Chapter 3

Transport layer

Liping Shen 申丽萍

lpshen@sjtu.edu.cn

These slides are based upon the exceptional slides provided by Kurose and Ross

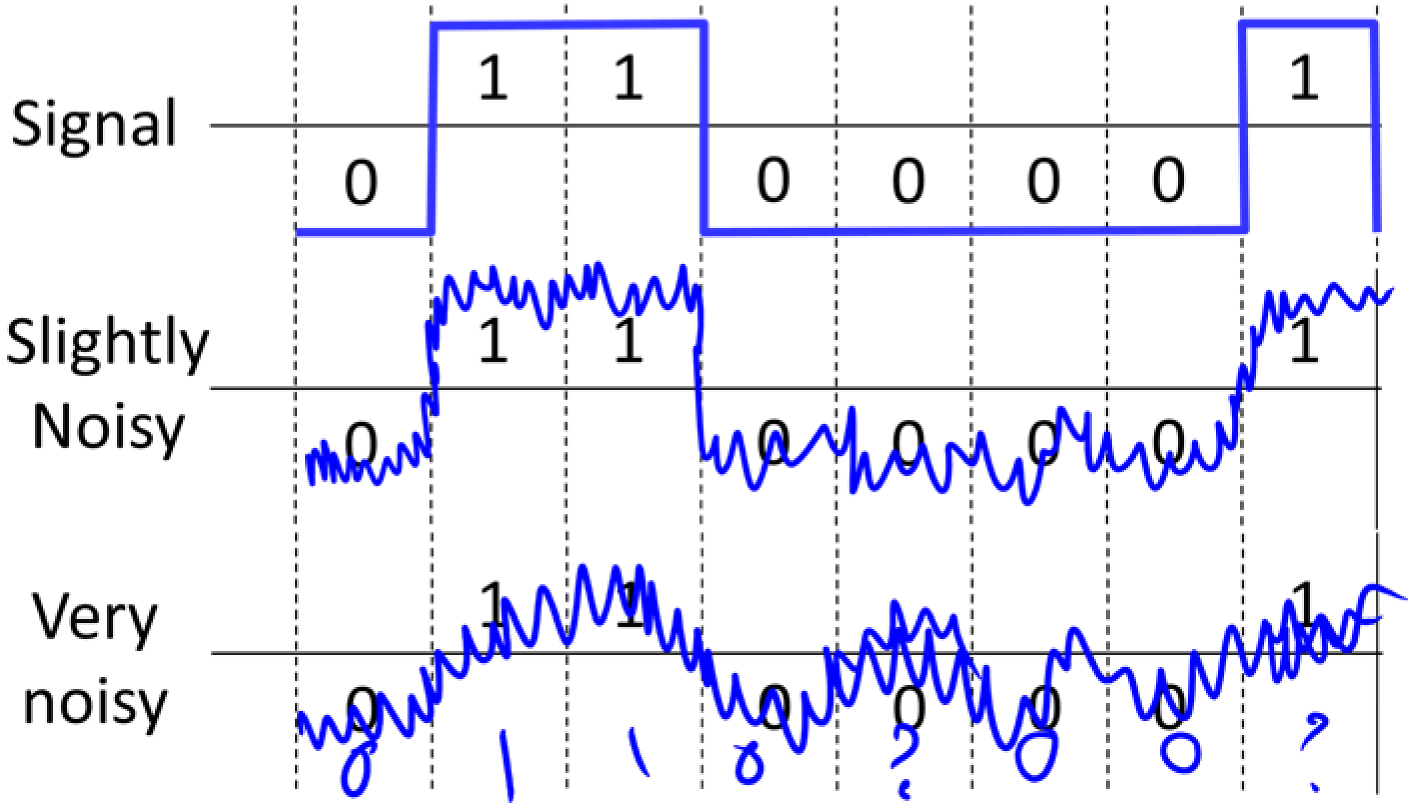
Chapter 3 outline

- 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

Contents

- Error Control: detecting and correcting both the bit level and packet level errors.
  - Error Detection and Retransmission
  - Forward Error Correction
- Flow Control: regulating data flow so that slow receivers not swamped by fast senders
  - stop and wait
  - pipelining (sliding window)
- Reliable Data Transfer protocols
  - RDT 1.0, 2.0, 2.1, 2.2, 3.0
  - Go-Back-N
  - Selective Repeat

Noise may flip bits



2017-03-23

Error Control Techniques

the process of detecting and correcting both the bit level and packet level errors

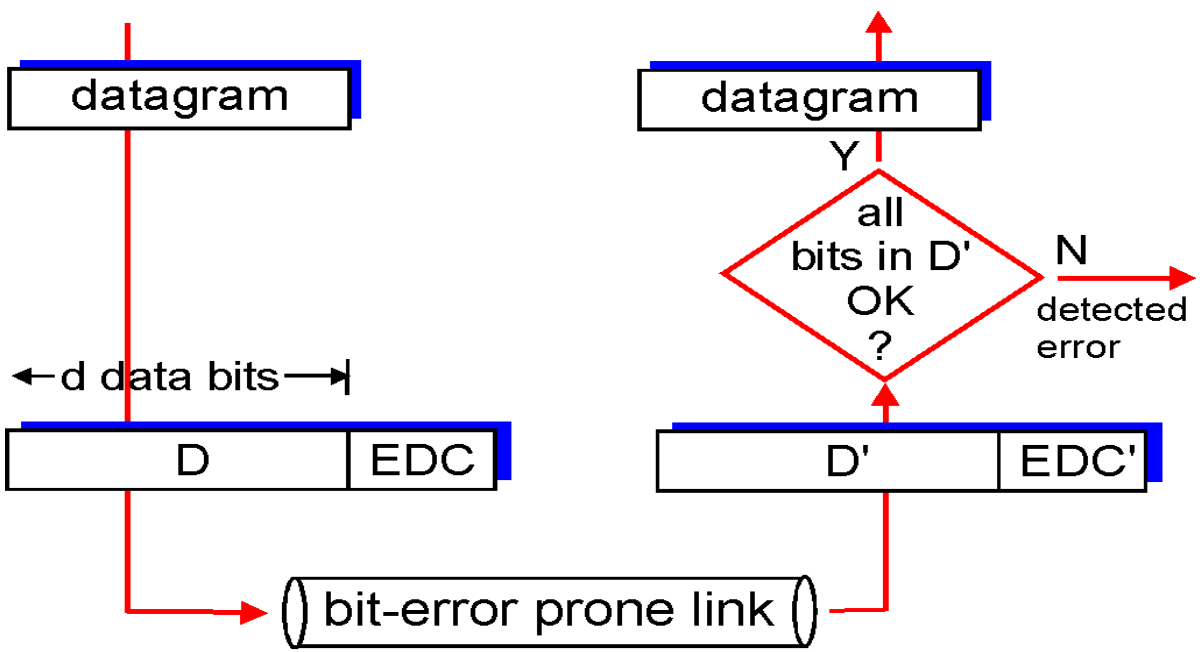
- **bit error**: error detection, acknowledgment and retransmission, error correction
  - Acknowledgment: receiver sends back special control frame: ACK when received OK, NAK when have errors.
  - Retransmission: If receive a NAK, the sender will retransmit the packet.
- **packet loss**: timer and retransmission
  - Timer: sender starts a timer once transmits a packet
  - Retransmission: If the timer goes off, the sender will retransmit the packet.
- **duplicate packet/out-of-order**: sequence number

Error Control Methods

- **Error Detection and Retransmission**: to include only enough redundancy to allow the receiver to deduce that an error occurred, but not which error (error-detecting codes), and have it request a retransmission.
- **Forward Error Correction**: to include enough redundant information along with each block of data sent (error-correcting codes), to enable the receiver to deduce what the transmitted data must have been.

Error Detection Codes

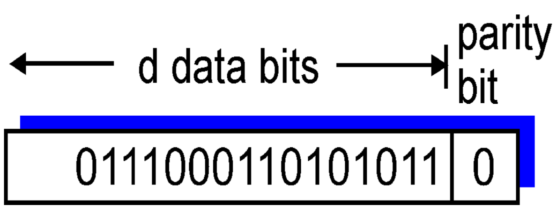
- A codeword consists of m data bits and r EDC bits
  - EDC= Error Detection and Correction bits (redundancy)
- Error detection not 100% reliable!
  - larger r (EDC field) and stronger algorithm yields better detection and correction



Error Detecting Codes - Parity Checking

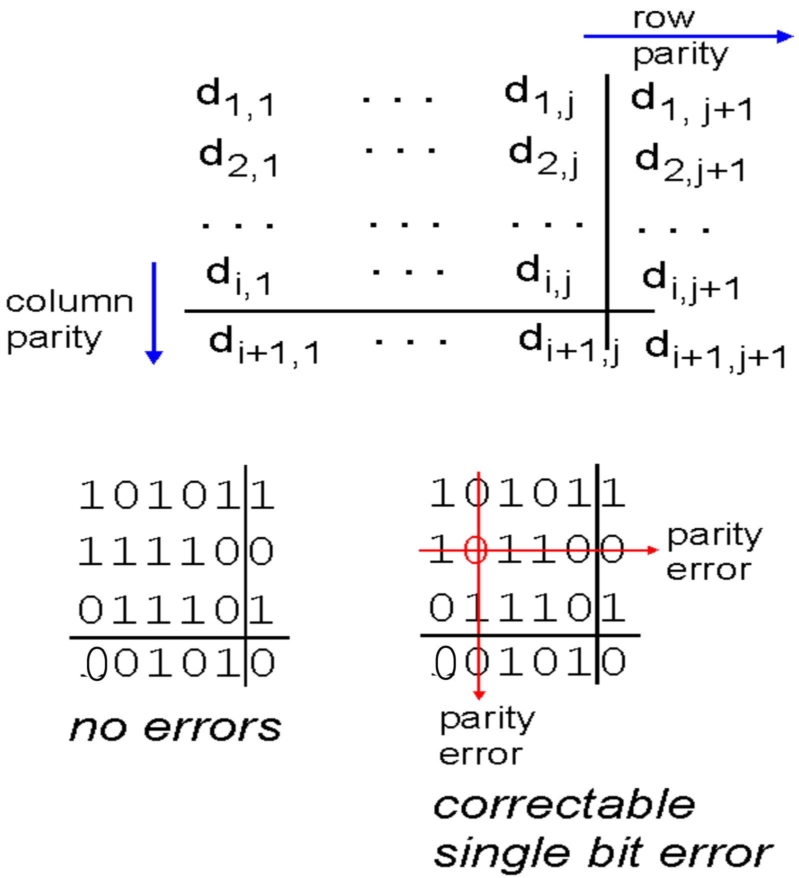
Single Bit Parity:

Detect odd number of bit errors



Two Dimensional Bit Parity:

Detect *and correct* single bit errors



Error-Detecting Codes - Checksum

- Checksum: divide data into 16-bit or 32-bit sections, then add them together. If have carries, add them to the checksum.
- Odds: simple and small check bits.
- Cons: couldn't detect all errors.

H	e	l	l	o		w	o	r	l	d	.
48	65	6C	6C	6F	20	77	6F	72	6C	64	2E

4865 + 6C6C + 6F20 + 776F + 726C + 642E + carry = 71FC

		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
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

Error-Detecting Codes -CRC Example

- Frame: <u>1101011011</u>
- $G(x) = x^4 + x + 1$
- Frame Transmitted:

