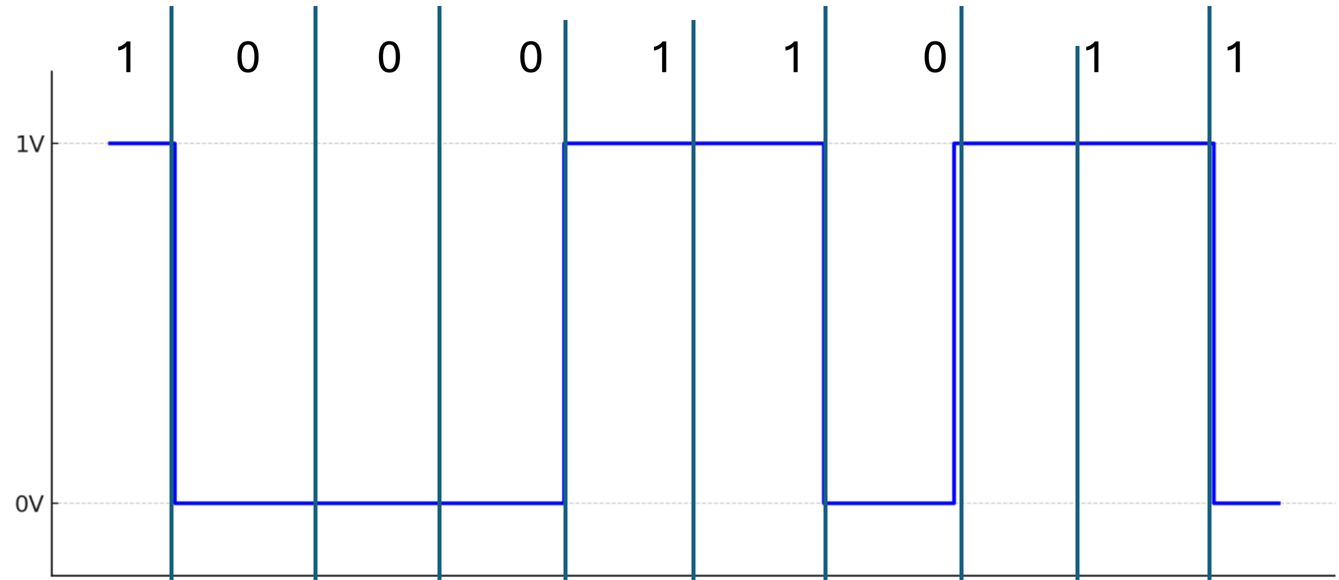
Lecture 9: Physical layer - Error detection and reliable transmission

