
Framing, Error Detection, ARQ

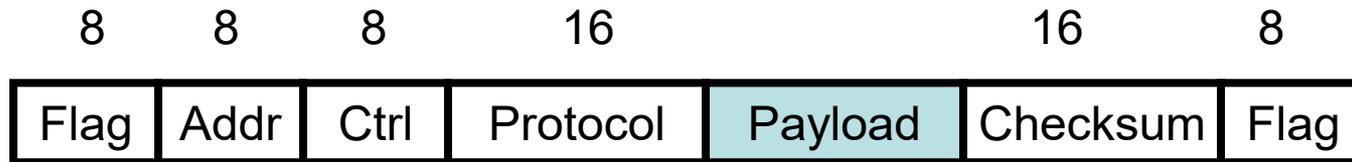
9 November 2025
Lecture 3

Slides Credits: Steve Zdancewic (UPenn)

Topics for Today

- Link Layer
 - Framing
- Errors
 - Error Detection
 - Error Correction
- Reliable Transmission
 - ARQ
 - Sliding Window
- Source: Peterson and Davie 2.1-2.5, 2.6, Tanenbaum 4.3

(PPP) Point-to-Point Protocol



PPP frame format

Used for dial-up connections (modem)

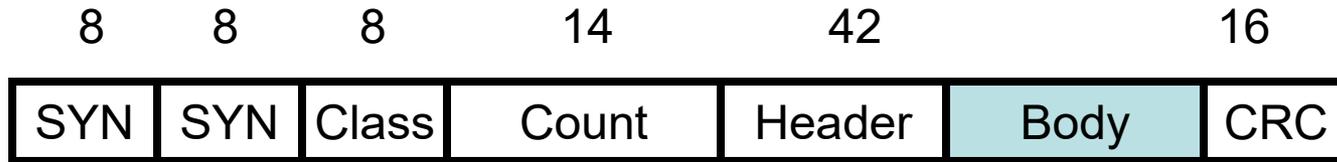
Flag – sentinel
01111110

Protocol – demux identifies high-level protocol such as IP or LCP

Payload size is negotiated

- 1500B default
- *Link Control Protocol (LCP)*

Byte-counting: DDCMP



DDCMP Frame Format

- Instead of sentinels, include *byte count* in frame.
- What happens if count is corrupted?

So Far

- Link Layer
 - Framing
- Errors
 - Error Detection
 - Error Correction
- Reliable Transmission
 - ARQ
 - Sliding Window

Problem: Error Detection & Correction

- Bit errors may be introduced into frames
 - Electrical interference
 - Thermal noise
- Could flip one bit or a few bits independently
- Could zero-out or flip a sequence of bits (*burst error*)

How do you
detect an
error?

What do you
do once you
find one?

Error Detection

- General principal: Introduce redundancy

Trivial example: send two copies

- High overheads: $2n$ bits to send n
- Won't detect errors that corrupt same bits in both copies

How can we do better?

- Minimize overhead
- Detect many errors
- General subject: *error detecting codes*

Simple Error Detection Schemes

Parity bit

- 7 bits of data
- 8th bit is sum of first seven bits *mod* 2

Overhead:
 $8n$ bits to
send $7n$

Detects: any
odd number
of bit errors

Simple Error Detection Schemes

Internet Checksum algorithm

- Add up the words of the message, transmit sum
- 16 bit ones-complement addition
- <https://www.youtube.com/watch?v=EmUuFRMJbss>

Overhead:
16 bits to
send n

Does not
detect all 2-
bit errors

Cyclic Redundancy Check



Used in link-level protocols

- CRC-32 used by Ethernet, 802.5, PKzip, ...
- CRC-CCITT used by HDLC
- CRC-8, CRC-10, CRC-32 used by ATM

Simple to implement

Better than parity or checksum

- (e.g. 32 bits to send 12000)

For more info

- Wikipedia entry on CRC
- https://en.wikipedia.org/wiki/Cyclic_redundancy_check

Cyclic Redundancy Check (CRC)

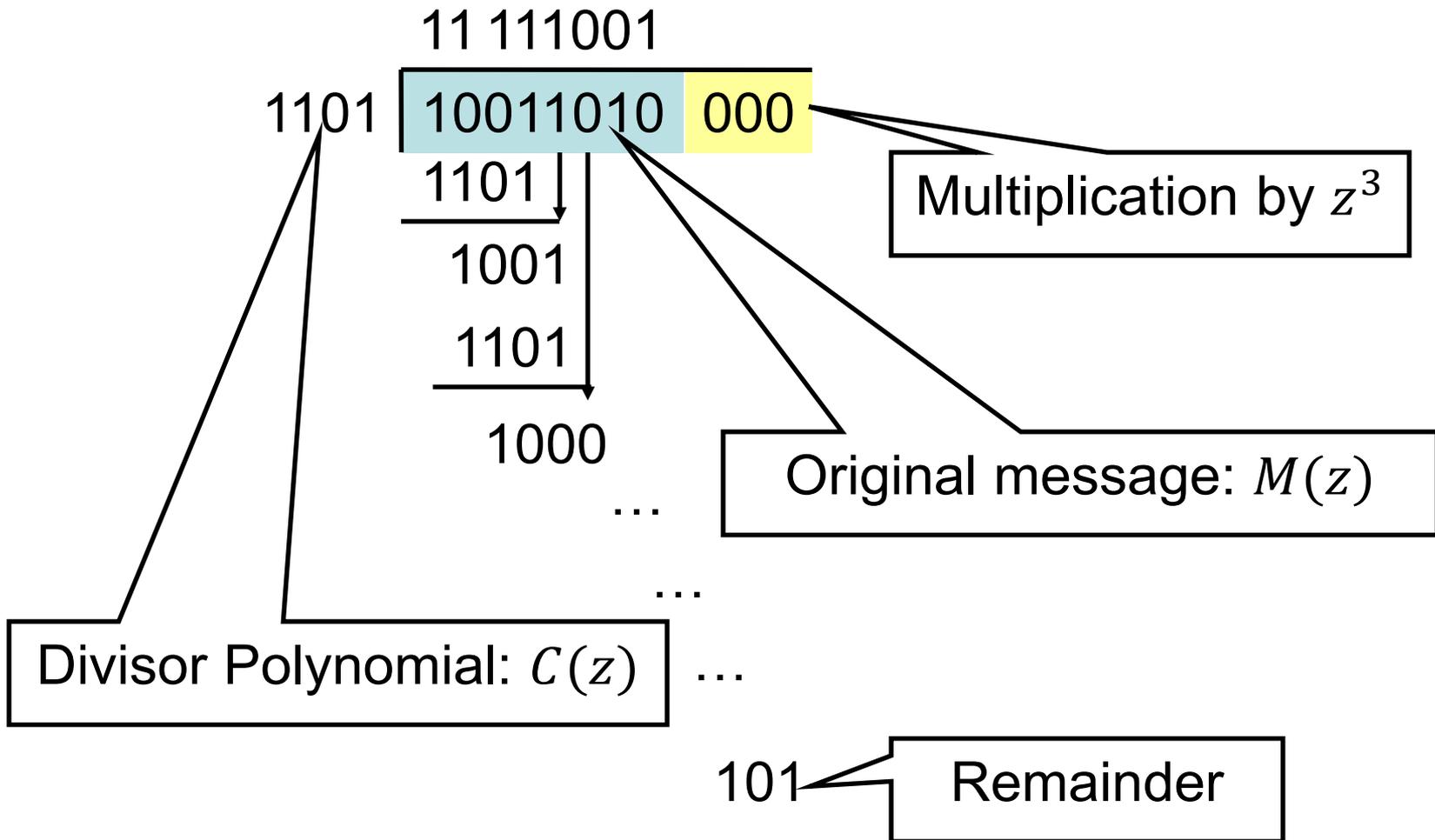
- Consider $(n + 1)$ –bit message as a n -degree polynomial
 - Polynomial arithmetic modulo 2
 - Bit values of message are coefficients
 - Message = 10011010
 - Polynomial:

$$\begin{aligned}M(z) &= (1 \times z^7) + (0 \times z^6) + (0 \times z^5) + (1 \times z^4) + (1 \times z^3) \\ &+ (0 \times z^2) + (1 \times z^1) + (0 \times z^0) \\ &= z^7 + z^4 + z^3 + z^1\end{aligned}$$

CRC Algorithm

1. Sender and receiver agree on a *divisor polynomial* $C(z)$ of degree k
 1. Example $k = 3$
 2. $C(z) = z^3 + z^2 + 1$
 3. Coefficients are 1101
2. Error correction bits are remainder of $(M(z) \times z^k)$ divided by $C(z)$
3. This yields a $n + k$ bit transmission polynomial $P(z)$ that is *exactly* divisible by $C(z)$

Example CRC Calculation



	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O
1					1	1	1	1	1	0	0	1			
2	1	1	0	1	1	0	0	1	1	0	1	0	0	0	0
3					1	1	0	1							
4					0	1	0	0	1						
5						1	1	0	1						
6						0	1	0	0	0					
7							1	1	0	1					
8							0	1	0	1	1				
9								1	1	0	1				
10								0	1	1	0	0			
11									1	1	0	1			
12									0	0	0	1	0	0	0
13												1	1	0	1
14												0	1	0	1

Example CRC Calculation

$$\begin{array}{r} z^3 \times \text{Original Message } M(z) = \quad 10011010 \quad 000 \\ \text{Remainder} = \quad + \quad \quad \quad 101 \\ \hline \text{Transmitted message } P(z) = \quad 10011010 \quad 101 \end{array}$$

- Recipient checks that $C(z)$ evenly divides the received message.

