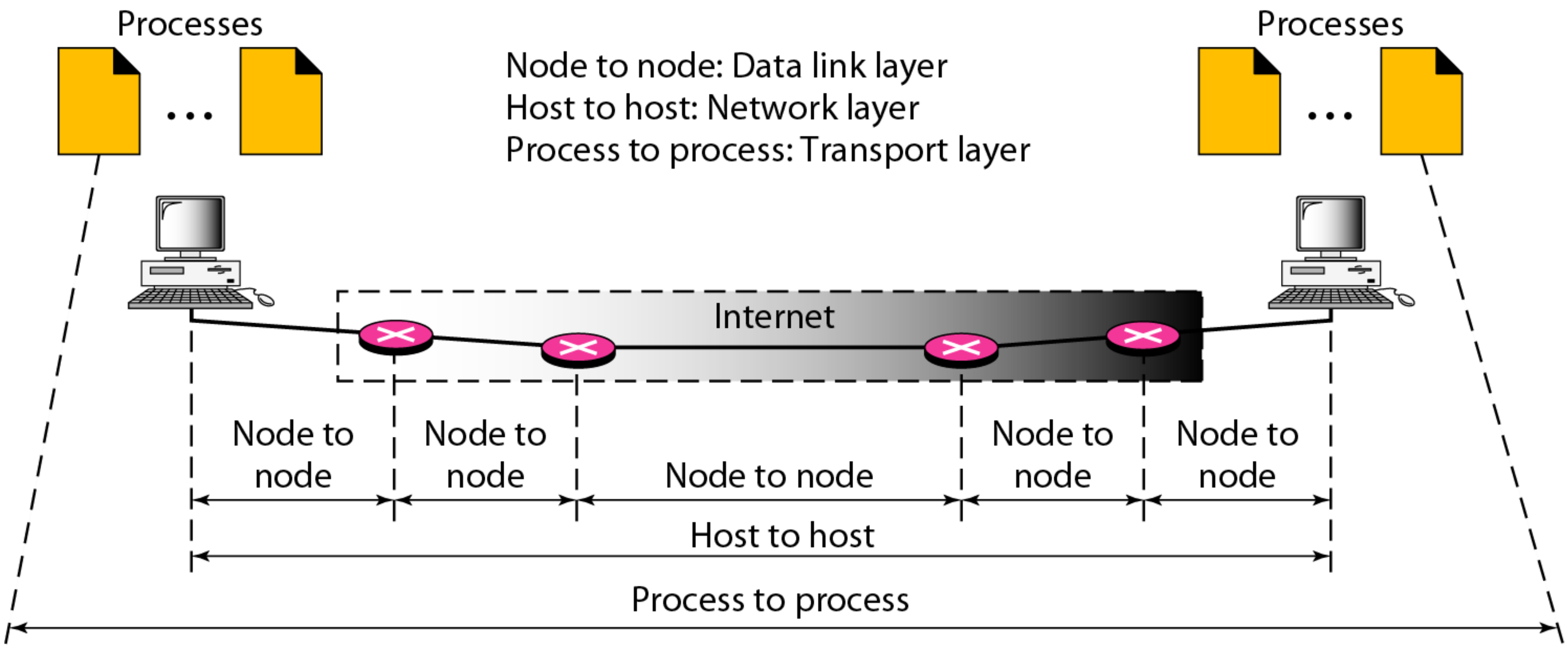
Computer Networks

Unit - 5
Transport Layer

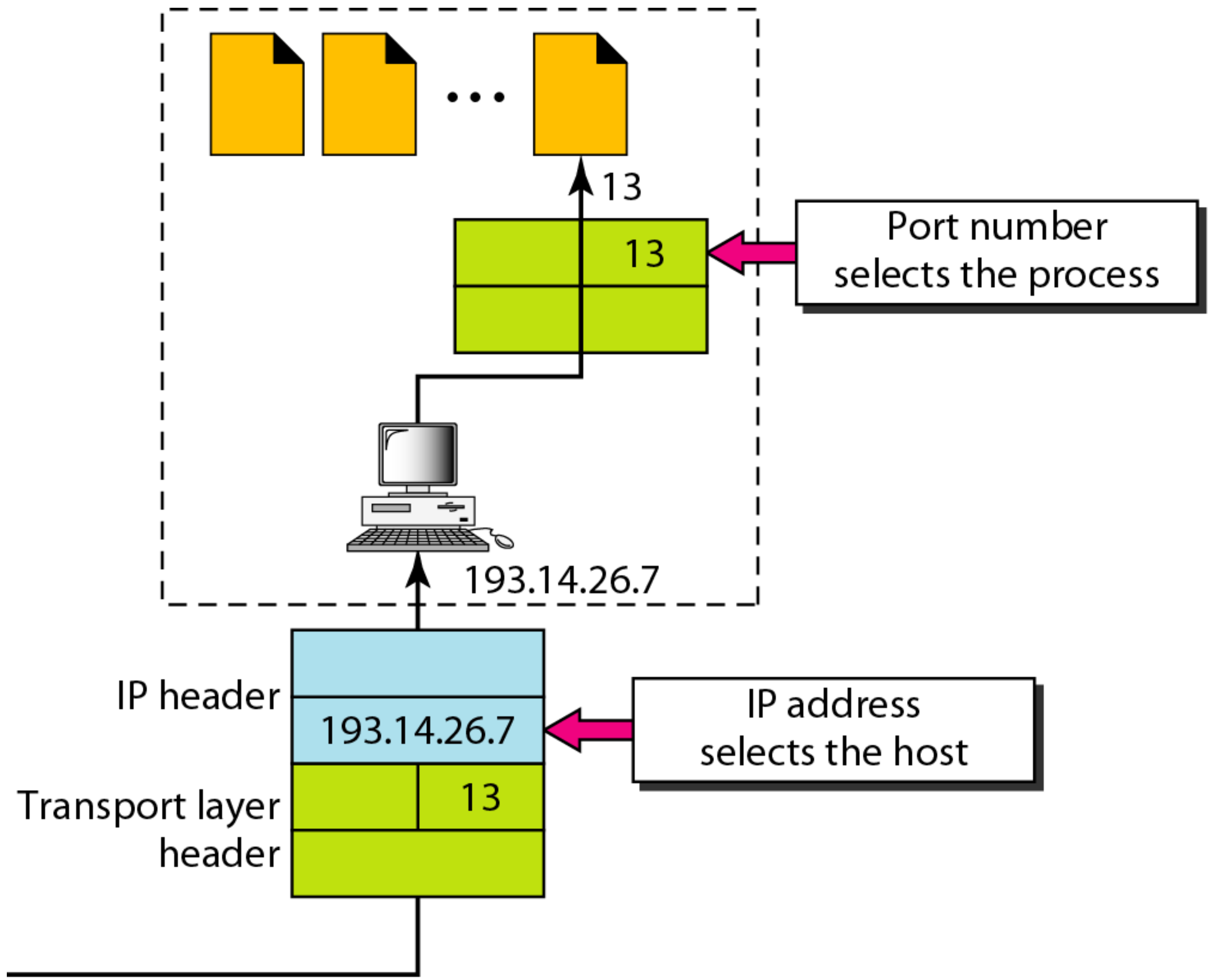
Outline

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

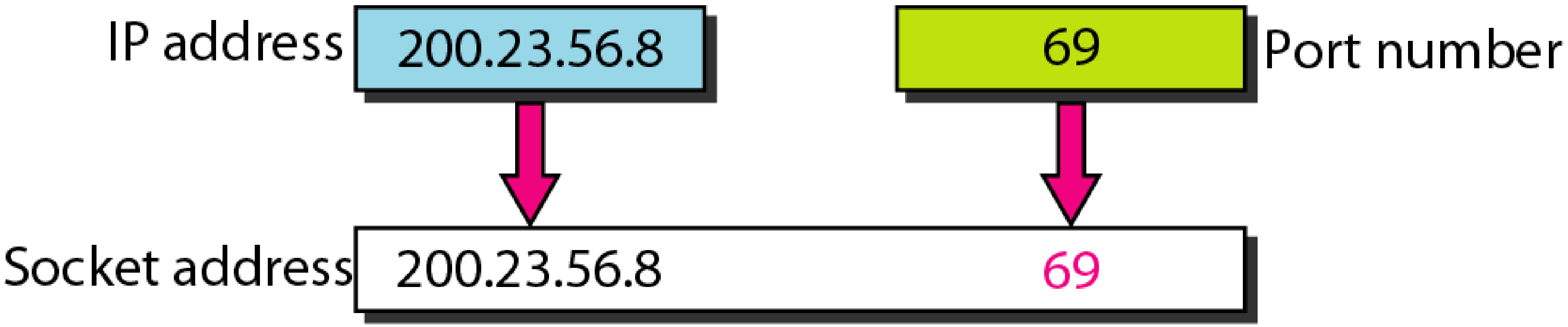
The transport layer is responsible for process-to-process delivery.



IP ADDRESSES VERSUS PORT NUMBERS

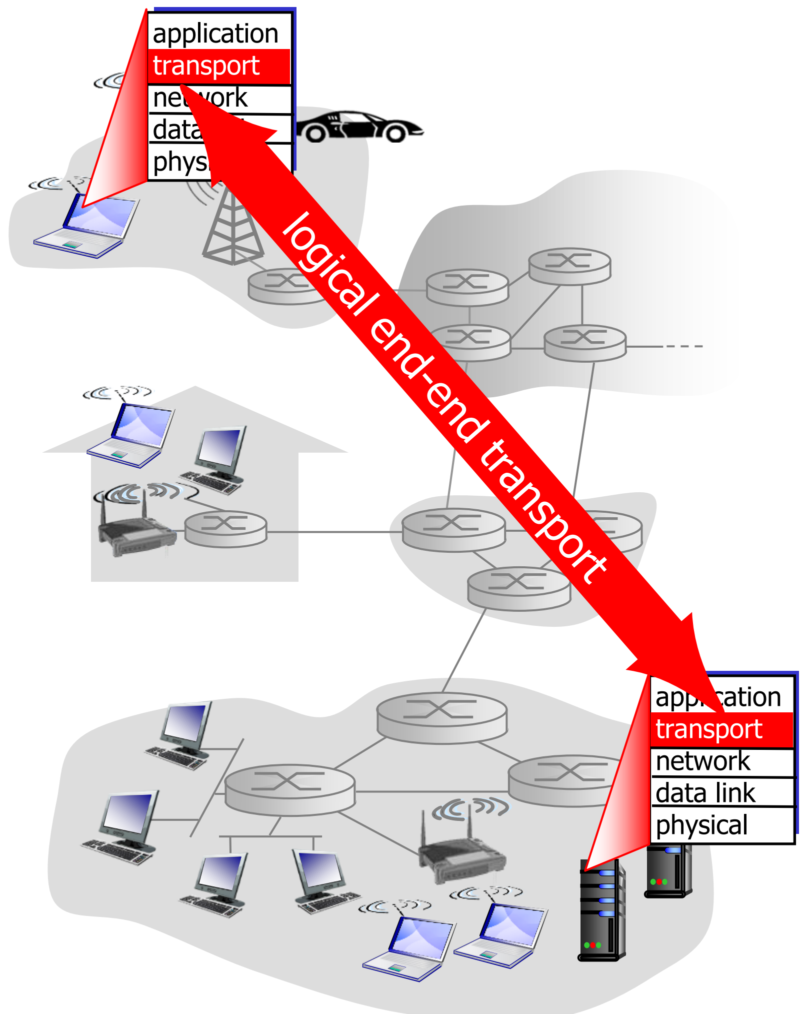


SOCKET ADDRESS



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.
- E.g. TCP and UDP



Transport Layer Services and Protocols – Example

West Side

Applications Messages = Letters in envelops

Network Layer Protocol = postal service
(including mail persons)

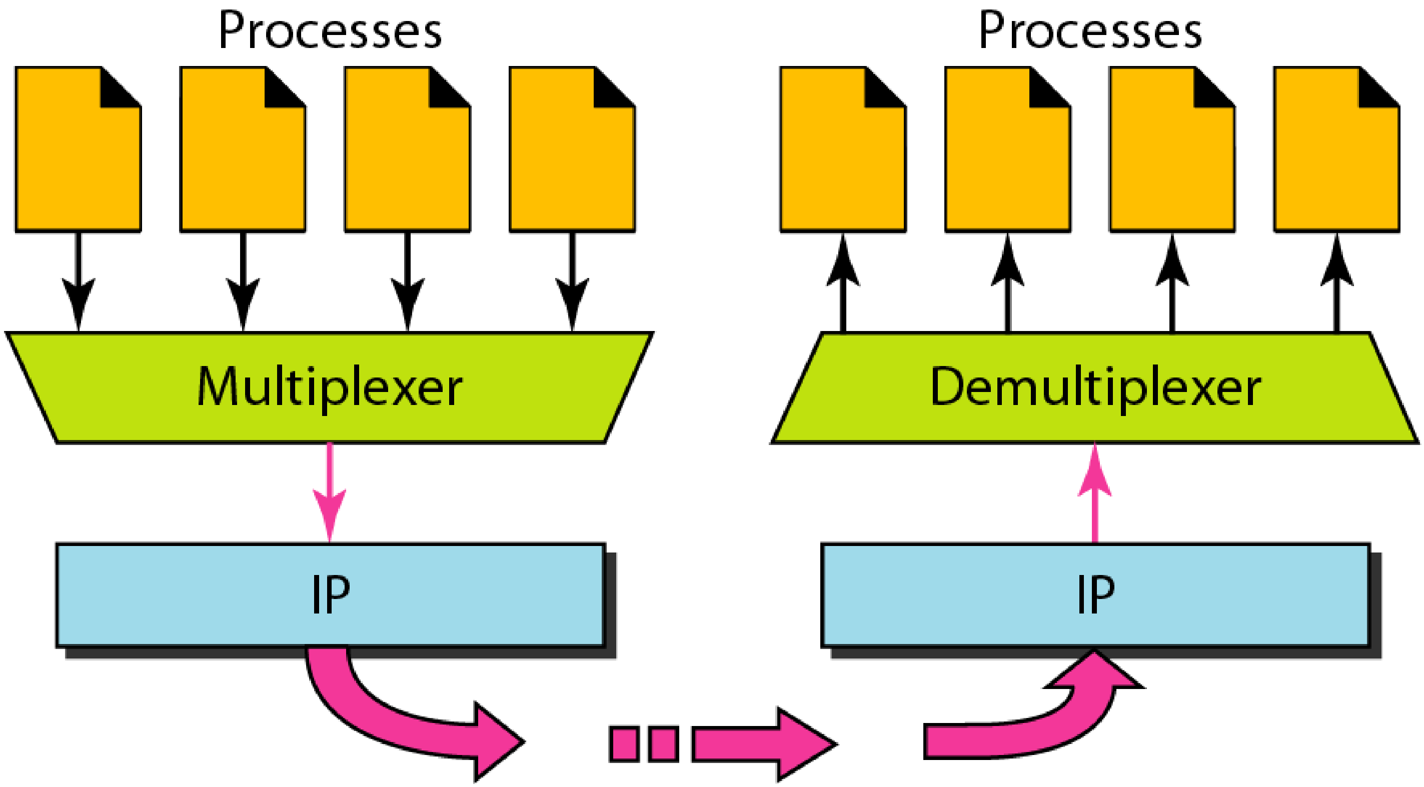
Hosts(End Systems) = Houses

Processes = Cousins

Transport Layer Protocol = Ann & Bill

East Side

MULTIPLEXING AND DEMULTIPLEXING



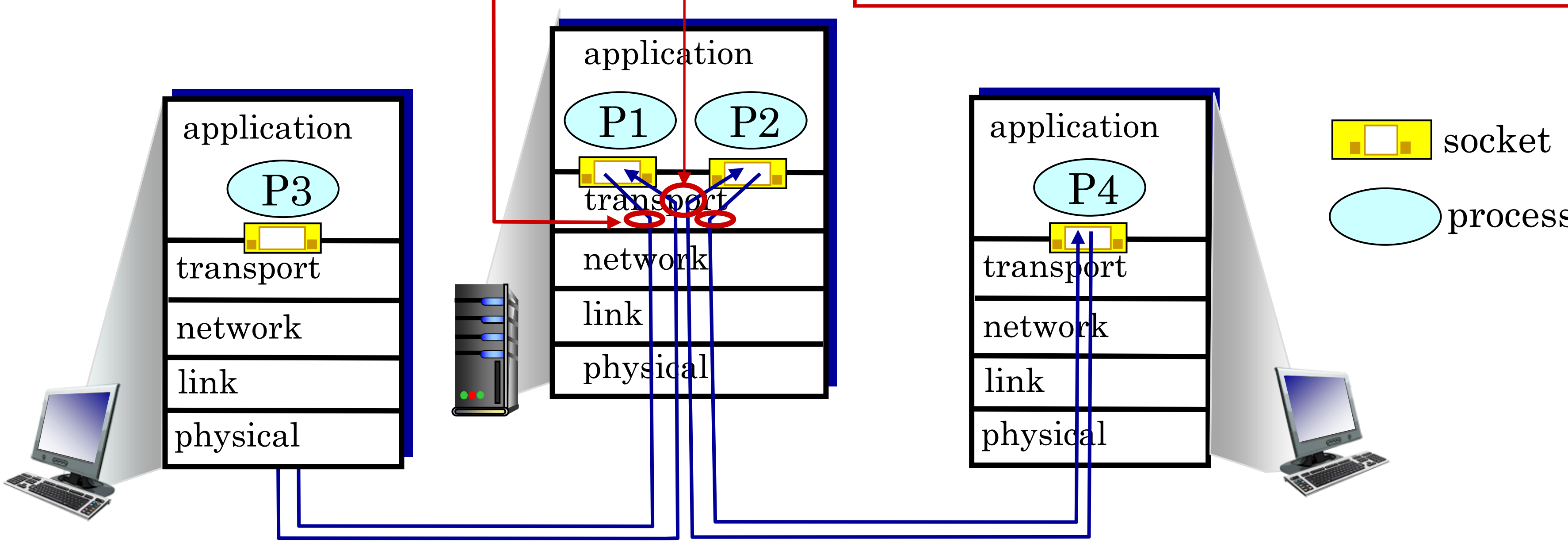
Multiplexing / Demultiplexing

Multiplexing at sender:

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

Demultiplexing at receiver:

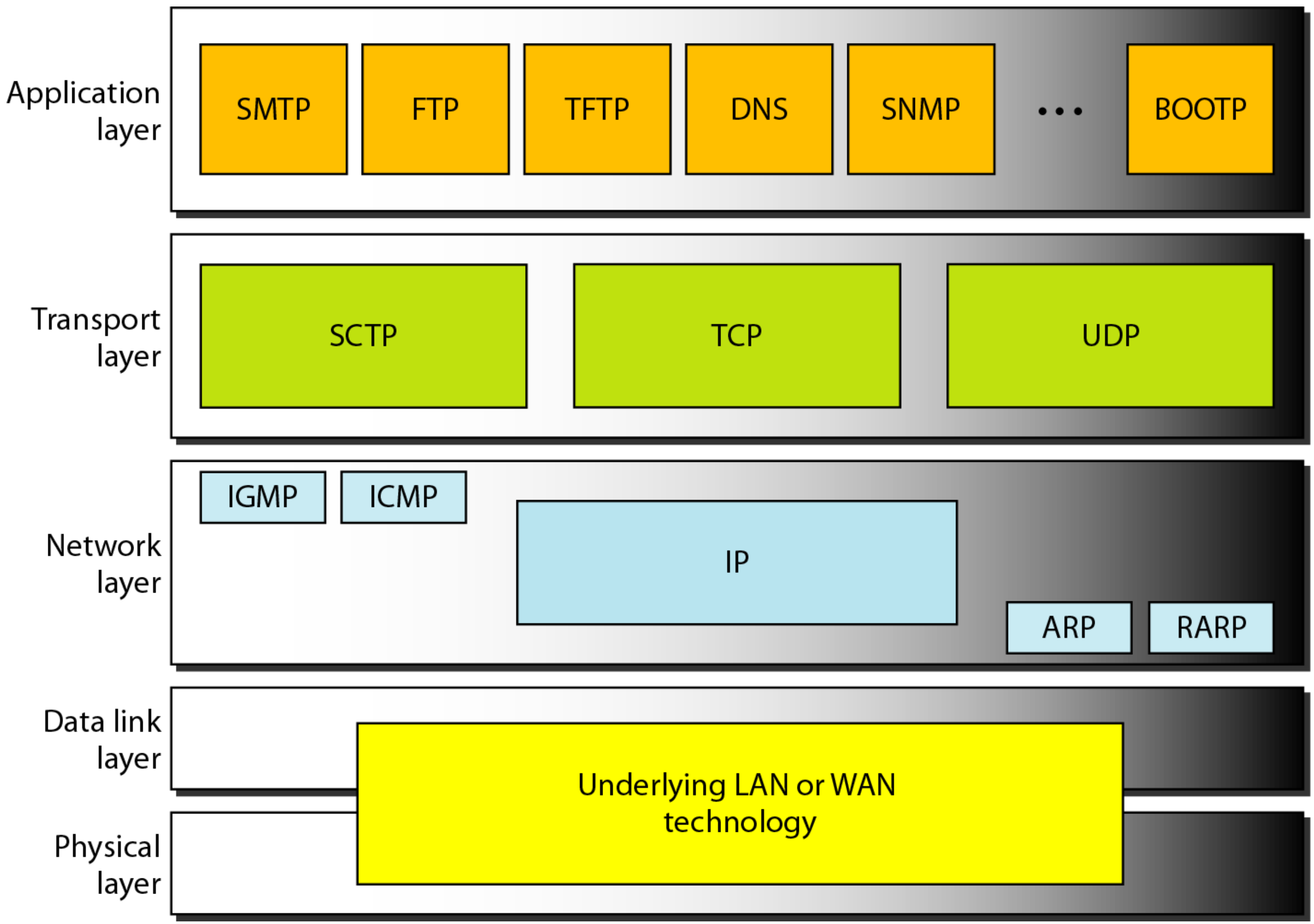
Use header information to deliver received segments to correct socket



Multiplexing / Demultiplexing

- The most fundamental responsibility of UDP and TCP is to extend IP's delivery service between two end systems to a delivery service between two processes running on the end systems.
- Extending host-to-host delivery to process-to-process delivery is called **transport-layer multiplexing** and **demultiplexing**.
- Each transport-layer segment has a set of fields in the segment for this purpose.
- At the receiving end, the transport layer examines these fields to identify the receiving socket and then directs the segment to that socket.
- This job of delivering the data in a transport-layer segment to the correct socket is called **demultiplexing**.
- The job of gathering data chunks at the source host from different sockets, encapsulating each data chunk with header information (that will later be used in demultiplexing) to create segments, and passing the segments to the network layer is called **multiplexing**.

