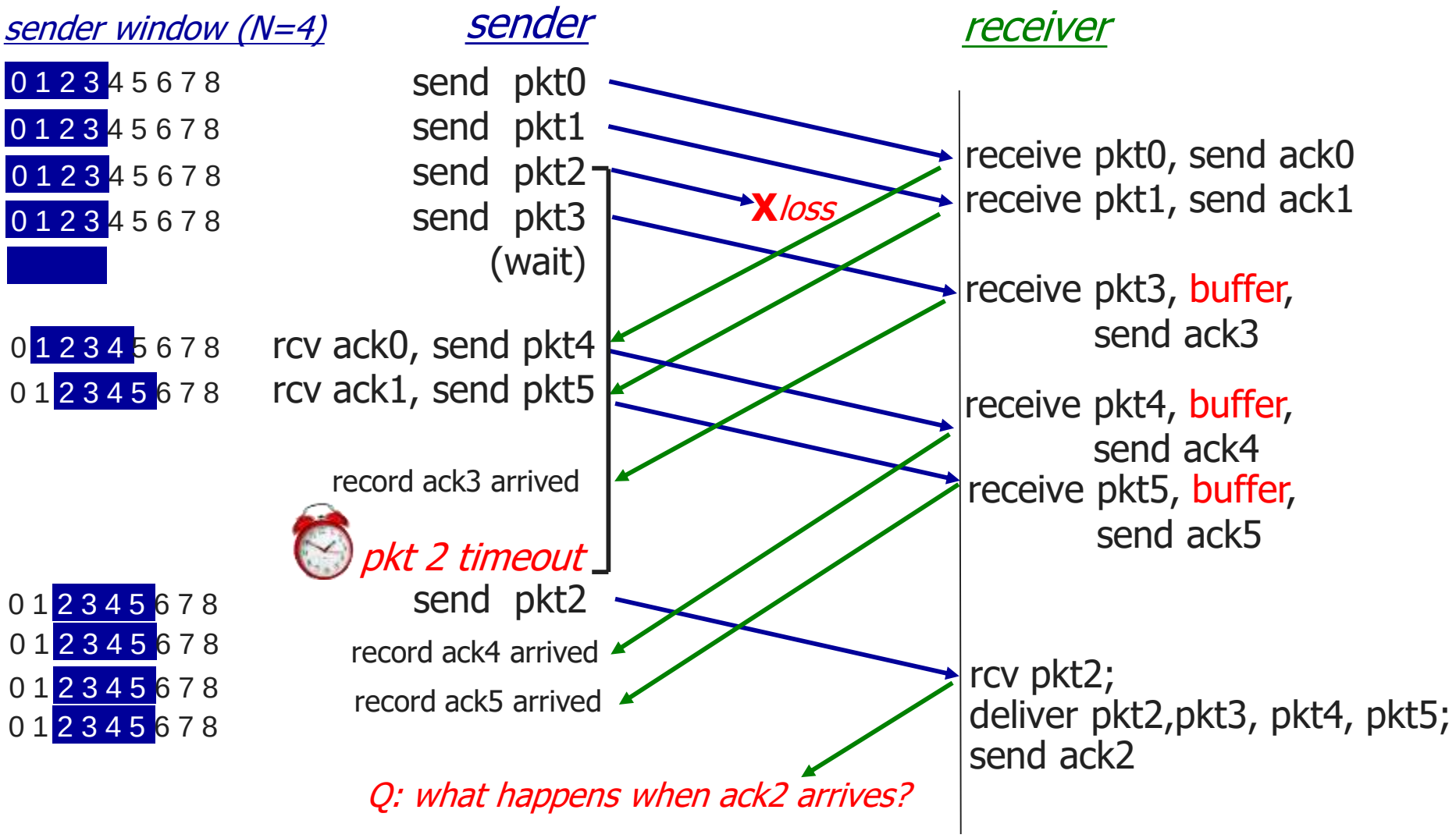
Selective Repeat

- A sender can have up to N unACKed packets in pipeline.
- Receiver sends individual ACK for each packet.
- Sender maintains timer for each unACKed packet.
- When timer expires, retransmit only that unACKed packet.