	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O
1					1	1	1	1	1	0	0	1			
2	1	1	0	1	1	0	0	1	1	0	1	0	1	0	1
3					1	1	0	1							
4					0	1	0	0	1						
5						1	1	0	1						
6						0	1	0	0	0					
7							1	1	0	1					
8							0	1	0	1	1				
9								1	1	0	1				
10								0	1	1	0	0			
11									1	1	0	1			
12									0	0	0	1	1	0	1
13												1	1	0	1
14												0	0	0	0

CRC Error Detection

- Must choose a good divisor $C(z)$
 - There are many standard choices:
 - CRC-8, CRC-10, CRC-12, CRC-16, CRC-32,
 - CRC-32: $0x04C11DB7$ (1 0000 0100 1100 0001 0001 1101 1011 0111
or $x^{32} + x^{26} + x^{23} + x^{22} + x^{16} + x^{12} + x^{11} + x^{10} + x^8 + x^7 + x^5 + x^4 + x^2 + x + 1$)

Detects:

All 1-bit errors as long as z^k and z^0 coefficients are 1

All 2-bit errors as long as $C(z)$ has three terms

Any odd number of errors if $(z + 1)$ divides $C(z)$

Any burst errors of length $\leq k$

CRC Implementations

- Easy to implement in hardware
 - Base 2 subtraction is XOR
 - Simple k -bit shift register with XOR gates inserted before 1's in $C(z)$ polynomial (except the first one)
 - Message is shifted in, registers fill with remainder

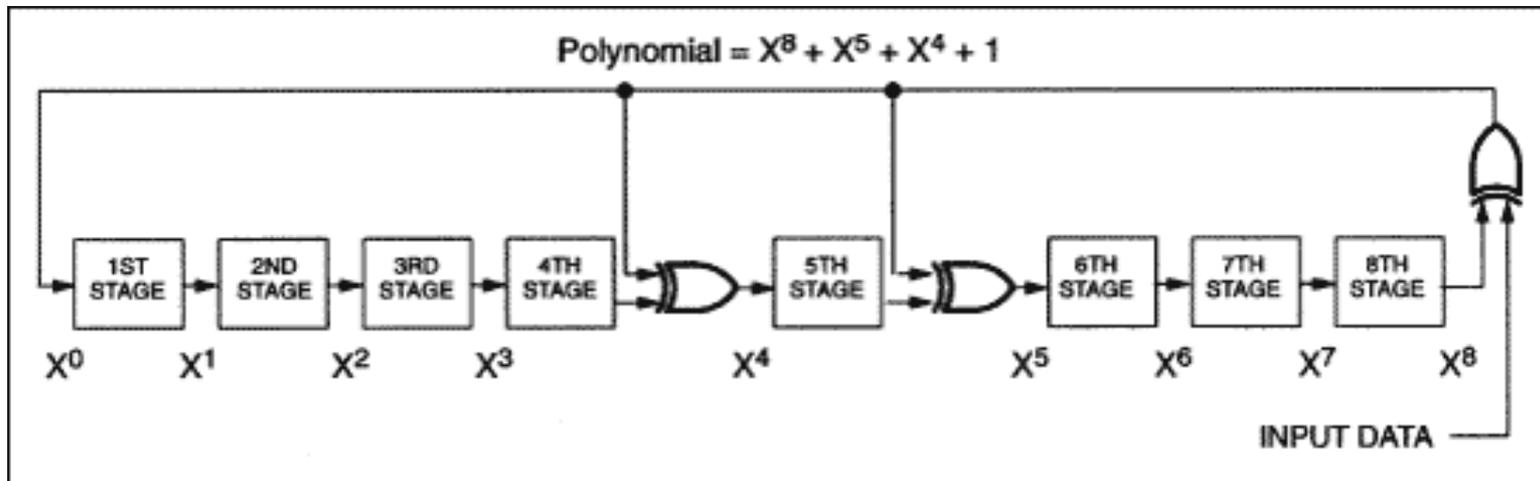
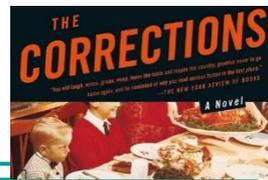


Image source: <https://www.analog.com/en/resources/technical-articles/understanding-and-using-cyclic-redundancy-checks-with-maxim-1-wire-and-ibutton-products.html>



Error Correction Codes

Redundant information can be used to *correct* some errors

- Typically requires more redundancy

Tradeoffs:

- Error detection requires retransmission
- Error correction sends more bits all the time

Forward Error Correction is useful:

- When errors are likely (e.g. wireless network)
- When latency is too high for retransmission (e.g. satellite link)

So Far

- Link Layer
 - Framing
- Errors
 - Error Detection
 - Error Correction
- Reliable Transmission
 - ARQ
 - Sliding Window

We found an error. Now what?

What should the sender and receiver do?

Acknowledgments (ACK)

Small control frame/packet (little data)