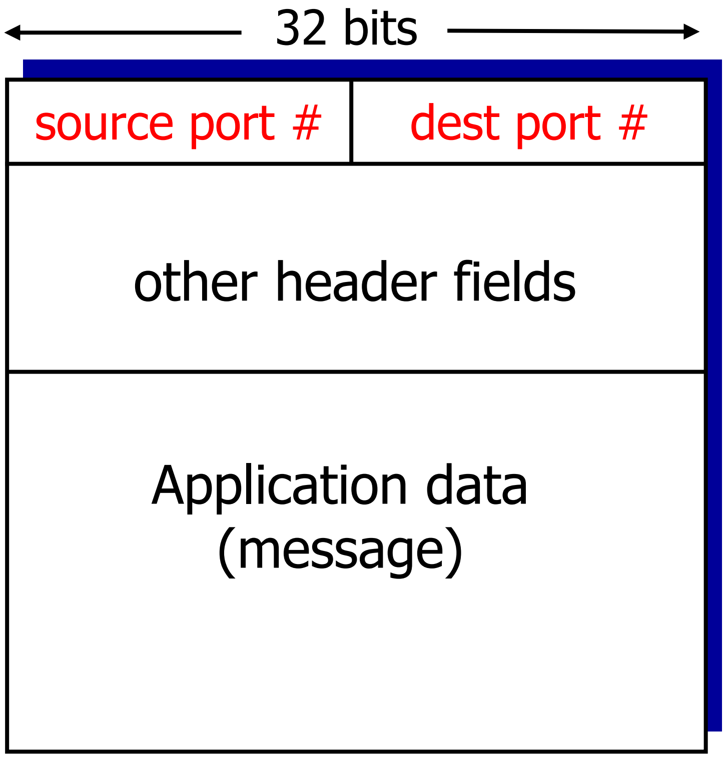
Position of UDP, TCP, and SCTP in TCP/IP suite



- Segment: Collection of bytes
- Byte Streaming
- Connection Oriented (Reliable)
- Full Duplex
- Piggybacking (Ack.)
- Error Control
- Flow Control
- Congestion Control

How demultiplexing 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.

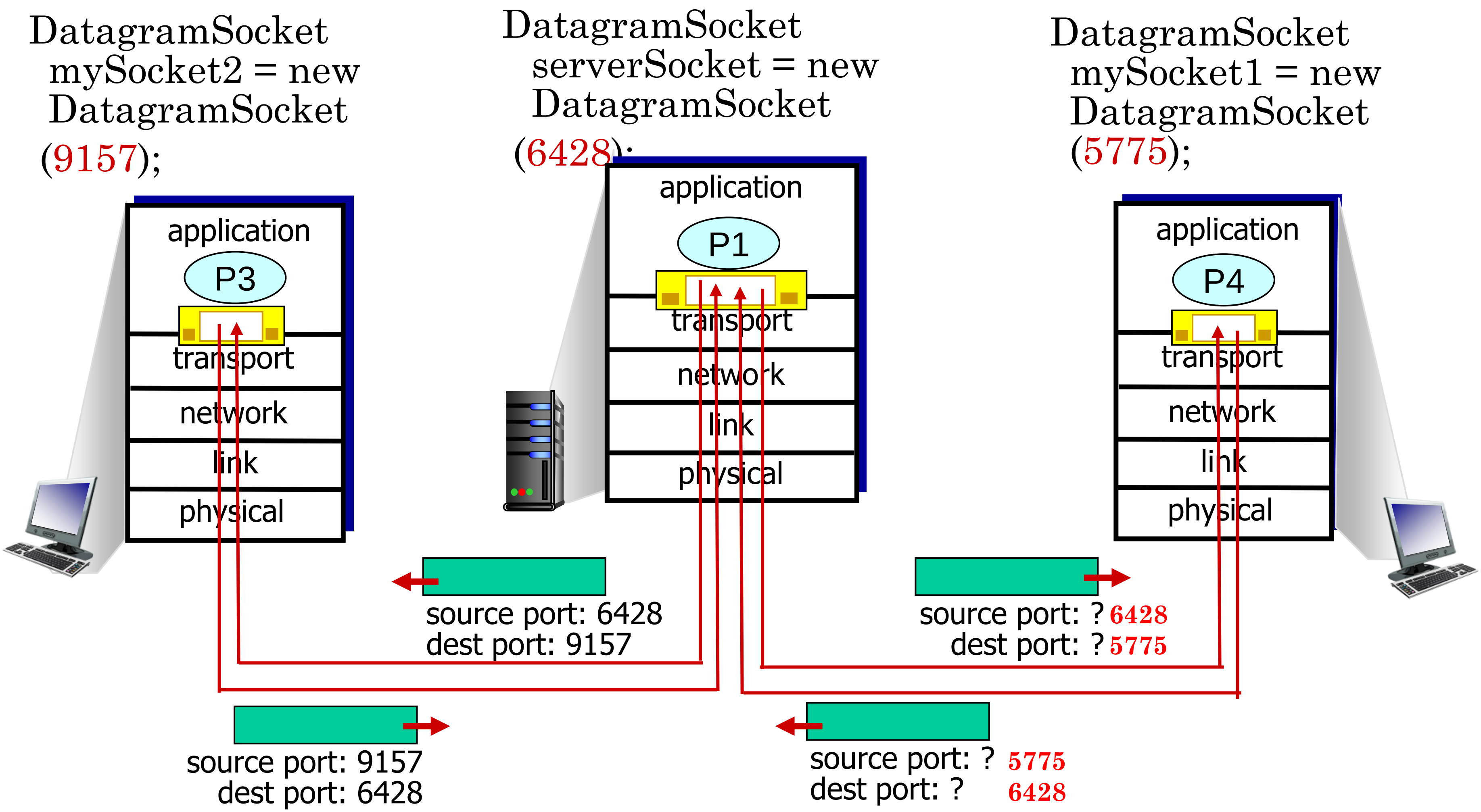


TCP/UDP segment format

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.

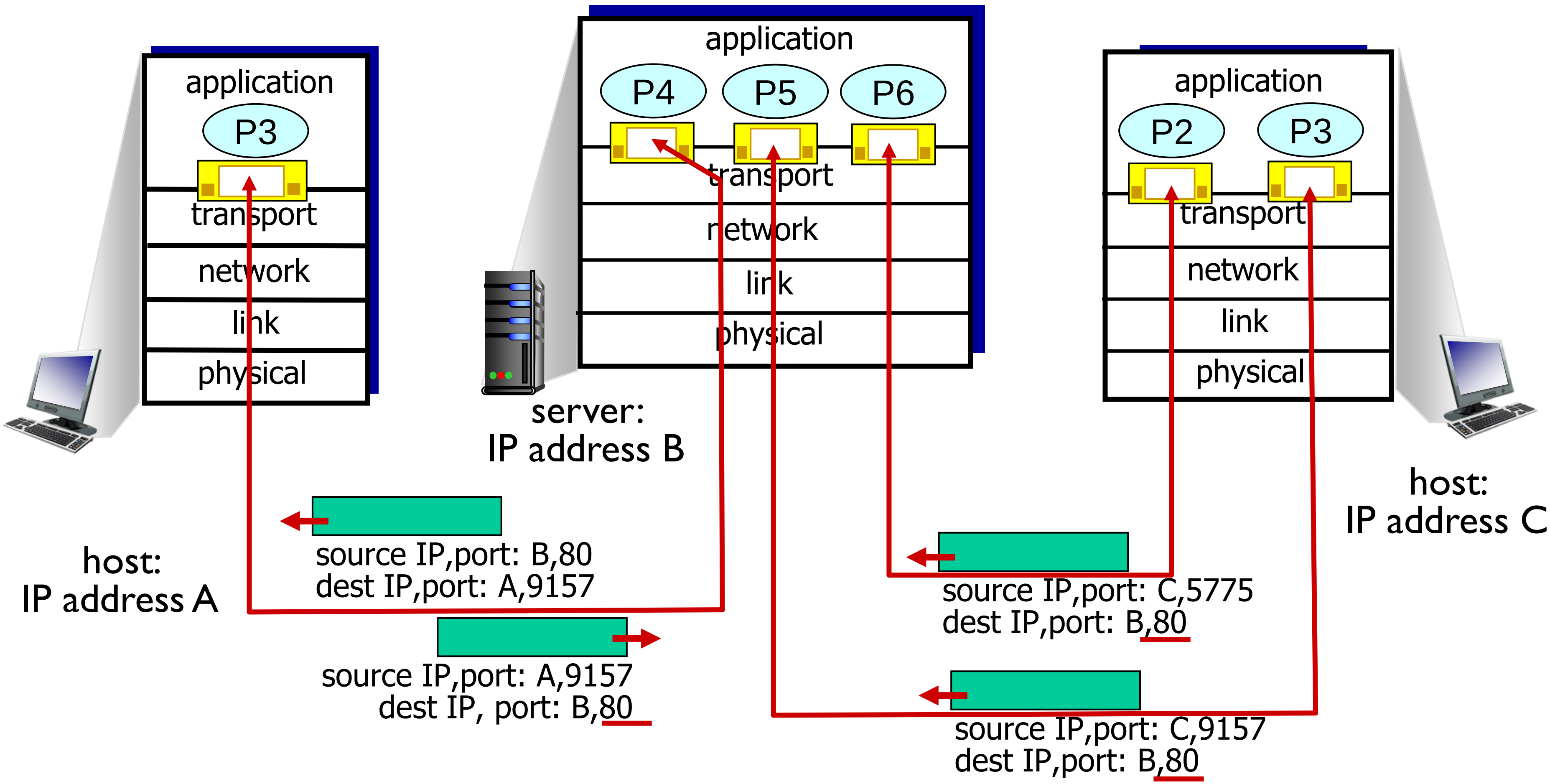
Example: Connectionless



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.

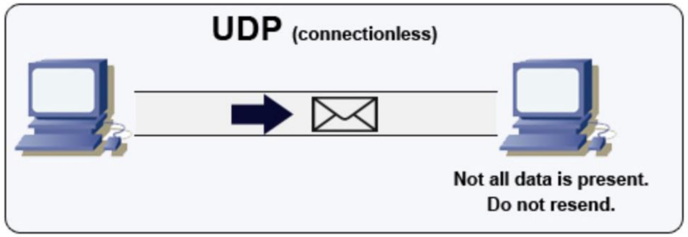
EXAMPLE: CONNECTION-ORIENTED



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 & dest. 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).



Distinguish between Connection-Oriented and Connectionless Service

Connection Oriented Services	**Connectionless Services–**
It can generate an end to end connection between the senders to the receiver before sending the data over the same or multiple networks.	It can transfer the data packets between senders to the receiver without creating any connection.
It generates a virtual path between the sender and the receiver.	It does not make any virtual connection or path between the sender and the receiver.
It needed a higher bandwidth to transmit the data packets.	It requires low bandwidth to share the data packets.
There is no congestion as it supports an end-to-end connection between sender and receiver during data transmission.	There can be congestion due to not providing an end-to-end connection between the source and receiver to transmit data packets.
It is a more dependable connection service because it assures data packets transfer from one end to the other end with a connection.	It is not a dependent connection service because it does not ensure the share of data packets from one end to another for supporting a connection.

S. No	Comparison Parameter	Connection-oriented Service	Connection Less Service
1.	Related System	It is designed and developed based on the telephone system.	It is service based on the postal system.
2.	Definition	It is used to create an end to end connection between the senders to the receiver before transmitting the data over the same or different network.	It is used to transfer the data packets between senders to the receiver without creating any connection.
3.	Virtual path	It creates a virtual path between the sender and the receiver.	It does not create any virtual connection or path between the sender and the receiver.
4.	Authentication	It requires authentication before transmitting the data packets to the receiver.	It does not require authentication before transferring data packets.
5.	Data Packets Path	All data packets are received in the same order as those sent by the sender.	Not all data packets are received in the same order as those sent by the sender.
6.	Bandwidth Requirement	It requires a higher bandwidth to transfer the data packets.	It requires low bandwidth to transfer the data packets.
7.	Data Reliability	It is a more reliable connection service because it guarantees data packets transfer from one end to the other end with a connection.	It is not a reliable connection service because it does not guarantee the transfer of data packets from one end to another for establishing a connection.
8.	Congestion	There is no congestion as it provides an end-to-end connection between sender and receiver during transmission of data.	There may be congestion due to not providing an end-to-end connection between the source and receiver to transmit of data packets.
9.	Examples	Transmission Control Protocol (TCP) is an example of a connection-oriented service.	User Datagram Protocol (UDP), Internet Protocol (IP), and Internet Control Message Protocol (ICMP) are examples of connectionless service.

Difference between Connection-oriented and Connection-less Services:

S.NO	Connection-oriented Service	Connection-less Service
1.	Connection-oriented service is related to the telephone system.	Connection-less service is related to the postal system.
2.	Connection-oriented service is preferred by long and steady communication.	Connection-less Service is preferred by bursty communication.
3.	Connection-oriented Service is necessary.	Connection-less Service is not compulsory.
4.	Connection-oriented Service is feasible.	Connection-less Service is not feasible.
5.	In connection-oriented Service, Congestion is not possible.	In connection-less Service, Congestion is possible.
6.	Connection-oriented Service gives the guarantee of reliability.	Connection-less Service does not give the guarantee of reliability.
7.	In connection-oriented Service, Packets follow the same route.	In connection-less Service, Packets do not follow the same route.
8.	Connection-oriented Services requires a bandwidth of high range.	Connection-less Service requires a bandwidth of low range.

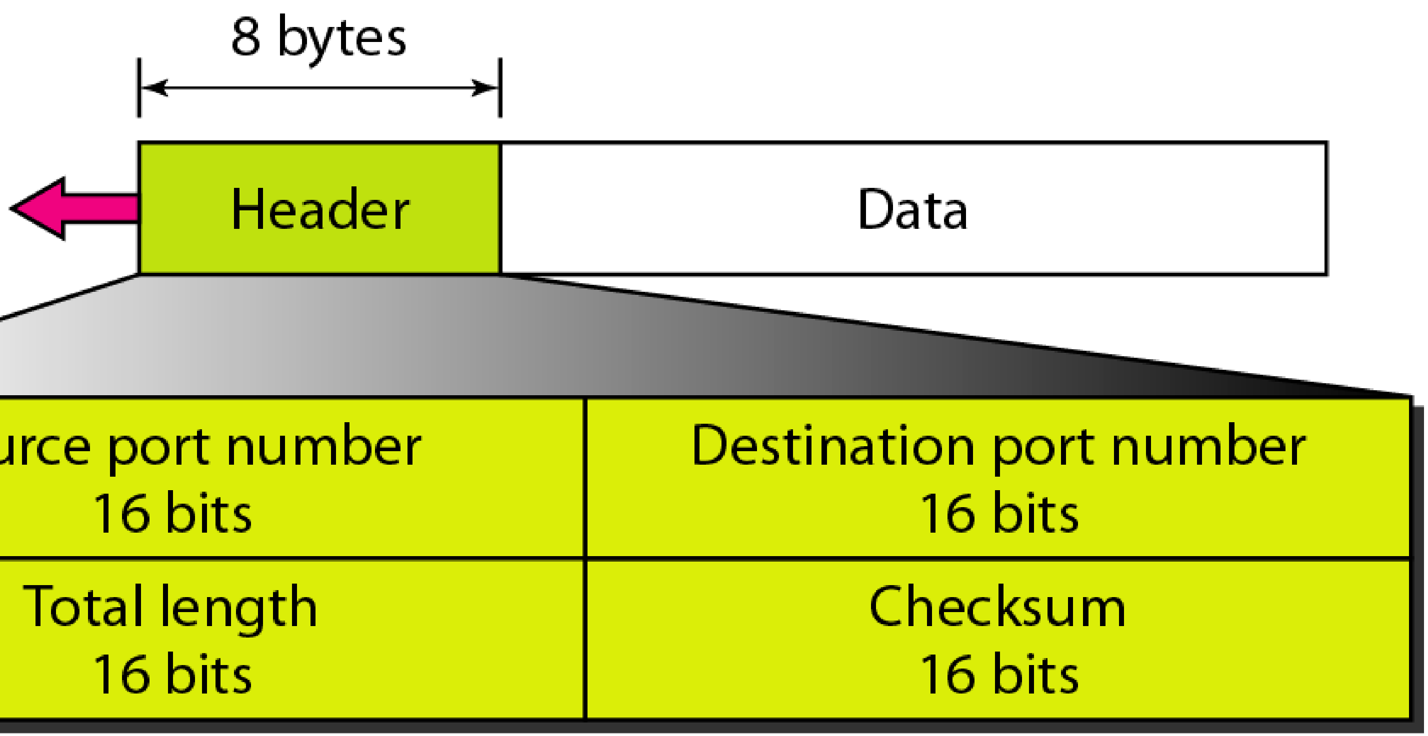
UDP

- *The User Datagram Protocol (UDP) is called a connectionless, unreliable transport protocol.*
- *It does not add anything to the services of IP except to provide process-to-process communication instead of host-to-host communication.*

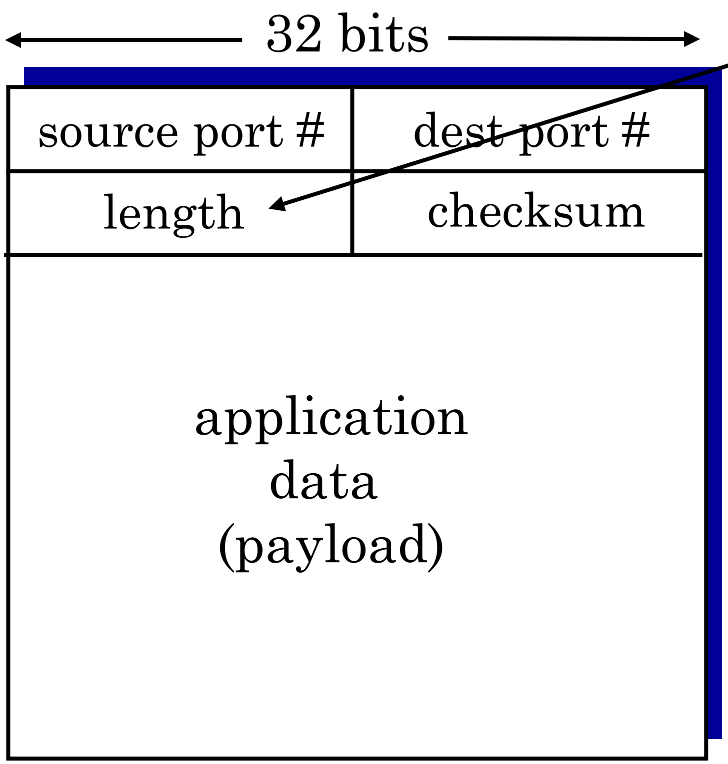
WELL-KNOWN PORTS USED WITH UDP

Port	Protocol	Description
7	Echo	Echoes a received datagram back to the sender
9	Discard	Discards any datagram that is received
11	Users	Active users
13	Daytime	Returns the date and the time
17	Quote	Returns a quote of the day
19	Chargen	Returns a string of characters
53	Nameserver	Domain Name Service
67	BOOTPs	Server port to download bootstrap information
68	BOOTPc	Client port to download bootstrap information
69	TFTP	Trivial File Transfer Protocol
111	RPC	Remote Procedure Call
123	NTP	Network Time Protocol
161	SNMP	Simple Network Management Protocol
162	SNMP	Simple Network Management Protocol (trap)

USER DATAGRAM FORMAT



UDP Segment - Header



UDP segment format

length (in bytes) UDP segment - including header

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

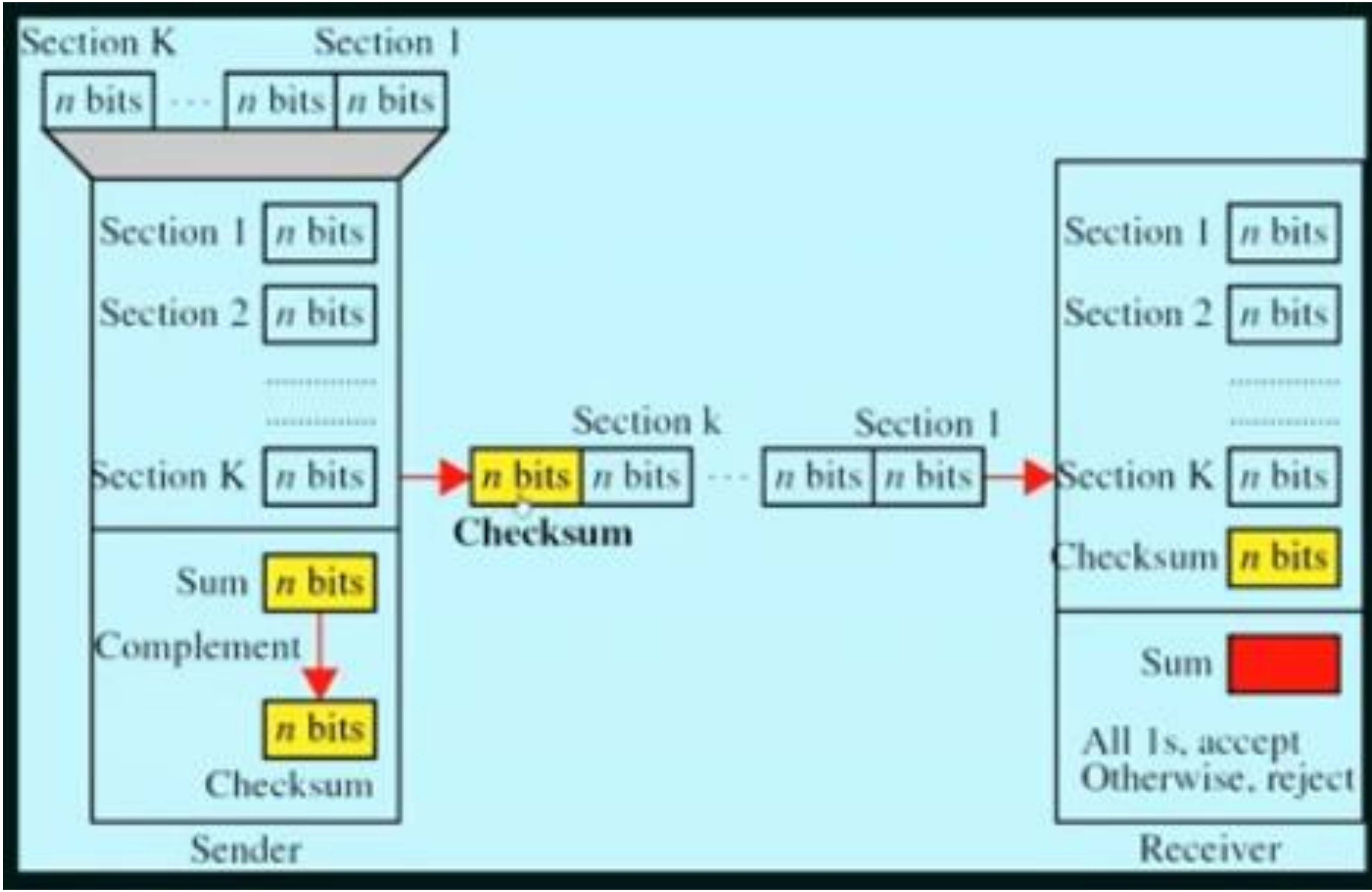
- Checksum is used to detect errors in transmitted segment.