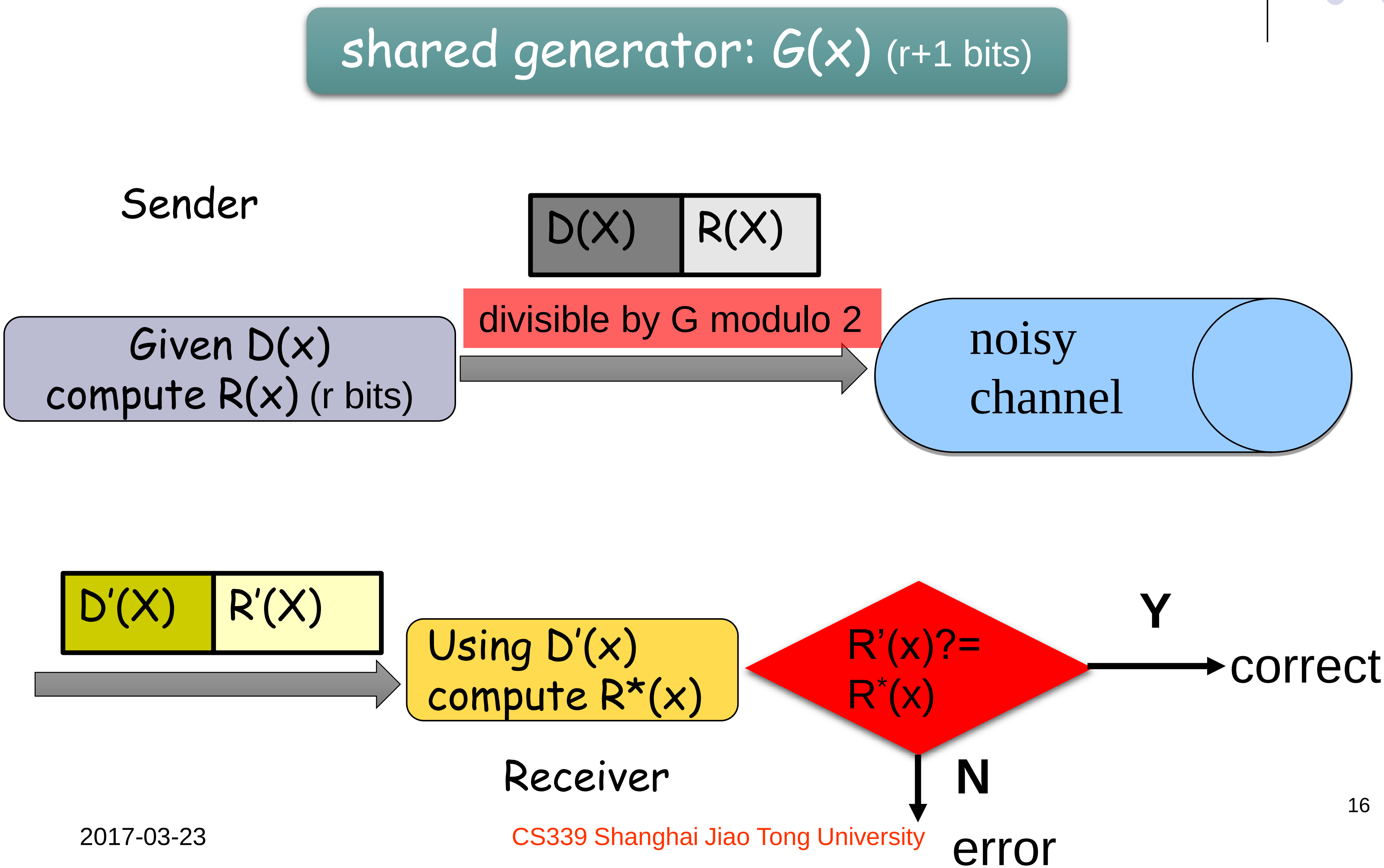
```
                1 1 0 0 0 0 1 0 1 0
          ______________________________
1 0 0 1 1 | 1 1 0 1 0 1 1 0 1 1 0 0 0 0
            1 0 0 1 1
            ---------
              1 0 0 1 1
              1 0 0 1 1
              ---------
                0 0 0 0 1
                0 0 0 0 0
                ---------
                  0 0 0 1 0
                  0 0 0 0 0
                  ---------
                    0 0 1 0 1
                    0 0 0 0 0
                    ---------
                      0 1 0 1 1
                      0 0 0 0 0
                      ---------
                        1 0 1 1 0
                        1 0 0 1 1
                        ---------
                          0 1 0 1 0
                          0 0 0 0 0
                          ---------
                            1 0 1 0 0
                            1 0 0 1 1
                            ---------
                              0 1 1 1 0
                              0 0 0 0 0
                              ---------
                                1 1 1 0
```

1101011011	1110
Data	Remainder

Error-Detecting Codes -cyclic redundancy check (CRC)

- Given m data bits, generator polynomial of degree r G(x), The algorithm for computing the CRC is as follows:
  - Append r zero bits to the low-order end of the frame so it now contains m + r bits.
  - Divide the m+r bits with r+1 bit string corresponding to G(x) using modulo 2 division (no carries for addition or borrows for subtraction. Both addition and subtraction are identical to exclusive OR).
  - The remainder is the r bits CRC. Append the r bits to the m data bits into a m+r codeword which is divisible (modulo 2) by G(x).
  - When the receiver gets the checksummed frame, it tries dividing it by G(x). If there is a remainder, there has been a transmission error.

Cyclic Redundancy Check used in link layer can detect all burst errors less than r+1 bits



Four International Standards Generator Polynomials

- CRC-12： $x^{12} + x^{11} + x^3 + x^2 + x^1 + 1$

  Used for 6-bit Characters
- CRC-16： $x^{16} + x^{15} + x^2 + 1$

  Used for 8-bit Characters
- CRC-CCITT： $x^{16} + x^{12} + x^5 + 1$

  Used for 8-bit Characters
- IEEE 802： $x^{32} + x^{26} + x^{23} + x^{22} + x^{16} + x^{12} + x^{11} + x^{10} + x^8$

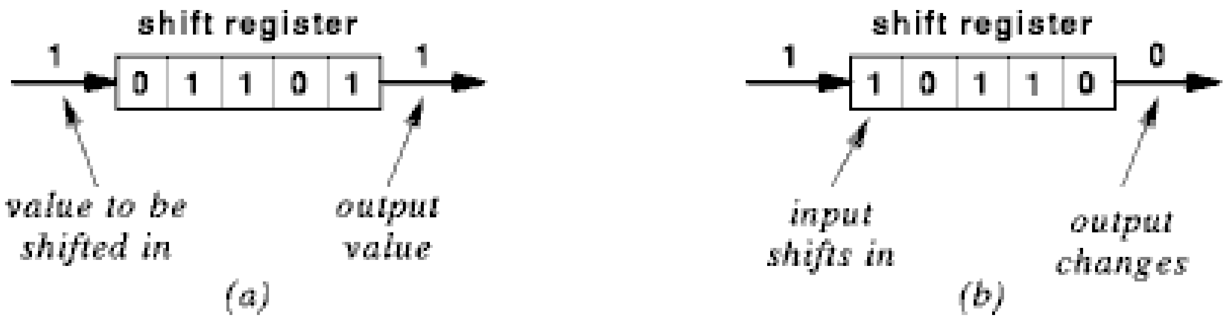
  $+ x^7 + x^5 + x^4 + x^2 + x^1 + 1$

Power of CRC

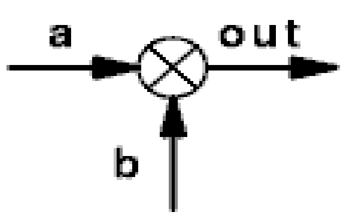
- The kinds of errors could be caught:
  - all single-bit errors will be detected
  - All two isolated single-bit errors will be detected
  - By making (x + 1) a factor of G(x), all errors consisting of an odd number of inverted bits will be detected
  - all burst errors of length <= r will be detected

A simple circuit can be constructed to compute and verify CRC in hardware

- **Shift Register**



- **OR Unit**

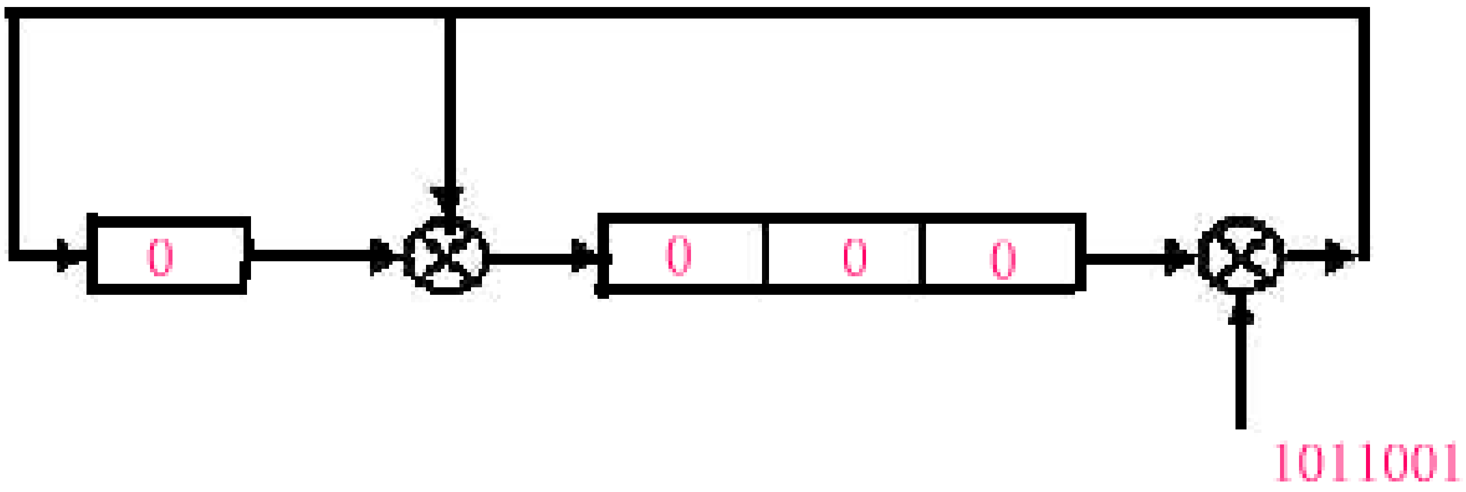


(a)

a	b	out
0	0	0
0	1	1
1	0	1
1	1	0

(b)

A simple shift register circuit can be constructed to compute and verify the CRC in hardware



Q: Why put checksum at trailer?

Demo

Error-Correcting Codes-Hamming Code

- Hamming Code can only correct one bit error.
- Every bit at position 2^k(k>=0) is used as a parity bit for those positions to which it contributes
- To see which check bits (k) the data bit in position i contributes to, rewrite i as a sum of 2^k .
- *E*ach check bit forces the parity of the collection of contribution bits, including itself, to be even (or odd).

	b1	b2	b3	b4	b5	b6	b7	b8	b9	b10	b11
1	X		X		X		X		X		X
2		X	X			X	X			X	X
4				X	X	X	X				
8								X	X	X	X

11	10	9	8	7	6	5	4	3	2	1
A7	A6	A5	P4	A4	A3	A2	P3	A1	P2	P1
1	0	0	1	0	0	1	1	1	0	1

Hamming Code- Correction

- Recompute check bits (with parity sum including the check bit)
- Arrange the check bits as a binary number
- Value (syndrome) tells error position,
  - Value of zero means no error
  - Otherwise, flip bit at position value to correct
  - Hamming codes can only correct single errors

	b1	b2	b3	b4	b5	b6	b7	b8	b9	b10	b11
1	X		X		X		X		X		X
2		X	X			X	X			X	X
4				X	X	X	X				
8								X	X	X	X

11	10	9	8	7	6	5	4	3	2	1
A7	A6	A5	P4	A4	A3	A2	P3	A1	P2	P1
1	0	**1**	1	0	0	1	1	1	0	1

Error-Correcting Codes-Hamming Code(2)

Use of a Hamming code to correct burst errors.

Char.	ASCII	Check bits
H	1001000	00110010000
a	1100001	10111001001
m	1101101	11101010101
m	1101101	11101010101
i	1101001	01101011001
n	1101110	01101010110
g	1100111	01111001111
	0100000	10011000000
c	1100011	11111000011
o	1101111	10101011111
d	1100100	11111001100
e	1100101	00111000101

CRC correct single bit error

Order of bit transmission

Error Control in Practice

- Error Detection and Retransmission (EDR)
  - CRCs are widely used on links (Ethernet, ADSL, Cable …)
  - Checksum used in Internet (IP, TCP, UDP …) , but it is weak
  - Parity is little used
- Forward Error Correction (FER)
  - Hamming code used when the error rate is low (computer Error Checking and Correction memory)
  - Convolutional codes and LDPC heavily used in wireless data link layer (802.11, WiMAX, LTE, power-line, …)

Detection vs. Correction

- Which is better depends on the pattern of errors.

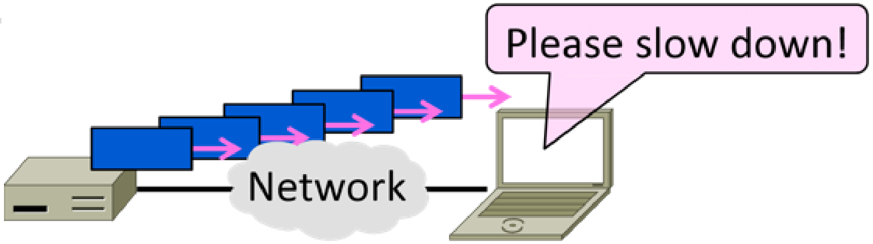
e.g. 1000 bit messages with a bit error rate(BER) of 1 in 10000,

  - in random: overhead FEC ~10, EDR ~100
  - in burst of 100: overhead FEC >100, EDR ~30

- Forward Error Correction :
  - when errors are expected
  - or when no time for retransmission
- Error Detection and Retransmission:
  - more efficient when errors are not expected
  - burst errors when they do occur