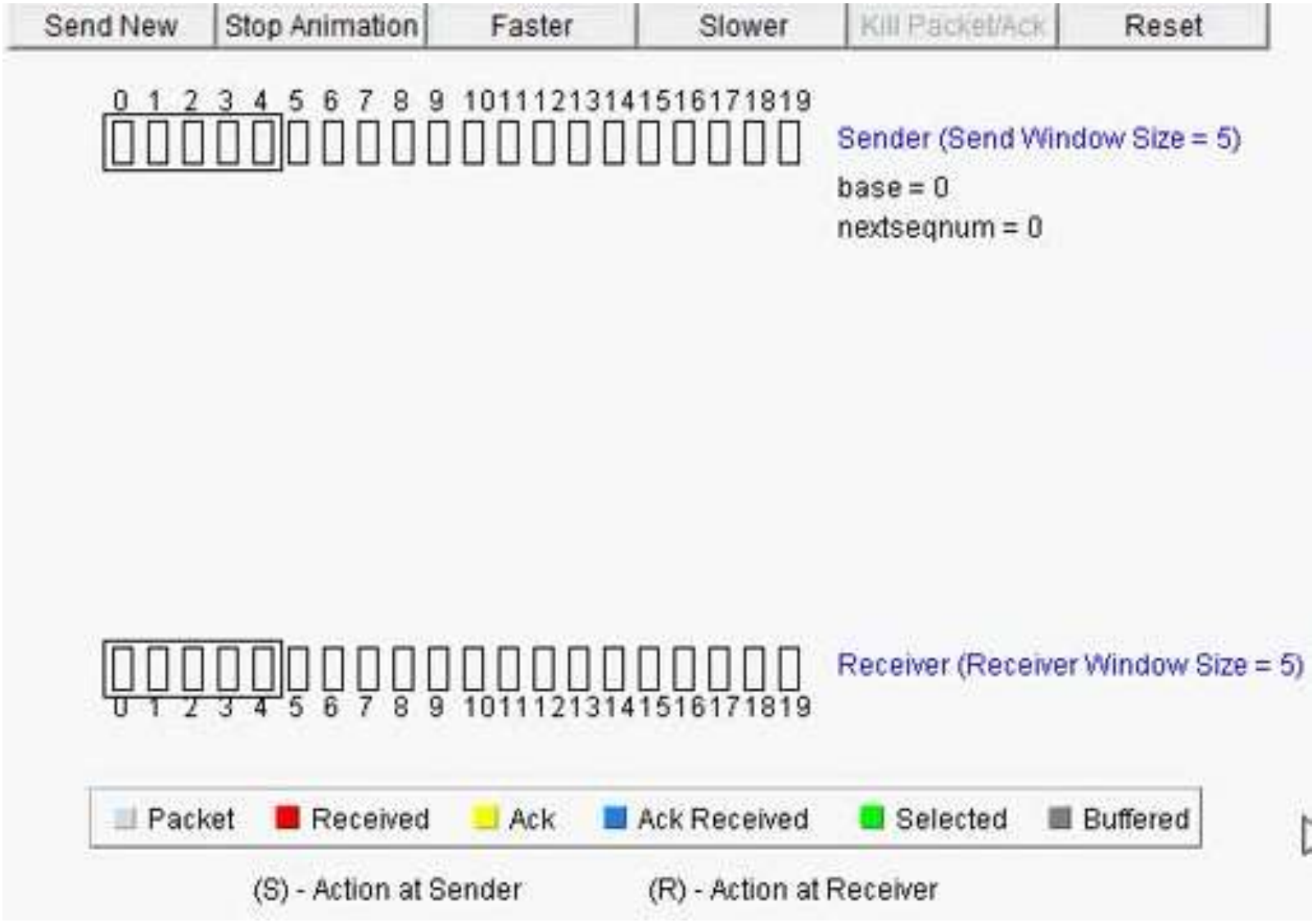
Selective Repeat

- Selective Repeat attempts to retransmit only those packets that are actually lost due to errors.
- Receiver must be able to accept packets out of order.
- Since receiver must release packets to higher layer in order, the receiver must be able to buffer some packets.
- The receiver acknowledges every good packet, packets that are not ACKed before a time-out are assumed lost or in error.
- This approach must be used to be sure that every packet is eventually received.
- An explicit NAK (selective reject) can request retransmission of just one packet.