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


Section K
Section 1
n bits
n bits
n bits
Section 1
n bits
Section 2
n bits
Section K
n bits
Sum
n bits
Complement
n bits
Checksum
Sender
Section k
Section 1
n bits
n bits
n bits
n bits
Checksum
Section 1
n bits
Section 2
n bits
Section K
n bits
Checksum
n bits
Sum
All 1s, accept
Otherwise, reject
Receiver

Checksum - Example

- Add two 16-bit integers word

Sender

	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
+	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

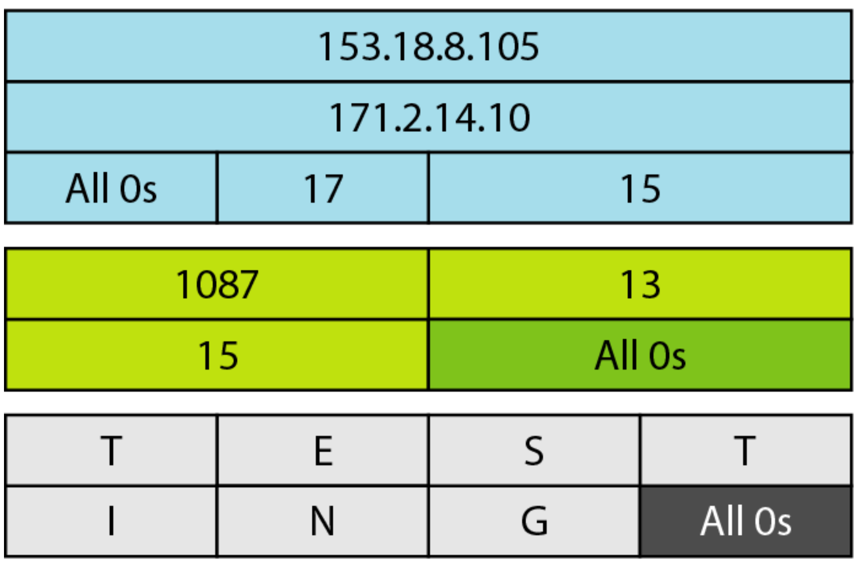
Receiver

	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
1	1 0 1 1 1 0 1 1 1 0 1 1 1 0 1 1
+	1
	1 0 1 1 1 0 1 1 1 0 1 1 1 1 0 0
	0 1 0 0 0 1 0 0 0 1 0 0 0 0 1 1
	1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1

If one of the bits is a 0, then we can say that error introduced into packet

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

CHECKSUM CALCULATION OF A SIMPLE UDP USER DATAGRAM

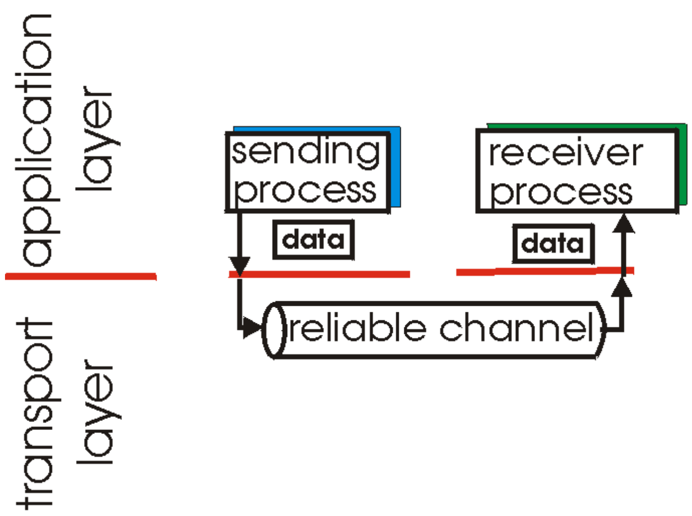


10011001 00010010	→	153.18
00001000 01101001	→	8.105
10101011 00000010	→	171.2
00001110 00001010	→	14.10
00000000 00010001	→	0 and 17
00000000 00001111	→	15
00000100 00111111	→	1087
00000000 00001101	→	13
00000000 00001111	→	15
00000000 00000000	→	0 (checksum)
01010100 01000101	→	T and E
01010011 01010100	→	S and T
01001001 01001110	→	I and N
01000111 00000000	→	G and 0 (padding)
10010110 11101011	→	Sum
01101001 00010100	→	Checksum

- Isn’t TCP always preferable, since TCP provides a reliable data transfer service, while UDP does not?
- The answer is no, as many applications are better suited for UDP for the following reasons:
  - ***Finer application-level control over what data is sent, and when.***
  - ***No connection establishment.***
  - ***No connection state.***
  - ***Small packet header overhead.***

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)

Unreliable Channel Characteristics

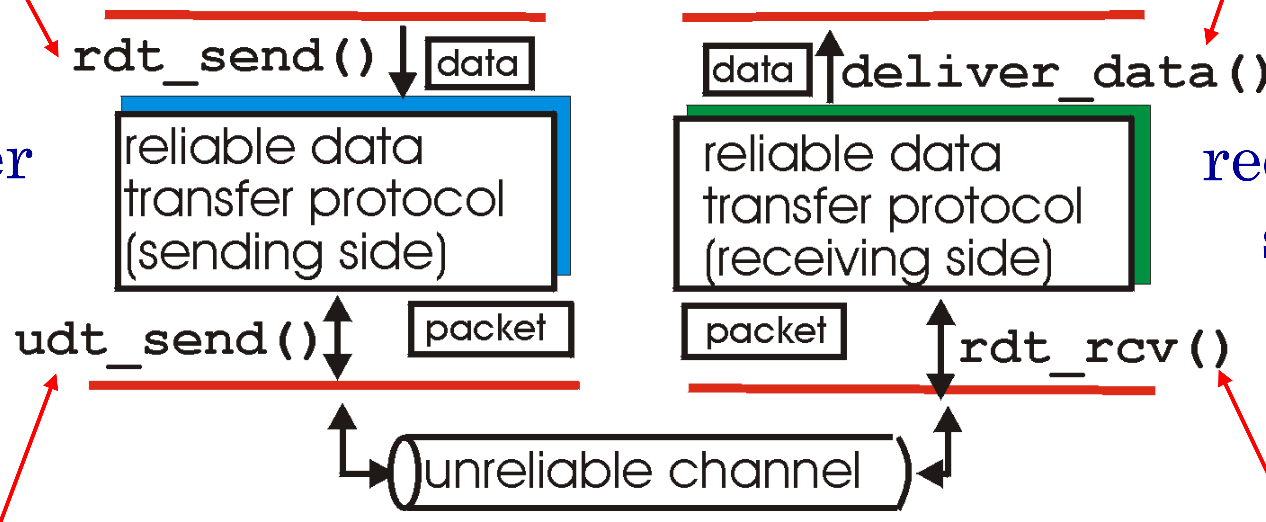
- Packet Errors:
  - packet content modified
  - Assumption: either no errors or detectable.
- Packet loss:
  - Can packet be dropped
- Packet duplication:
  - Can packets be duplicated.
- Reordering of packets
  - Is channel FIFO?
- Internet: Errors, Loss, Duplication, non-FIFO

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

sender side



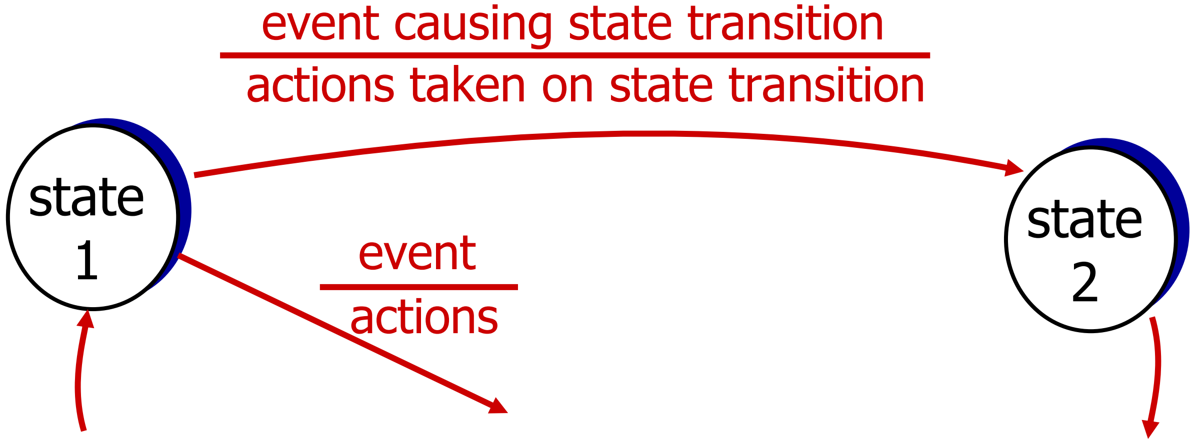
receiver side

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

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

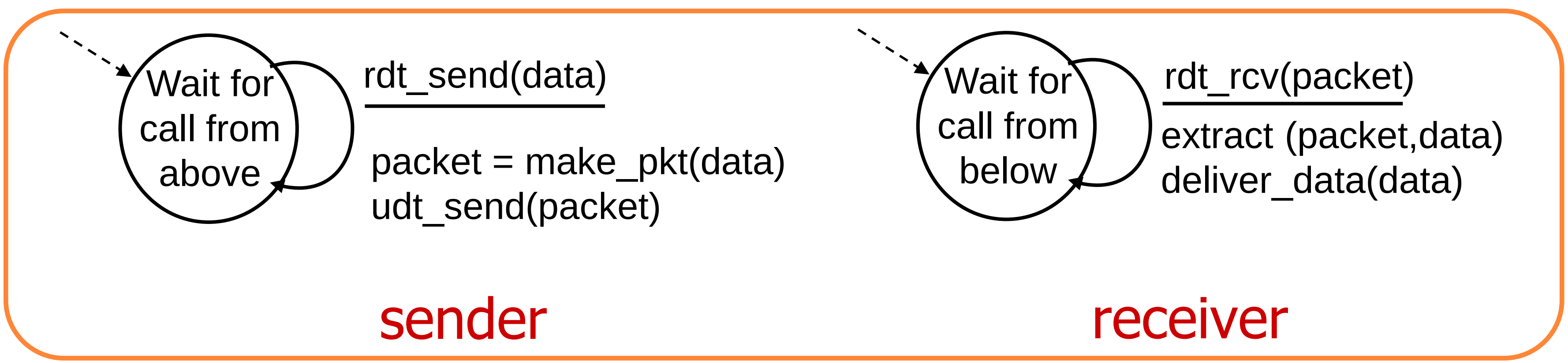
Reliable Data Transfer – Cont…

- 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

- There is no packet loss
- Underlying channel may flip bits in packet.
  - checksum to detect bit errors.
- Question: How do we 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.

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

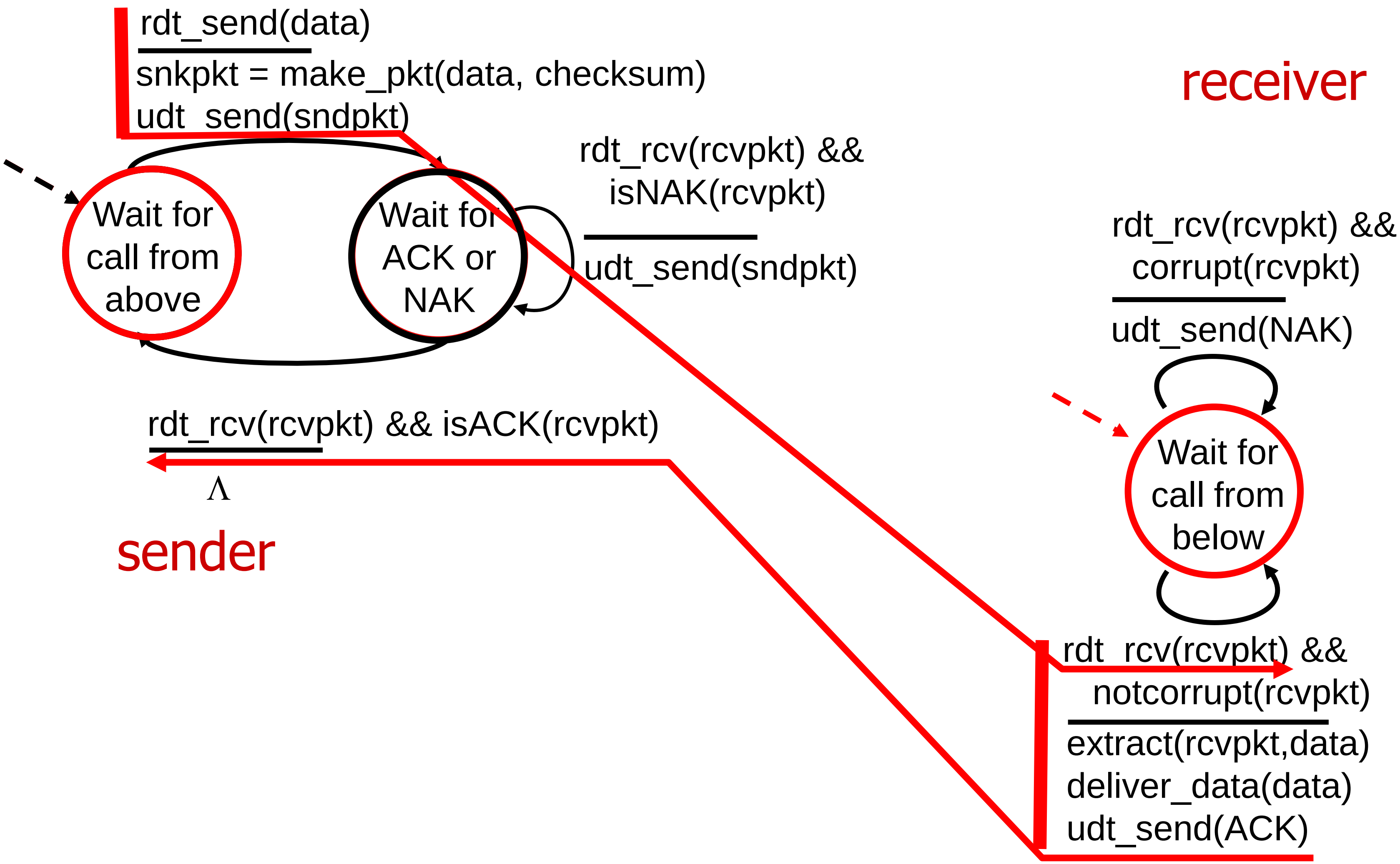
handling duplicates:

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

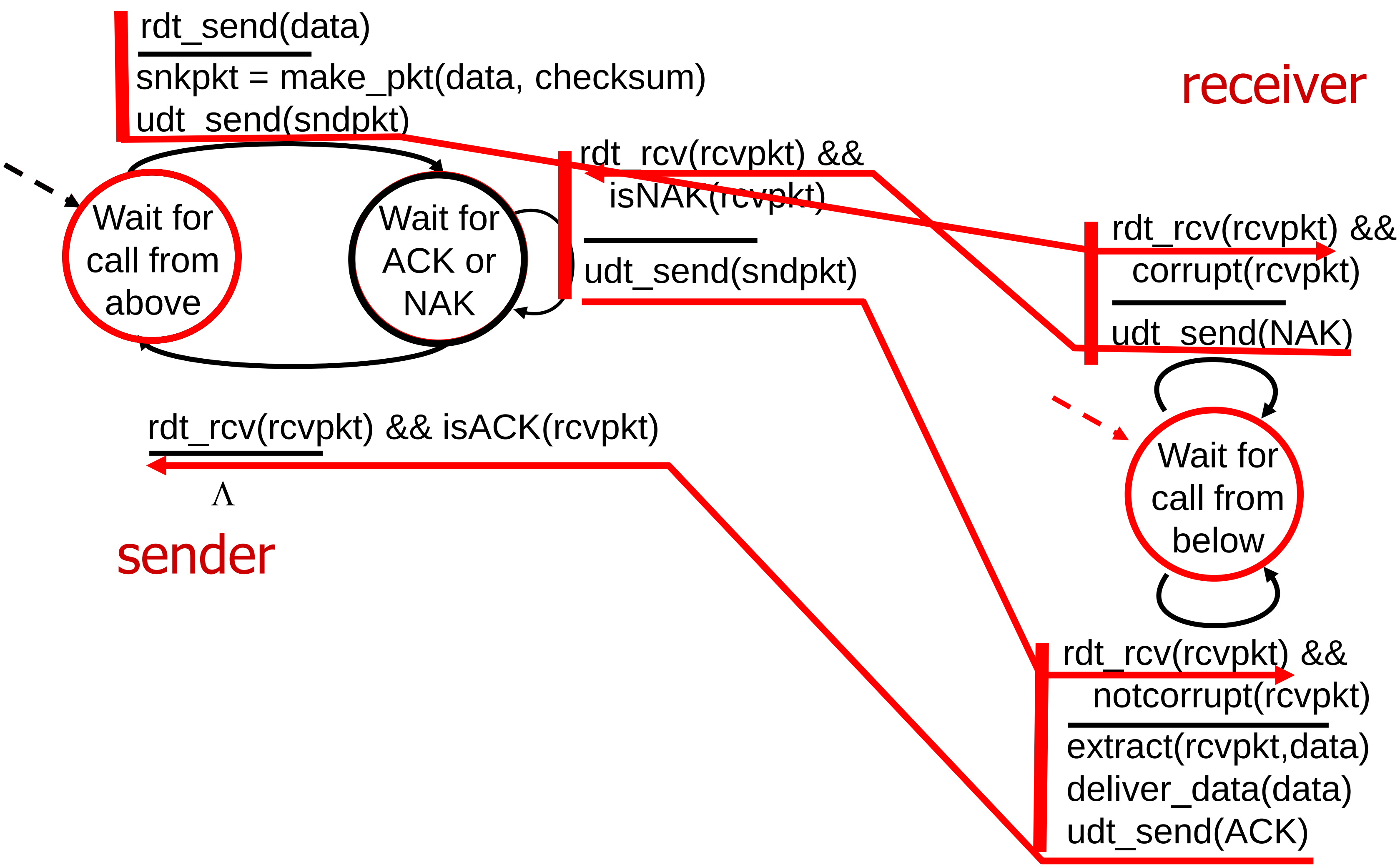
stop and wait

sender sends one packet, then waits for receiver response

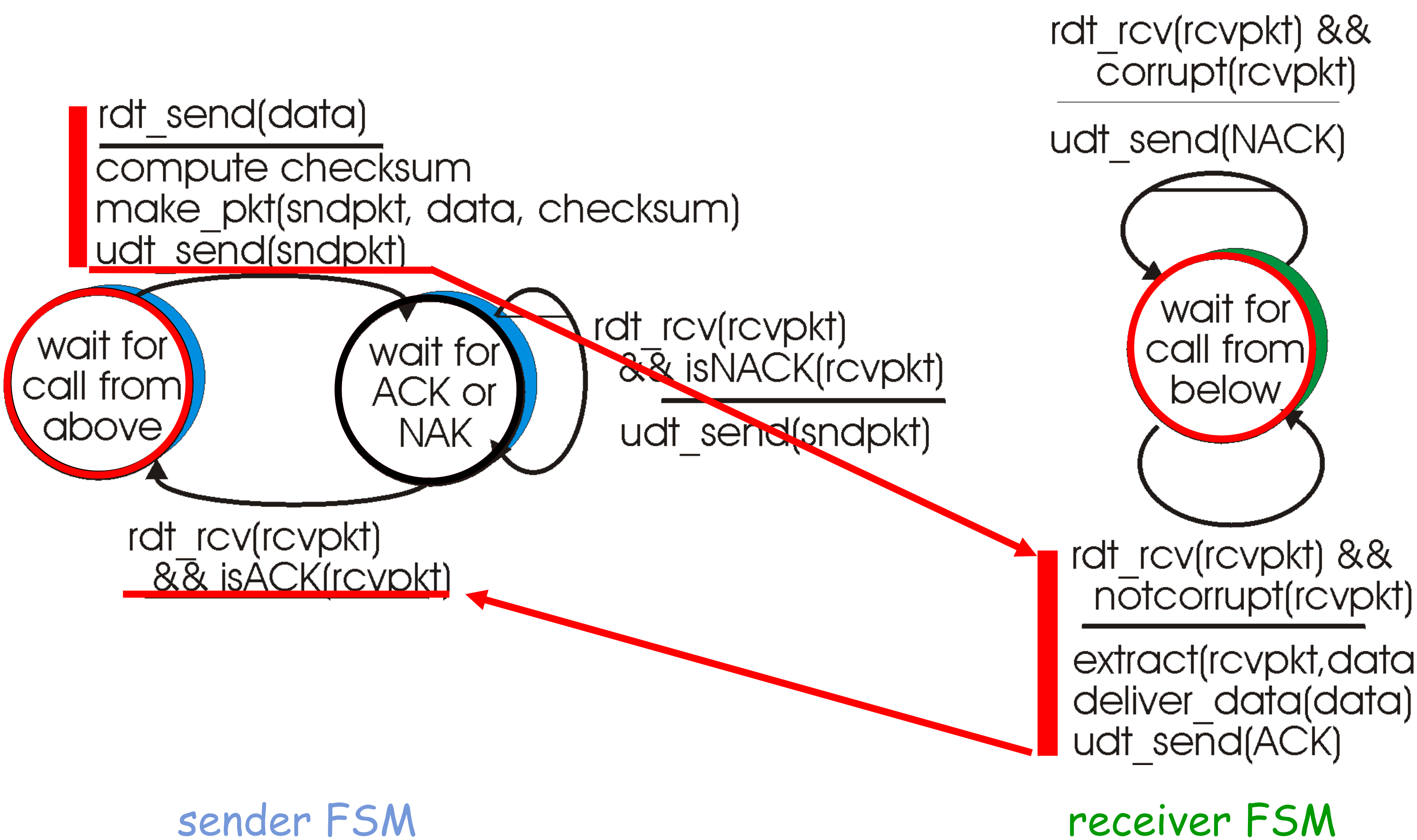
RDT 2.0 – WITH NO ERROR



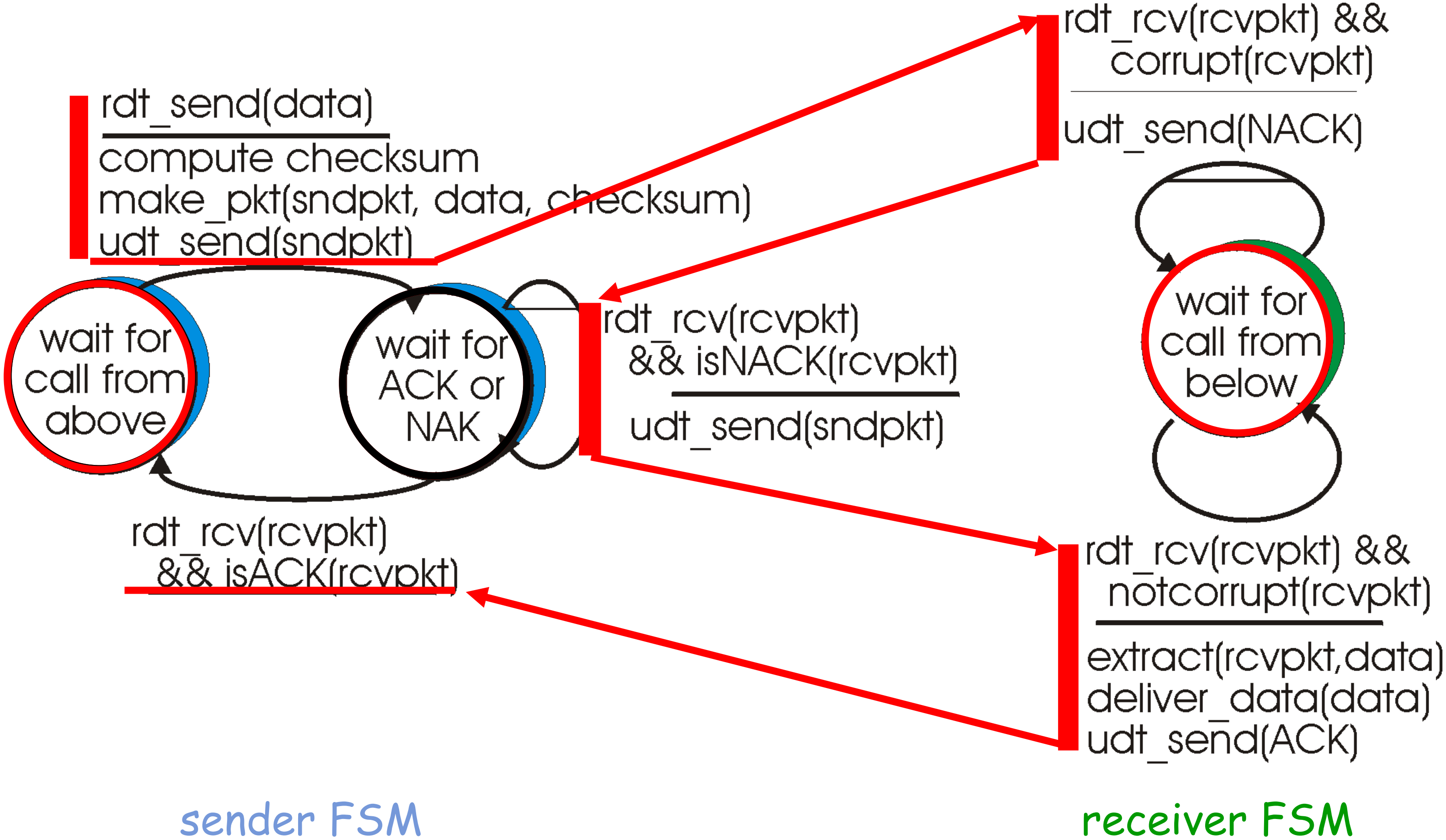
RDT 2.0 – WITH ERROR



RDT2.0: IN ACTION (NO ERRORS)