When sender gets an ACK, recipient has successfully gotten a frame

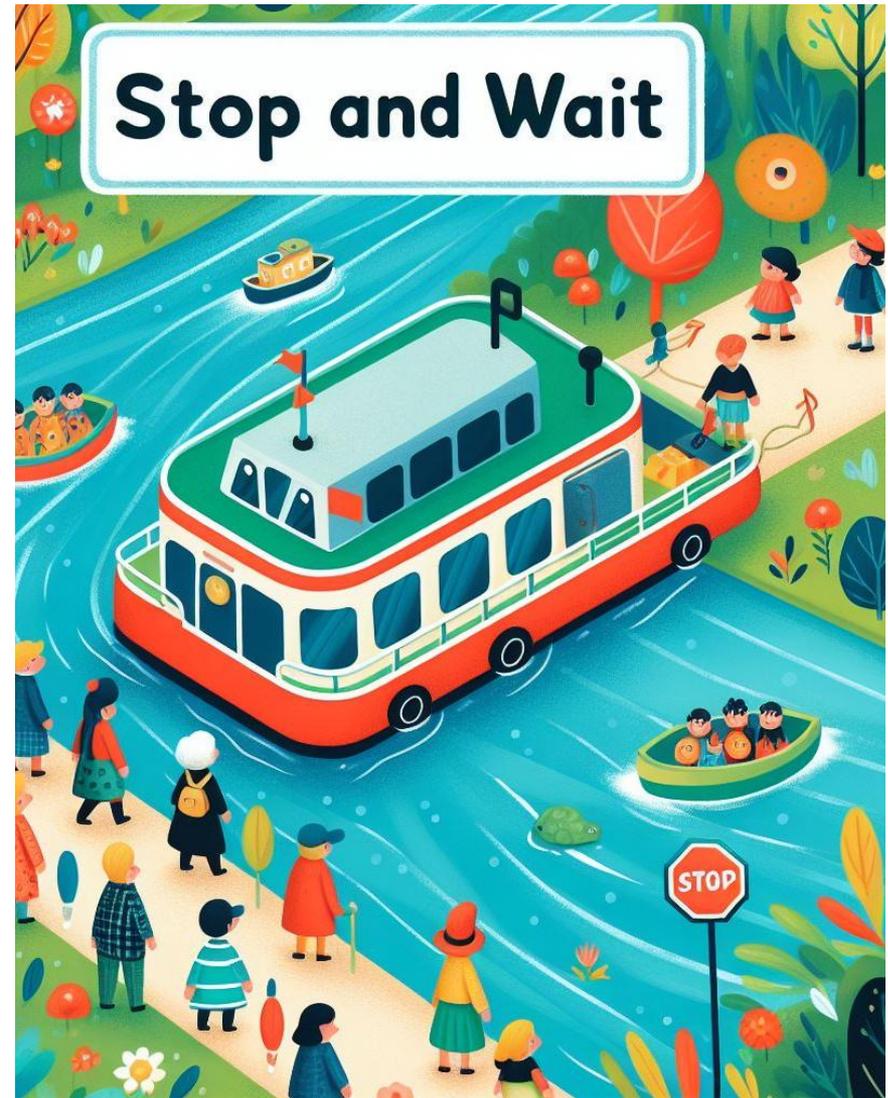
Timeouts

If sender doesn't get an ACK after "reasonable" time it retransmits the original

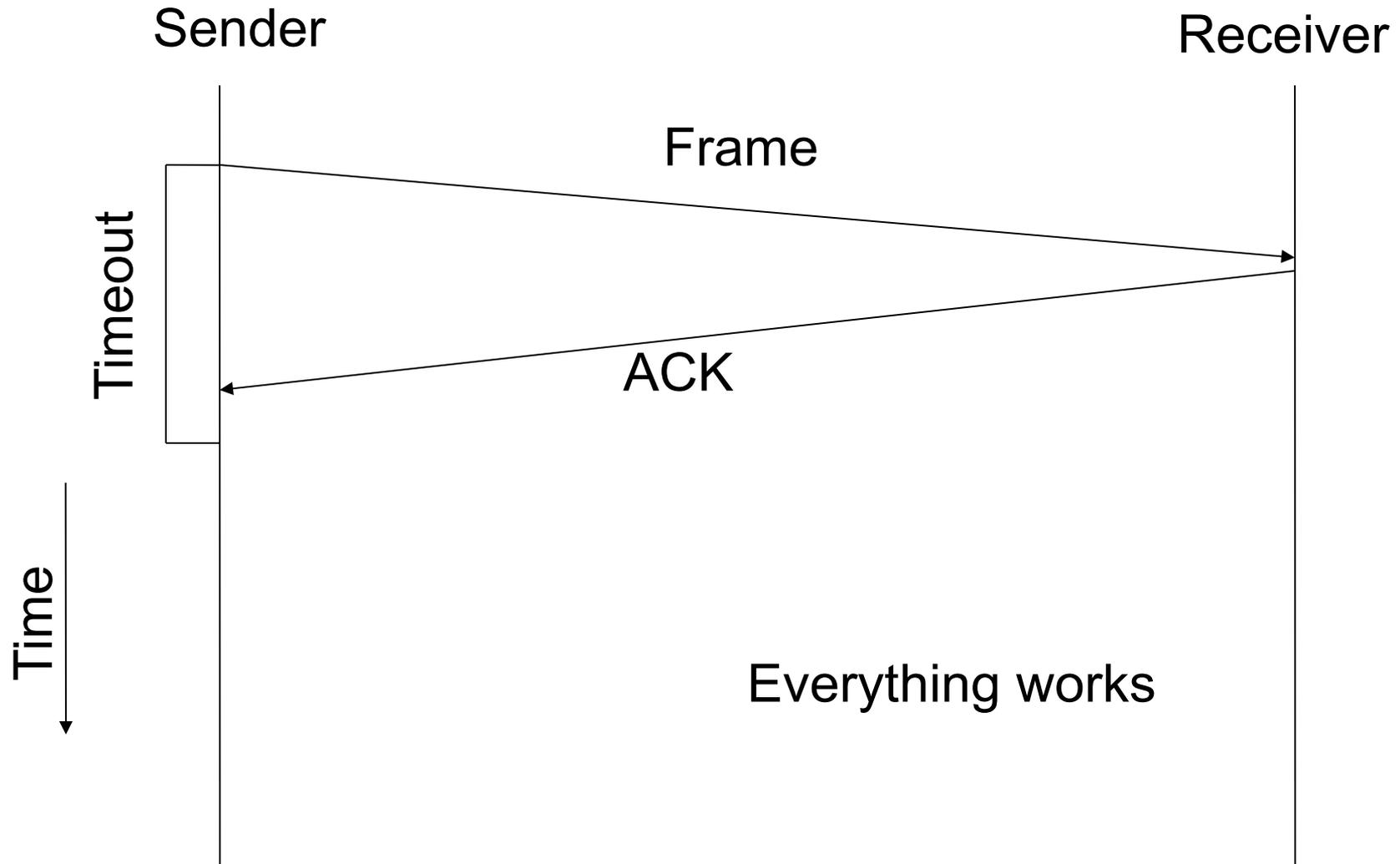
General strategy called *Automatic Repeat Request (ARQ)*

Stop-and-Wait - Simplest scheme

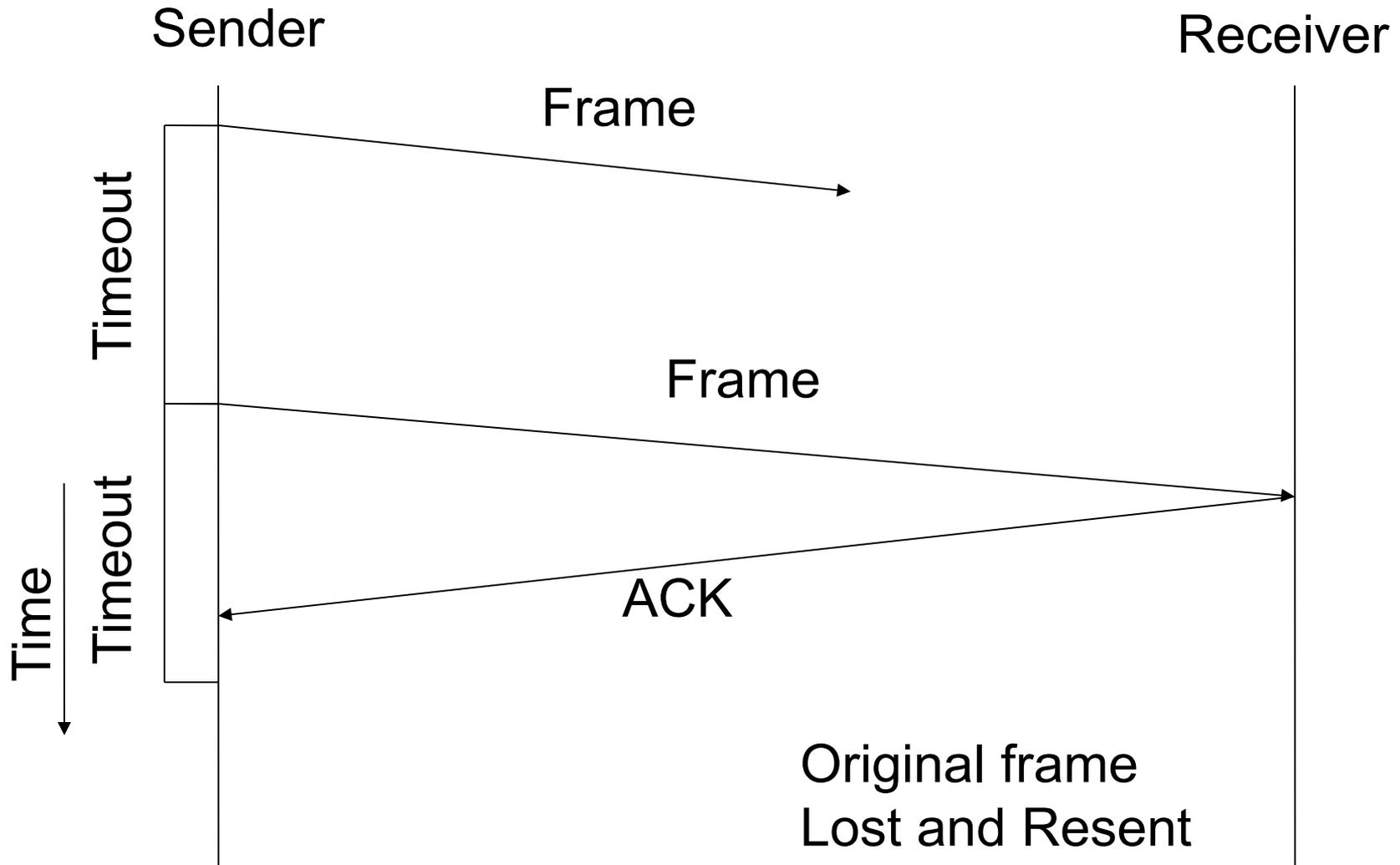
1. After transmitting one frame, sender waits for an ACK
2. If the ACK doesn't arrive, sender retransmits