Lecturer: Venkat Arun

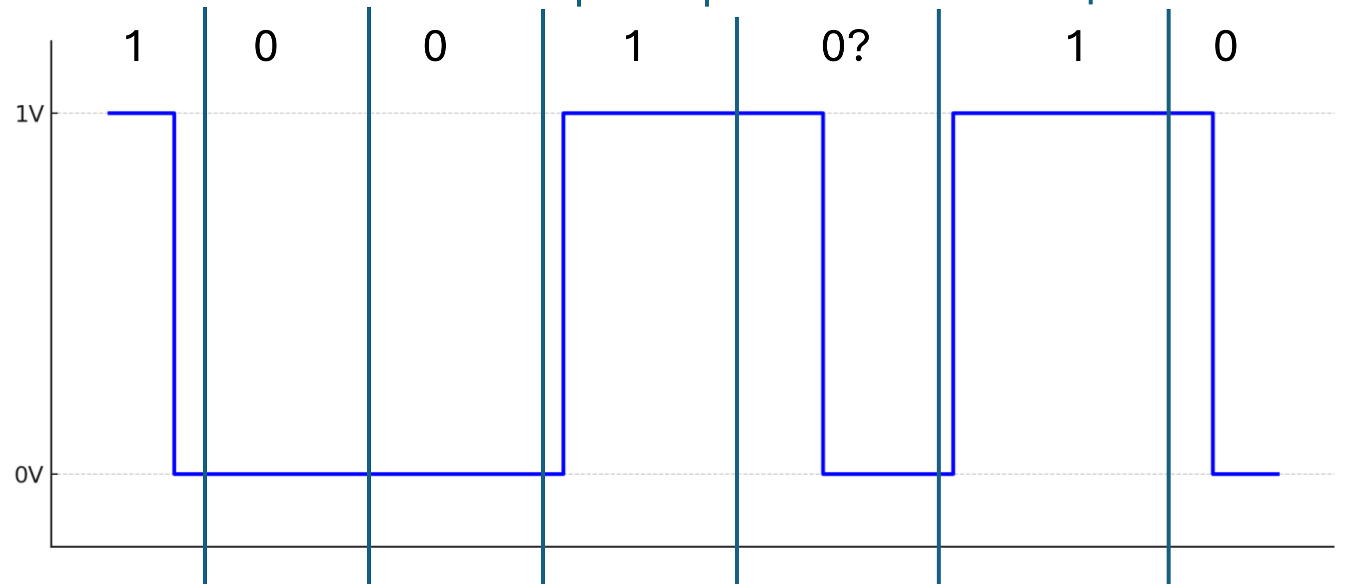
Some slides borrowed from Daehyoek Kim

Recap: bit synchronization

What the sender intended



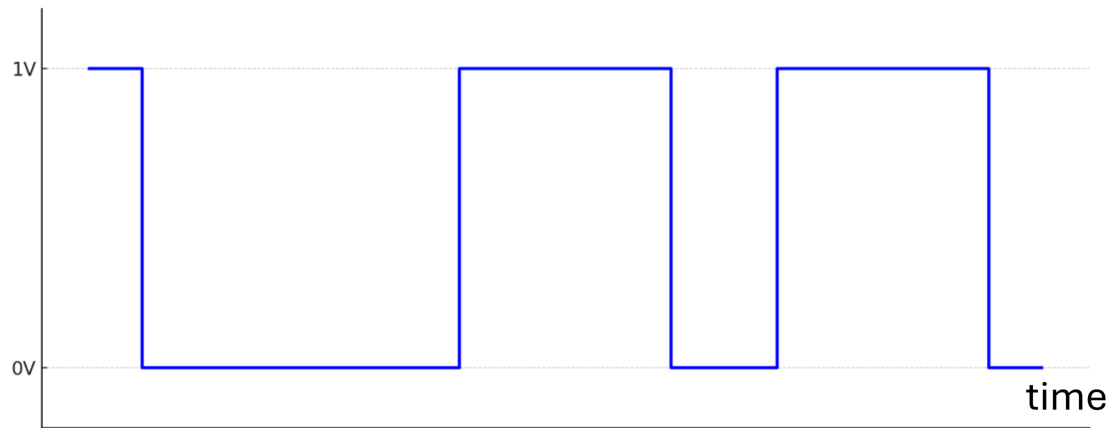
What the receiver saw because its clock was slower (effect exaggerated here for clarity)