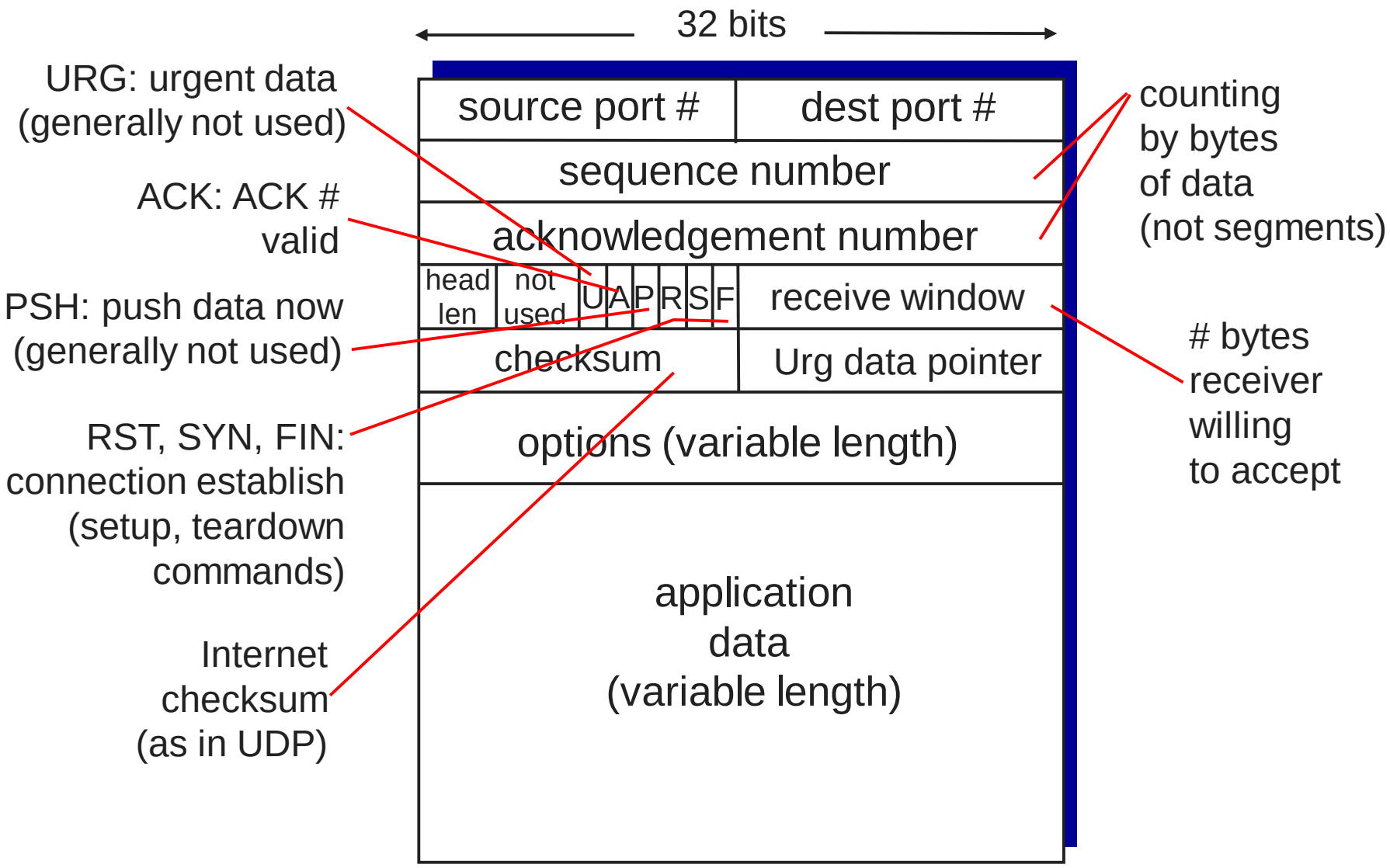
Selective Repeat Works



Selective Repeat Visualization



TCP Segment Structure



TCP Segment – Cont…

- The unit of transmission in TCP is called segments.
- The header includes source and destination port numbers, which are used for multiplexing/demultiplexing data from/to upper-layer applications.
- The 32-bit sequence number field and the 32-bit acknowledgment number field are used by the TCP sender and receiver in implementing a reliable data transfer service.
- The sequence number for a segment is the byte-stream number of the first byte in the segment.
- The acknowledgment number is the sequence number of the next byte a Host is expecting from another Host.

TCP Segment – Cont…

- The 4-bit header length field specifies the length of the TCP header in 32-bit words. The TCP header can be of variable length due to the TCP options field.
- The 16-bit receive window field is used for flow control. It is used to indicate the number of bytes that a receiver is willing to accept.
- The 16-bit checksum field is used for error checking of the header and data.
- Unused 6 bits are reserved for future use and should be sent to zero.
- Urgent Pointer is used in combining with the URG control bit for priority data transfer. This field contains the sequence number of the last byte of urgent data.

TCP Segment – Cont…

- **Data**: The bytes of data being sent in the segment.
- **URG (1 bit)**: indicates that the Urgent pointer field is significant.
- **ACK (1 bit)**: indicates that the Acknowledgment field is significant.
- **PSH (1 bit)**: Push function. Asks to push the buffered data to the receiving application.
- **RST (1 bit)**: Reset the connection.
- **SYN (1 bit)**: Synchronize sequence numbers. Only the first packet sent from each end should have this flag set. Some other flags and fields change meaning based on this flag, and some are only valid for when it is set, and others when it is clear.
- **FIN (1 bit)**: No more data from sender.