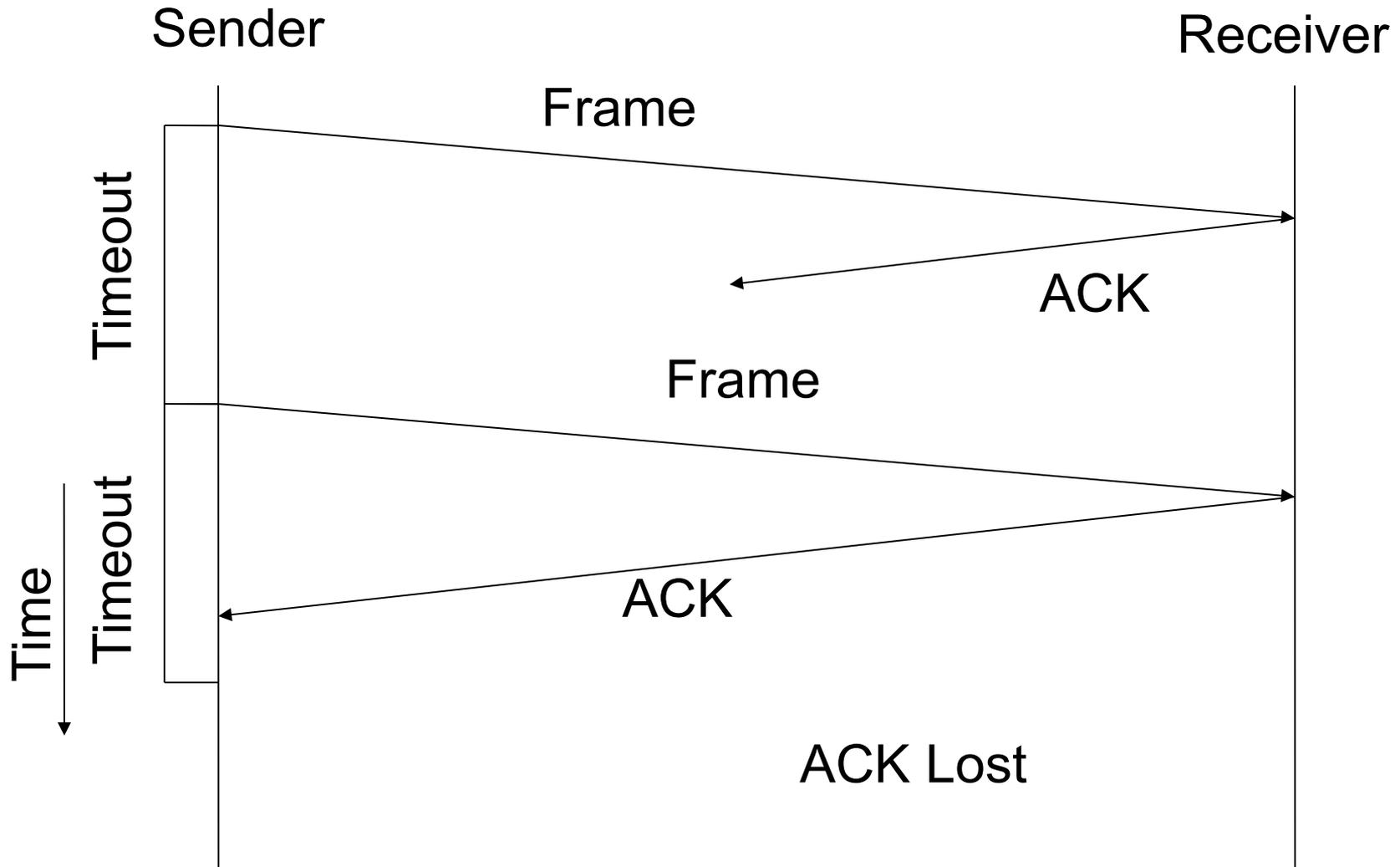
Stop-and-Wait scenario 1



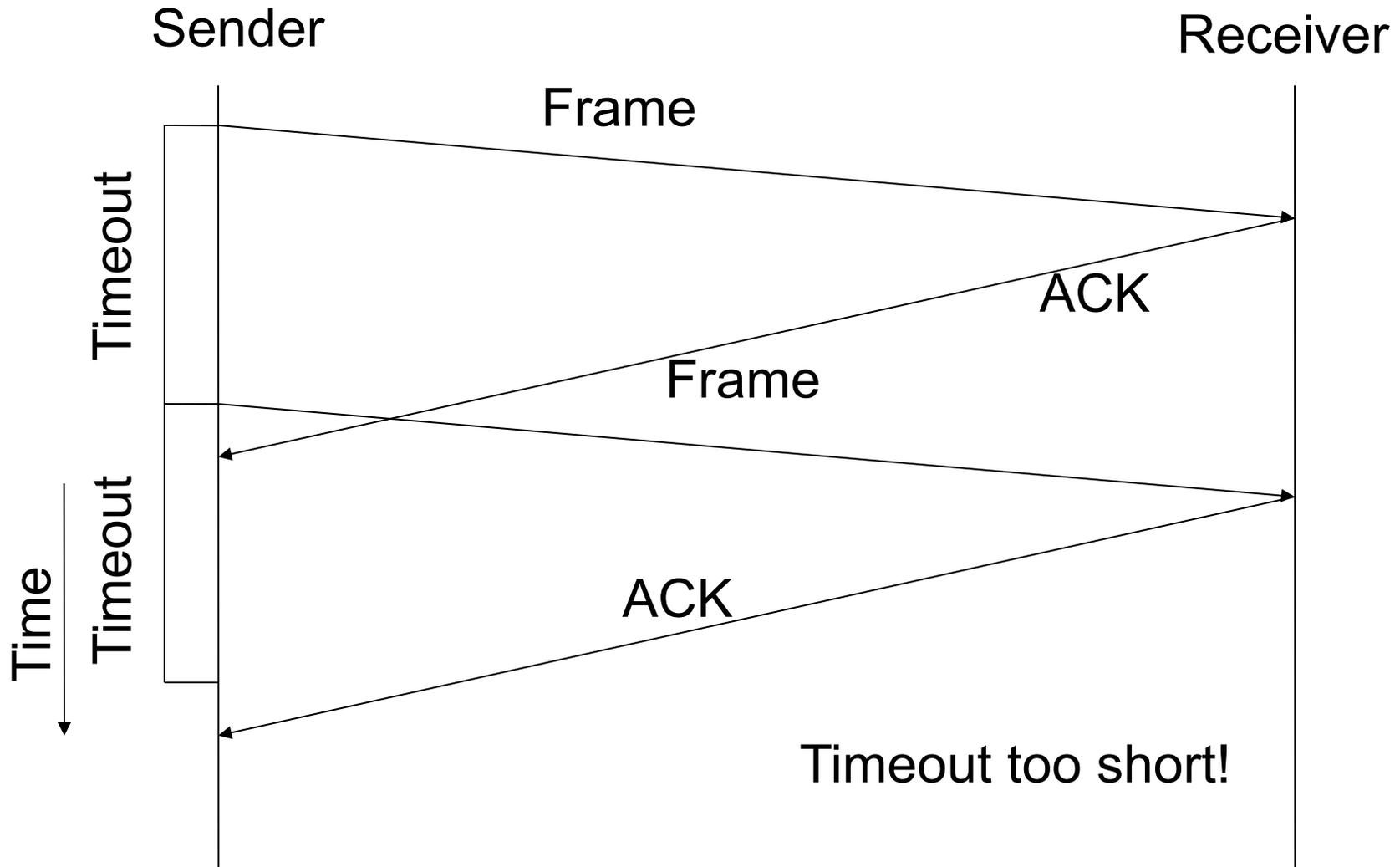
Stop-and-Wait scenario 2



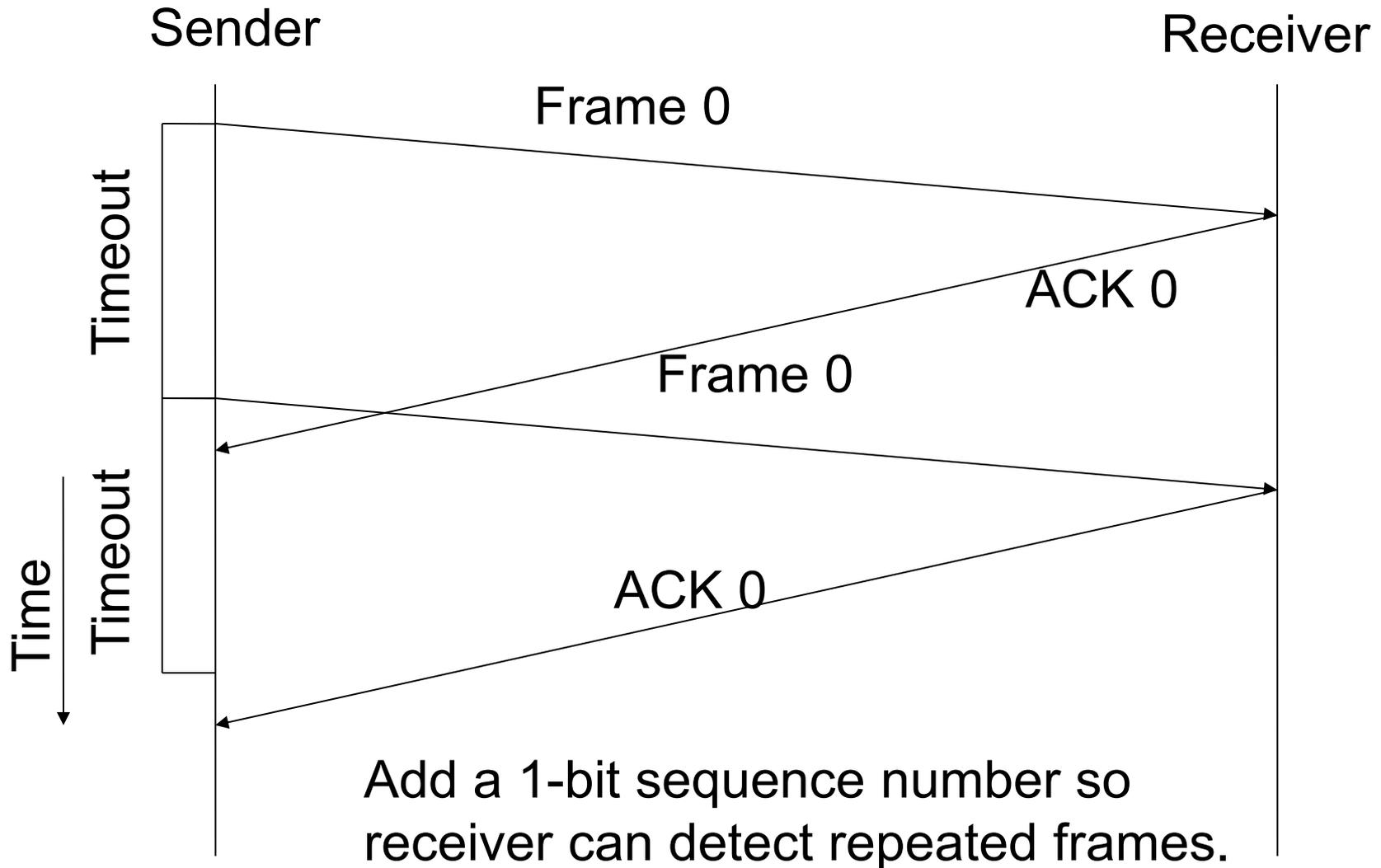
Stop-and-Wait scenario 3



Stop-and-Wait scenario 4



Sequence numbers



Stop-and-Wait

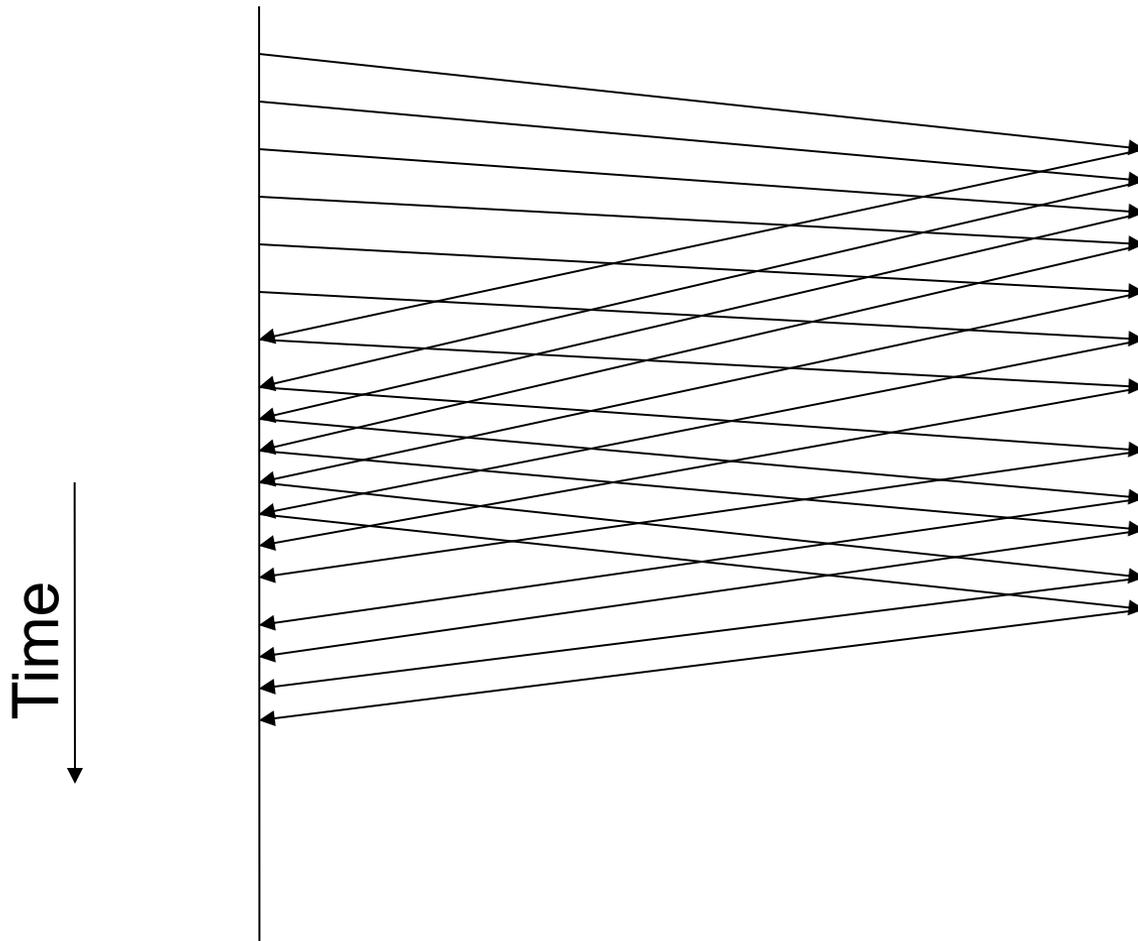
- Inefficient use of link's capacity
- Sends 1 frame per RTT