RDT2.0: IN ACTION (ERROR SCENARIO)



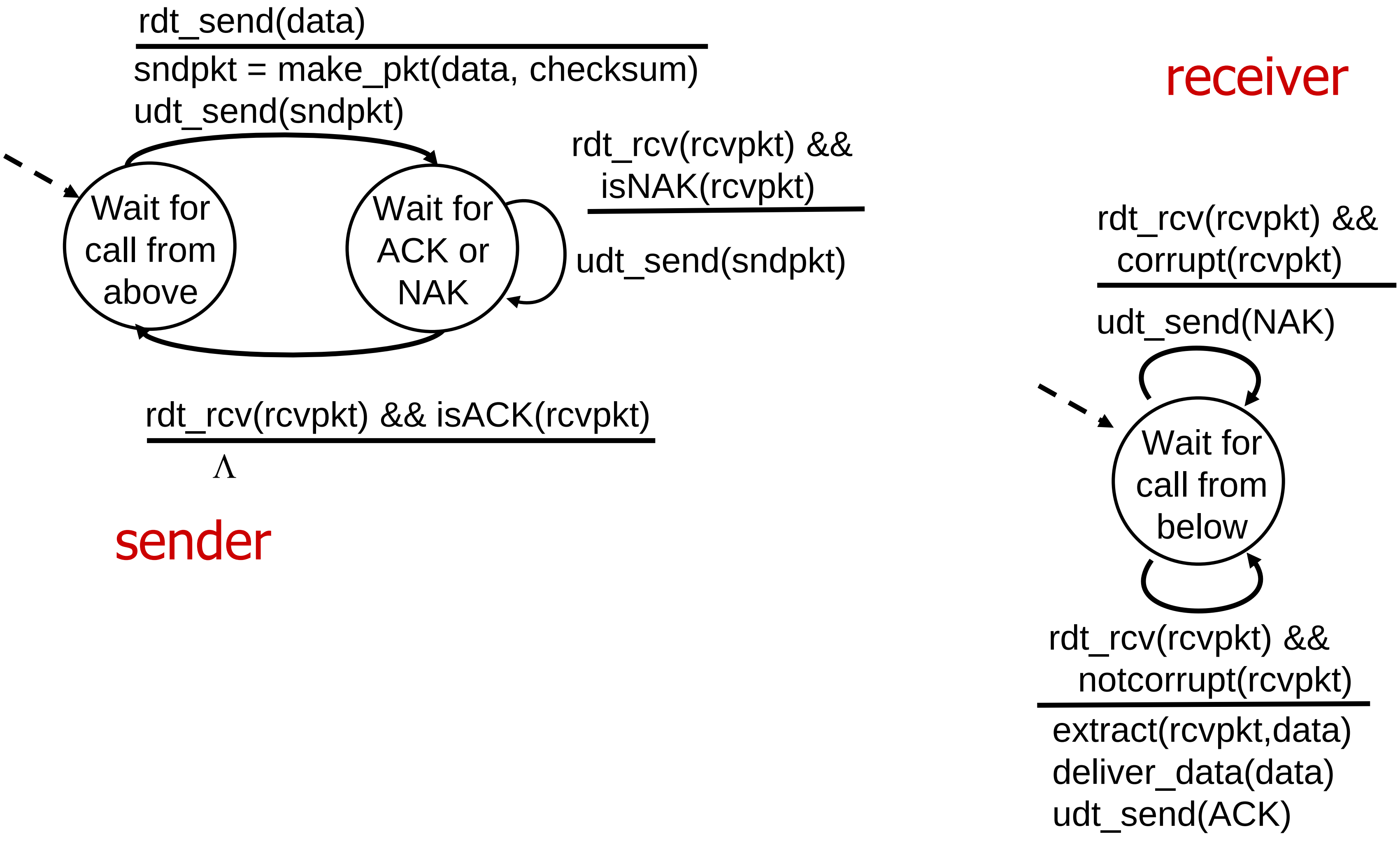
RDT 2.0: TYPICAL BEHAVIOR

Typical sequence in sender FSM:

"wait for call"
rdt_send(data)
udt_send(data)
"wait for Ack/Nack"
udt_send(NACK)
udt_send(data) udt_send(NACK)
. . .
udt_send(data) udt_send(NACK)
udt_send(data) udt_send(ACK)
"wait for call"

Claim A: There is at most one packet in transit.

RDT 2.0 – FSM SPECIFICATION



RDT 2.1 - HANDLED GARBLED 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 - GARBLED ACK/NACK

What happens if ACK/NACK corrupted?

- sender doesn't know what happened at receiver!
- If ACK was corrupt:
  - Data was delivered
  - Needs to return to "wait for call"
- If NACK was corrupt:
  - Data was not delivered.
  - Needs to re-send data.

What to do?

- Assume it was a NACK - retransmit, but this might cause retransmission of correctly received pkt! Duplicate.
- Assume it was an ACK - continue to next data, but this might cause the data to never reach the receiver! Missing.
- Solution: sender ACKs/NACKs receiver's ACK/NACK.

  What if sender ACK/NACK corrupted?

RDT 2.1 - GARBLED ACK/NACK

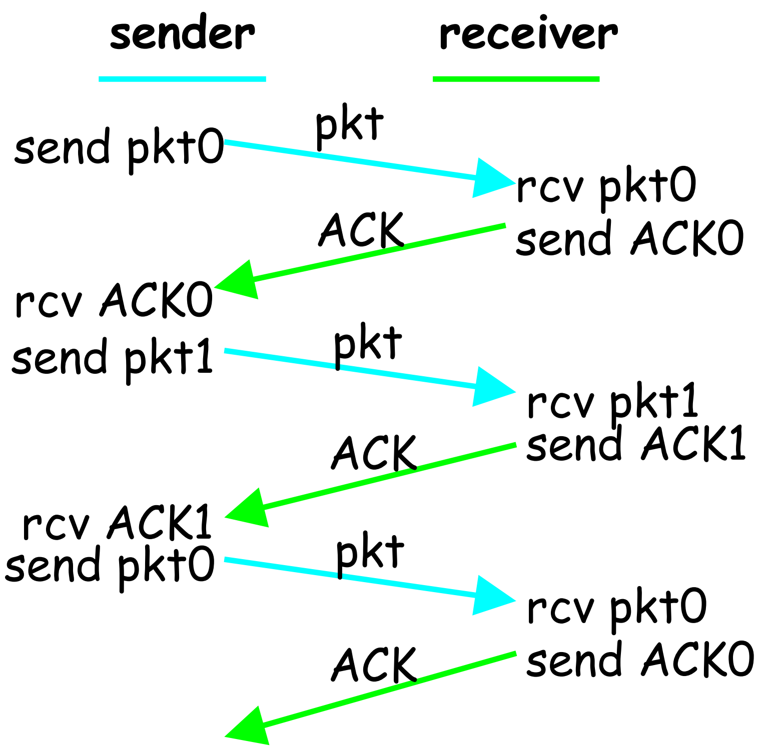
Handling duplicates:

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

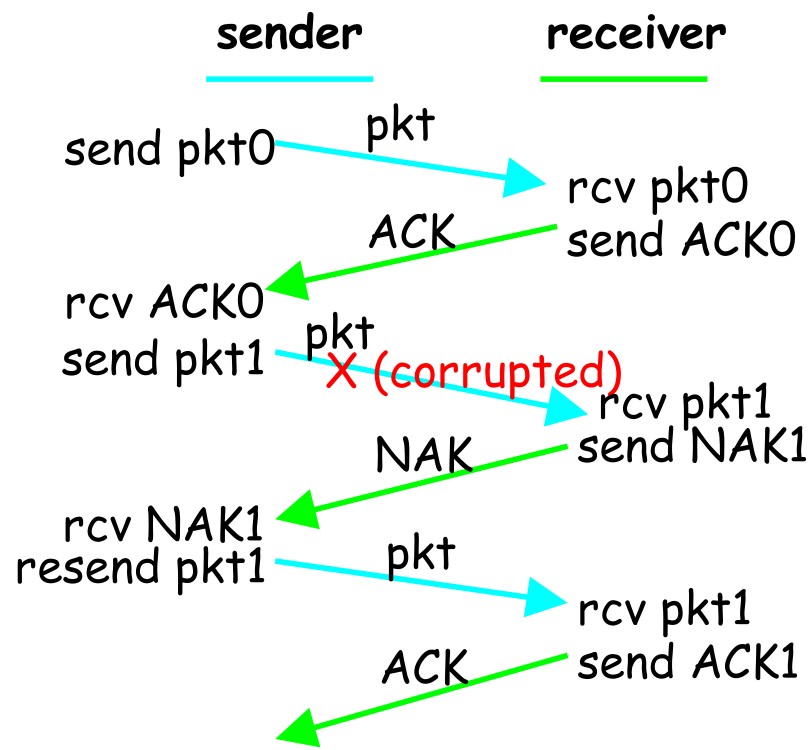
stop and wait

Sender sends one packet, then waits for receiver response

RDT 2.1 IN ACTION

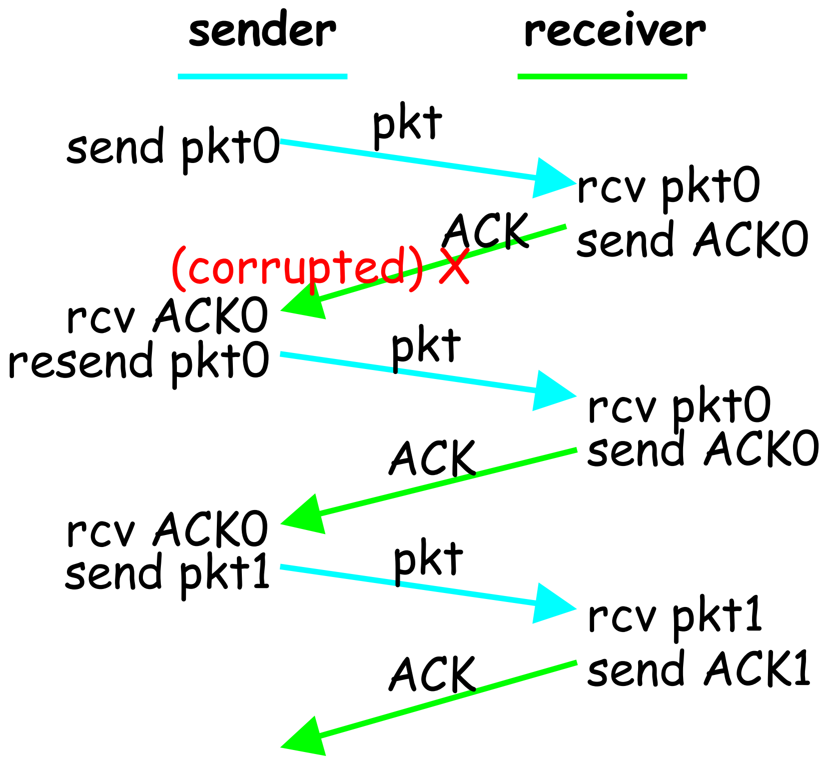


a) operation with no corruption



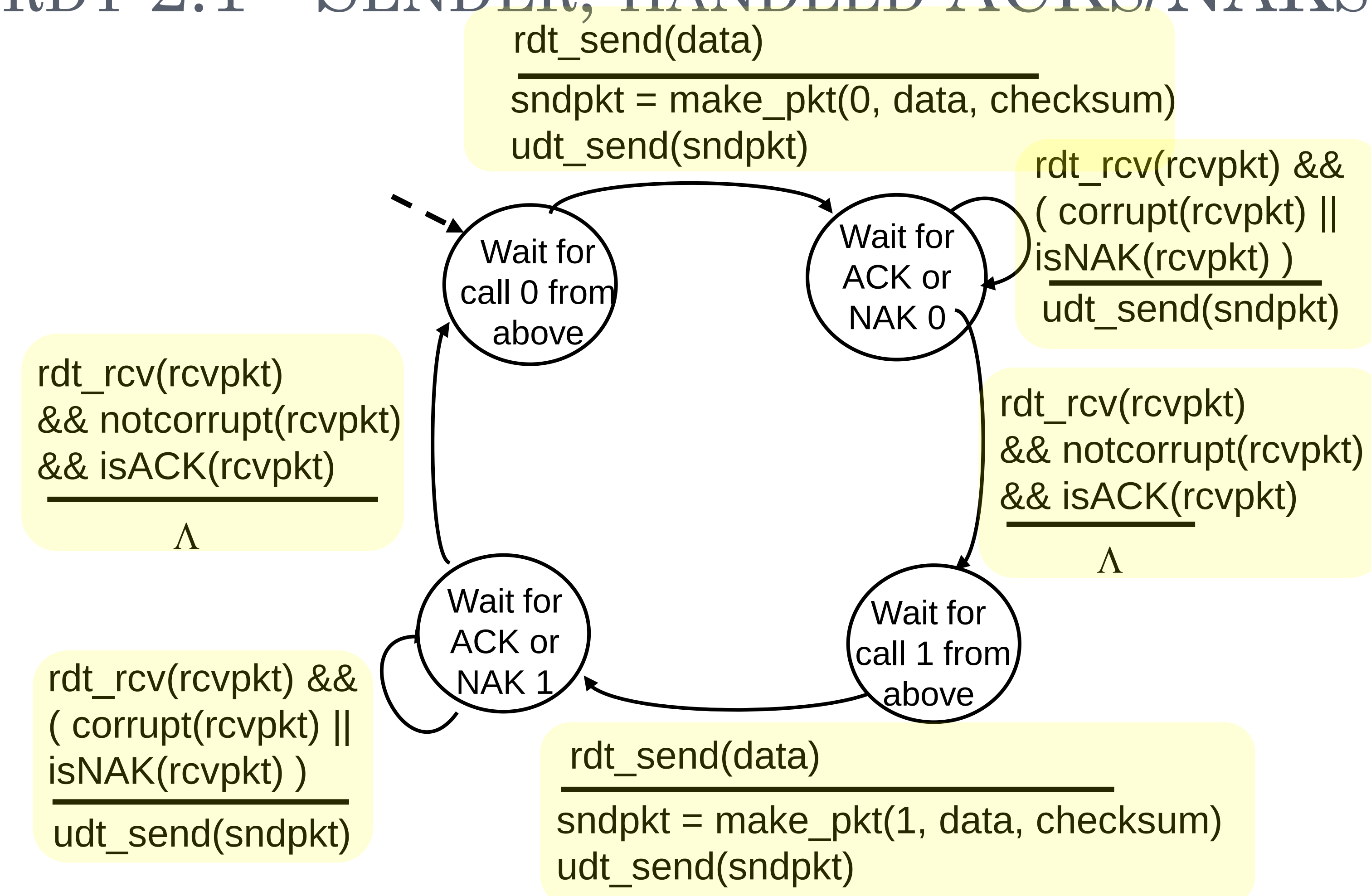
b) packet corrupted

RDT 2.1 IN ACTION (CONT)

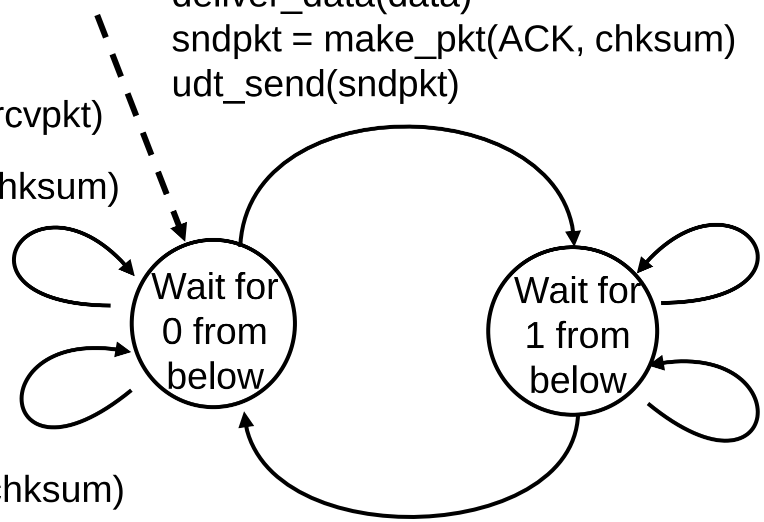


c) ACK corrupted

rdt 2.1 - Sender, handled ACKs/NAKs



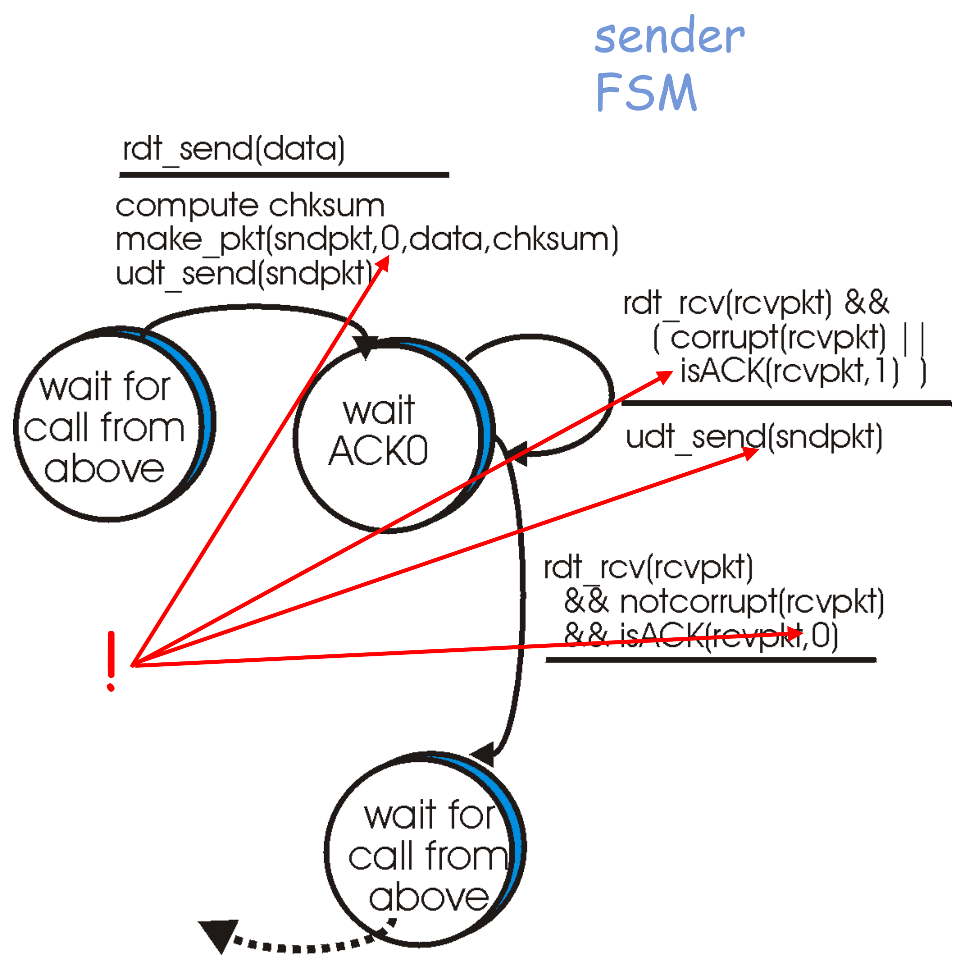
RDT 2.1 - RECEIVER, HANDLED ACKS/NAKS



RDT 2.2 & RDT 3.0

RDT2.2: A NACK-FREE PROTOCOL

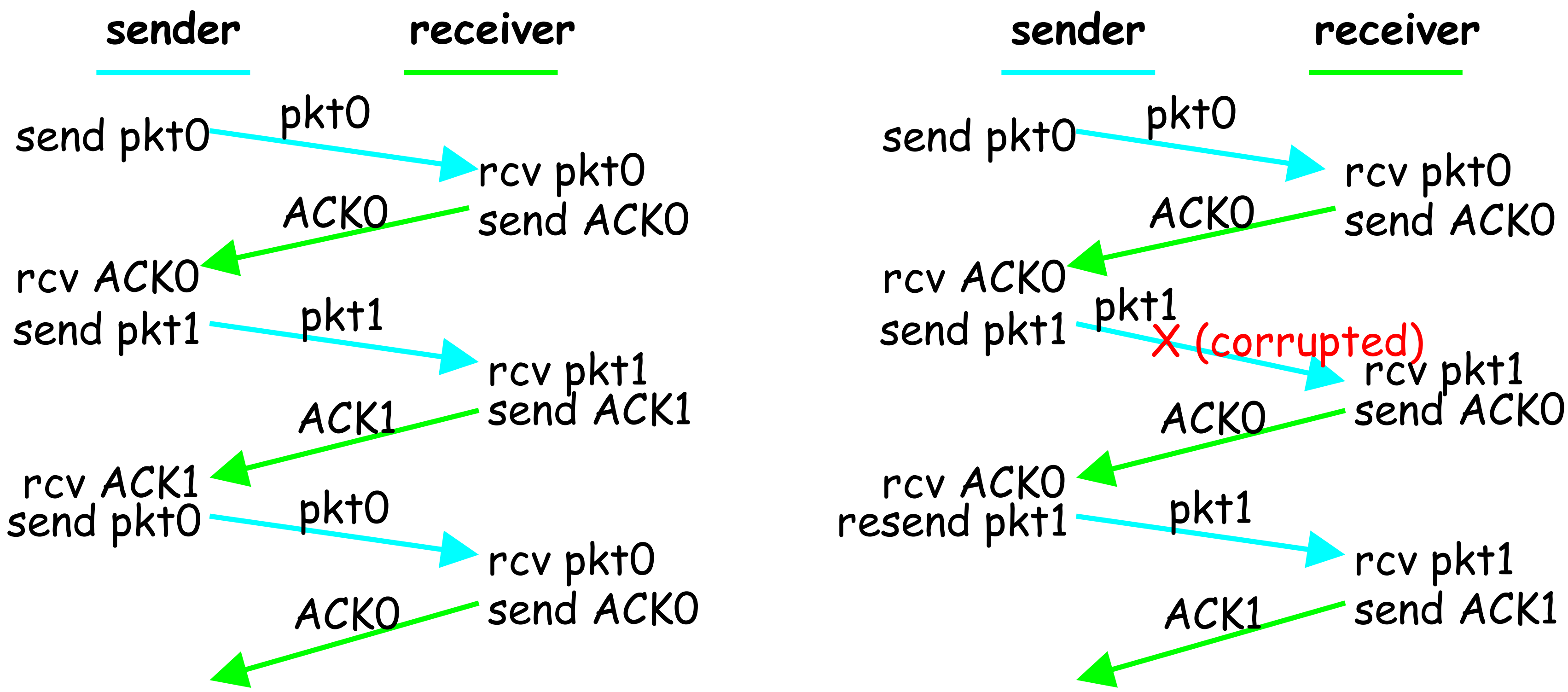
- same functionality as rdt2.1, using ACKs only
- instead of NACK, receiver sends ACK for last pkt received OK
  - receiver must *explicitly* include seq # of pkt being ACKed
- duplicate ACK at sender results in same action as NACK: *retransmit current pkt*



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.