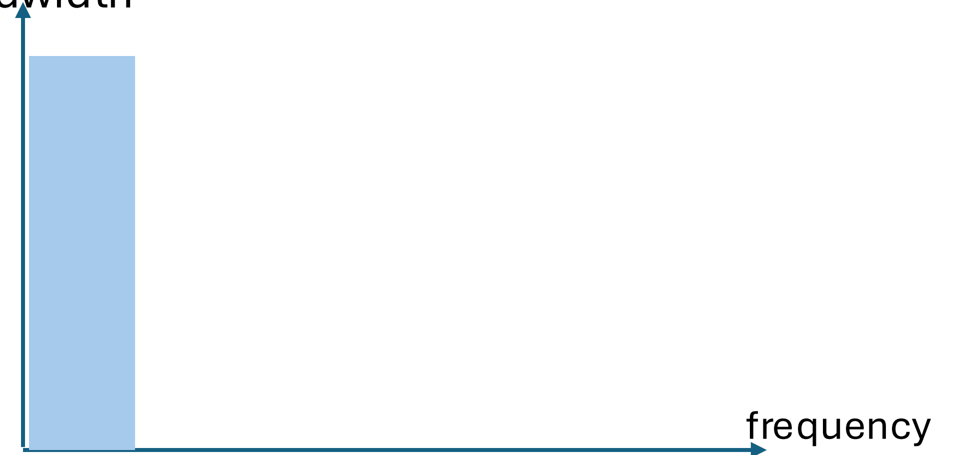
Recap: Signal bandwidth

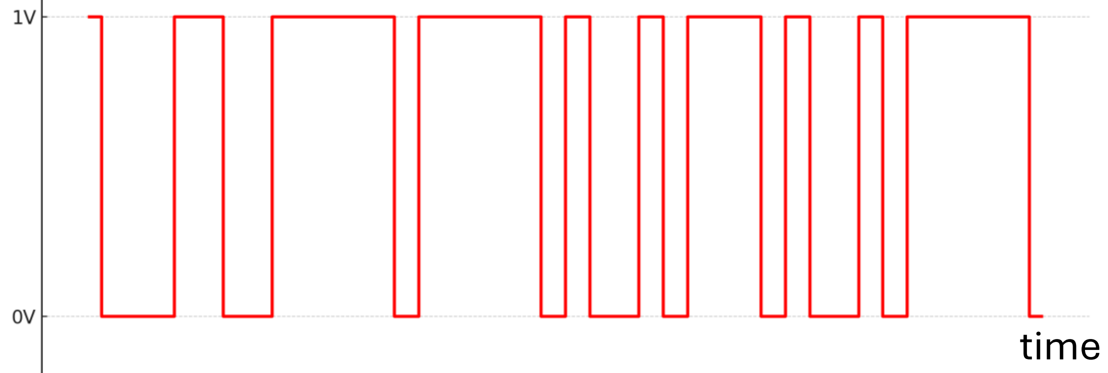
Note: we are trying to walk a fine line between giving enough background that you understand the engineering tradeoffs, without making this a signals course