- Example:
 - 10Mbps Link
 - 16ms RTT
 - Delay x Bandwidth product is about 20KB
 - Frame size of 1K yields about 5% link capacity

More efficient solution

Sender

Receiver



$$N = \frac{RTT}{2} \times BW$$

frame size

Sender ready
to transmit $N + 1^{st}$
frame just as
first ACK arrives.

So Far

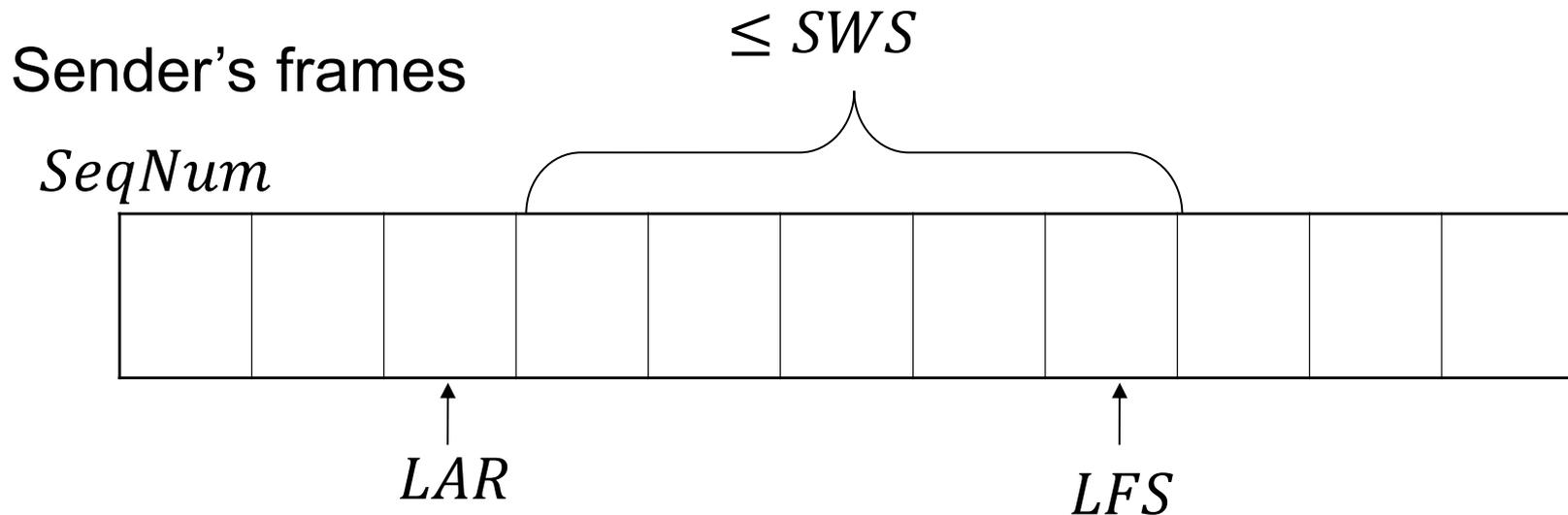
- Link Layer
 - Framing
- Errors
 - Error Detection
 - Error Correction
- Reliable Transmission
 - ARQ
 - Sliding Window

Sliding Window Algorithm

- Sender assigns a *sequence number* to each frame:
SeqNum
 - For now, assume *SeqNum* can grow infinitely
- Send Window Size (*SWS*)
 - Upper bound on # of unacknowledged frames sender will transmit
- Last ACK Received (*LAR*)
 - Sequence number of last ACK
- Last Frame Sent (*LFS*)

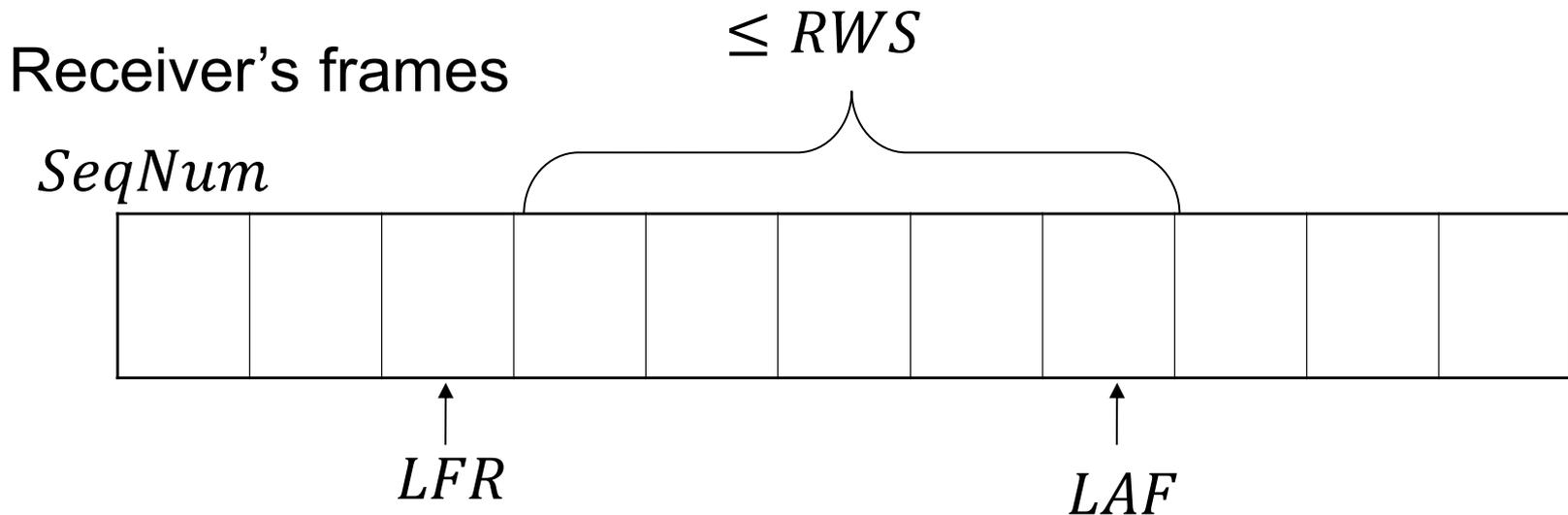


Sender Invariant



- $LFS - LAR \leq SWS$
- Associates timeout with each frame sent
 - Retransmits if no ACK received before timeout
- When ACK arrives, increase *LAR*
 - Means another frame can be sent