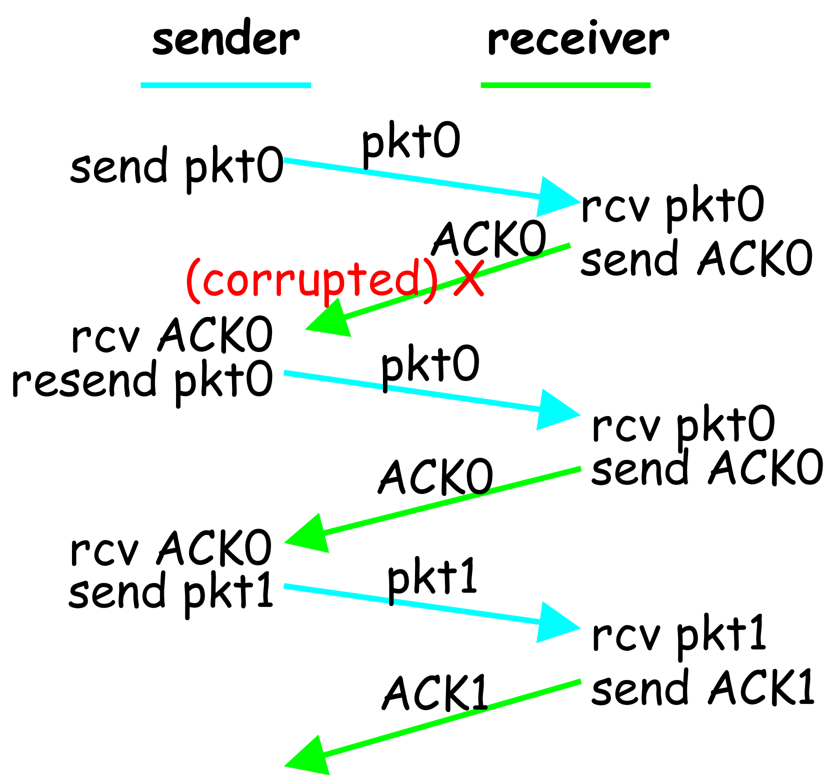
RDT 2.2 IN ACTION



a) operation with no corruption

b) packet corrupted

RDT 2.2 IN ACTION (CONT)

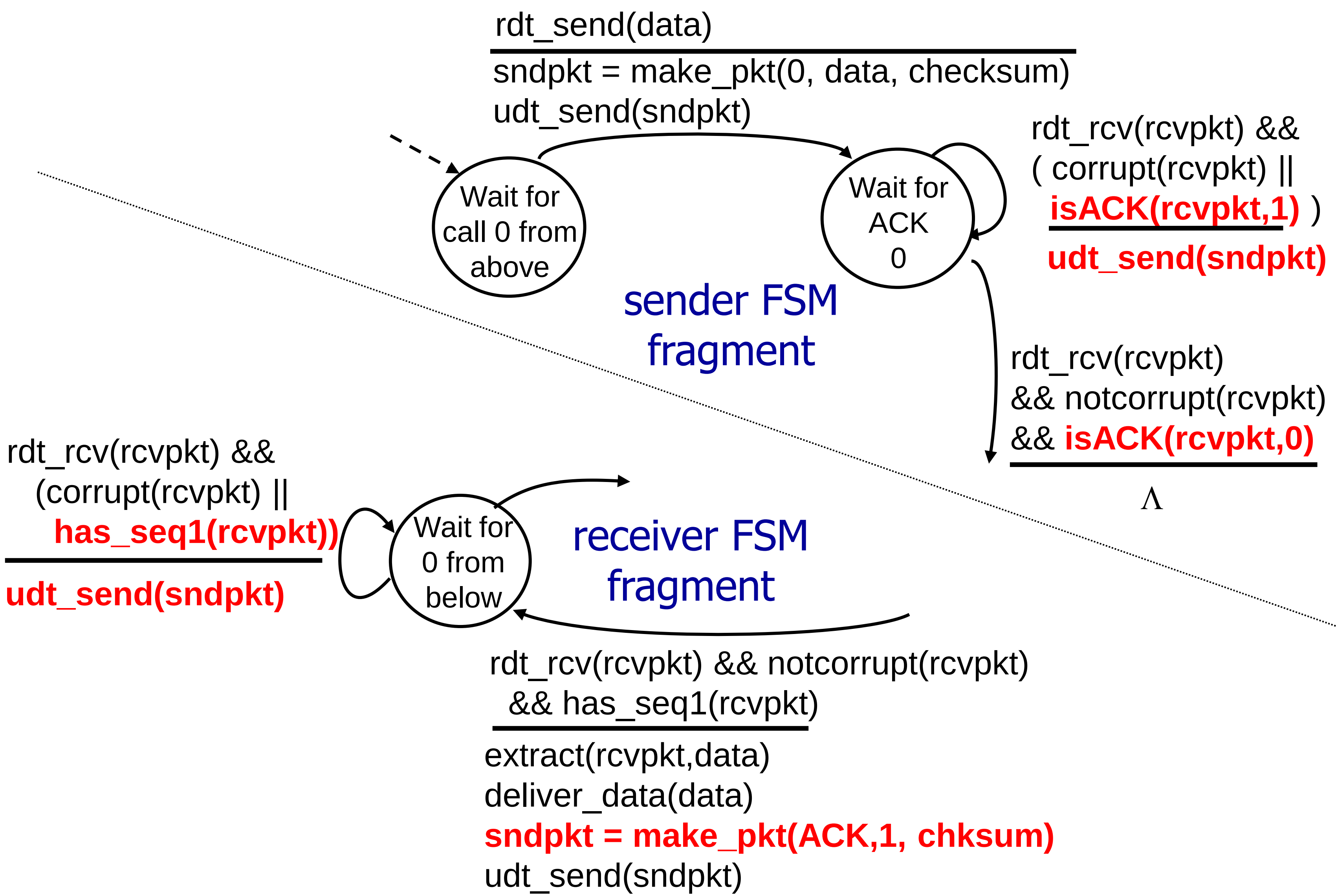


c) ACK corrupted

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 2.2 – SENDER AND RECEIVER



Channel uc 3.0

- FIFO:
  - Data packets and Ack packets are delivered in order.
- Errors and Loss:
  - Data and ACK packets might get corrupt or lost
- No duplication: but can handle it!
- Liveness:
  - If continuously sending packets, eventually, an uncorrupted packet received.

RDT3.0: CHANNELS WITH ERRORS *AND* LOSS

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

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

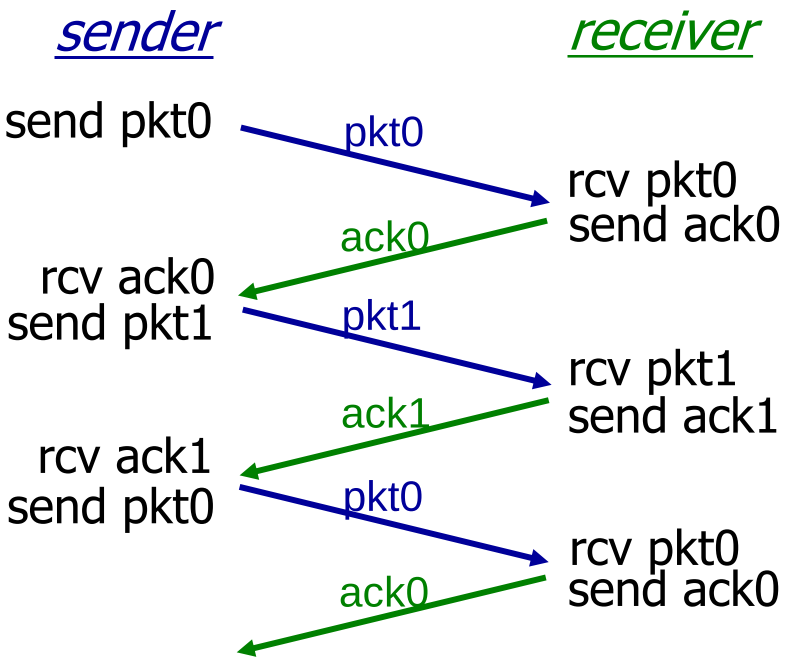
Q: how to deal with loss?

- sender waits until certain data or ACK lost, then retransmits
- feasible?

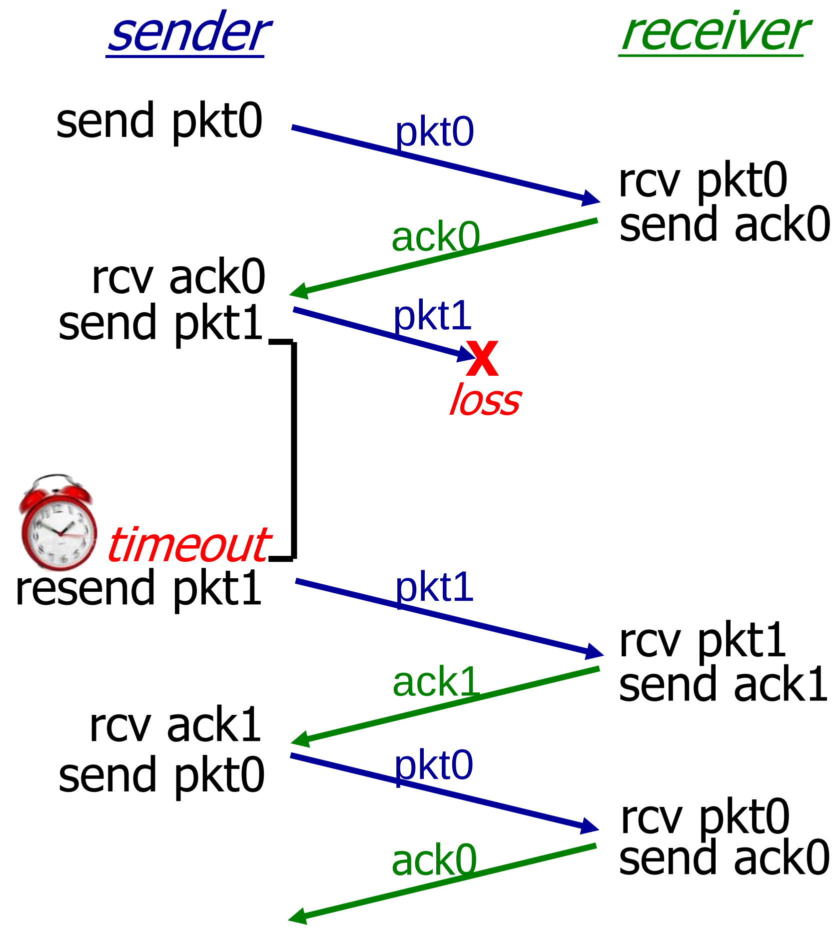
Approach: sender waits "reasonable" amount of time for ACK

- 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

Rdt 3.0: Alternating-bit protocol

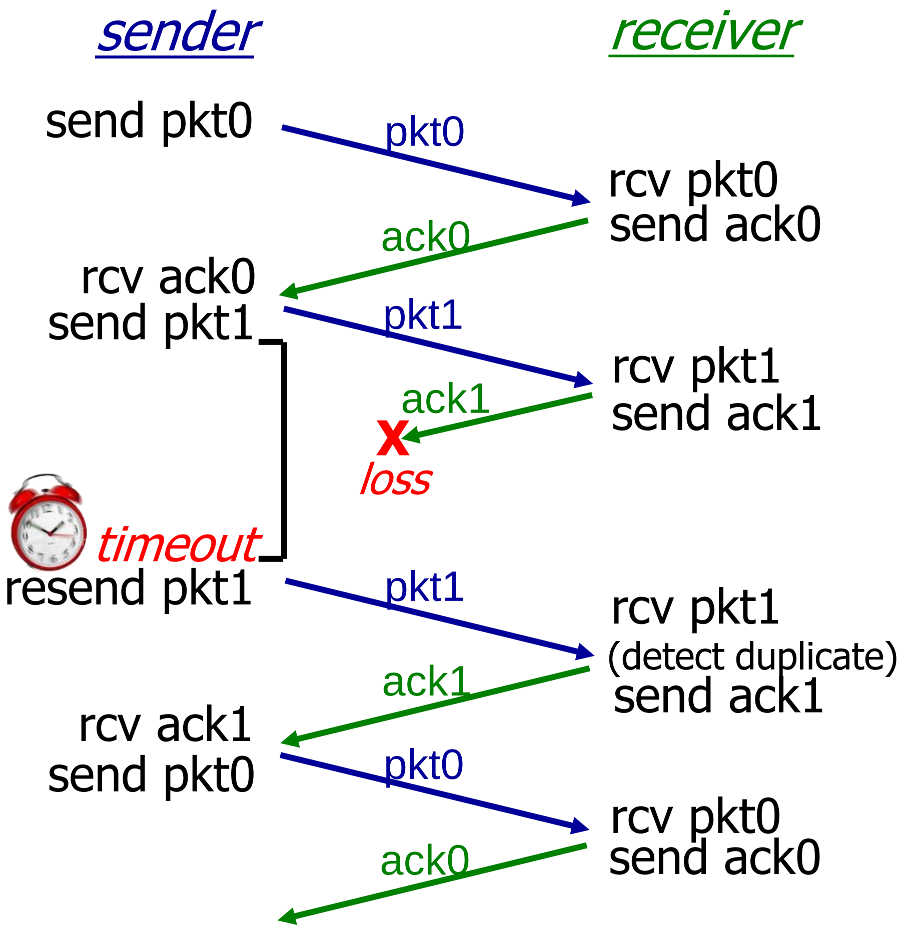


(a) no loss

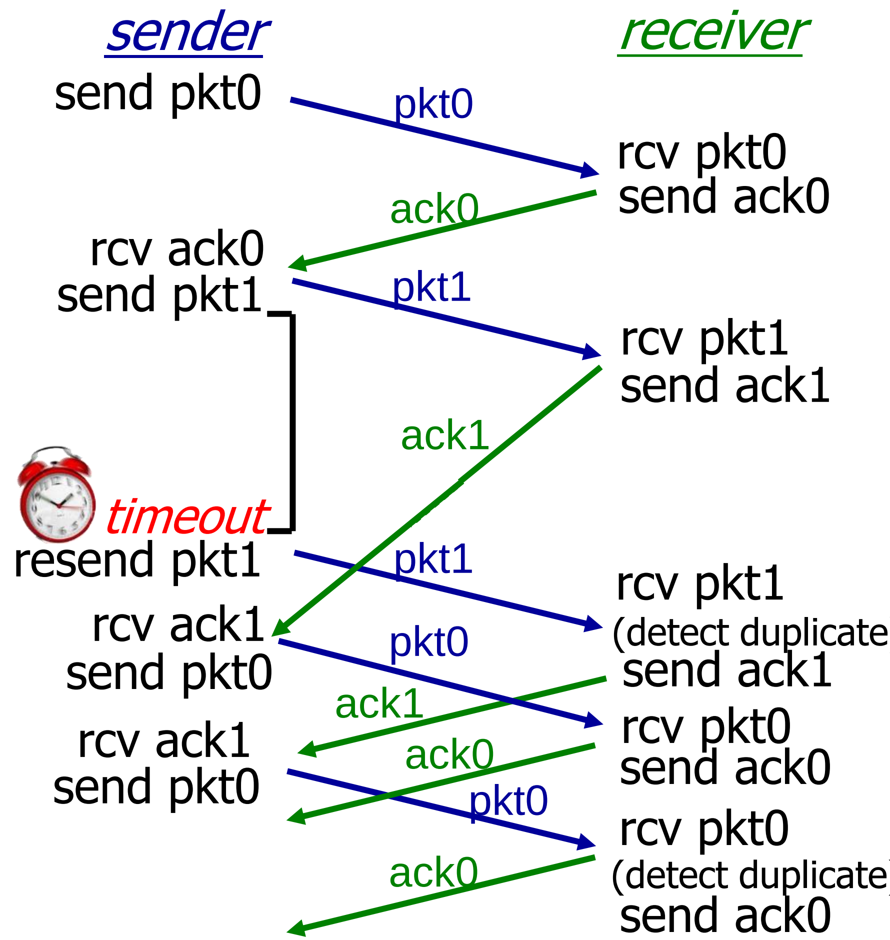


(b) packet loss

RDT 3.0 – CONT…

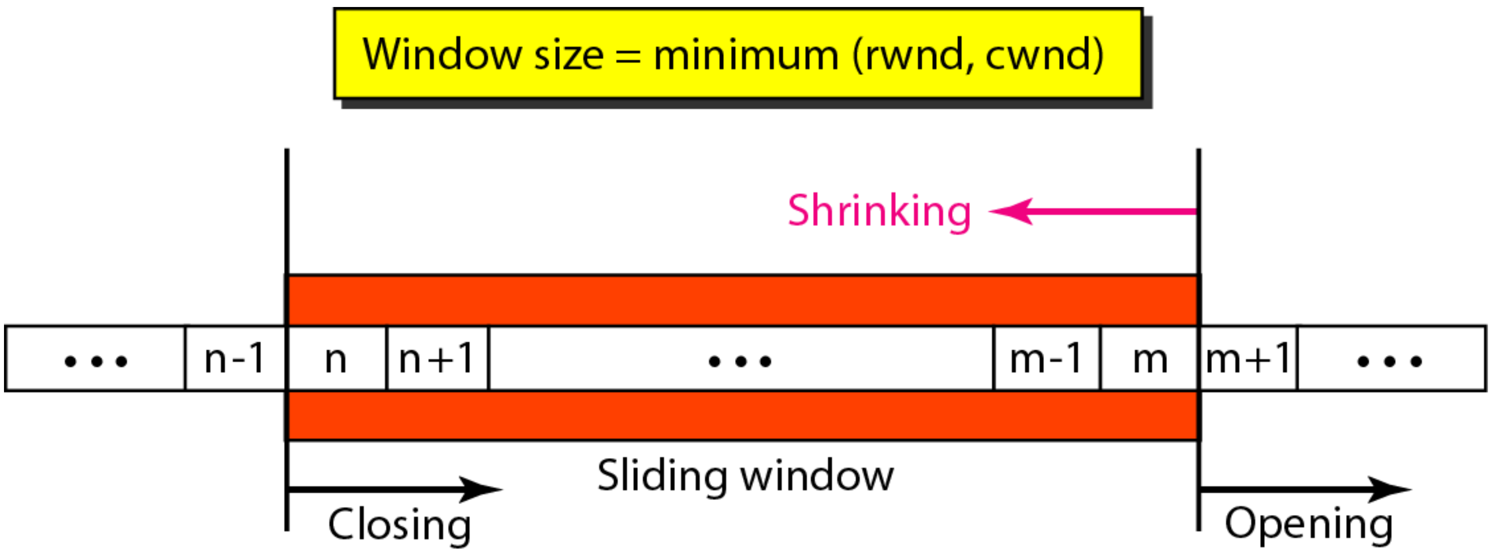


(c) ACK loss



(d) premature timeout/ delayed ACK

Figure 23.22 *Sliding window*

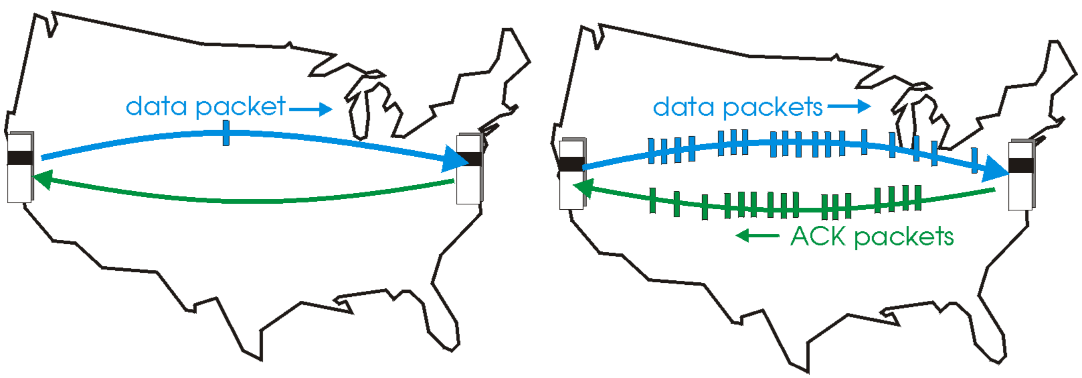


Pipelined Protocol

- 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.