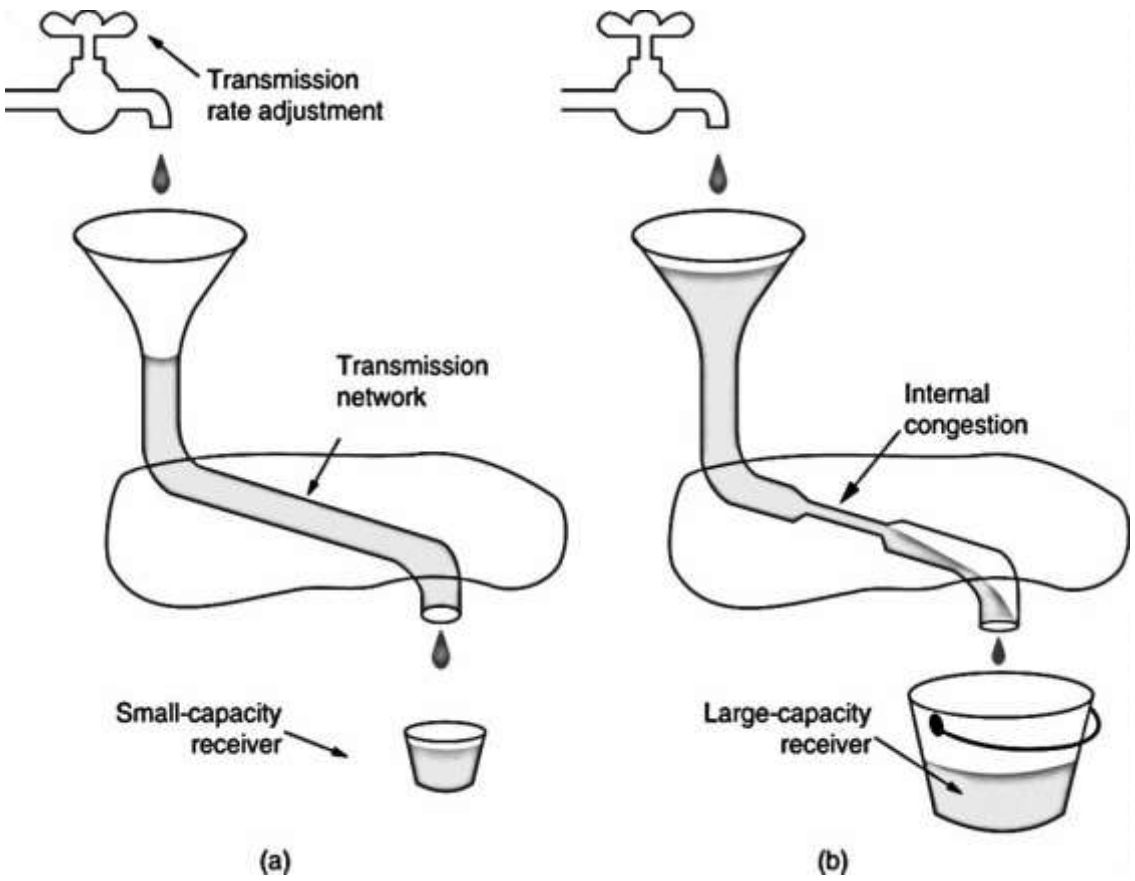
Flow Control

- Flow control is the process of managing the rate of data transmission between two nodes to prevent a fast sender from overwhelming a slow receiver.
- It prevent receiver from becoming overloaded.
- Receiver advertises a window rwnd(receiver window) with each acknowledgement.
- Window:
  - Closed (by sender) when data is sent and ack'd
  - Opened (by receiver) when data is read
- The size of this window can be the performance limit.

Congestion Control

- Too many sources sending too much data too fast for network to handle.
- When a connection is established, a suitable window size has to be chosen.
- The receiver can specify a window based on its buffer size.
- If the sender sticks to this window size, problems will not occur due to buffer overflow at the receiving end, but they may still occur due to internal congestion within the network.