Receiver Invariant

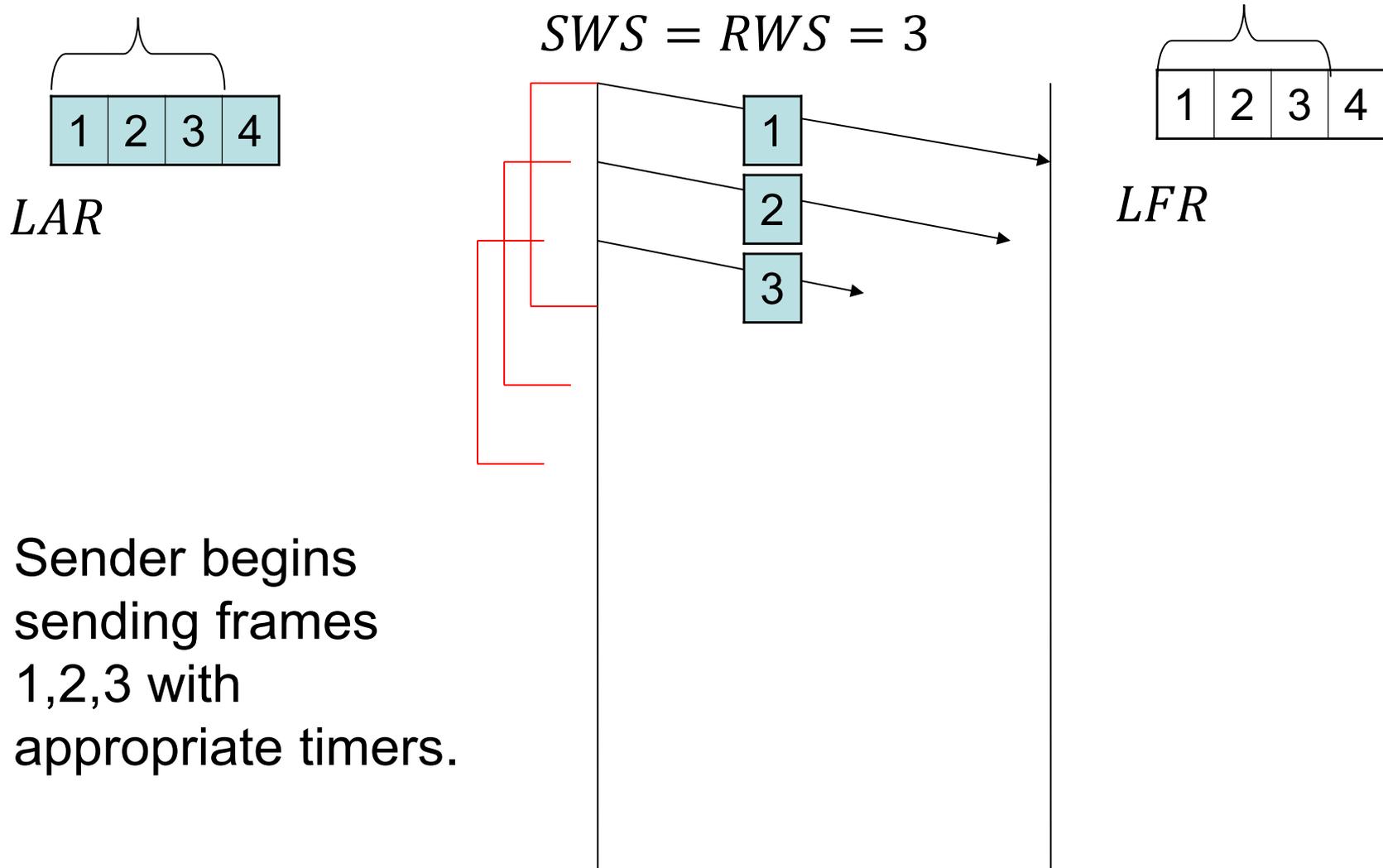


- Receive Window Size (*RWS*)
 - Number of out-of-order frames it will accept
- Largest Acceptable Frame (*LAF*)
 - Largest Frame Received (*LFR*)
- $LAF - LFR \leq RWS$

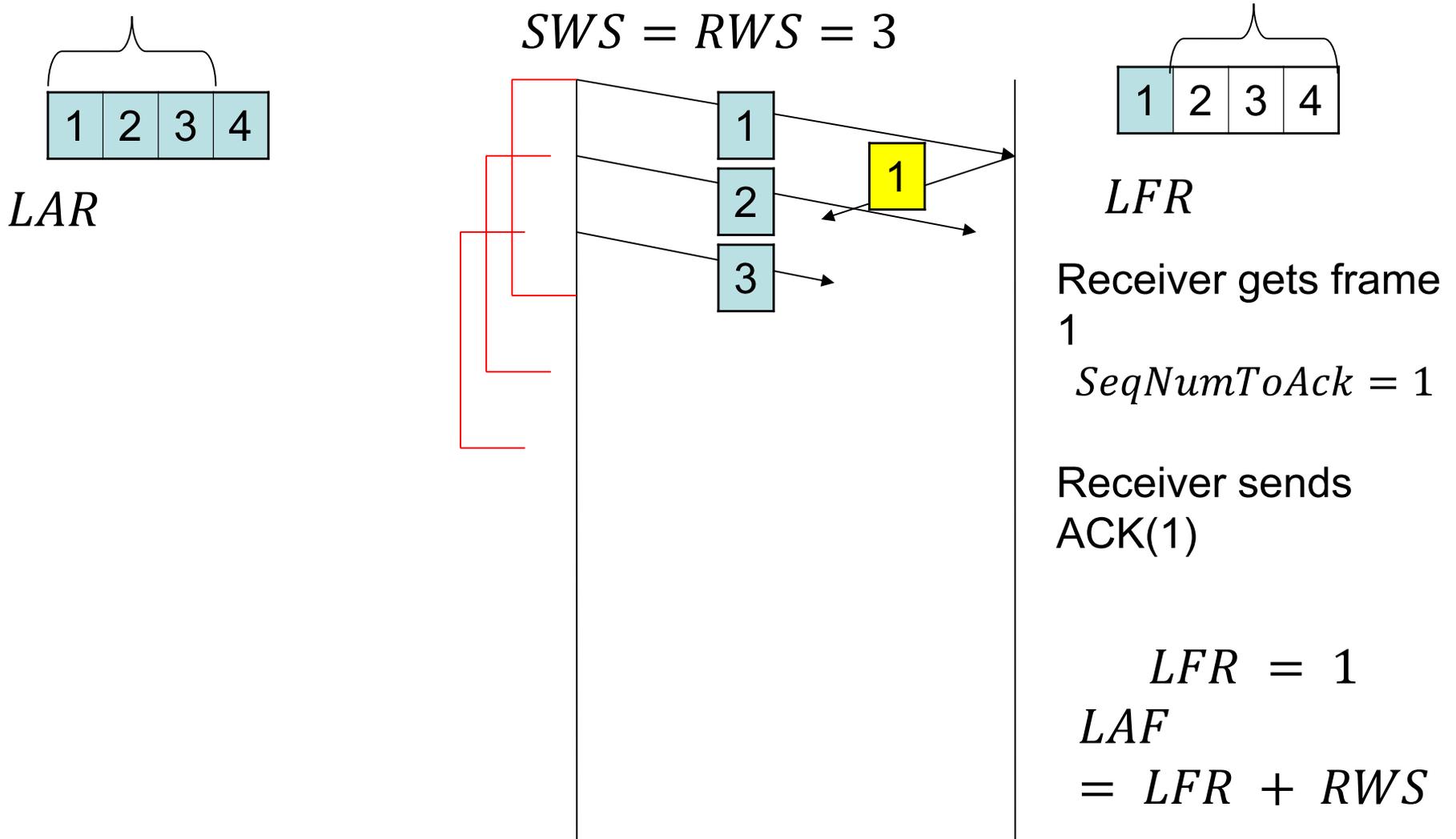
Receiver Algorithm

- When packet numbered $SeqNum$ arrives
 - If $(SeqNum \leq LFR)$ or $(SeqNum > LAF)$ discard
 - Else accept the packet
- Define: $SeqNumToAck$
 - Largest unACK'ed sequence number such that all earlier frames have been accepted
- Receiver sends $ACK(SeqNumToAck)$
- $LFR = SeqNumToAck$
- $LAF = LFR + RWS$