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

- Two generic forms of pipelined protocols:
  1. *Go-back-N*
  2. *Selective Repeat*

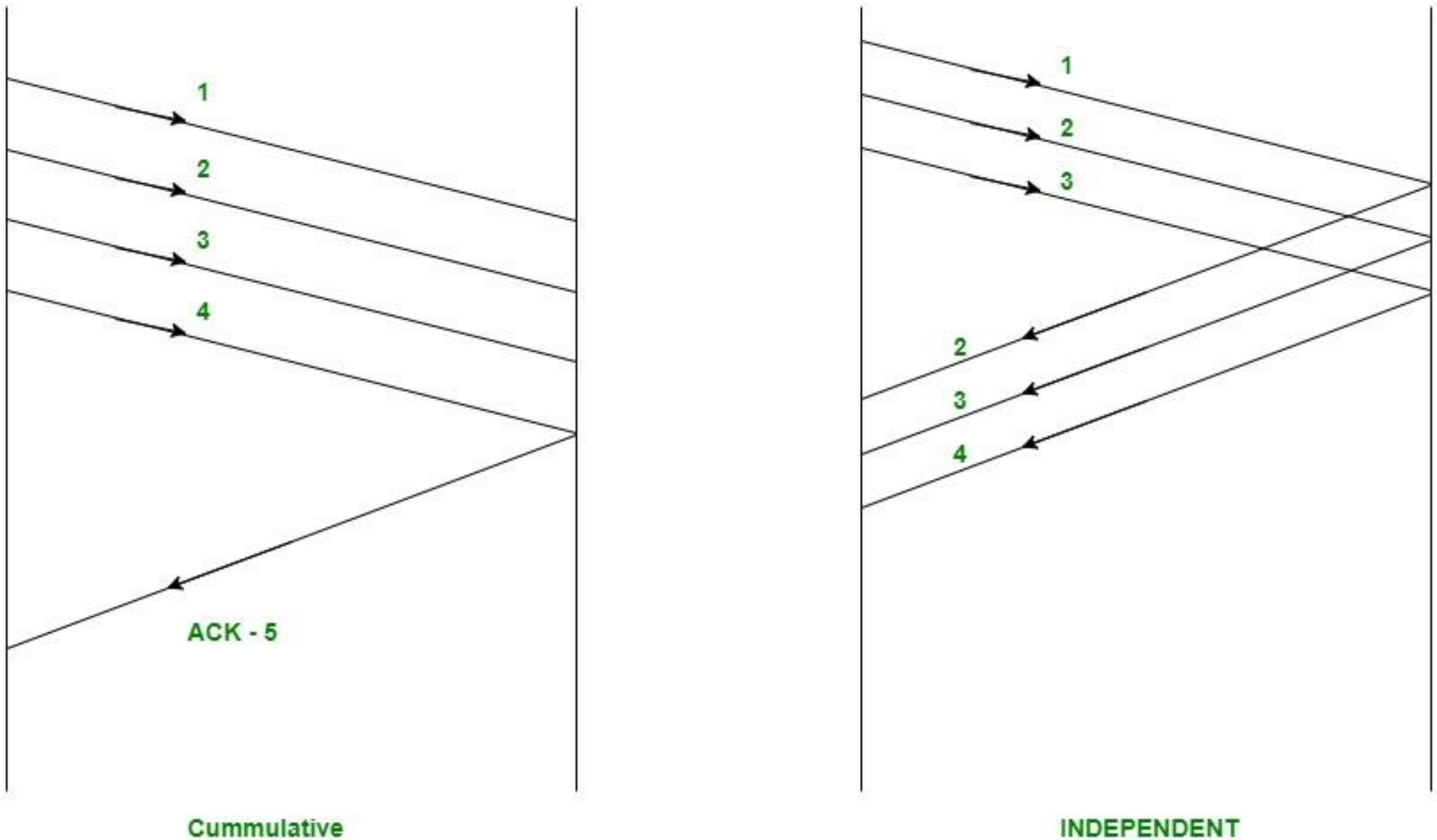
Go-back-N Protocol

- 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.

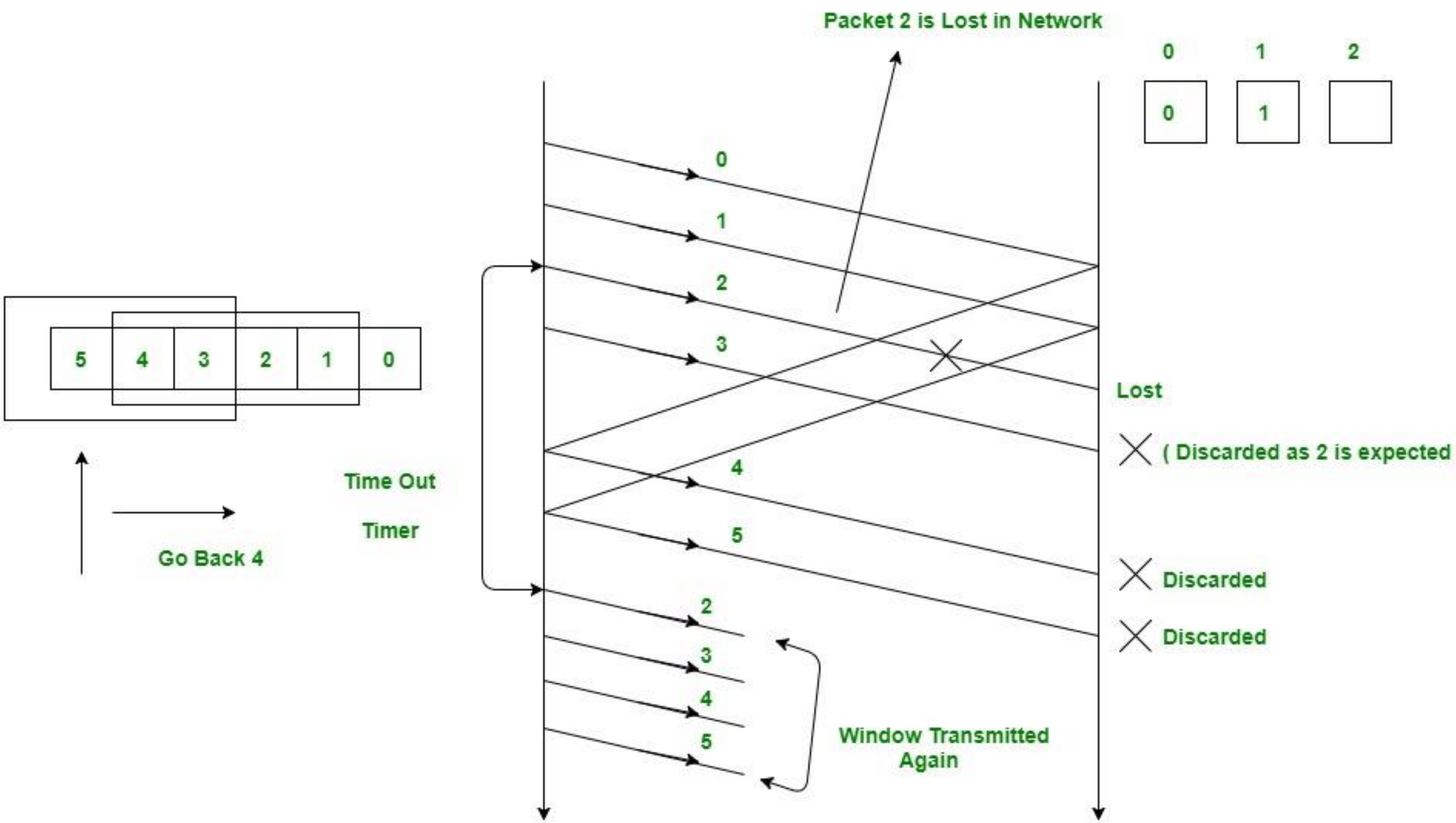
Go-back-N Protocol Cont…

- 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.

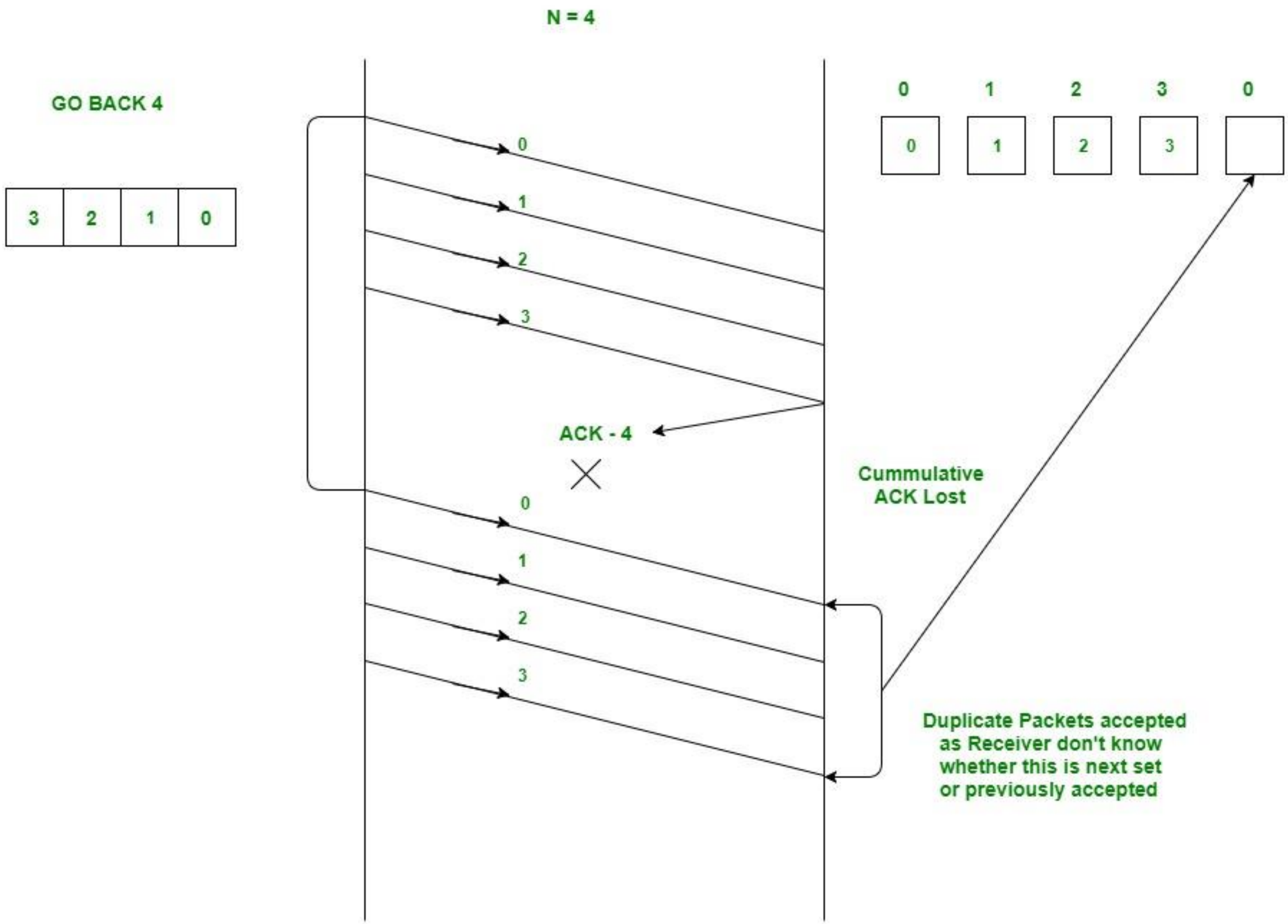
Acknowledgement



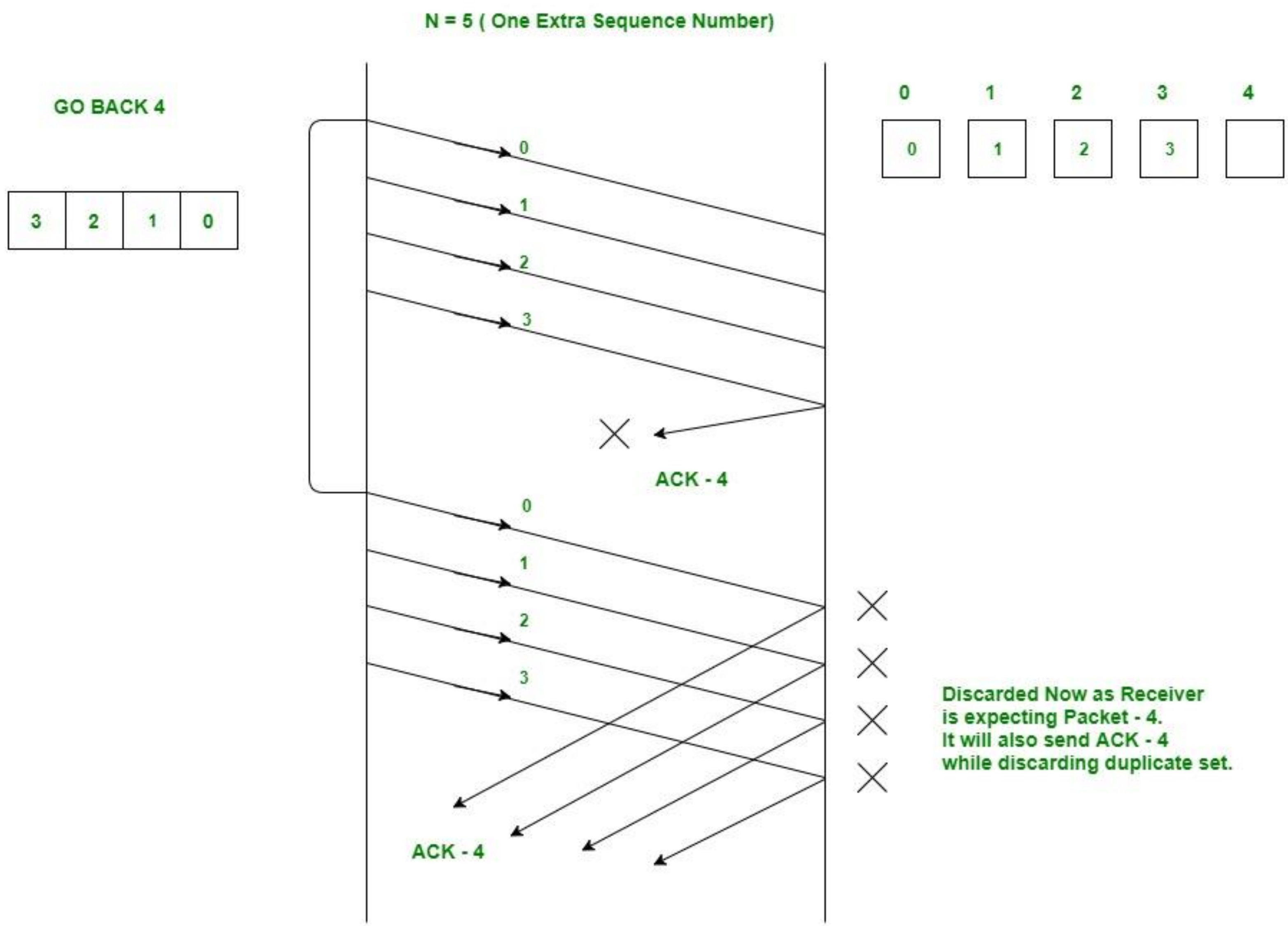
Receiver Window Size = 1



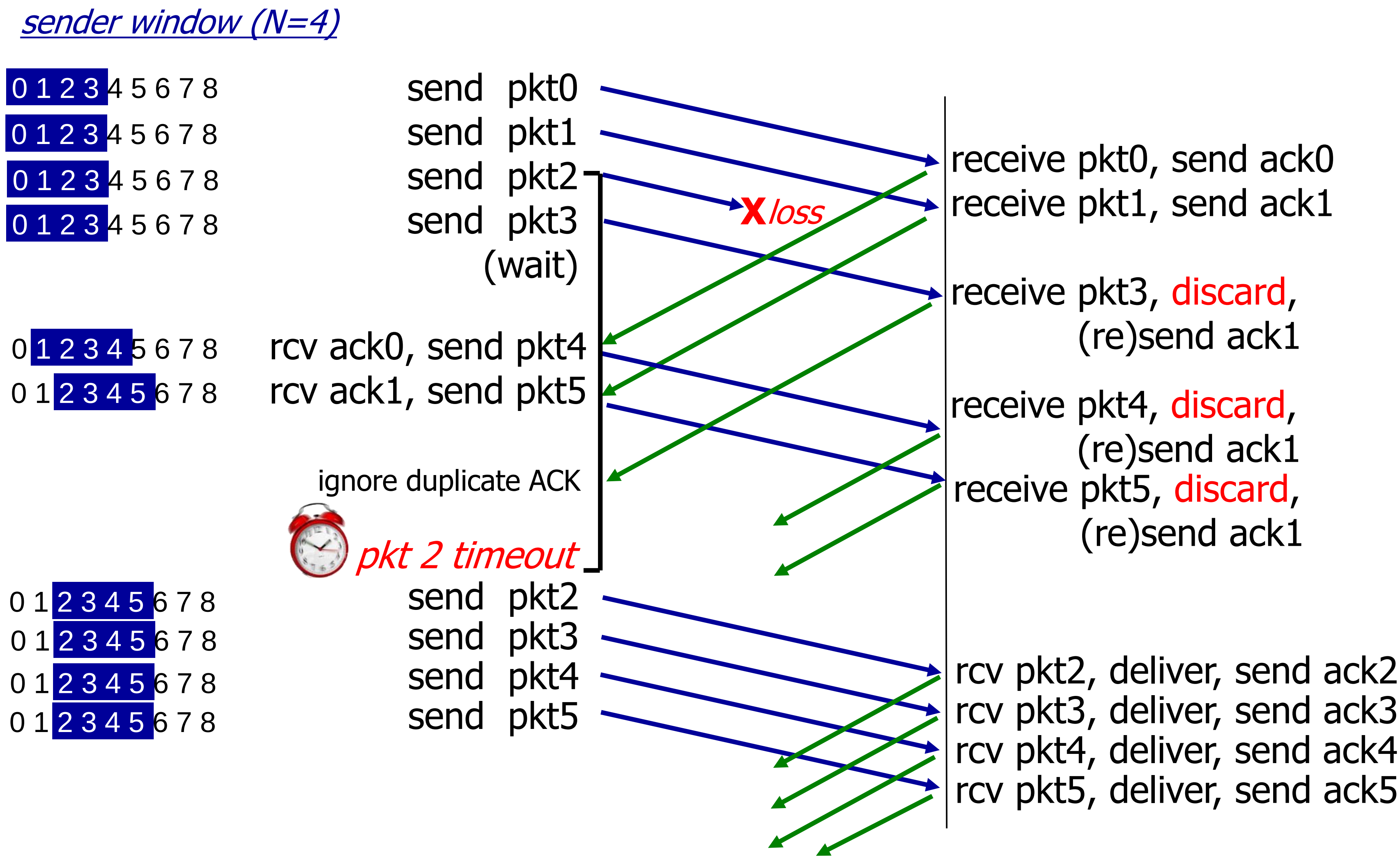
WITHOUT SEQUENCE NUMBER

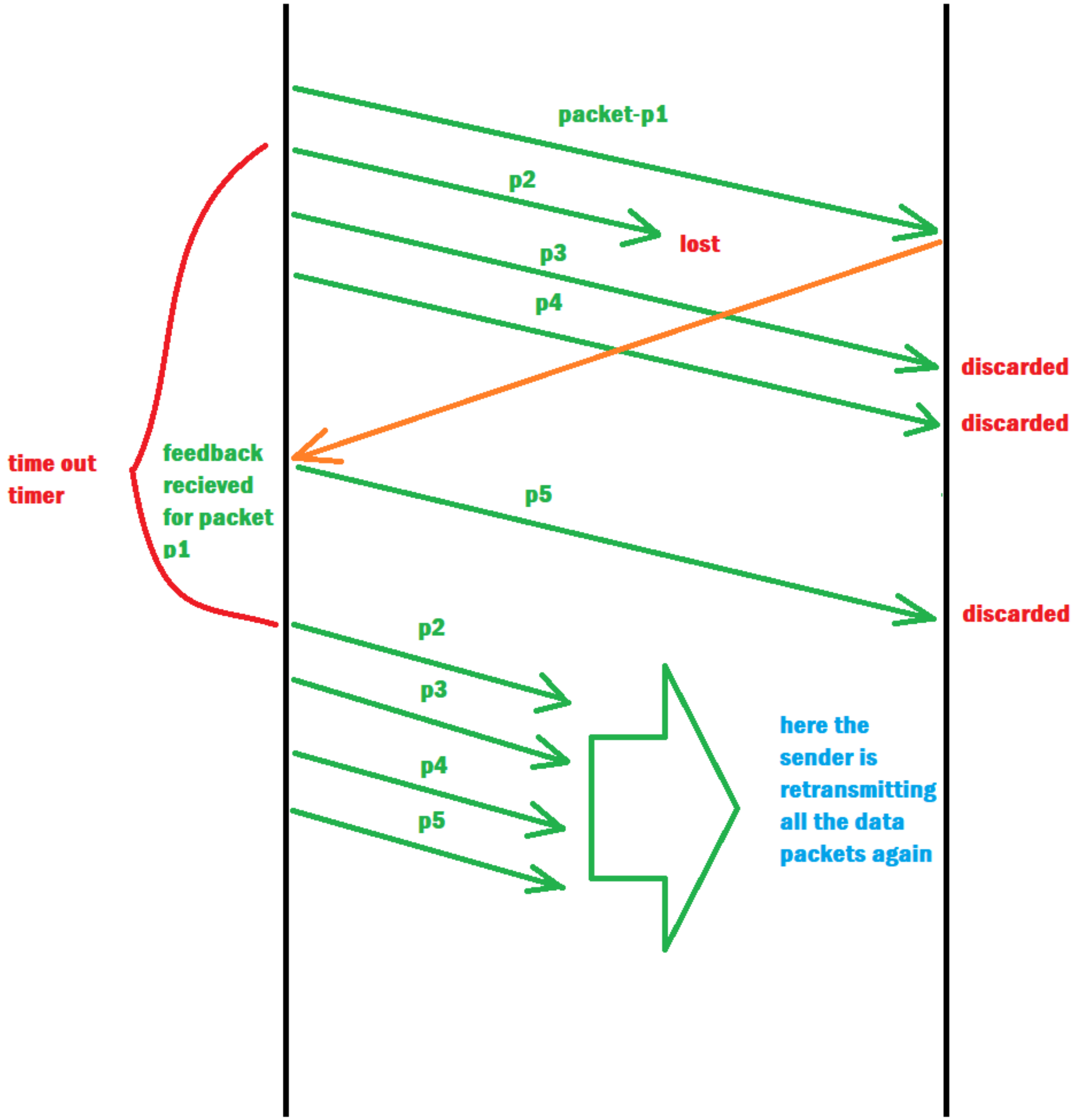


With Sequence Number



Go-back-N Protocol works




packet-p1
p2
lost
p3
p4
discarded
discarded
time out
timer
feedback
recieved
for packet
p1
p5
discarded
p2
p3
p4
p5
here the
sender is
retransmitting
all the data
packets again