Low baud rate

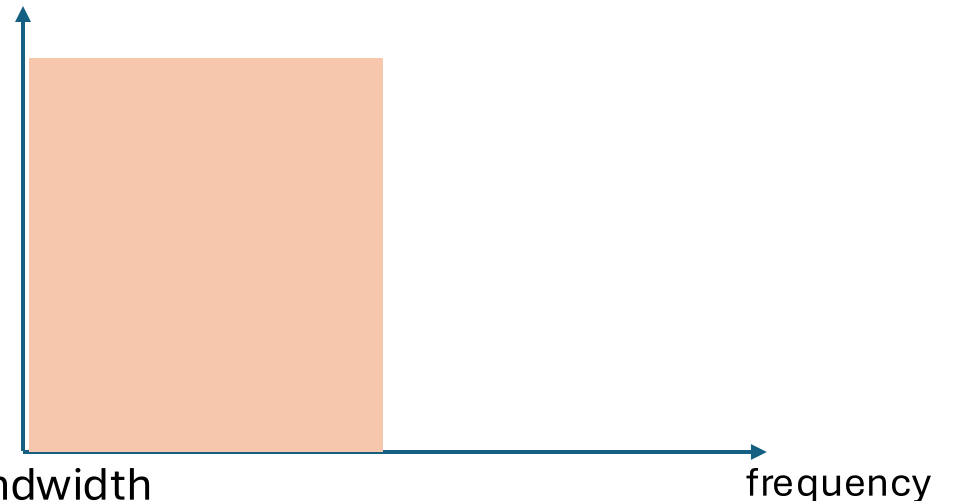


Low bandwidth





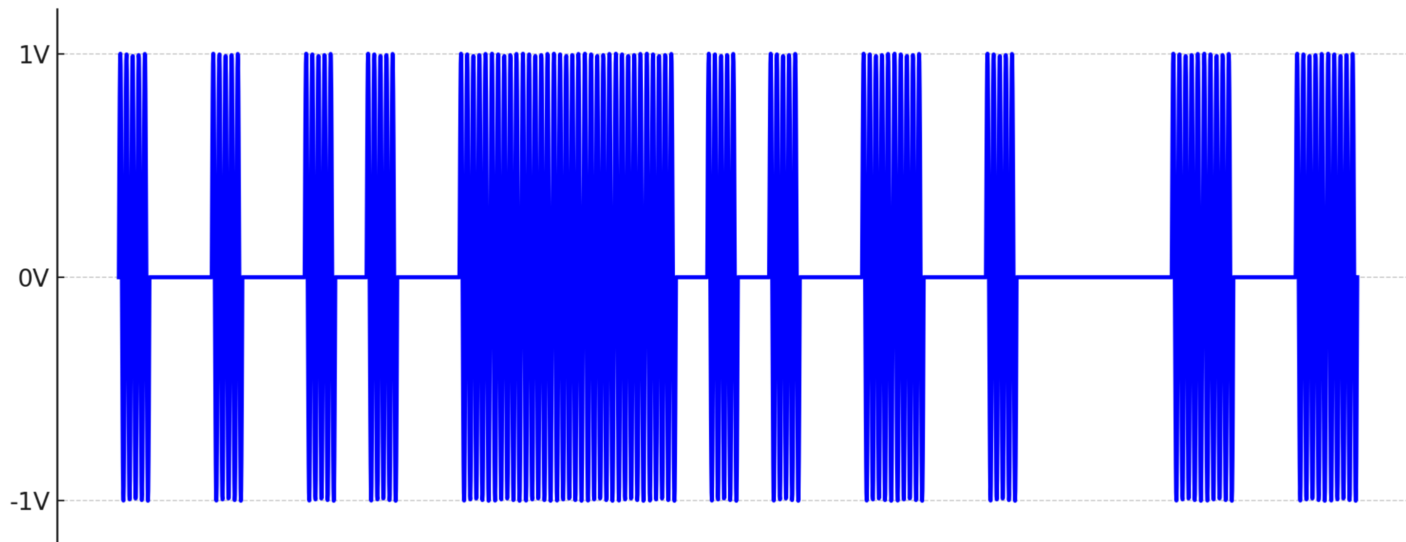
High baud rate

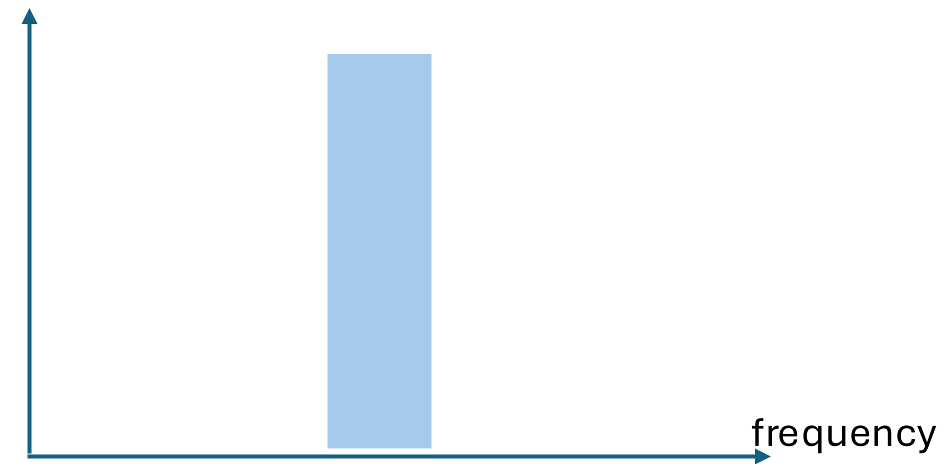


High bandwidth

Recap: Sending multiple signals on the same medium

It is possible to "shift" the signal in the frequency domain by multiplying it with a sinusoid