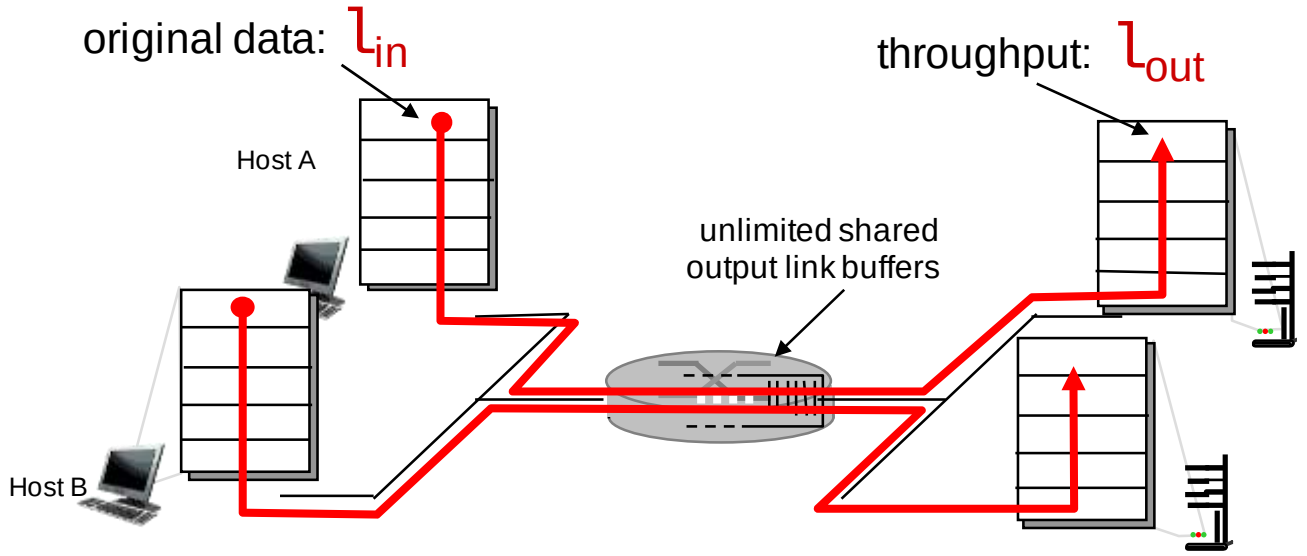
Congestion Control – Cont…



- In Figure(a), we see a thick pipe leading to a small-capacity receiver.
- In Figure(b), the limiting factor is not the bucket capacity, but the internal carrying capacity of the network.
- As long as the sender does not send more water than the bucket can contain, no water will be lost.