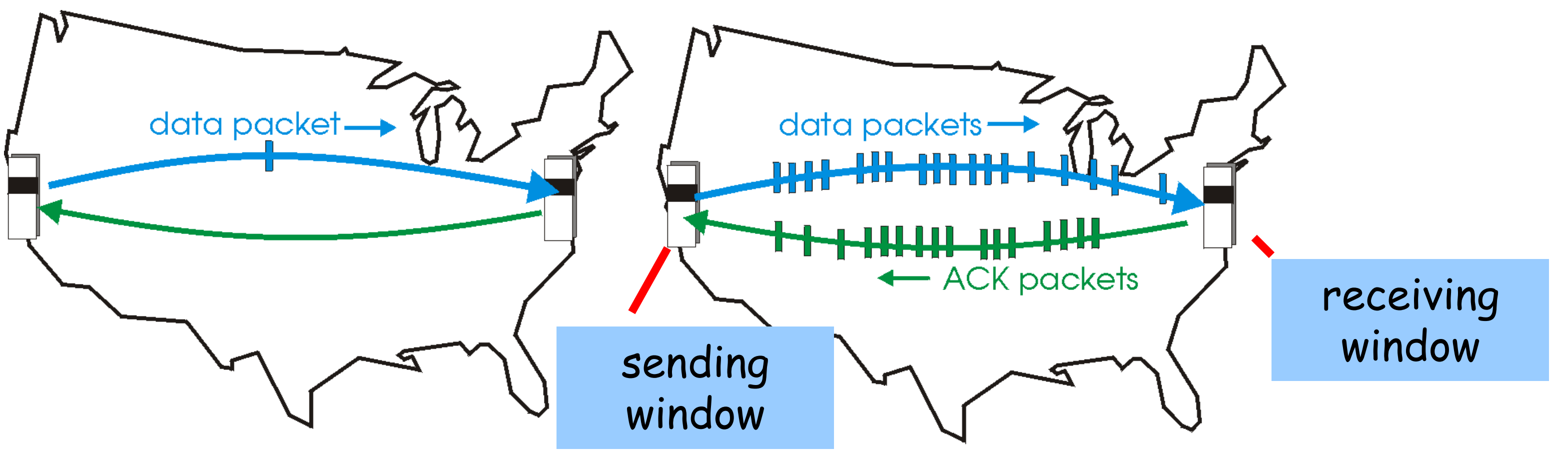
Principles of reliable data transfer

- Error Control: detecting and correcting both the bit level and packet level errors.
  - Error Detection and Retransmission
  - Forward Error Correction
- Flow Control: regulating data flow so that slow receivers not swamped by fast senders
  - stop and wait
  - pipelining (sliding window)
- Reliable Data Transfer pr
  - RDT 1.0, 2.0, 2.1, 2.2, 3.0
  - Go-Back-N
  - Selective Repeat



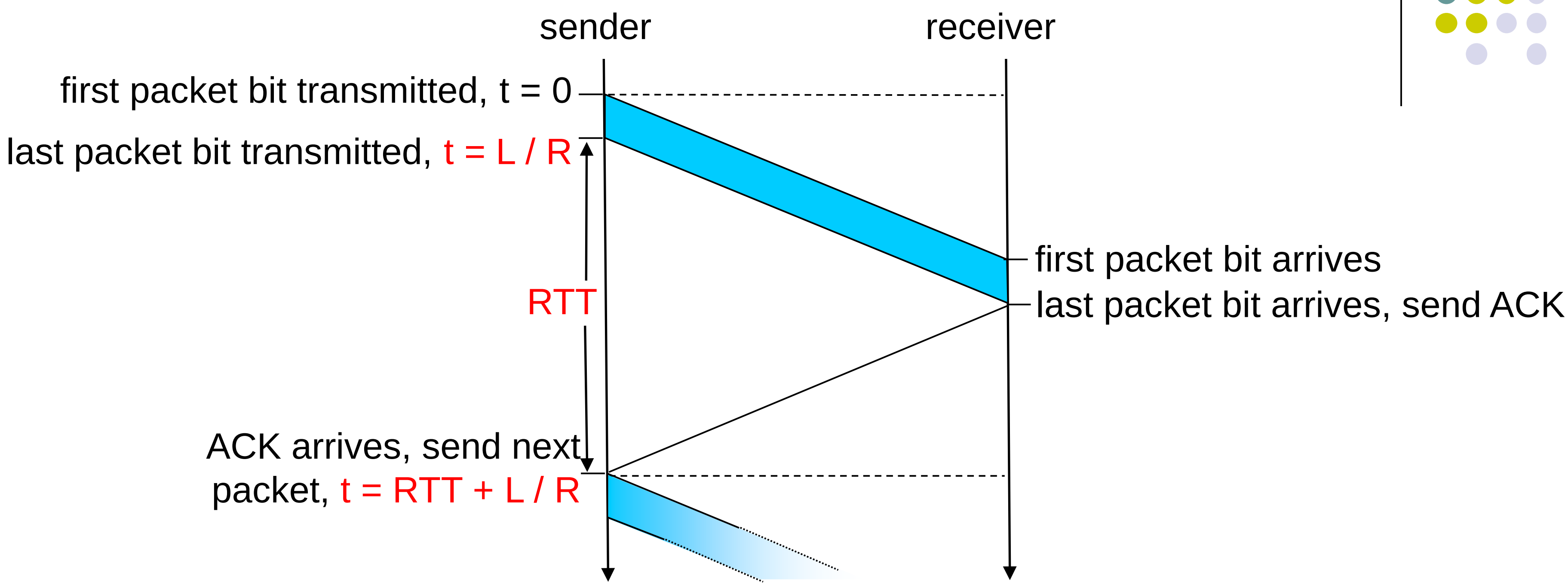
Flow Control

- Stop and Wait: Sender sends one packet, then waits for receiver response.
- Sliding Window (pipelining): allowing the sender to transmit up to *multiple* frames before the first acknowledgement arrives.
  - sending window to keep unacknowledged packets for possible retransmission
  - receiving window to hold out-of-order packets received or ordered packets undelivered to upper layer.



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

Performance of stop-and-wait

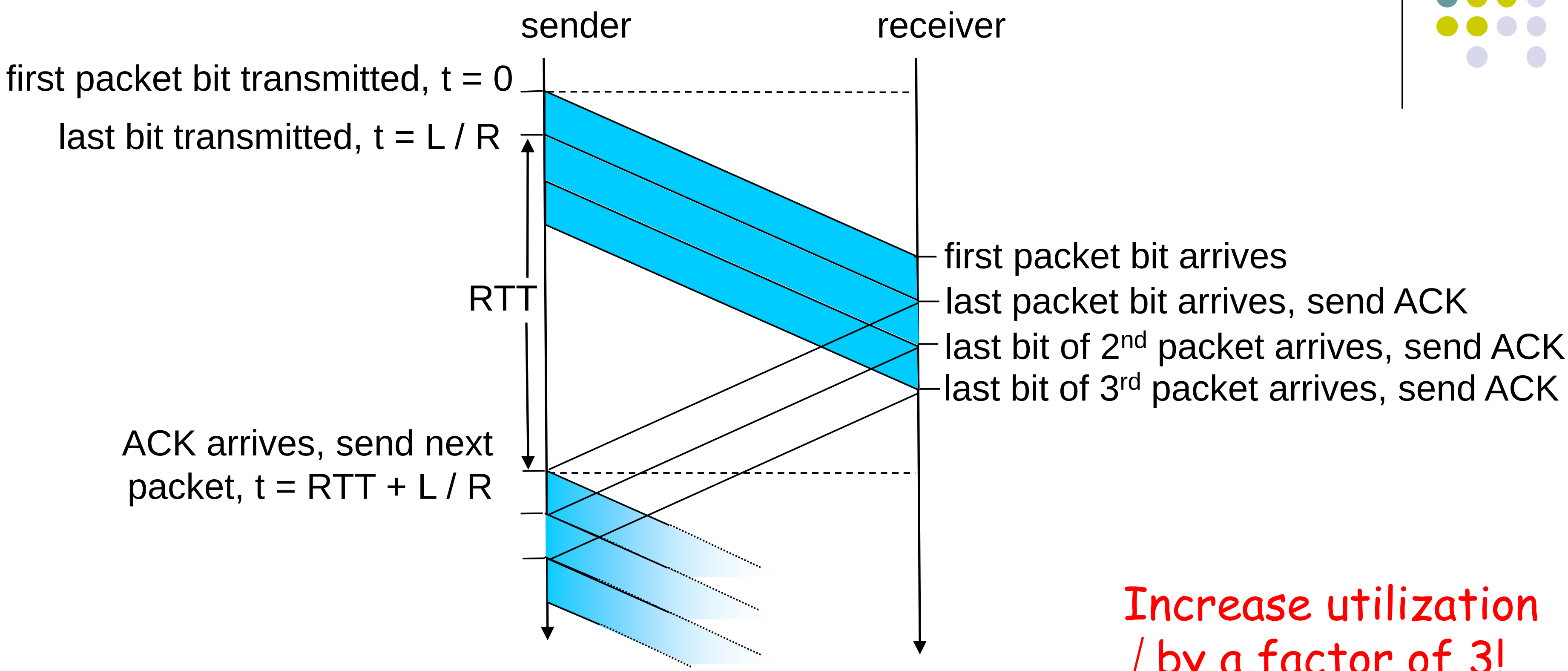


- utilization : fraction of time sender busy sending
- ex: 1 Gbps link, 15 ms prop. delay, 8000 bit packet

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

1KB pkt every 30 msec -> 33kB/sec thruput over 1 Gbps link

Pipelining: increased utilization



Increase utilization by a factor of 3!

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

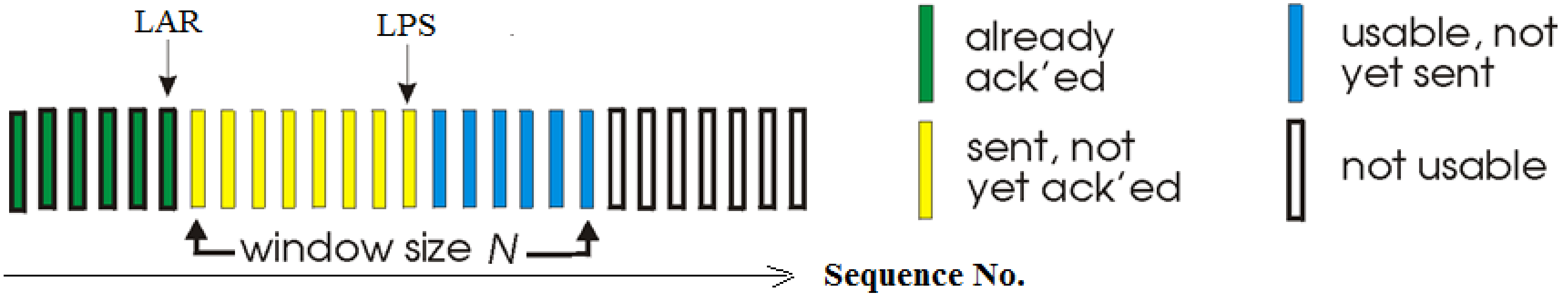
With an appropriate choice of window W= RTT*R pipelining can keep the line busy (100%efficiency)

Sliding Sender Window

Sender buffers up to N unacknowledged packets.

- LPS=Last Packet Sent, LAR=Last ACK Received
- Sends while LPS – LAR < N
- Sender blocked while LPS – LAR = N, used for flow control.
- when next ACK arrived:
  - LAR=LAR+1
  - Sending window advances/slides, buffer is freed

Demo

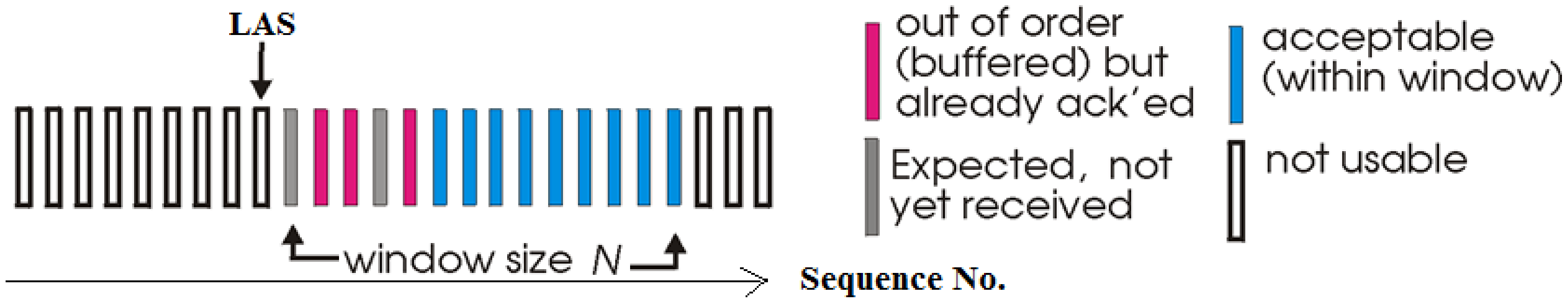


Sliding Receiver Window

Receiver buffers up to N out-of-order or undelivered packets.