Selective Repeat

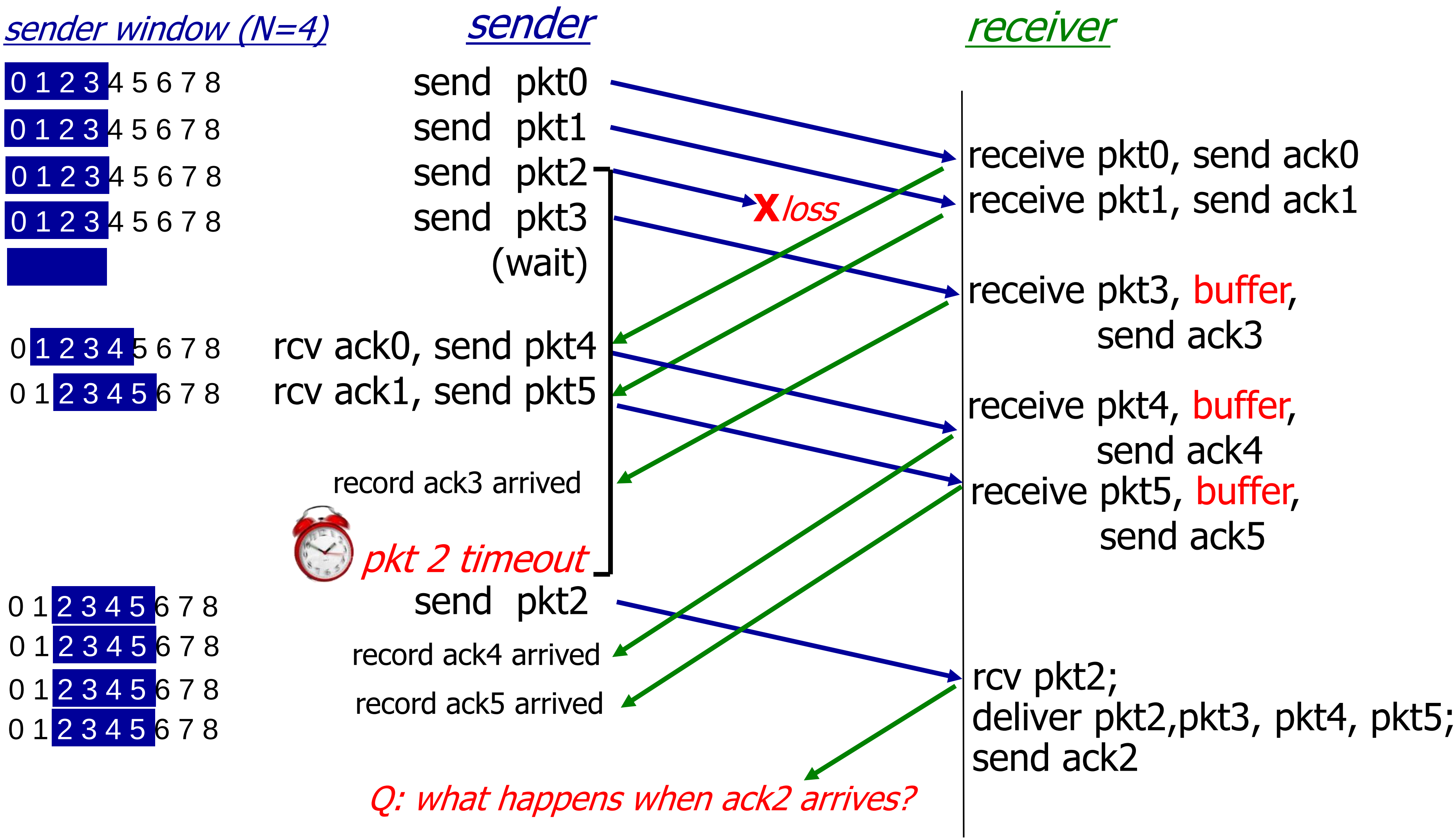
Selective Repeat

- 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



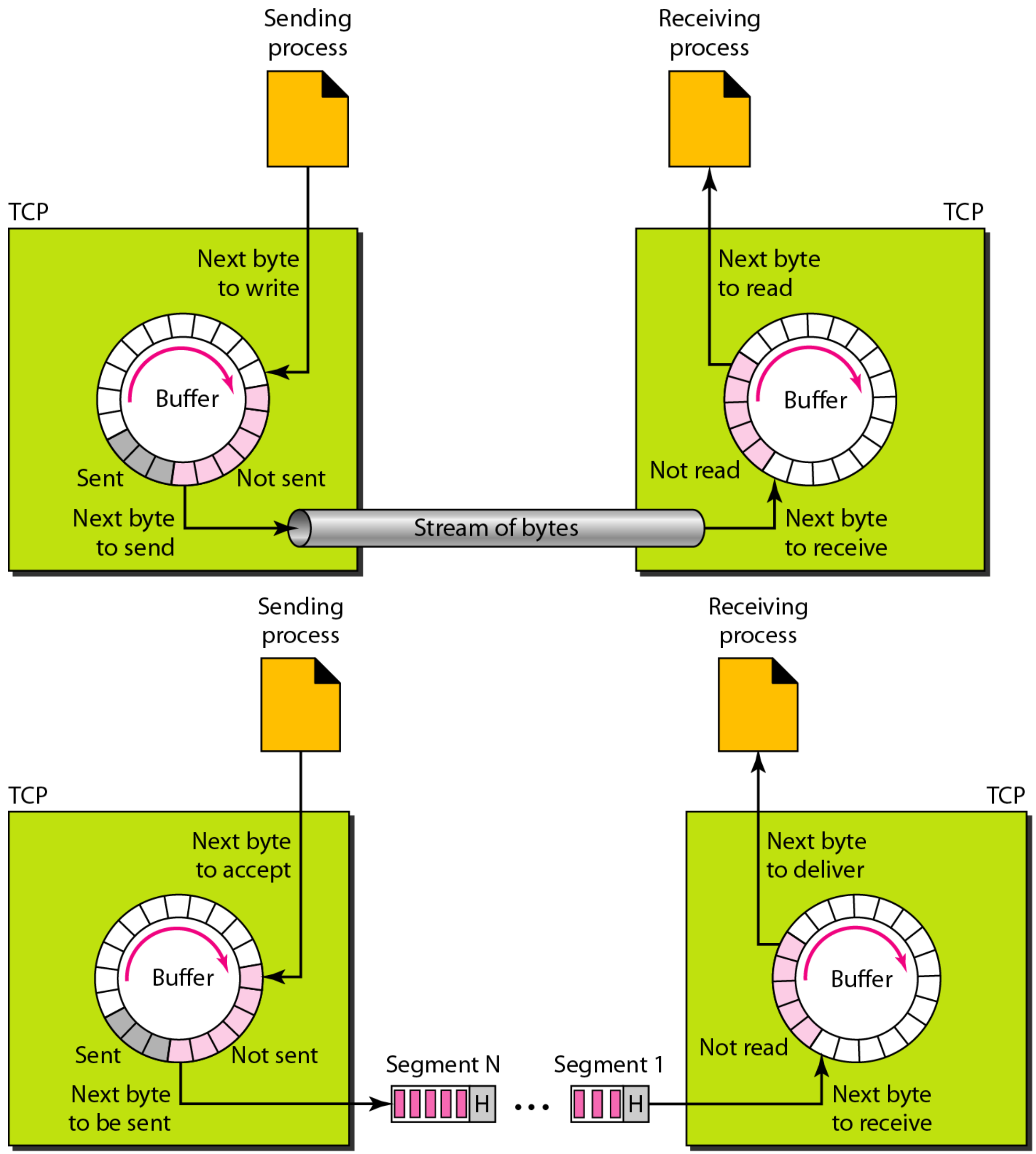
TCP (Transmission Control Protocol)

- *TCP is a connection-oriented protocol;*
- *it creates a virtual connection between two TCPs to send data.*
- *In addition, TCP uses flow and error control mechanisms at the transport level.*

WELL-KNOWN PORTS USED BY TCP

Port	*Protocol*	*Description*
7	Echo	Echoes a received datagram back to the sender
9	Discard	Discards any datagram that is received
11	Users	Active users
13	Daytime	Returns the date and the time
17	Quote	Returns a quote of the day
19	Chargen	Returns a string of characters
20	FTP, Data	File Transfer Protocol (data connection)
21	FTP, Control	File Transfer Protocol (control connection)
23	TELNET	Terminal Network
25	SMTP	Simple Mail Transfer Protocol
53	DNS	Domain Name Server
67	BOOTP	Bootstrap Protocol
79	Finger	Finger
80	HTTP	Hypertext Transfer Protocol
111	RPC	Remote Procedure Call

SENDING AND RECEIVING BUFFERS & TCP SEGMENTS



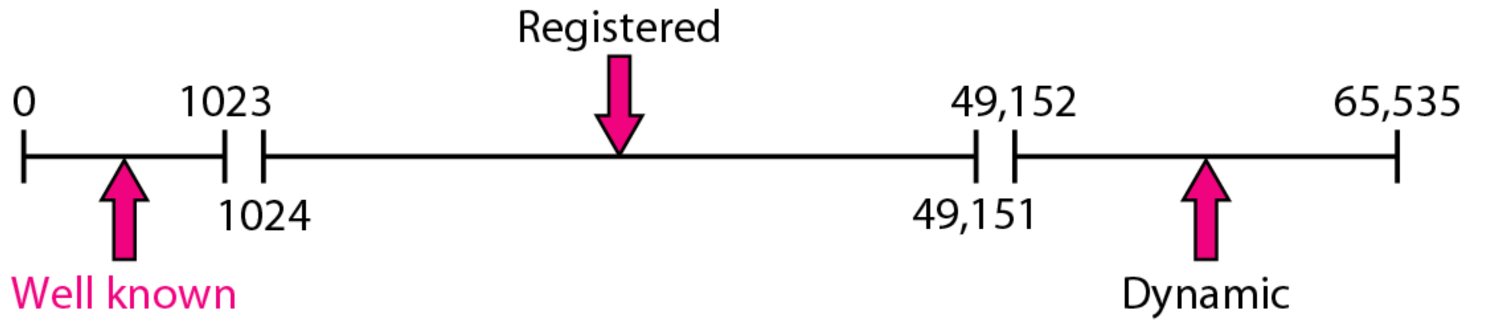
- The bytes of data being transferred in each connection are numbered by TCP.
- The numbering starts with a randomly generated number.
- The value in the sequence number field of a segment defines the number of the first data byte contained in that segment.
- The value of the acknowledgment field in a segment defines the number of the next byte a party expects to receive.
- The acknowledgment number is cumulative.

The following shows the sequence number for each segment:

Segment 1	→	**Sequence Number: 10,001 (range: 10,001 to 11,000)**
Segment 2	→	**Sequence Number: 11,001 (range: 11,001 to 12,000)**
Segment 3	→	**Sequence Number: 12,001 (range: 12,001 to 13,000)**
Segment 4	→	**Sequence Number: 13,001 (range: 13,001 to 14,000)**
Segment 5	→	**Sequence Number: 14,001 (range: 14,001 to 15,000)**

TCP Segment Structure

IANA RANGES



Registered
0
1023
49,152
65,535
1024
49,151
Well known
Dynamic

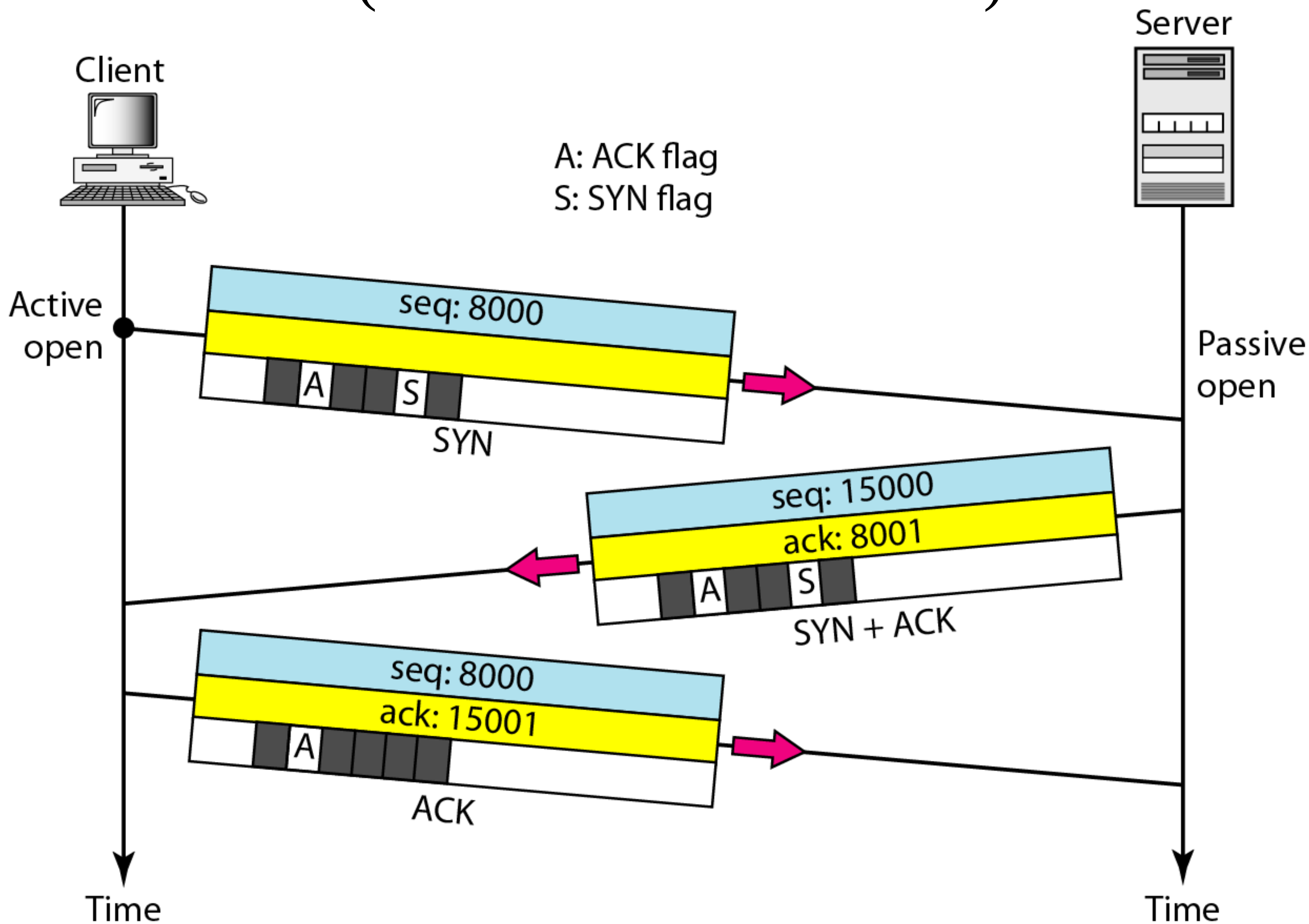
CONTROL FIELD & DESCRIPTION OF FLAGS

URG: Urgent pointer is valid
ACK: Acknowledgment is valid
PSH: Request for push
RST: Reset the connection
SYN: Synchronize sequence numbers
FIN: Terminate the connection

URG	ACK	PSH	RST	SYN	FIN

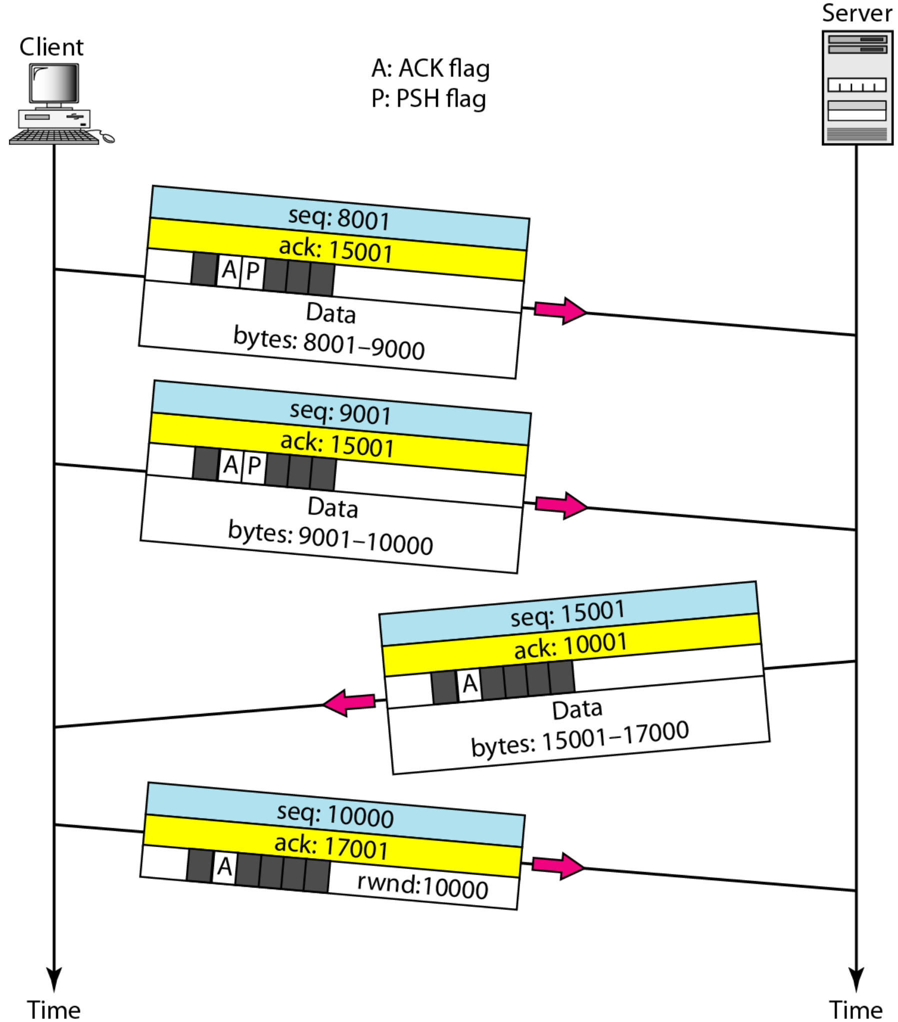
Flag	*Description*
URG	The value of the urgent pointer field is valid.
ACK	The value of the acknowledgment field is valid.
PSH	Push the data.
RST	Reset the connection.
SYN	Synchronize sequence numbers during connection.
FIN	Terminate the connection.

TCP Connection establishment using three-way handshaking (Connection Oriented)

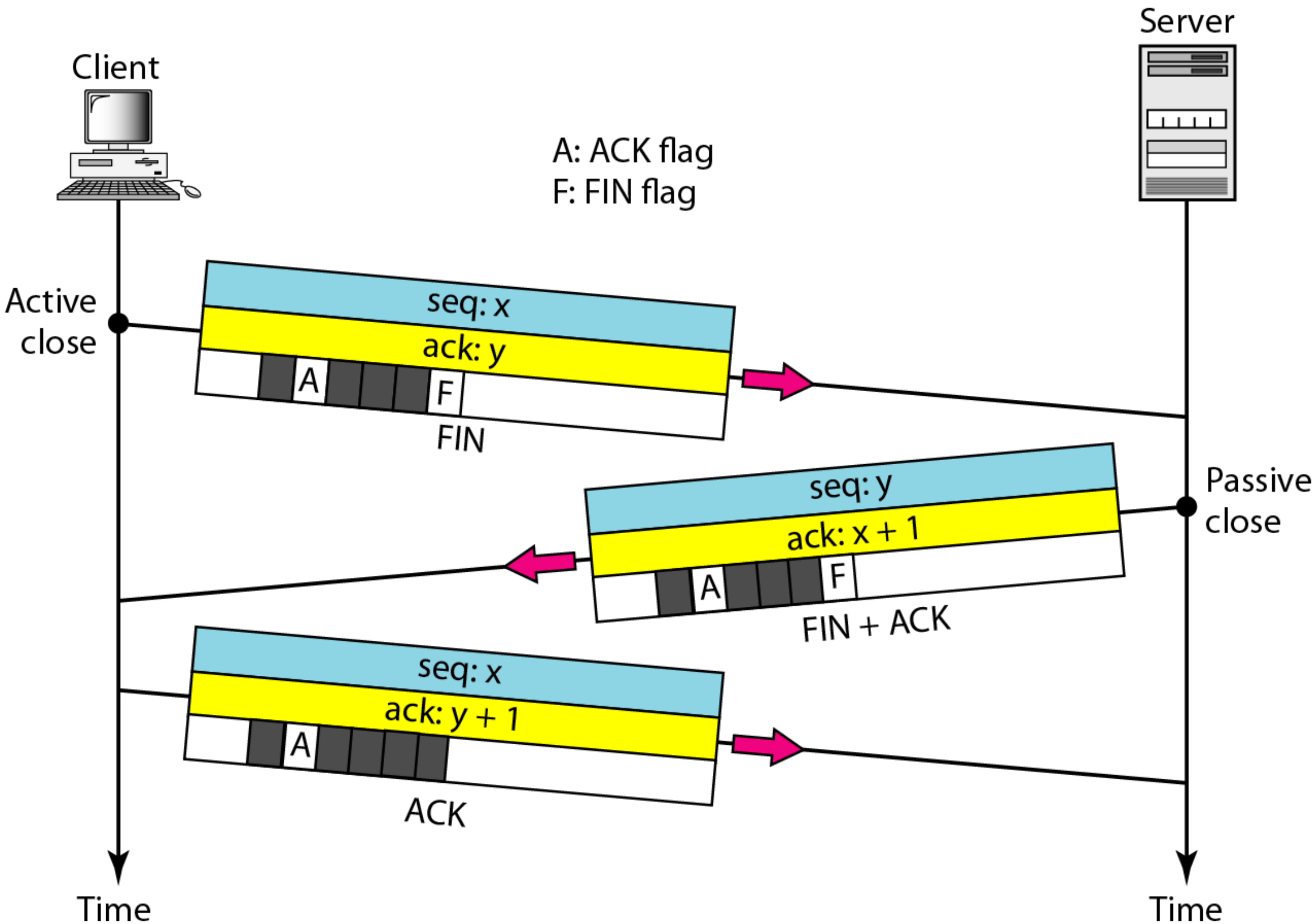


- A SYN segment cannot carry data, but it consumes one sequence number.
- A SYN + ACK segment cannot carry data, but does consume one sequence number.
- An ACK segment, if carrying no data, consumes no sequence number.