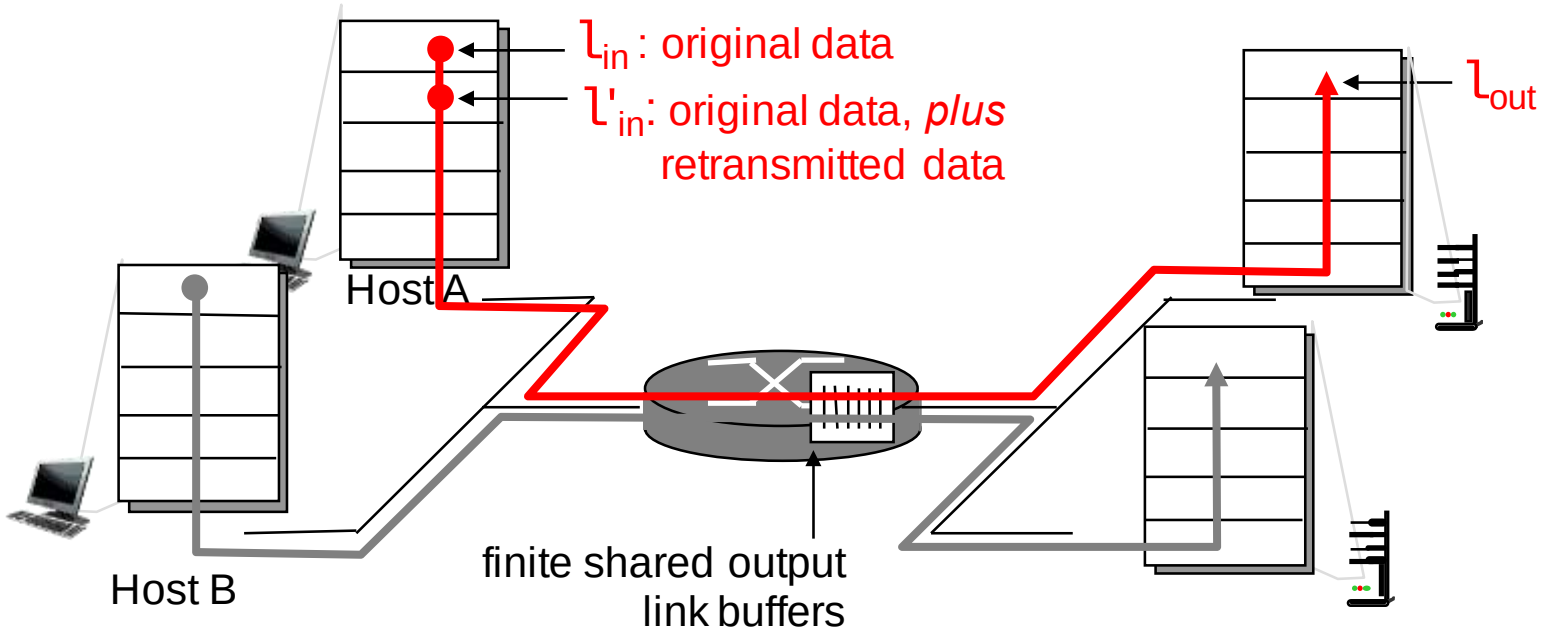
Causes/costs of Congestion: Scenario 1

- Two senders, Two receivers
- One router, Infinite buffers
- No retransmission



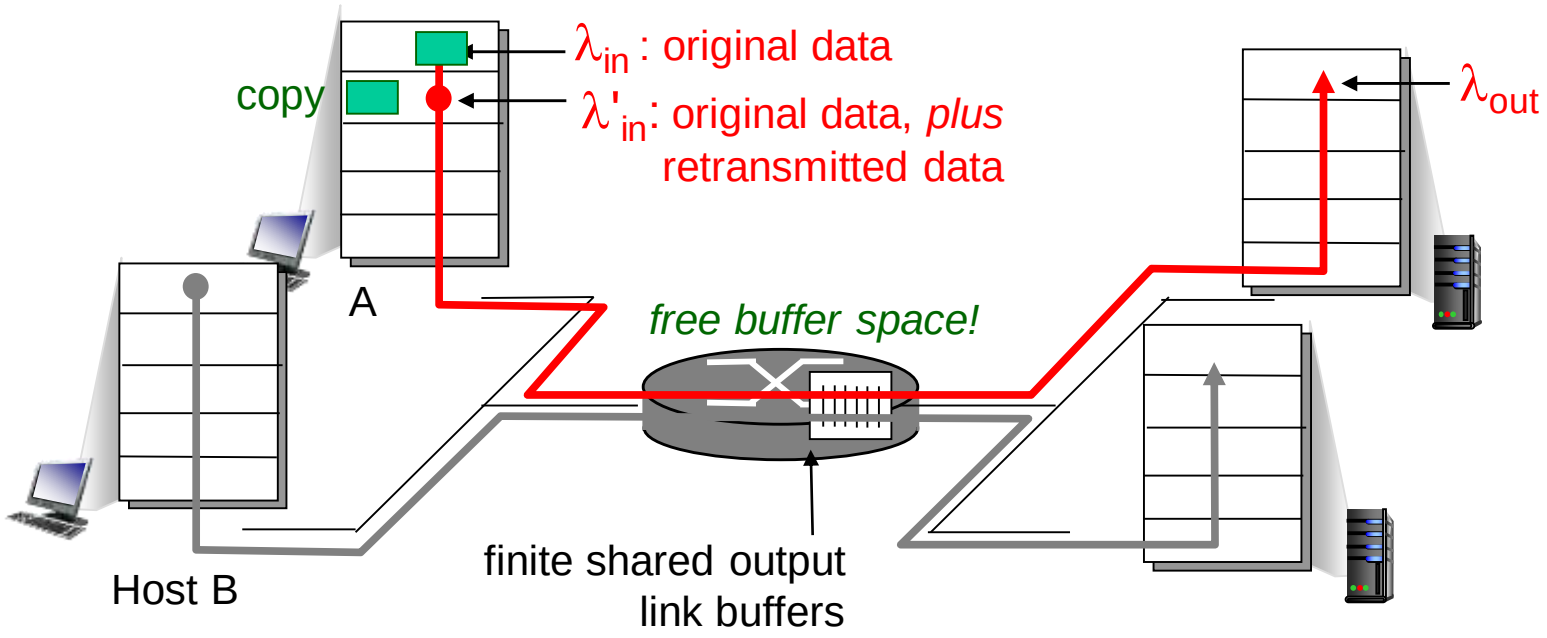
Causes/costs of Congestion: Scenario 2

- One router, finite buffers
- Sender retransmission of timed-out packet
  - application-layer input = application-layer output: $\lambda_{in} = \lambda_{out}$
  - transport-layer input includes *retransmissions* : $\lambda'_{in} >= \lambda_{in}$