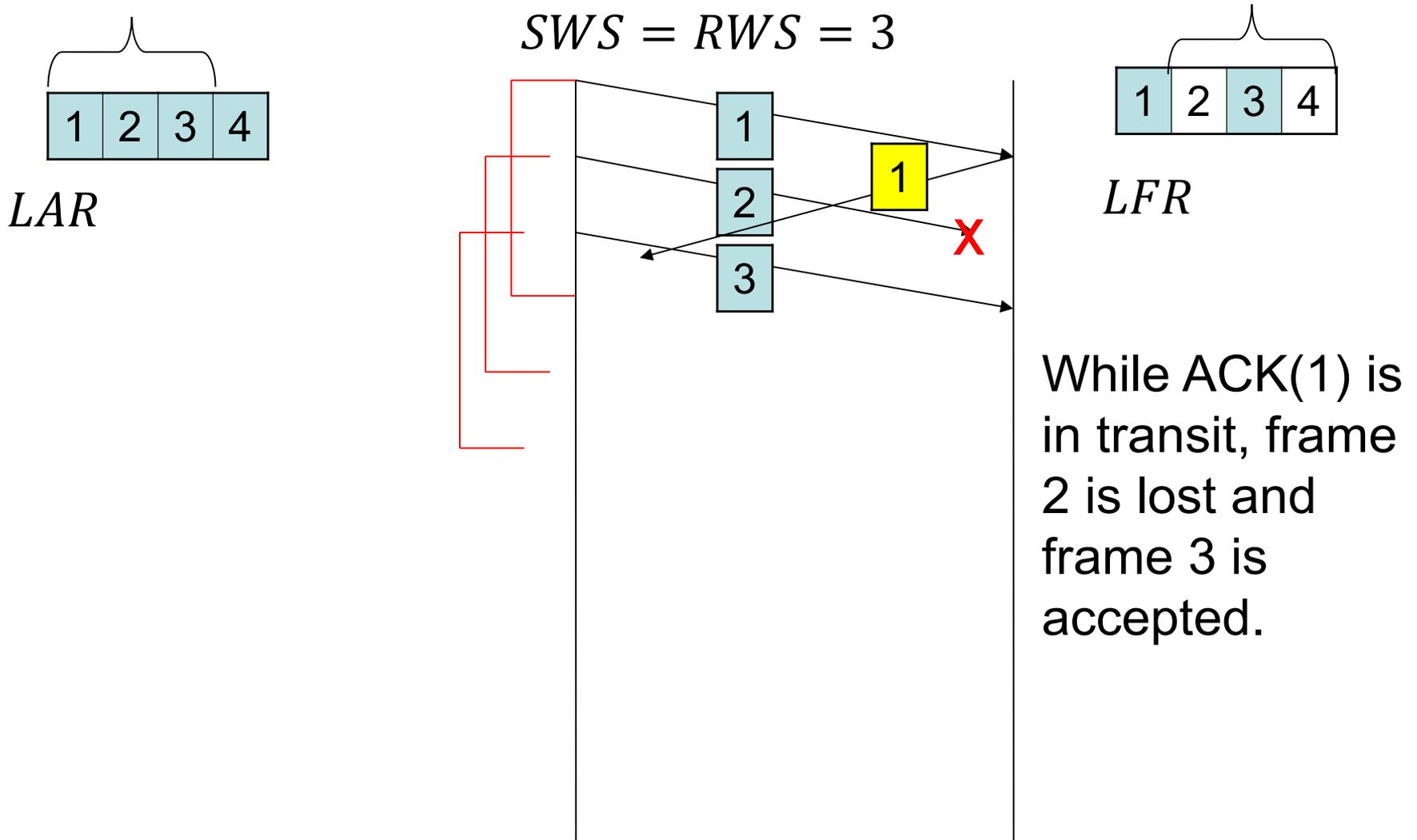
Example Sliding Window Protocol



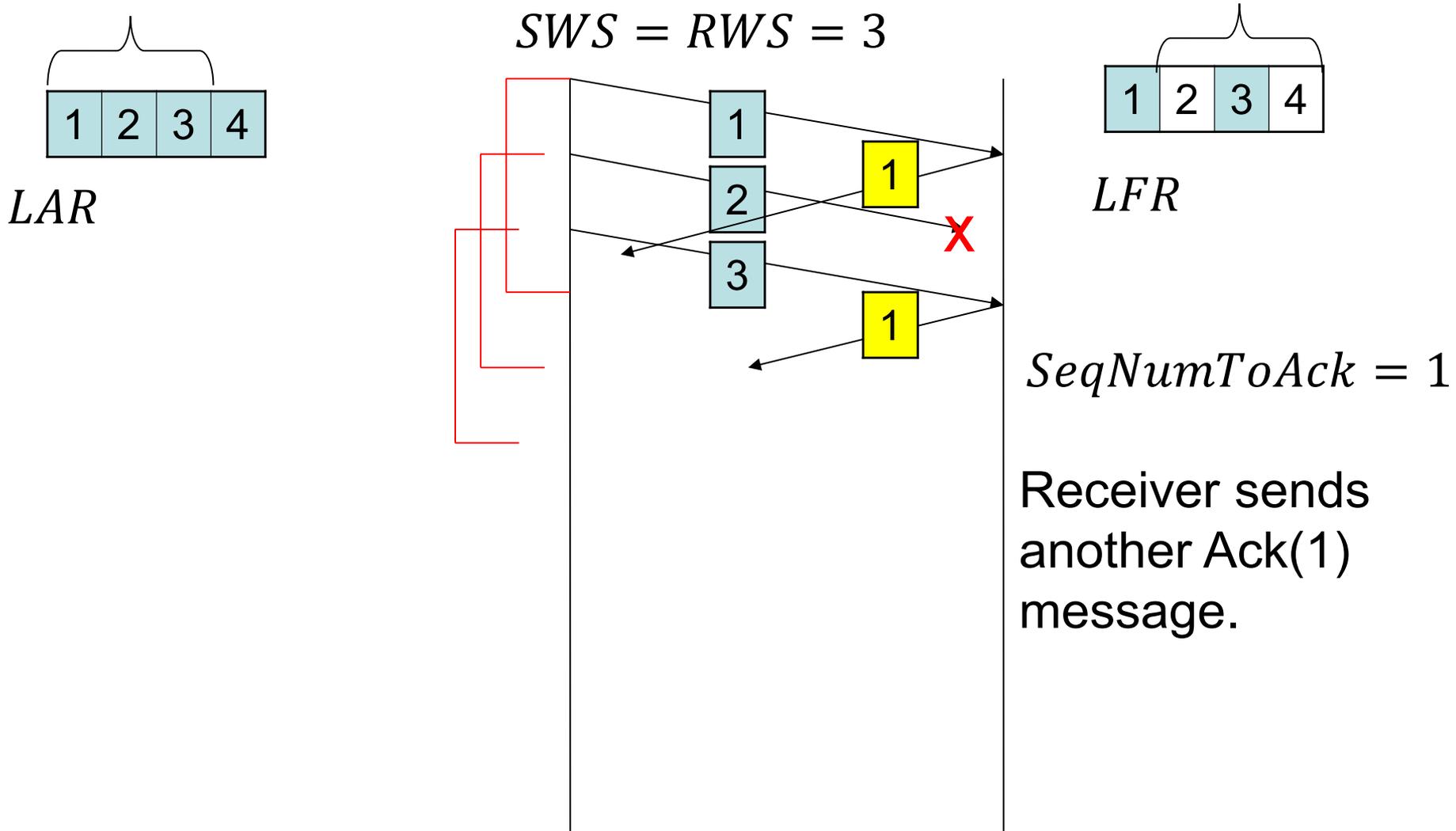
Example Sliding Window Protocol



Example Sliding Window Protocol

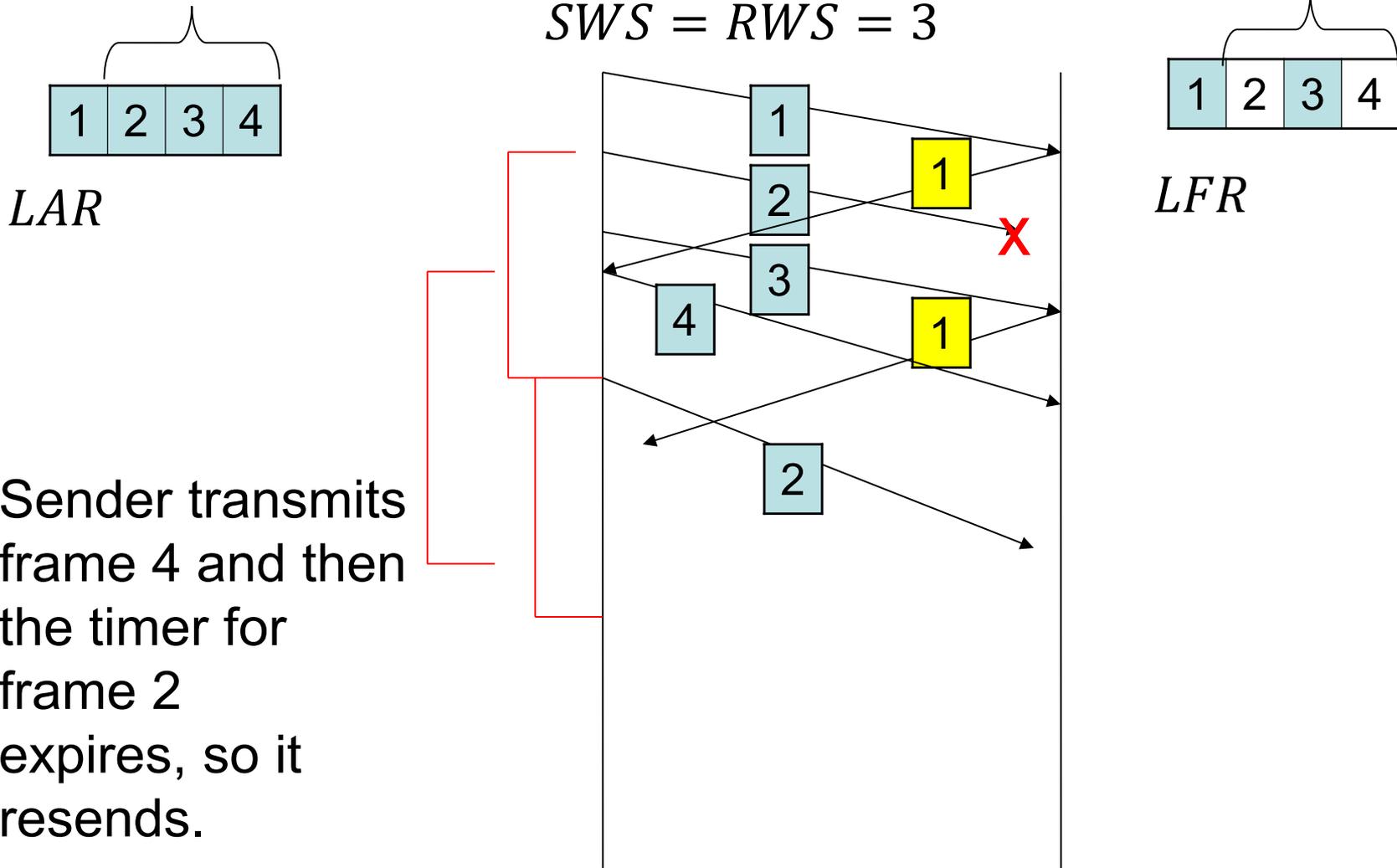


Example Sliding Window Protocol

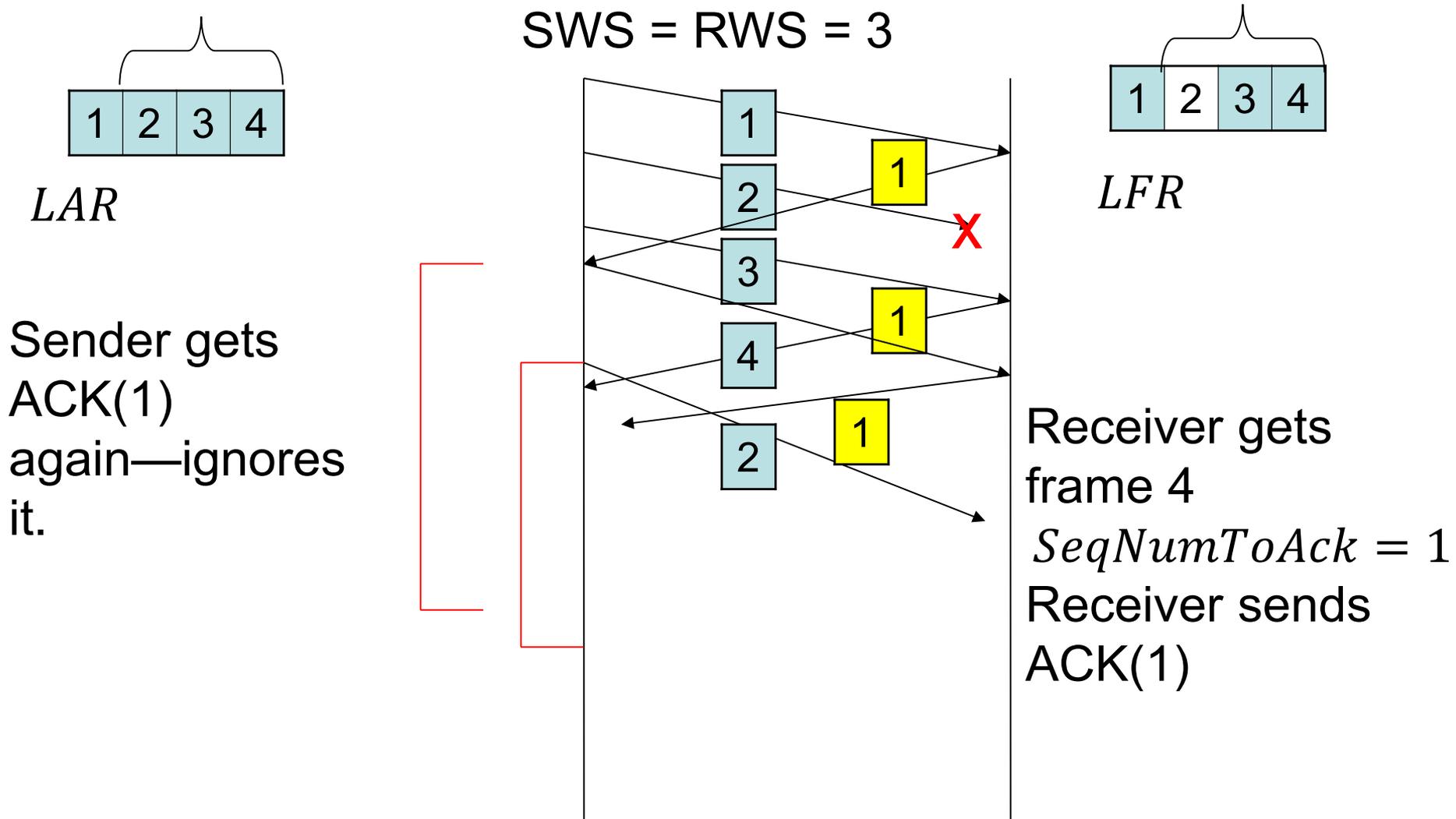


Example Sliding Window Protocol

$$SWS = RWS = 3$$

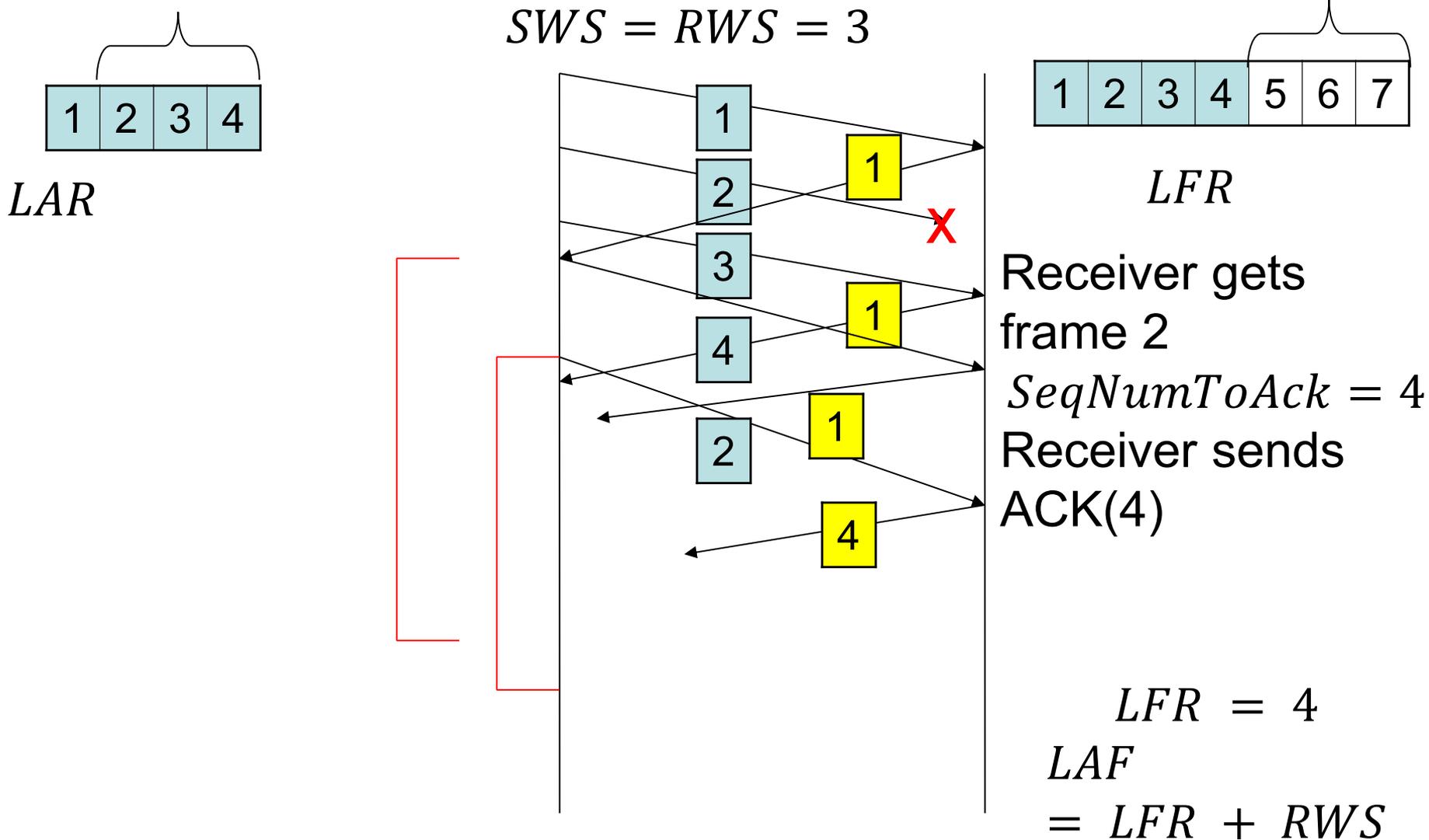


Example Sliding Window Protocol

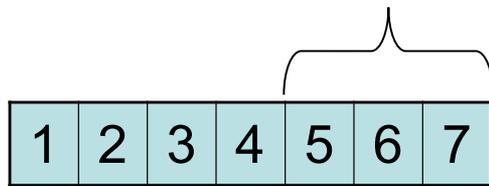


Example Sliding Window Protocol

$$SWS = RWS = 3$$

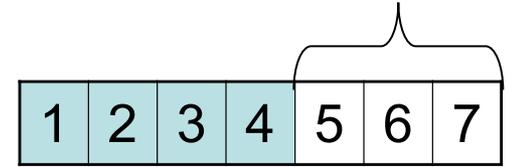