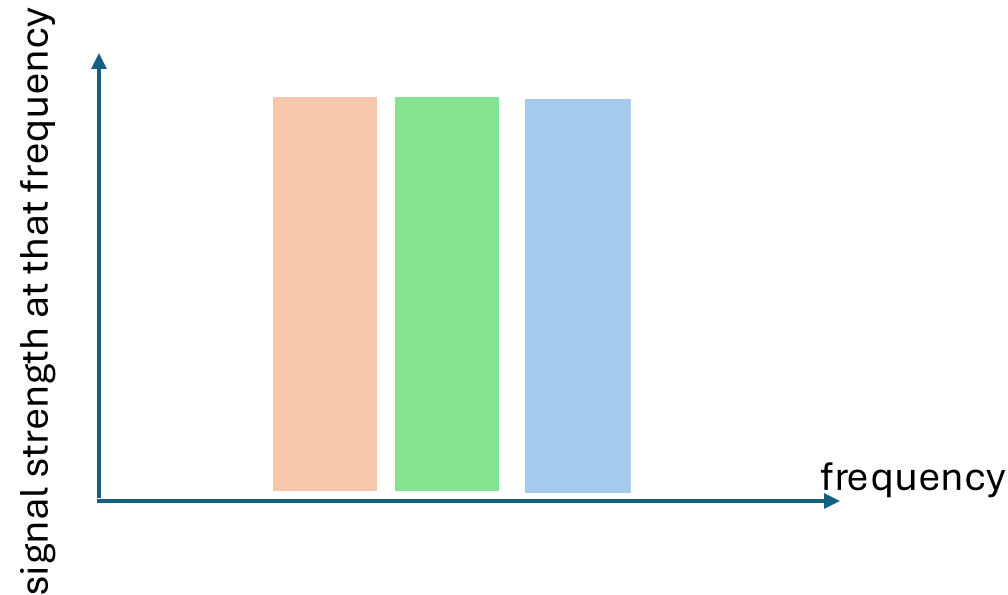


Recap: Sending multiple signals on the same medium

This means, we can send multiple independent streams on the same physical channels by shifting them by different amounts

Why would we want to do this?

1. If the electronics limit the baud rate. Usually, it is difficult to go beyond 1-4 billion flips/s. Often happens in fiber optics where the available frequency range is much broader than what the electronics can fill
2. If we want multiple transmitter to be able to independently send signals on the same medium. Often used in wireless links where each transmitter/receiver pair can pick a frequency range and use it in any way that they like

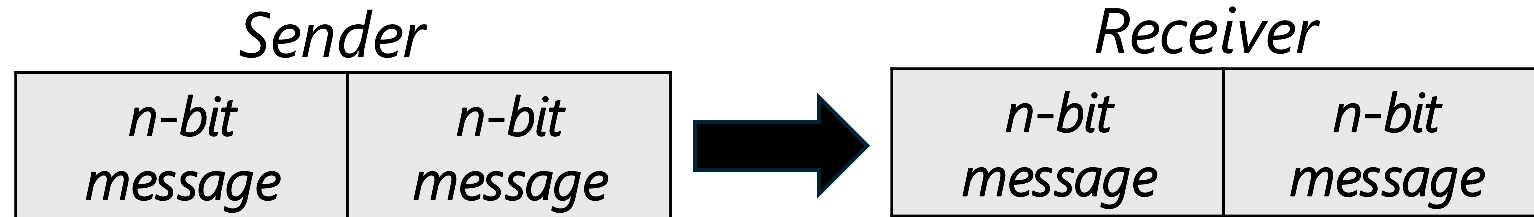


Basic idea of error detection

Adding redundant information to a frame that can be used to determine if errors have been introduced

Naïve approach: Transmitting two complete copies of data

- Identical ➔ No error
- Differ ➔ Error



Is this good enough? If not, why?

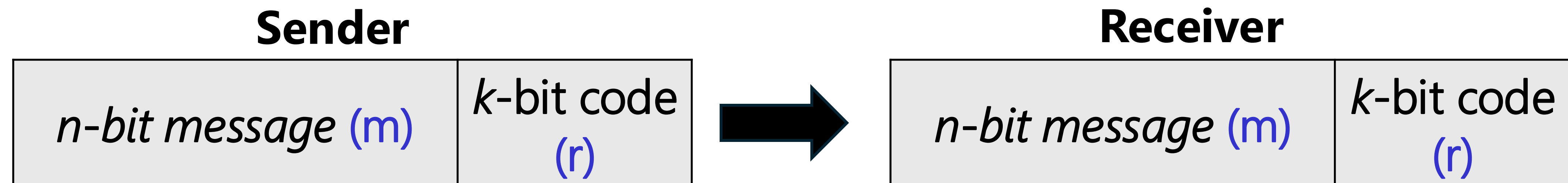
- *n* redundant bits for an *n*-bit message
- Error can go undetected – possibility to corrupt the same bit positions in the first and second copies of the message

More efficient error detection methods

k redundant bits for n bits message, where $k \ll n$

How to construct k-bit redundant information?