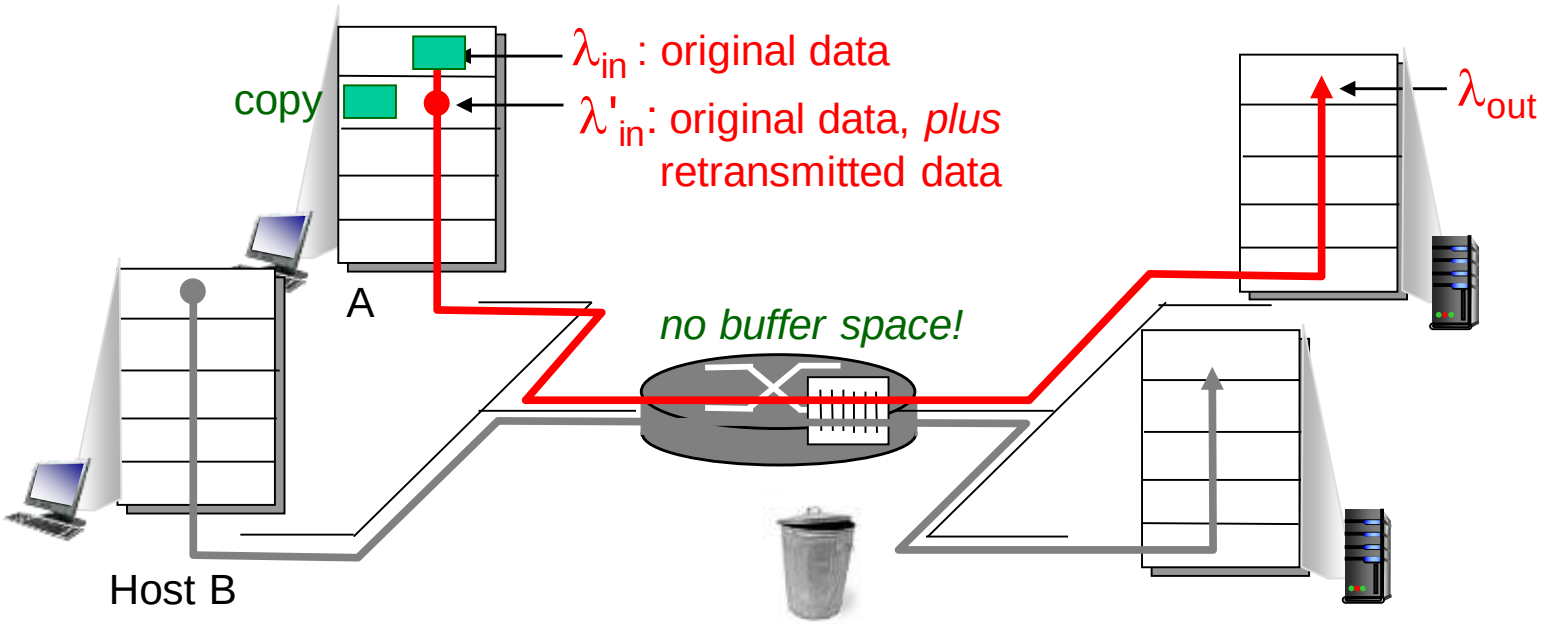
Causes/costs of Congestion: Scenario 2

- Idealization: Perfect knowledge about buffer space
- Sender sends only when router buffers available



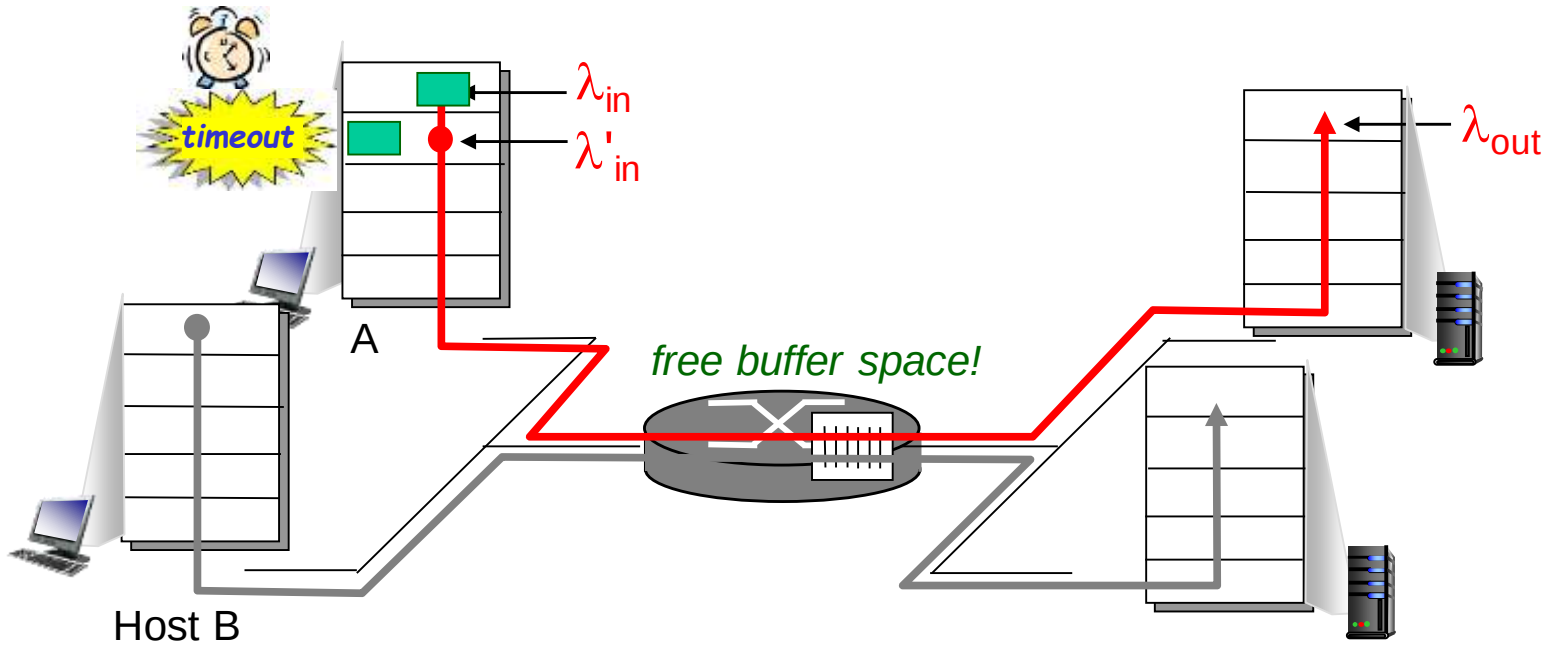
Causes/costs of Congestion: Scenario 2

- Idealization: Known loss
- Packets can be lost, dropped at router due to full buffers
- Sender only resends if packet known to be lost