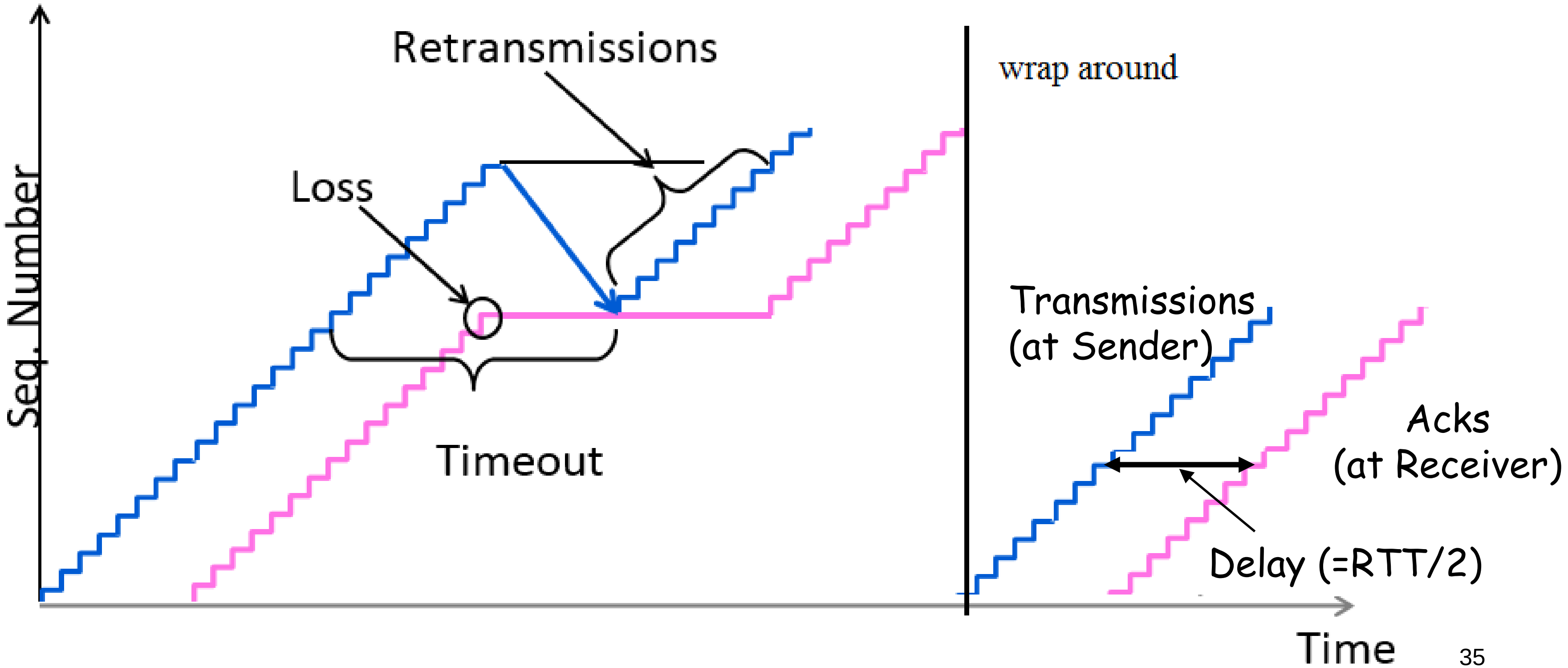
- keeps state variable LAS = Last in-order-ACK Sent
- receive and keep packets of [LAS+1, LAS+N], send ACKs.
- window full, discard packets > LAS+N, flow control needed.
- When in-order packets delivered to upper layer:
  - update LAS to next expected packets.
  - receiving window advances/slides, buffer is freed

Demo



Sliding Window – Sequence No.

- Seq. No. with an n-bit counter wraps around at 2^n—1
  - *e.g.*, n=8: …, 253, 254, 255, 0, 1, 2, 3, …
- Seq. No. space must be larger than number of outstanding pkts.



Summary of reliable data transfer mechanisms

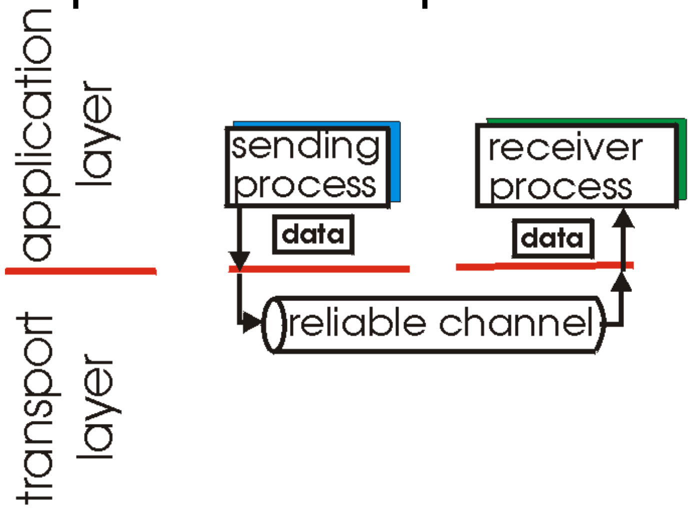
Mechanism	Use and Comments
Checksum	Used to detect bit errors in a transmitted pocket.
Timer	Used to timeout/retransmit a packet, possibly because the packet (or its ACK) was lost within the channel. Because timeouts can occur when a pocket is delayed but not lost (premature timeout), or when a packet has been received by the receiver but the receiver-to-sender ACK has been lost, duplicate copies of a packet may be received by a receiver.
Sequence number	Used for sequential numbering of packets of data flowing from sender to receiver. Gaps in the sequence numbers of received packets allow the receiver to detect a lost packet. Packets with duplicate sequence numbers allow the receiver to detect duplicate copies of a packet.
Acknowledgment	Used by the receiver to tell the sender that a packet or set of packets has been received correctly. Acknowledgments will typically carry the sequence number of the packet or packets being acknowledged. Acknowledgments may be individual or cumulative, independently or piggybacked, depending on the protocol.
Negative acknowledgment	Used by the receiver to tell the sender that a packet has not been received correctly. Negative acknowledgments will typically carry the sequence number of the packet that was not received correctly.
Sliding window, pipelining	The sender may be able to send multiple packets with sequence numbers that fall within a given range. By allowing multiple packets to be transmitted but not yet acknowledged, sender utilization can be increased over a stop-and-wait mode of operation. The window size may be set on the basis of the receiver's ability to receive and buffer messages, or the level of congestion in the network, or both .

Principles of reliable data transfer

- Error Control: detecting and correcting both the bit level and packet level errors.
  - Error Detection and Retransmission
  - Forward Error Correction
- Flow Control: regulating data flow so that slow receivers not swamped by fast senders
  - stop and wait
  - pipelining (sliding window)
- Reliable Data Transfer protocols
  - RDT 1.0, 2.0, 2.1, 2.2, 3.0
  - Go-Back-N
  - Selective Repeat

Reliable data transfer (rdt) service

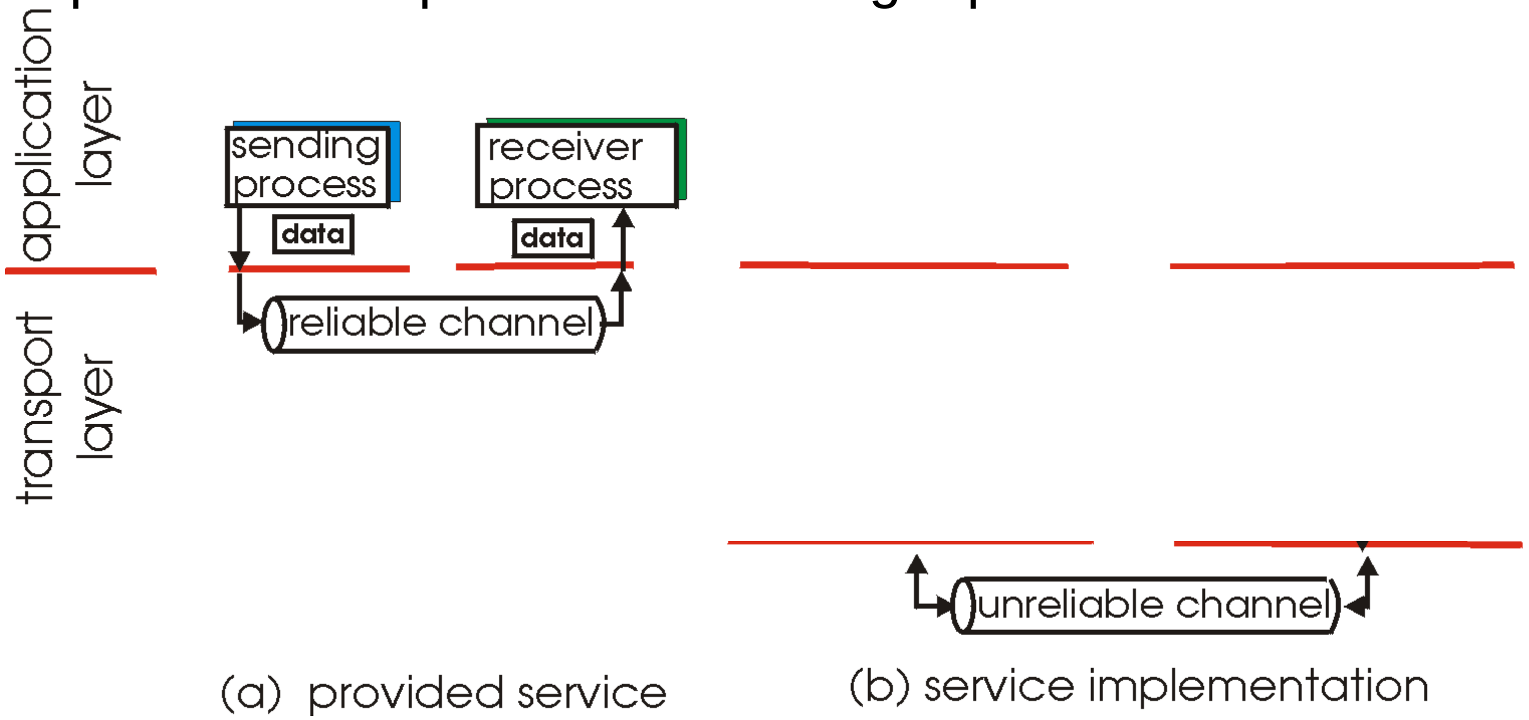
- important in app., transport, link layers
- top-10 list of important networking topics!



(a) provided service

Reliable data transfer (rdt) service

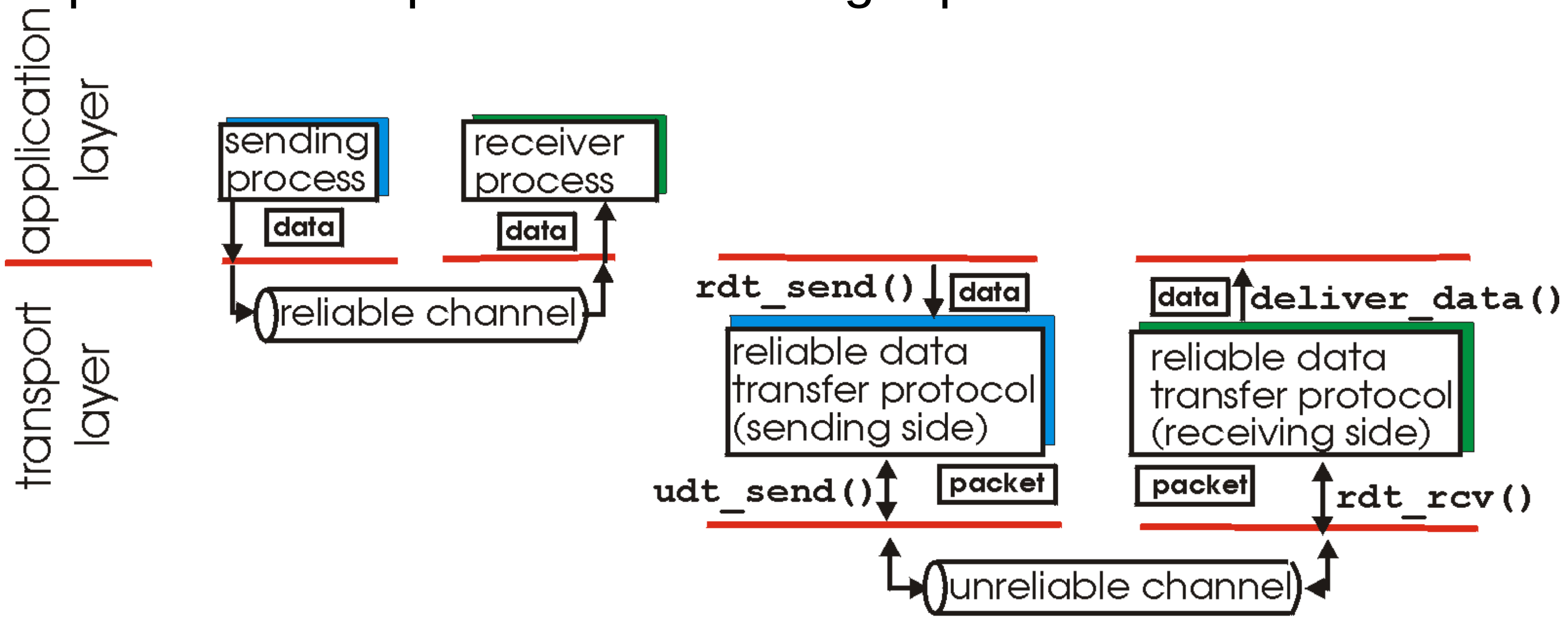
- important in app., transport, link layers
- top-10 list of important networking topics!



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

Reliable data transfer (rdt) service

- important in app., transport, link layers
- top-10 list of important networking topics!