DATA TRANSFER

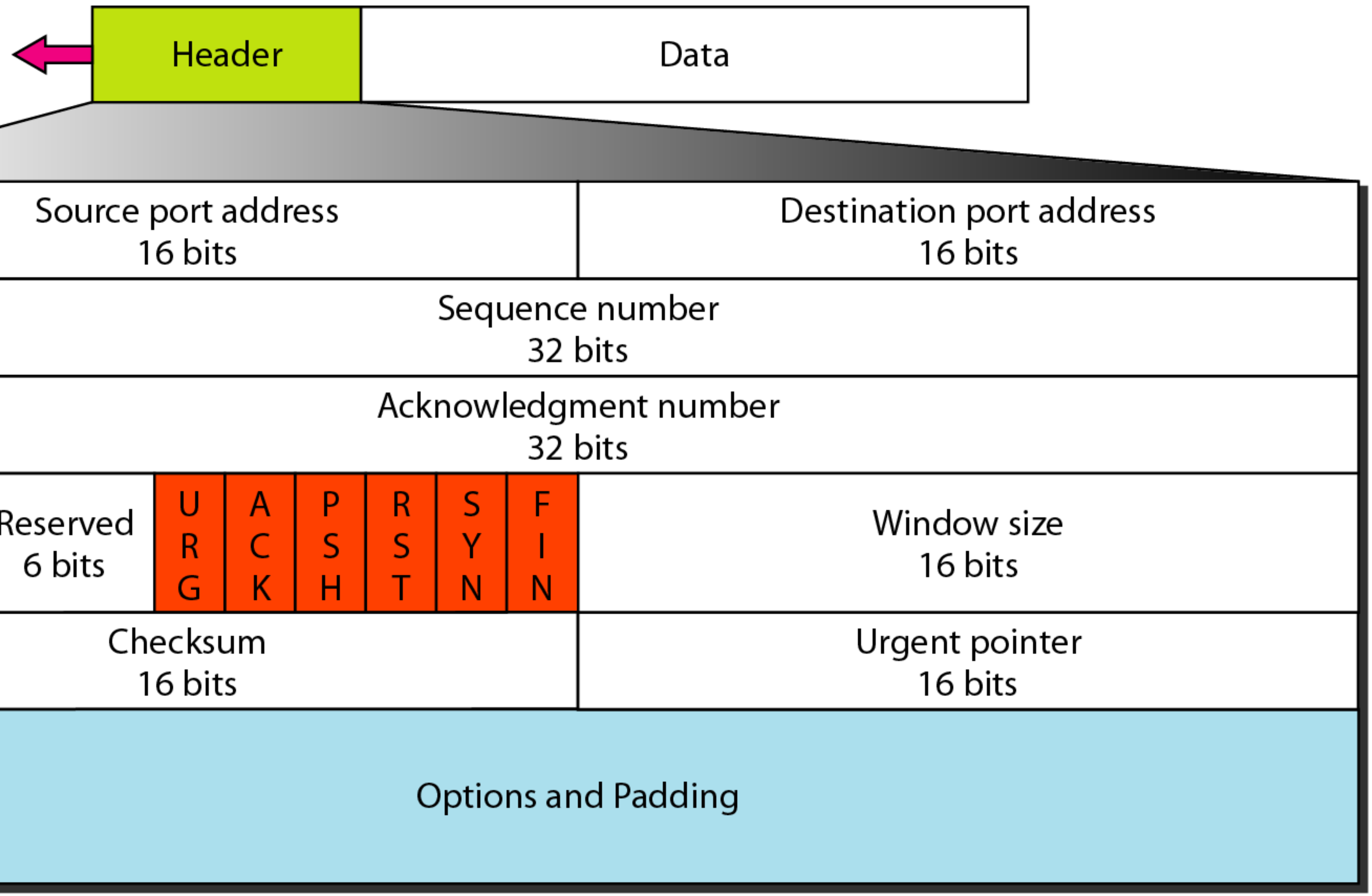


CONNECTION TERMINATION USING THREE-WAY HANDSHAKING



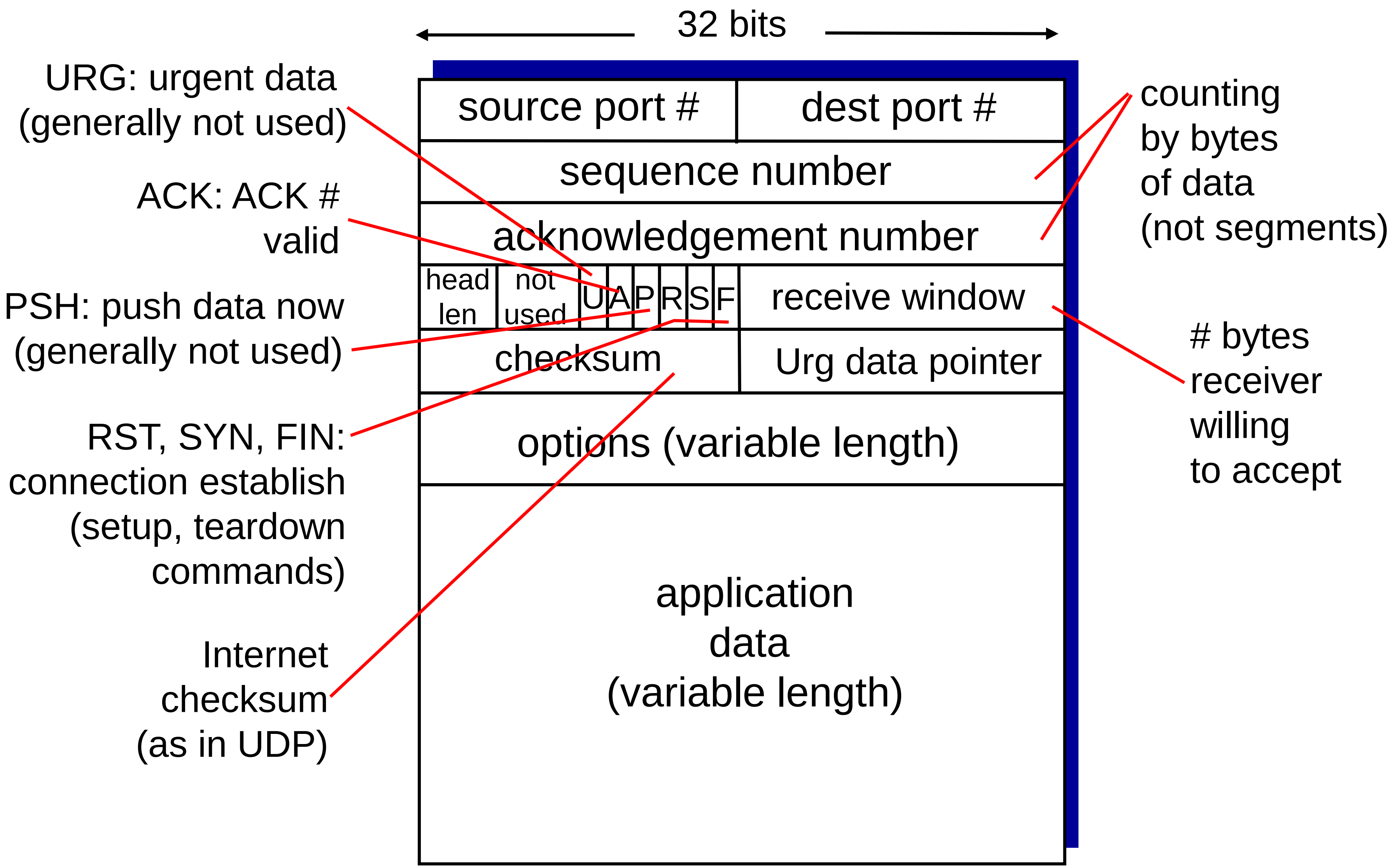
- The FIN segment consumes one sequence number if it does not carry data.
- The FIN + ACK segment consumes one sequence number if it does not carry data.

TCP SEGMENT FORMAT



- TCP Header Size= Min 20 to Max 60 Bytes
- (i.e. 20 Bytes = 20 x 8 = 160 Bits)

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 &
Congestion Control

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.

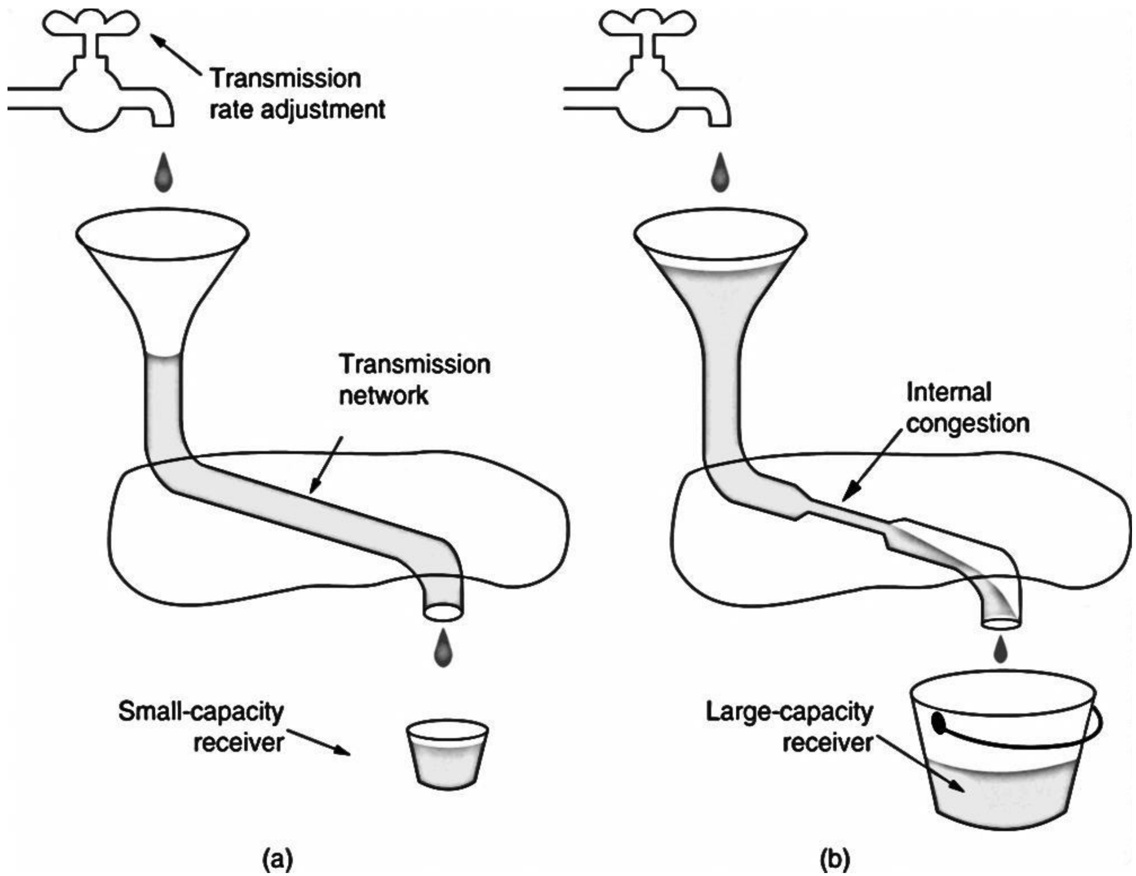
Congestion Control

- 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.

Effects of Congestion

- Delay Increase
  - If Delay Increases, retransmission occurs, making situation worse.
- Performance Decrease

Congestion Control – Cont…



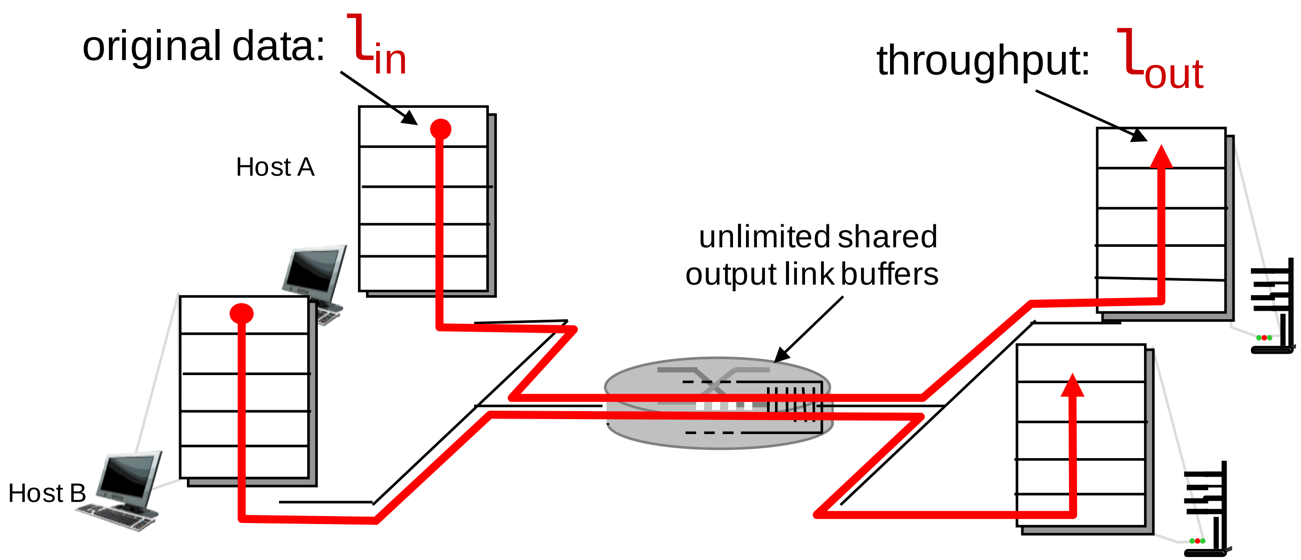
In Figure(a), we see a thick pipe leading to a small-capacity receiver.

As long as the sender does not send more water than the bucket can contain, no water will be lost.

In Figure(b), the limiting factor is not the bucket capacity, but the internal carrying capacity of the network.

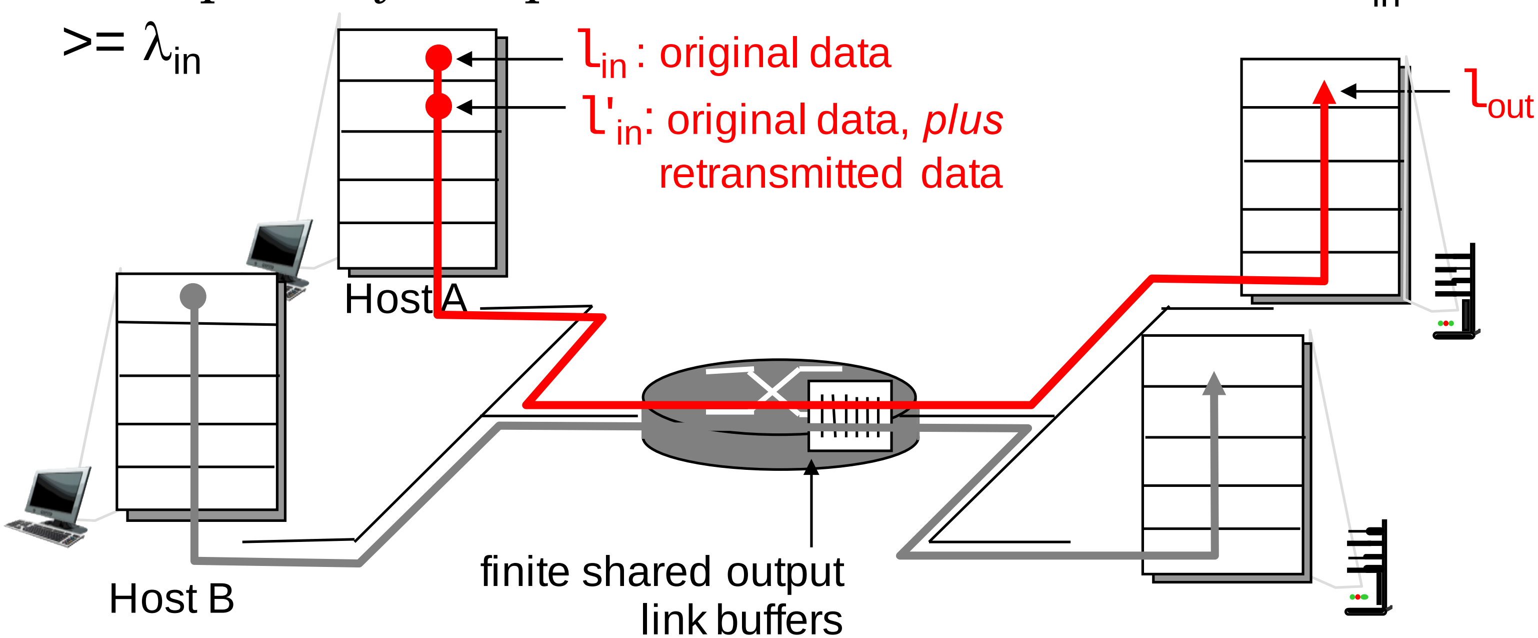
Causes/costs of Congestion: Scenario 1

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



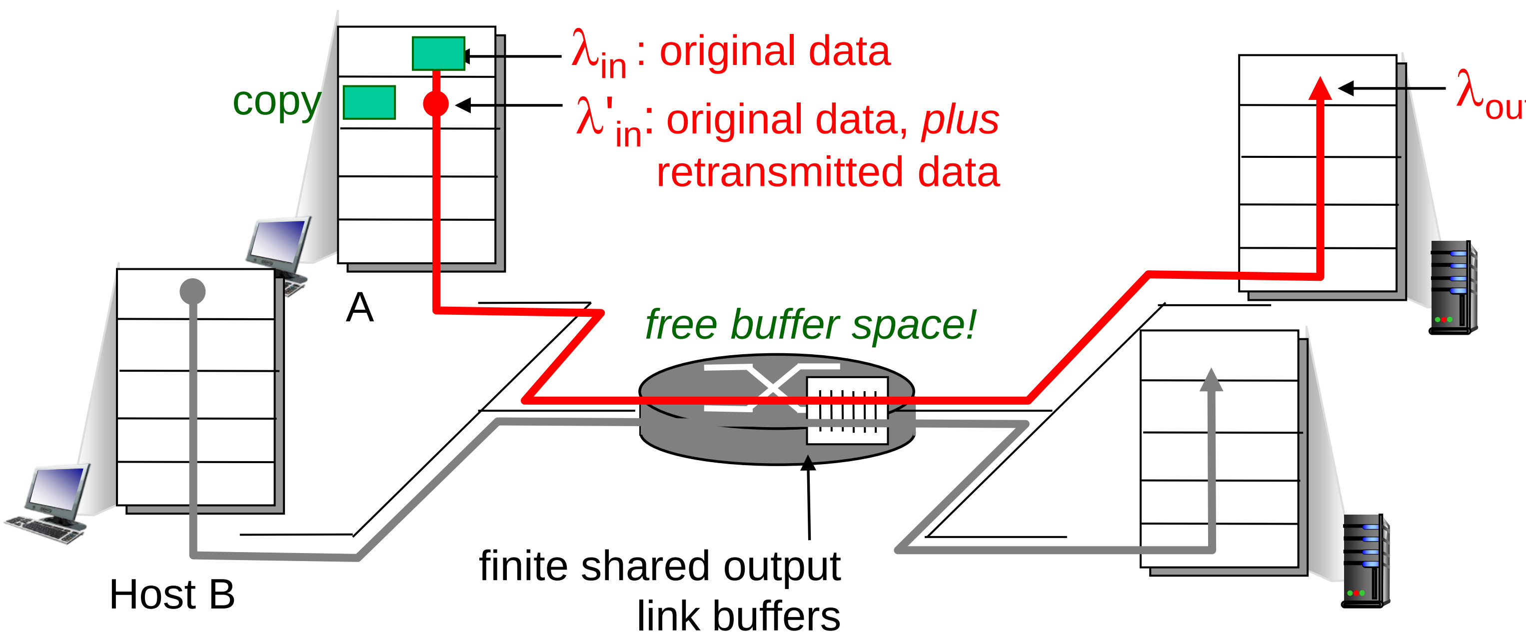
Causes/costs of Congestion: Scenario 2

- 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* : $\lambda'_{in} >= \lambda_{in}$



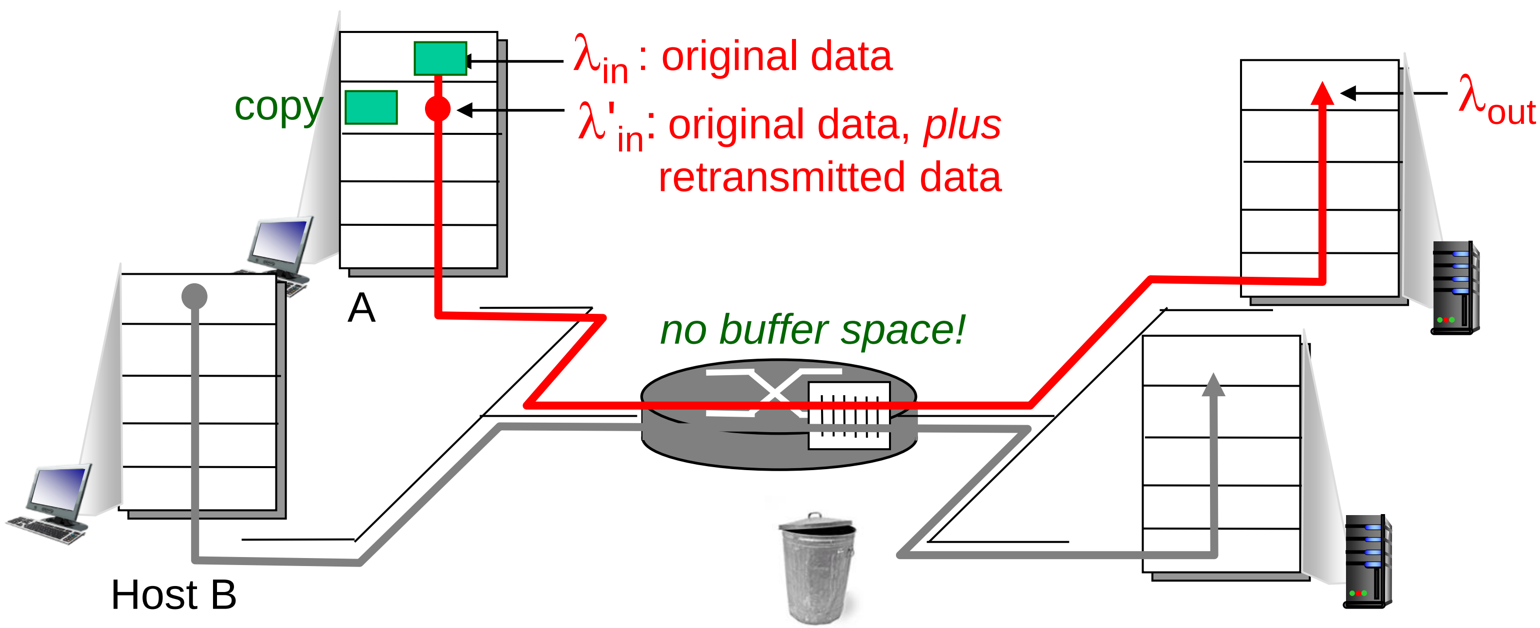
Causes/costs of Congestion: Scenario 2

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



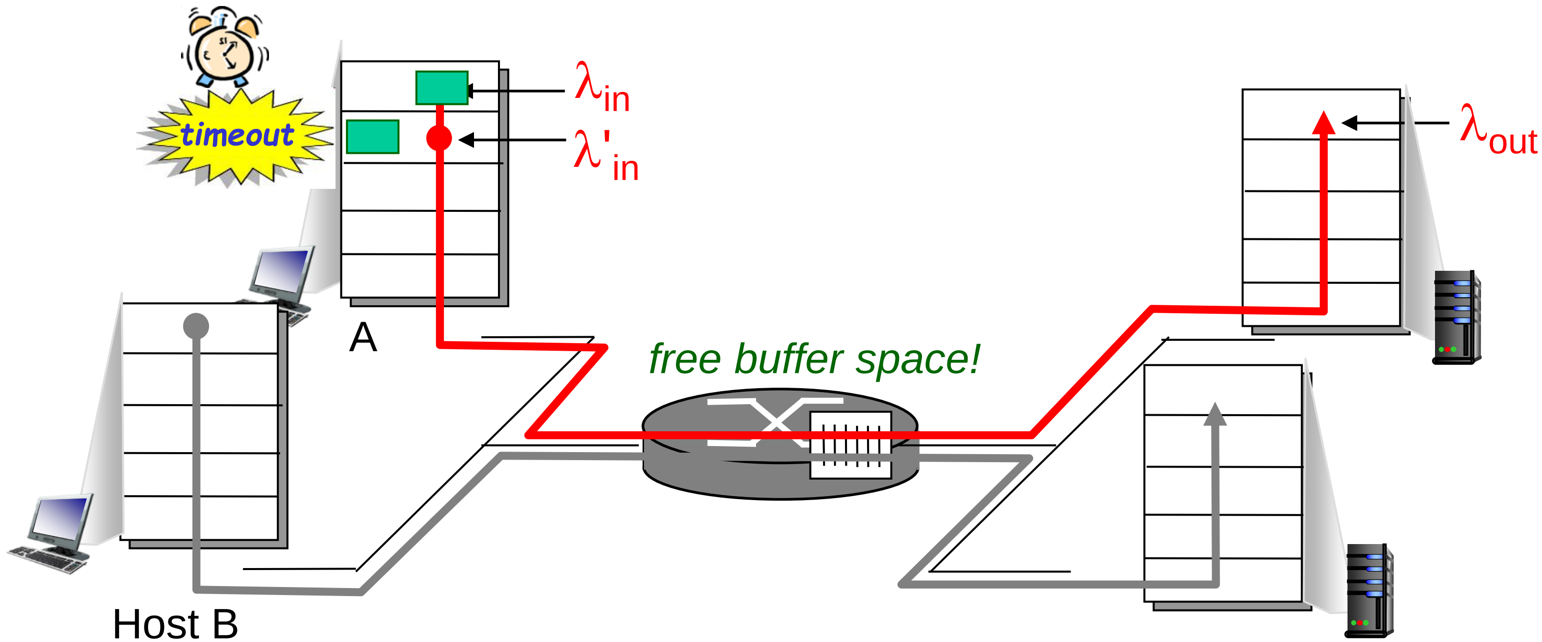
Causes/costs of Congestion: Scenario 2

- 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

THANK YOU