Example Sliding Window Protocol



LAR

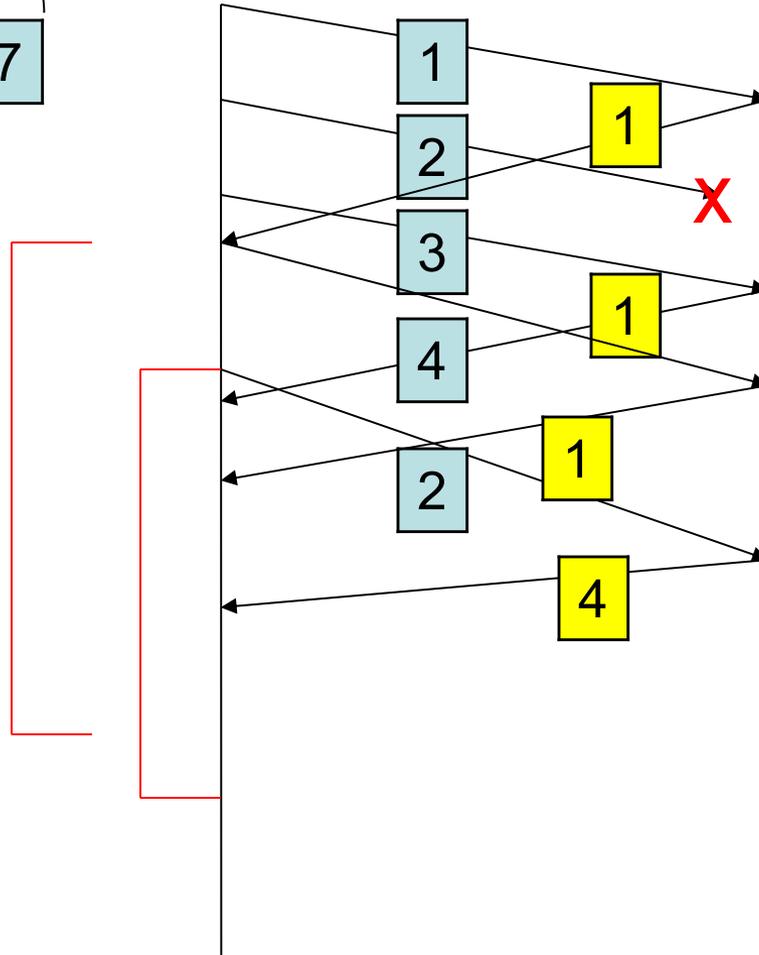
$$SWS = RWS = 3$$



LFR

Sender gets
ACK(1)
again—ignores it.

Sender gets
ACK(4)
Sets *LAR* = 4
Increases *LFS*



Variants on Sliding Window

Receiver doesn't transmit redundant ACKs

- Just ignore out of order arrivals

Receiver transmits *selective ACKS*

- ACK indicates exactly which frames have been accepted

Conclusion

- Link Layer
 - Framing
- Errors
 - Error Detection
 - Error Correction
- Reliable Transmission
 - ARQ
 - Sliding Window