(a) provided service

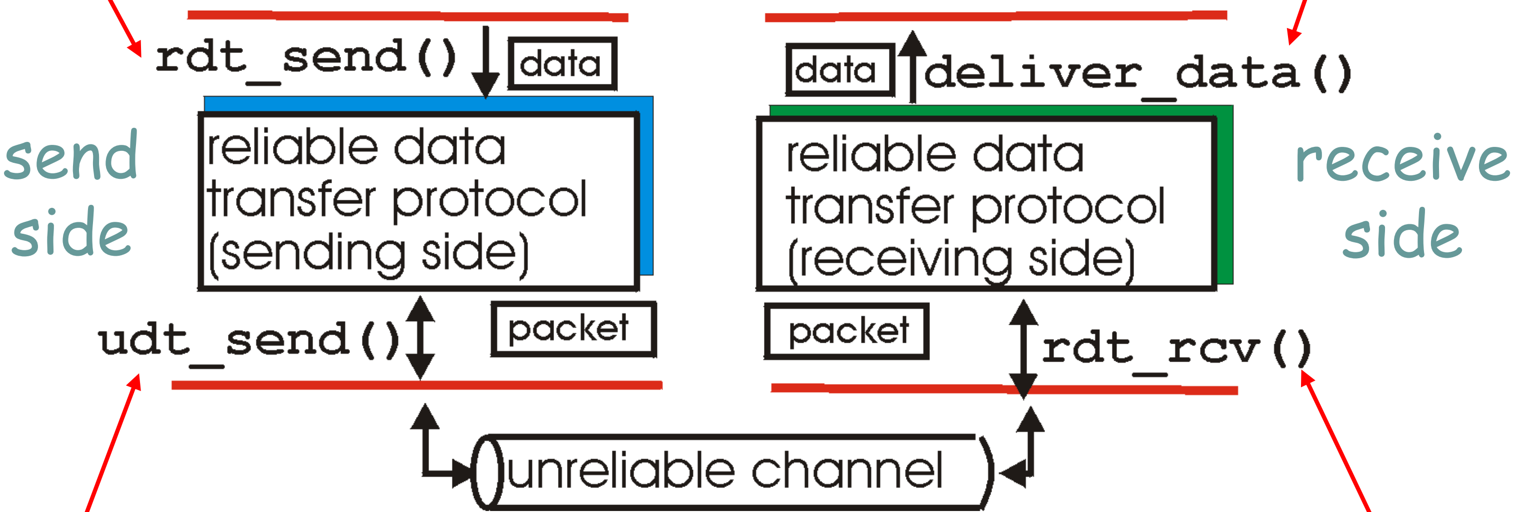
(b) service implementation

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

rdt protocol design -getting started

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

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

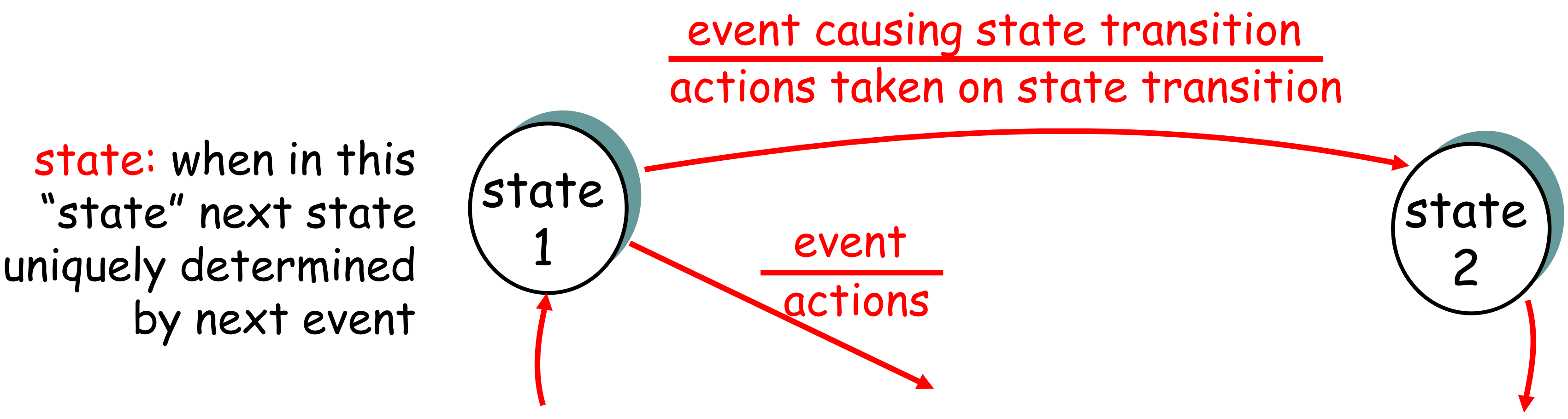


udt_send(): called by rdt, to transfer packet over unreliable channel to receiver

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

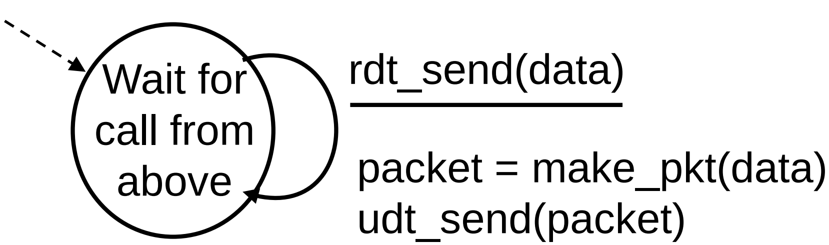
rdt Protocol Design -getting started

- incrementally develop reliable data transfer protocols (rdt)
  - consider only unidirectional data transfer, but control info will flow on both directions!
  - use finite state machines (FSM) to specify sender, receiver

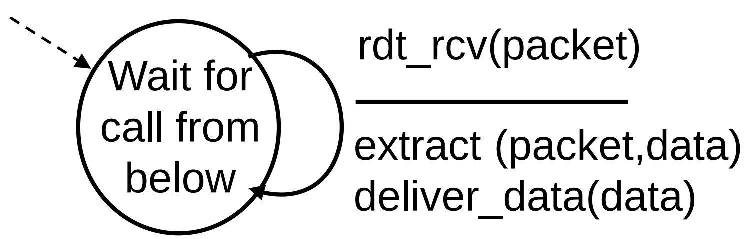


rdt1.0: a reliable channel

- underlying channel perfectly reliable
  - no bit errors, no loss of packets, so no error control
- Receiver has enough buffer and CPU power
  - no flow control
- separate FSMs for sender, receiver:
  - sender sends data into underlying channel
  - receiver read data from underlying channel



sender



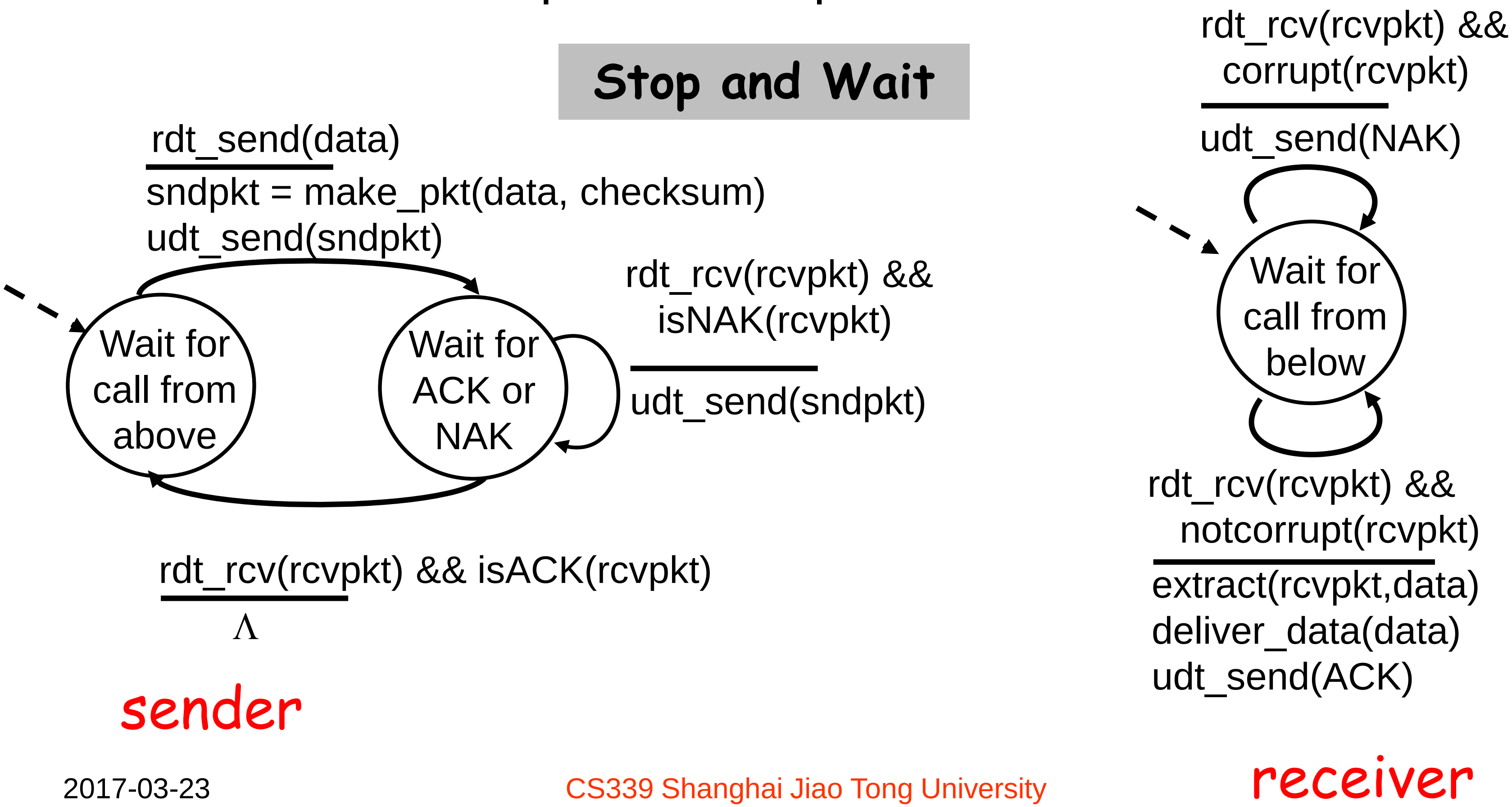
receiver

rdt2.0: channel with errors

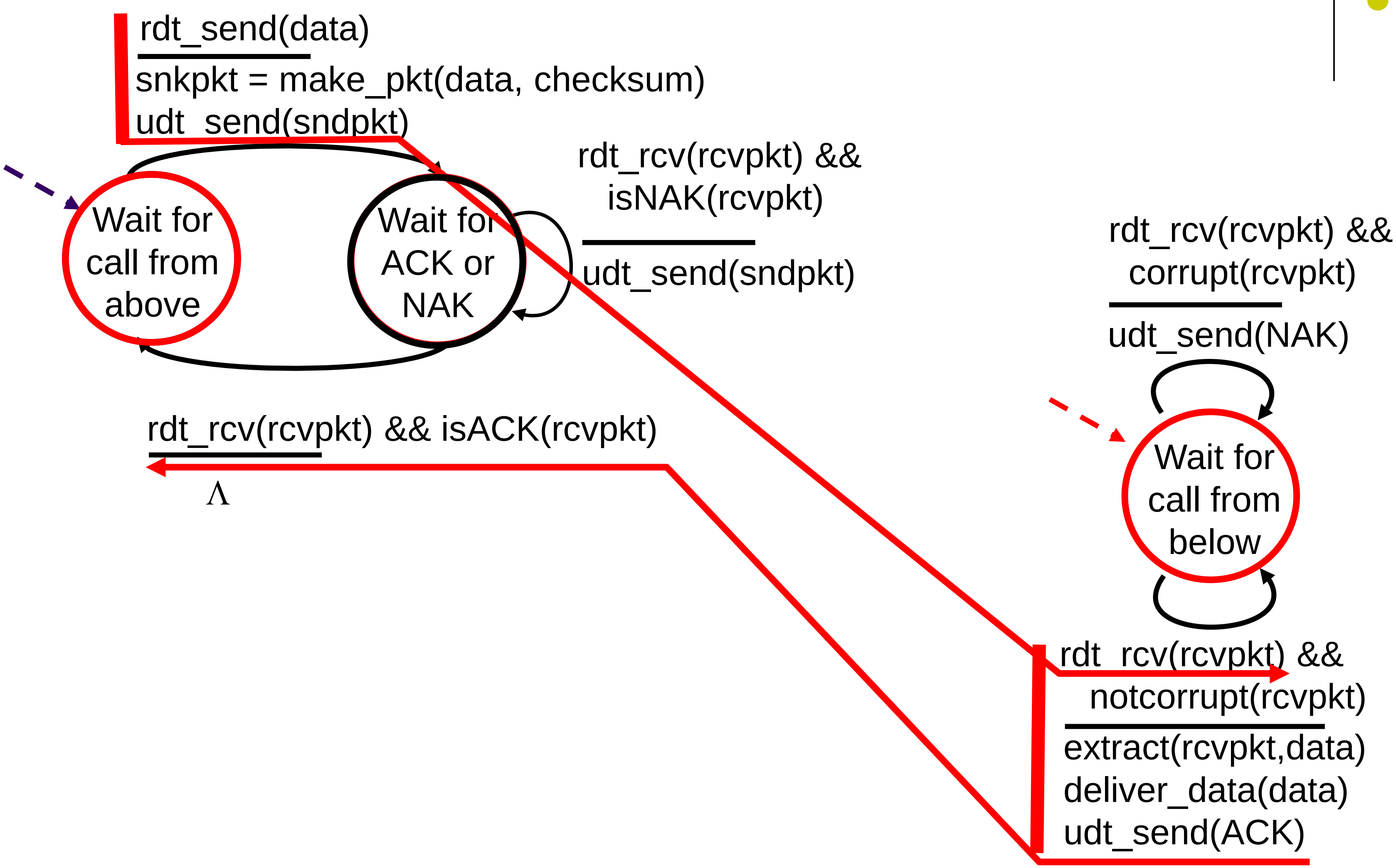
ARQ (Automatic Repeat reQuest)