Causes/costs of Congestion: Scenario 2

- Realistic: duplicate packets
- Packets can be lost, dropped at router due to full buffers
- Sender times out prematurely, sending two copies, both of which are delivered

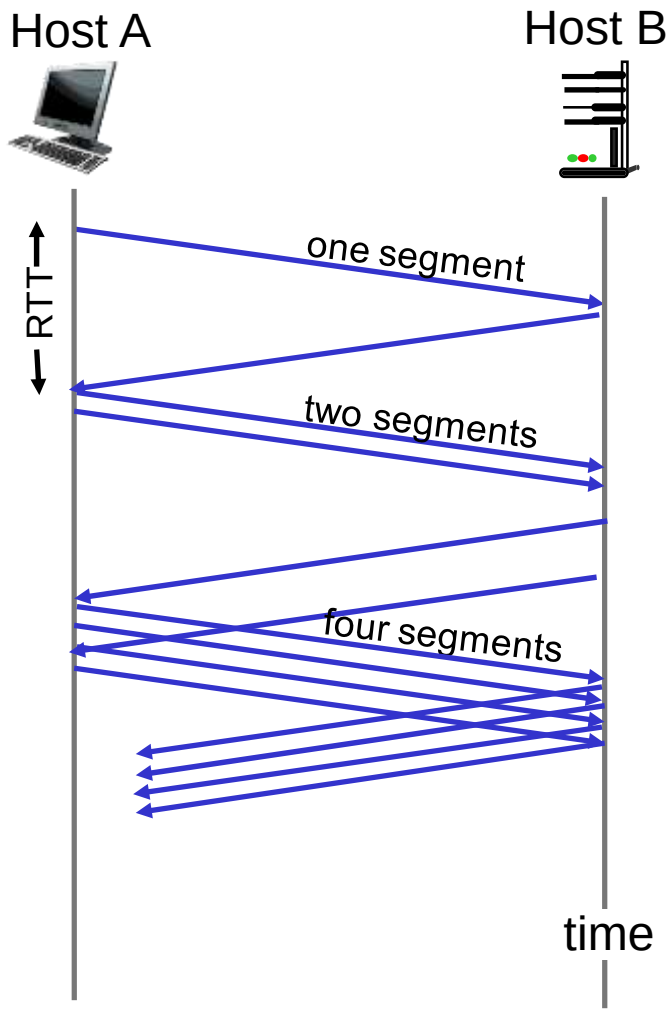


Approaches towards Congestion Control

- Two broad approaches towards congestion control
  1. End to End congestion control
     - No explicit feedback from network
     - Congestion inferred from end-system observed loss, delay
     - Approach taken by TCP
  2. Network-assisted congestion control
     - Routers provide feedback to end systems
     - Single bit indicating congestion (SNA, DECbit, TCP/IP ECN, ATM)
     - Explicit rate for sender to send

TCP Slow Start

- An algorithm which balances the speed of a network connection.
- TCP slow start is one of the first steps in the congestion control process.
- It balances the amount of data a sender can transmit (known as the congestion window) with the amount of data the receiver can accept (known as the receiver window).



TCP Slow Start – Cont…

- Slow start gradually increases the amount of data transmitted until it finds the network's maximum carrying capacity.
- A sender communicate to a receiver. Initial packet contains a small congestion window, which is determined based on the sender's maximum window.
- A receiver acknowledges the packet and responds with its own window size.
- If the receiver fails to respond, the sender knows not to continue sending data.
- After receiving the acknowledgement, the sender increases the next packet's window size.
- The window size gradually increases until the receiver can no longer acknowledge each packet, or until either the sender or the receiver's window limit is reached.

Summary

- Introduction about Transport Layer Services and Protocol
- Multiplexing and Demultiplexing (Connection-less & Connection Oriented)
- Connection less transport (UDP, Checksum)
- Principles of Reliable Data Transfer(rdt 1.0, rdt 2.0, rdt 2.1, rdt 2.2, rdt 3.0)
- Pipelined Protocol (GobackN, Selective Repeat)
- Connection oriented transport (TCP)
- Flow Control & Congestion Control
- TCP Slow Start approach

Thank You