- underlying channel may flip bits in packet (no lost)
  - checksum to detect bit errors
  - receiver feedback: control msgs (ACK,NAK)
  - sender retransmits pkt on receipt of NAK

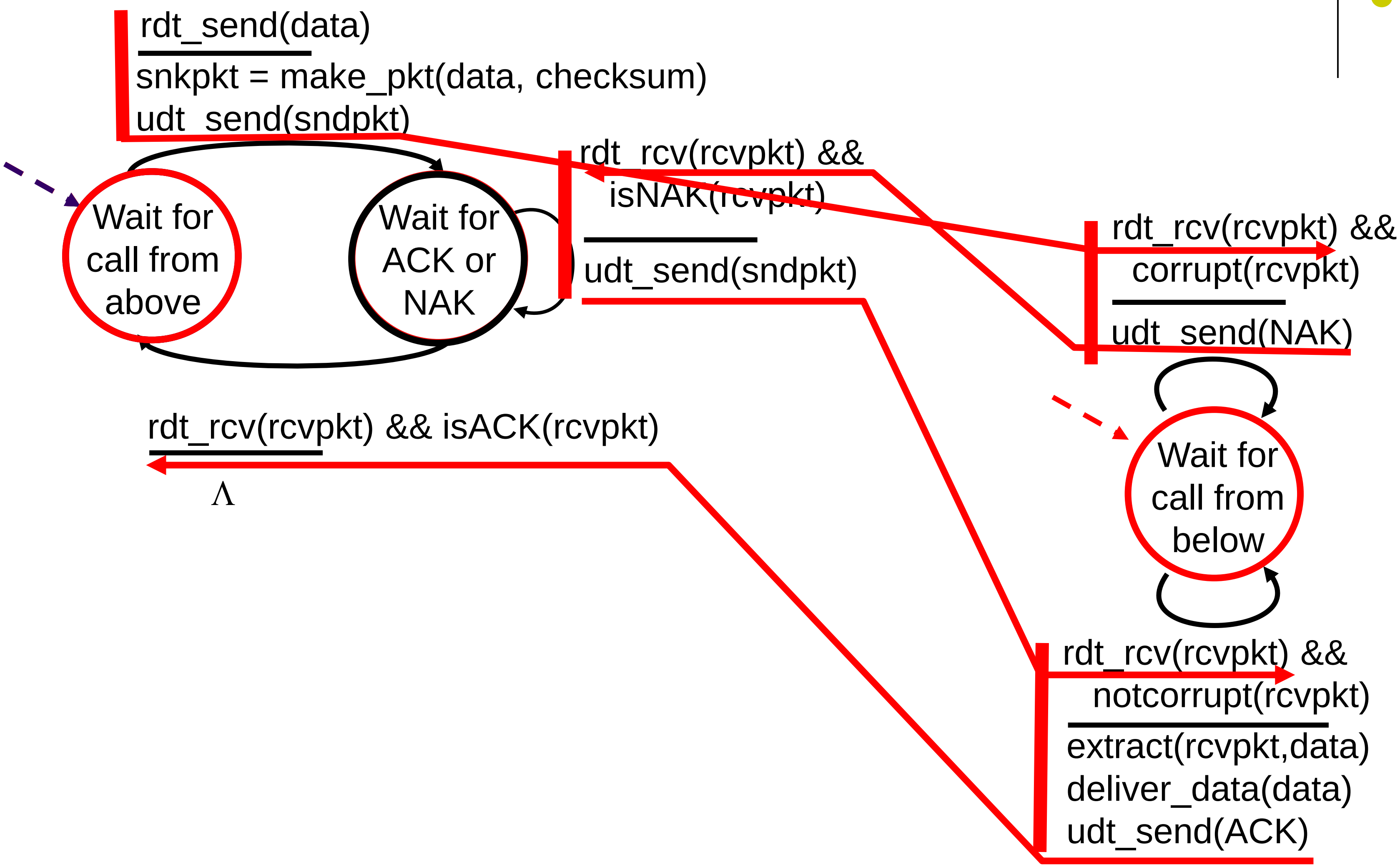
Stop and Wait



rdt2.0: operation with no errors



rdt2.0: error scenario



rdt2.0 has a fatal flaw!

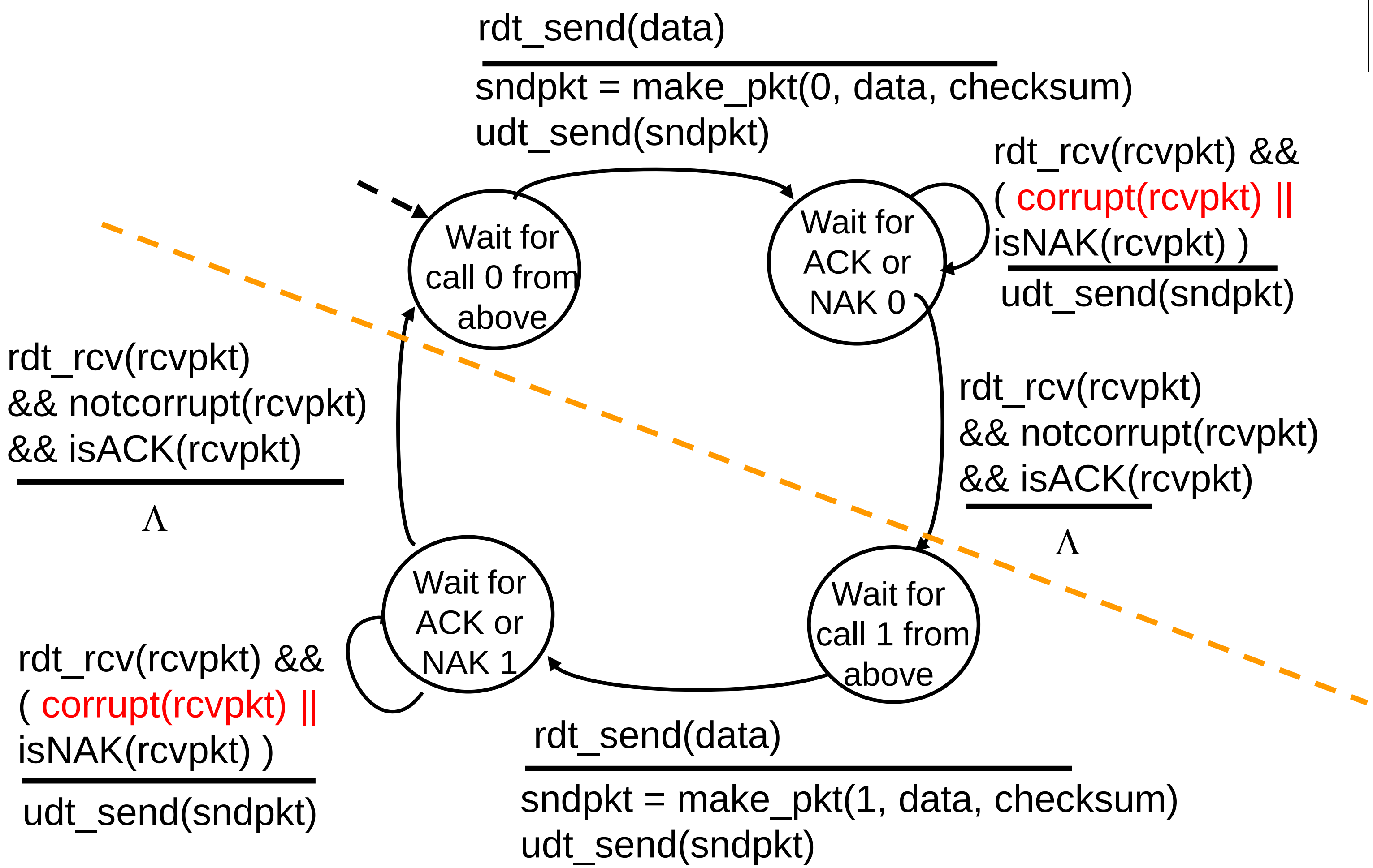
What happens if

- ACK/NAK corrupted? sender doesn't know what happened at receiver!
- possible duplicate pkt with retransmission

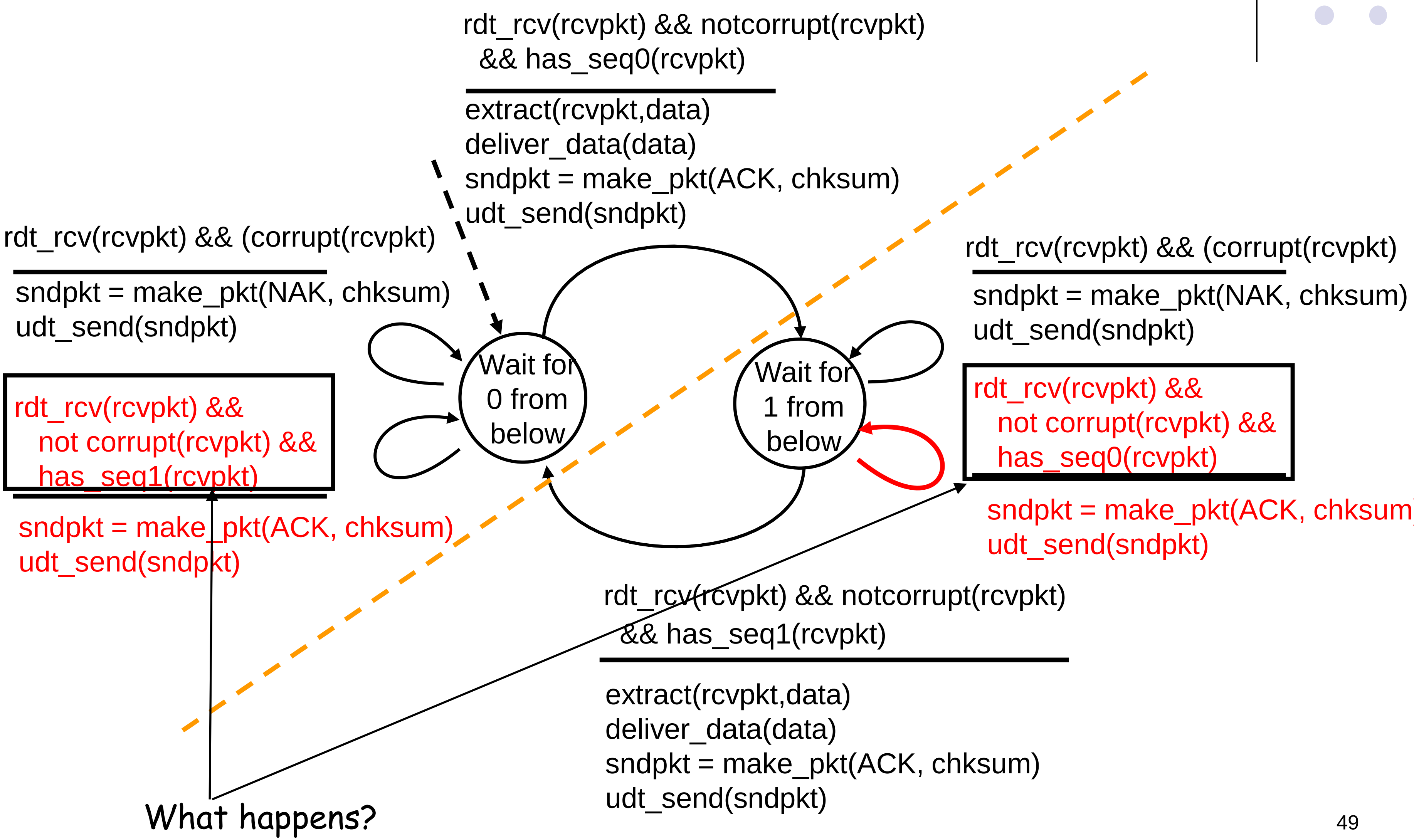
Handling duplicates:

- 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



rdt2.1: discussion

<u>Sender:</u>

- seq No. added to pkt
- two seq. #s (0,1) will suffice. Why?
- must check if received ACK/NAK corrupted
- twice as many states
  - state must “remember” whether “current” pkt has 0 or 1 seq. #.

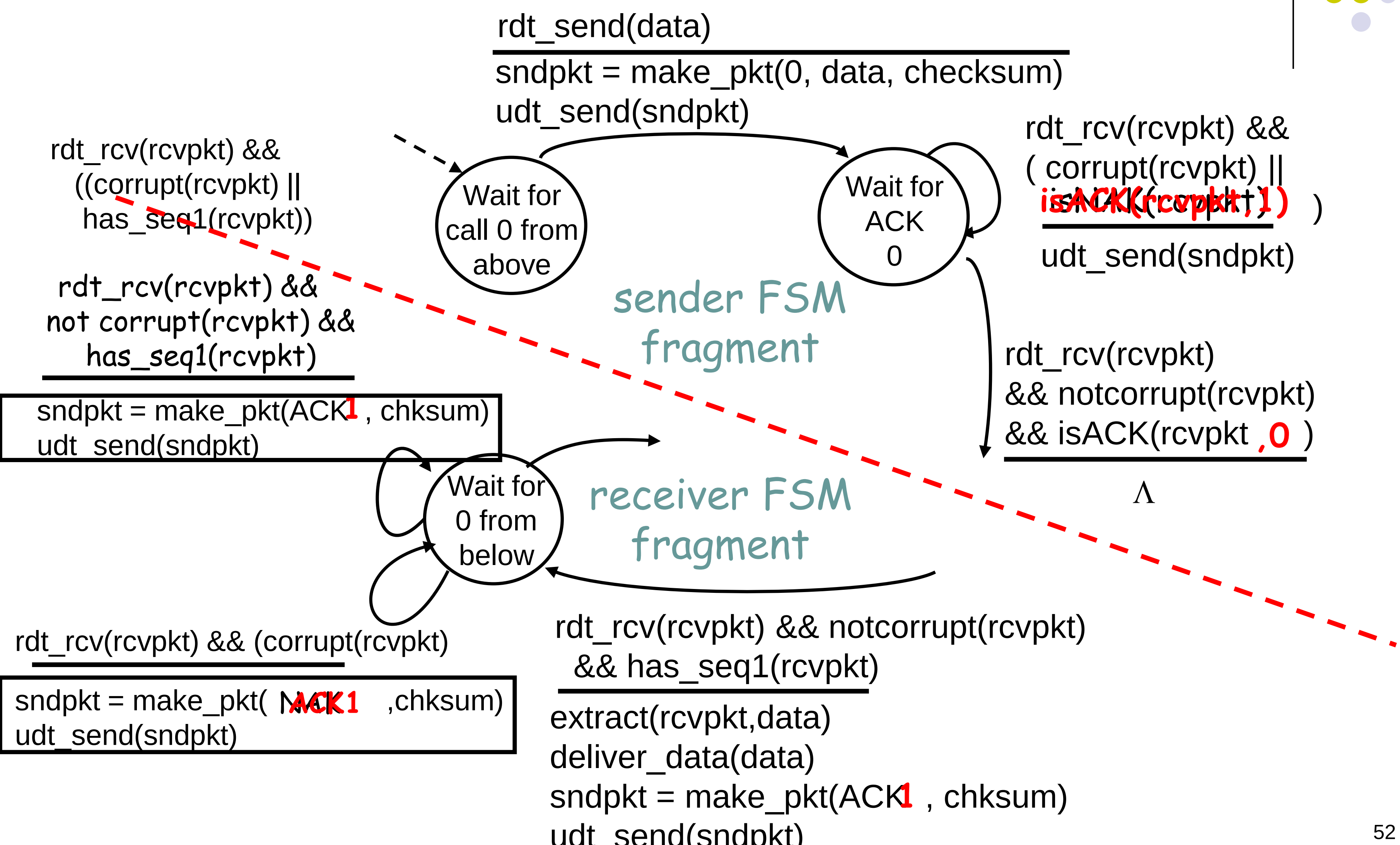
<u>Receiver:</u>

- must check if received packet is duplicate
  - state indicates whether 0 or 1 is expected pkt seq #
- note: receiver can *not* know if its last ACK/NAK received OK at sender

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 also lose packets (data or ACKs)

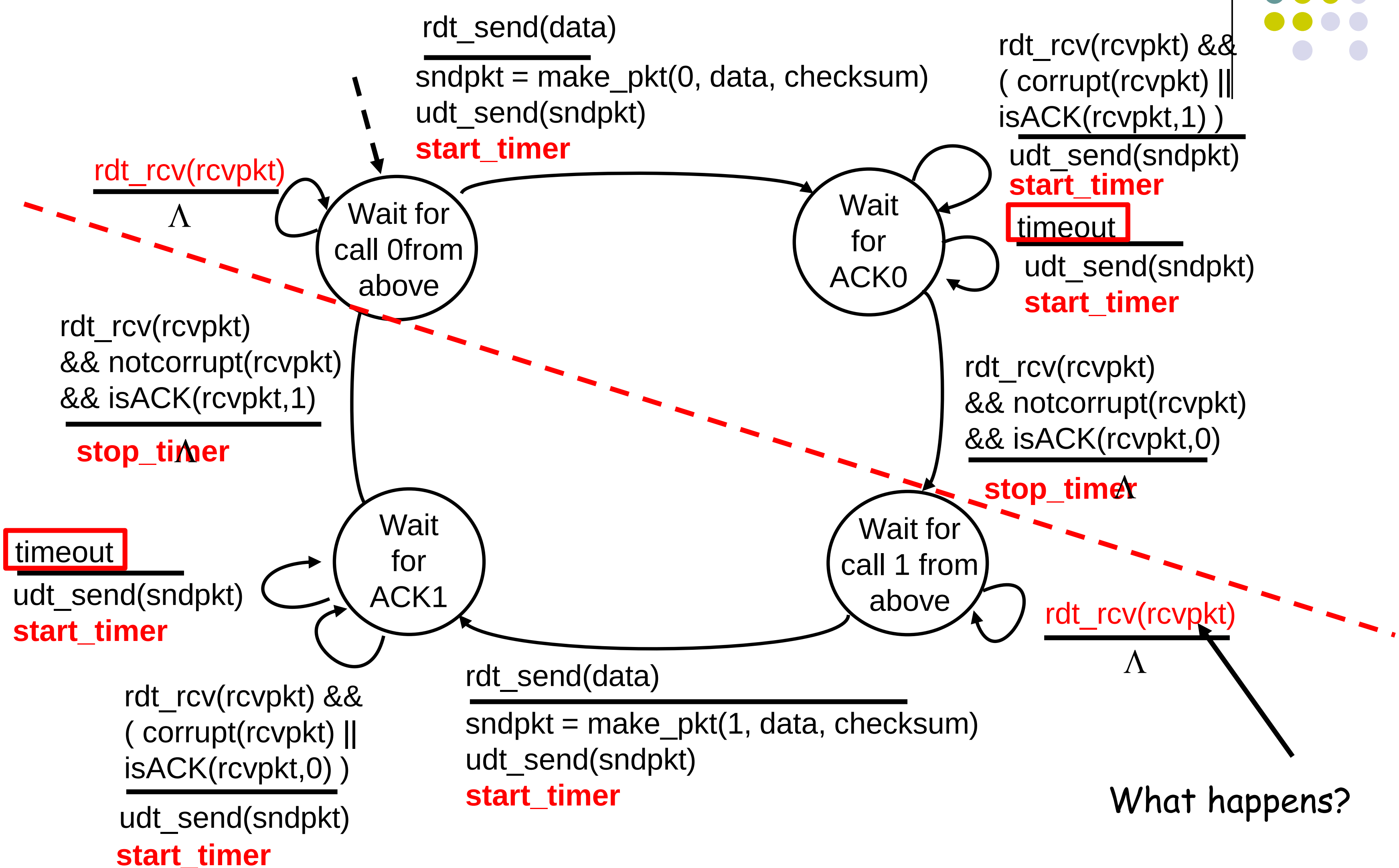
- checksum, seq. #, ACKs, retransmissions will be of help, but not enough

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

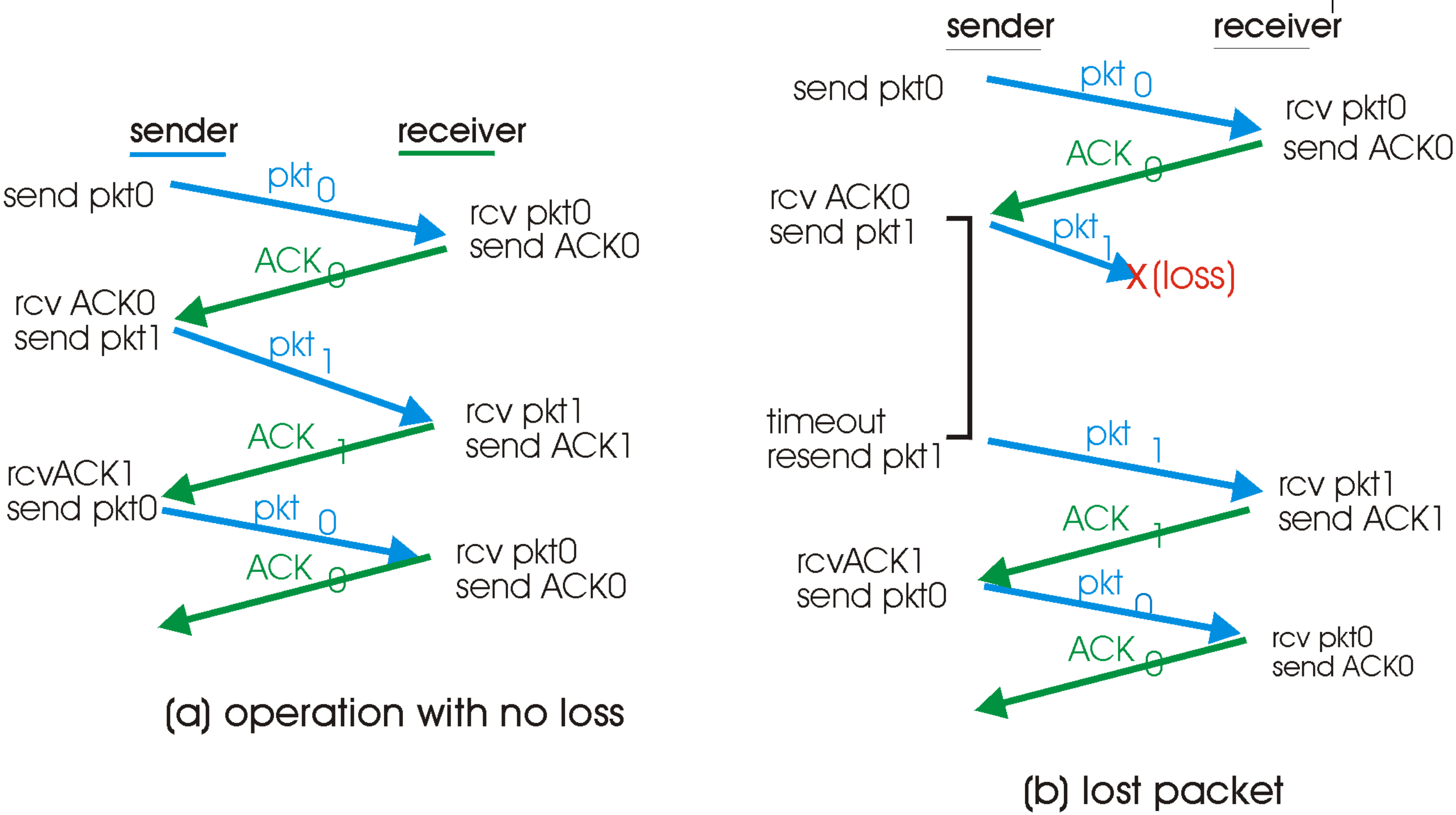
- requires countdown **timer**
- retransmits if no ACK received in this time
- if pkt (or ACK) just delayed (not lost):
  - retransmission will be duplicate, but use of seq. #'s already handles this
  - receiver must specify seq # of pkt being ACKed

rdt3.0 sender

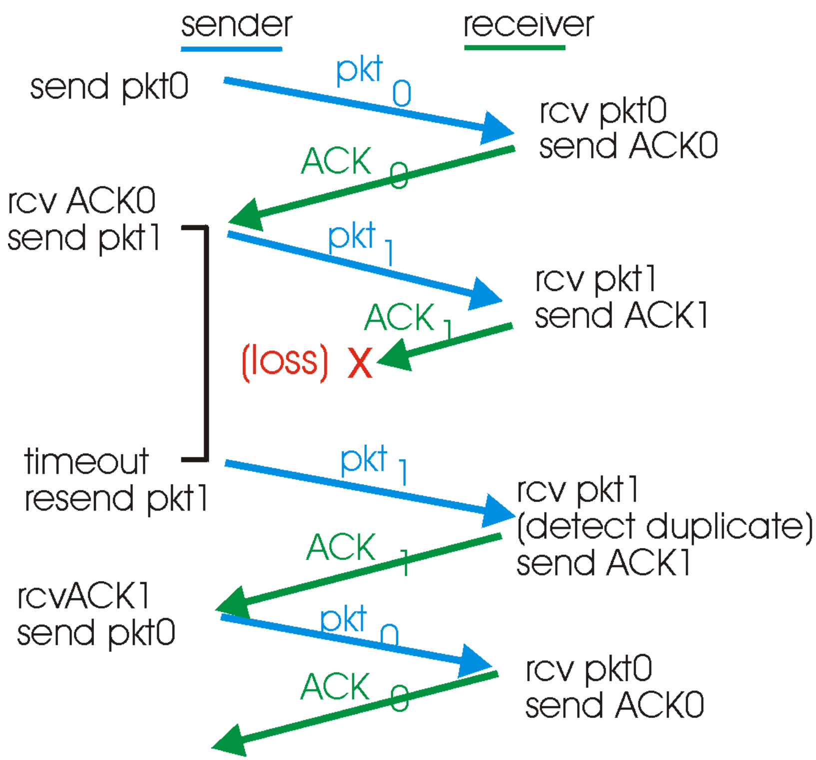
alternating-bit protocol



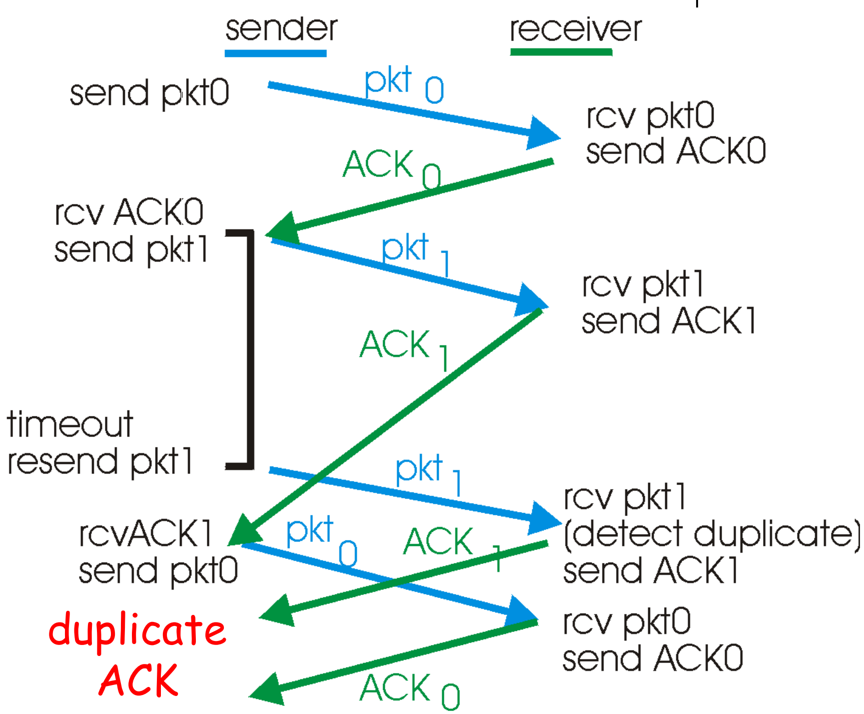
rdt3.0 in action



rdt3.0 in action



(c) lost ACK

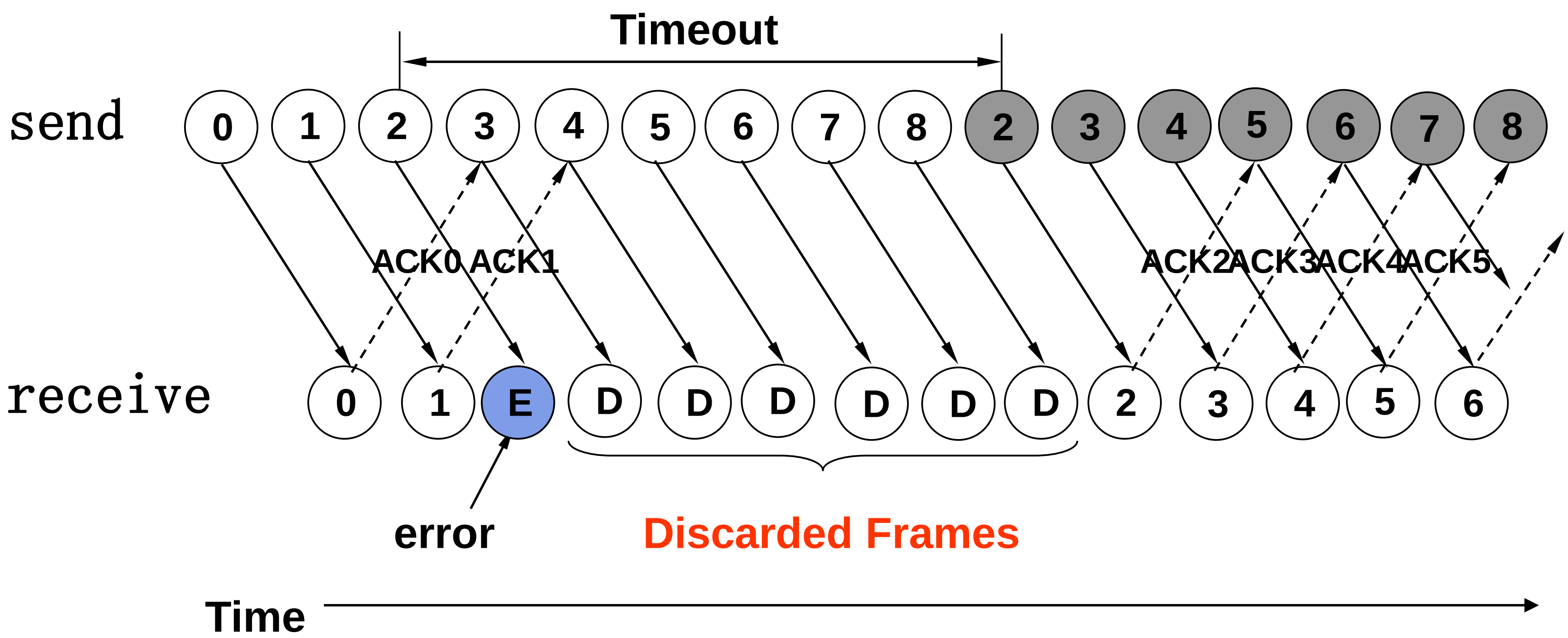


(d) premature timeout

Review

RDT	Error Control			Flow Control
	bit error	pkt loss	duplicate	
rdt1.0	x	x	x	x
rdt2.0	pkt checksum ACK/NAK+retrans	x	x	stop & wait
rdt2.1	duplex checksum ACK/NAK+retrans	x	seq# in pkt	stop & wait
rdt2.2	duplex checksum ACK+retrans	x	seq# in pkt & ack	stop & wait
rdt3.0	duplex checksum ACK+retrans	timer	seq# in pkt & ack	stop & wait

Go-Back-N （$W_S = 2^n - 1$， $W_R = 1$）



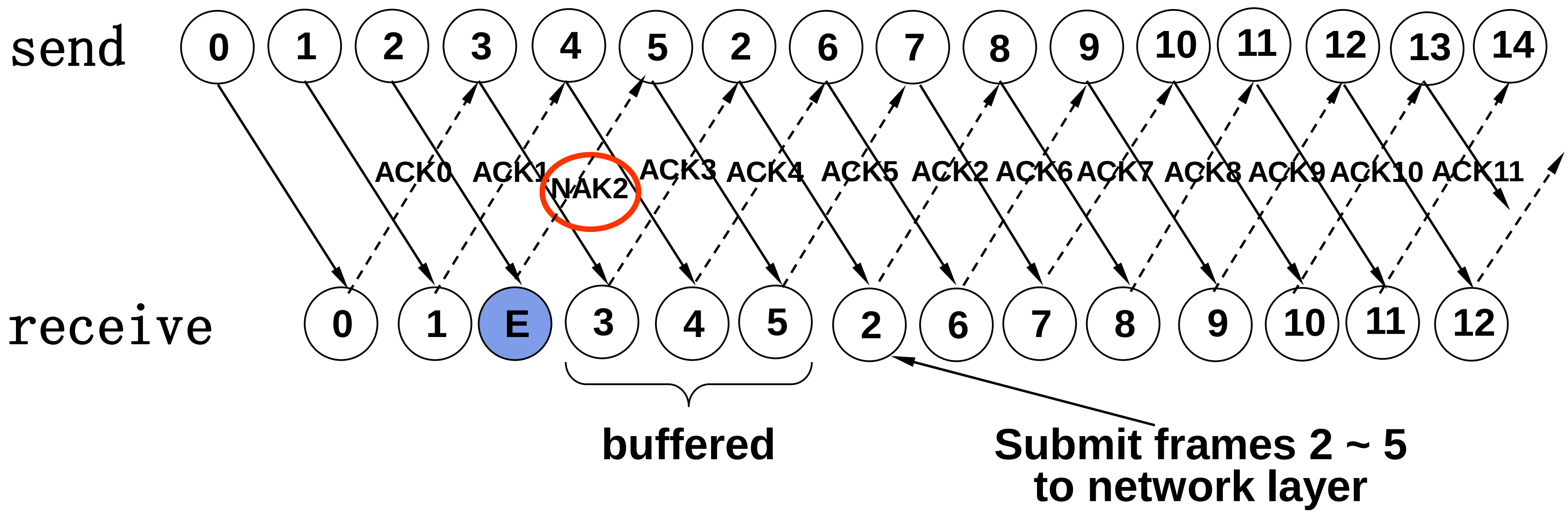
Go-Back-N （W_S=7， W_R=1）

Demo

Go-Back-N （$W_S = 2^n - 1$， $W_R = 1$）

- sending window
  - window size=2^n -1, Seq No. is n-bit counter.
  - Go back to the whole window when error detected.
- receiving window
  - window size=1
  - discard all out-of-order pkts.
- cumulative ACK: ACK(k) all pkts up to, including seq # k.
- timer for oldest in-flight pkt
  - retransmit pkt k and all higher seq # pkts in window when timeout(k).

Selective Repeat
（$W_S = W_R = 2^k/2 = 2^{k-1}$）



Selective Repeat （$W_S = 4$， $W_R = 4$）

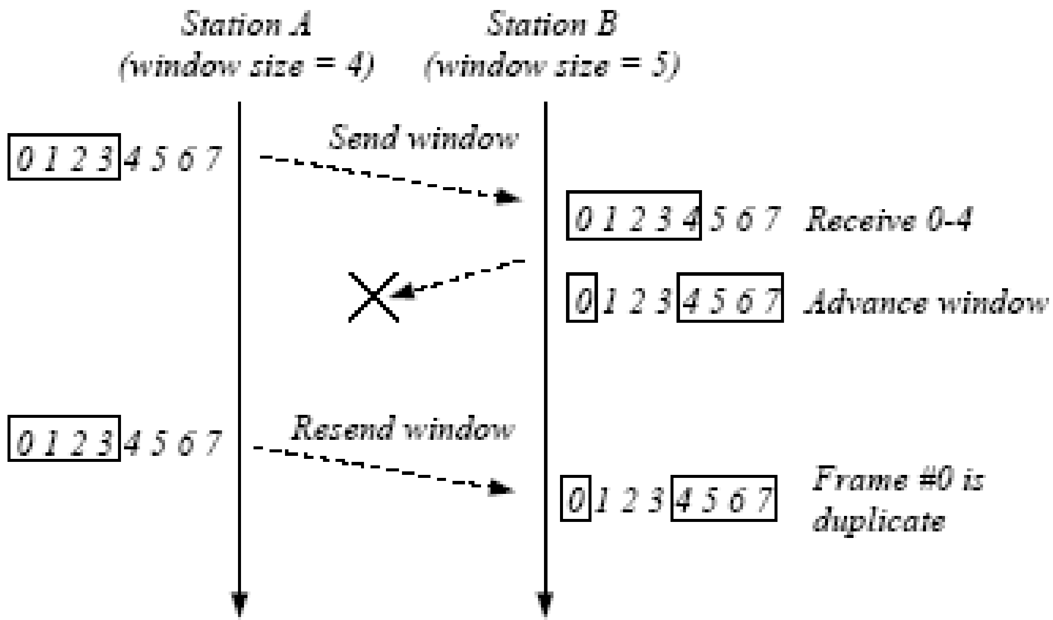
Demo

Selective Repeat （$W_S = W_R = 2^k/2 = 2^{k-1}$）

- sending window
  - N consecutive seq #'s
  - limits seq #s of sent, unACKed pkts
- receiving window
  - N consecutive seq #'s
  - buffers out-of-order pkts, as needed, for eventual in-order delivery to upper layer.
- receiver *indivirdually* acknowledges all correctly received pkts
- sender only resends pkts for which ACK not received
  - sender timer for each unACKed pkt

The Size of the Sliding Window

- **Window overlapping problem:**



- To avoid overlapping new receiving window with the original window, the maximum window size should satisfy: sending window + receiving window <= (MAX_SEQ + 1)

Summary

RDT	Error Control			Flow Control
	bit error	pkt loss	duplicate	
rdt1.0	x	x	x	x
rdt2.0	pkt checksum ACK/NAK+retrans	x	x	stop & wait
rdt2.1	duplex checksum ACK/NAK+retrans	x	seq# in pkt	stop & wait
rdt2.2	duplex checksum ACK+retrans	x	seq# in pkt & ack	stop & wait
rdt3.0	duplex checksum ACK+retrans	timer	seq# in pkt & ack	stop & wait
GBN	cumulative ACK retrans N+…	single timer	seq# in pkt & ack	sliding window
SR	individual ACK/NAK single retrans	multiple timer	seq# in pkt & ack	sliding window

Summary of reliable data transfer mechanisms

Mechanism	Use and Comments
Checksum	Used to detect bit errors in a transmitted pocket.
Timer	Used to timeout/retransmit a packet, possibly because the packet (or its ACK) was lost within the channel. Because timeouts can occur when a pocket is delayed but not lost (premature timeout), or when a packet has been received by the receiver but the receiver-to-sender ACK has been lost, duplicate copies of a packet may be received by a receiver.
Sequence number	Used for sequential numbering of packets of data flowing from sender to receiver. Gaps in the sequence numbers of received packets allow the receiver to detect a lost packet. Packets with duplicate sequence numbers allow the receiver to detect duplicate copies of a packet.
Acknowledgment	Used by the receiver to tell the sender that a packet or set of packets has been received correctly. Acknowledgments will typically carry the sequence number of the packet or packets being acknowledged. Acknowledgments may be individual or cumulative, independently or piggybacked, depending on the protocol.
Negative acknowledgment	Used by the receiver to tell the sender that a packet has not been received correctly. Negative acknowledgments will typically carry the sequence number of the packet that was not received correctly.
Sliding window, pipelining	The sender may be able to send multiple packets with sequence numbers that fall within a given range. By allowing multiple packets to be transmitted but not yet acknowledged, sender utilization can be increased over a stop-and-wait mode of operation. The window size may be set on the basis of the receiver's ability to receive and buffer messages, or the level of congestion in the network, or both .