- Add no new information to the message
- Derived from the original message using some algorithm
- Both the sender and receiver know the algorithm



Receiver computes r using m
If they match, no error

Error detection – parity bits

- If I send some bits and they get corrupted on the way, say due to physical noise, what happens?
- We can add extra "error detection" bits to ensure that we at least find out that this has happened
- Simplest error detection mechanism is the parity check bit.

Rule: the final bit is the parity bit ensures that there are always an even number of 1s

Examples:

- 0000 → 0000 0
- 1011 → 1011 1
- 0100 → 0100 1

If one bit is flipped, we can detect it

Error detection - checksum

- How do we detect more bit errors? One method is to view the transmitted bits as a sequence of k-bit integers
- To the message of size n = m * k, add another k bits such that the sum of all the k-bit integers in the message is 0

Example:

k = 8

Message: [10, 237, 213]

(10 + 237 + 213) % 256 = 204

Now 204 + 52 = 256

Thus, we can append 52 to get the encoded message: [10, 237, 213, 52]

Error detection - checksum

How good is checksum?

- It catches all single-bit errors and most multi-bit errors.
- It cannot catch *all* 2-bit errors. E.g. if we flipped the least-significant bits of two 8-bit integer, the sum remains the same
- Better error detection mechanisms have been designed to detect specific types of errors that are common in practice
- The book describes CRC, which catch catch the following (not a part of this course)
  - All single- and double-bit errors
  - All odd-bit errors (and hence all triple-bit errors)
  - Any burst error less than k-bits long
  - Most other multi-bit errors

Reliable transmission

Ok, we've detected an error. How what?

Chapter 2.5

Reliable transmission

Reliable transmission is needed to retransmit data in case of packet corruption or loss

Reliable transmission occurs in several layers. We'll focus on link-layer reliability for now

Building blocks for reliable transmission

Acknowledgements

- A small control frame that a protocol sends back to its peer saying that it has received the earlier frame

Timeout

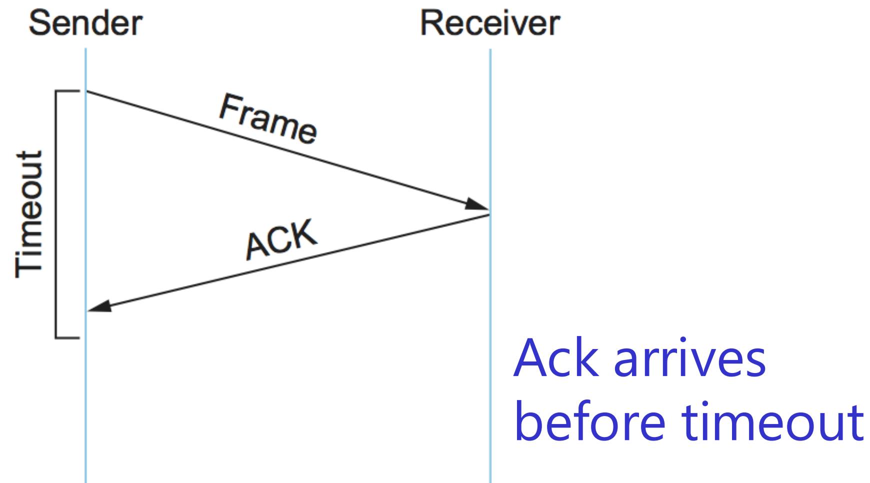
- If the sender does not receive an acknowledgment after a reasonable amount of time, then it retransmits the original frame
- The action of waiting a reasonable amount of time is called a **timeout**

Stop and wait protocol

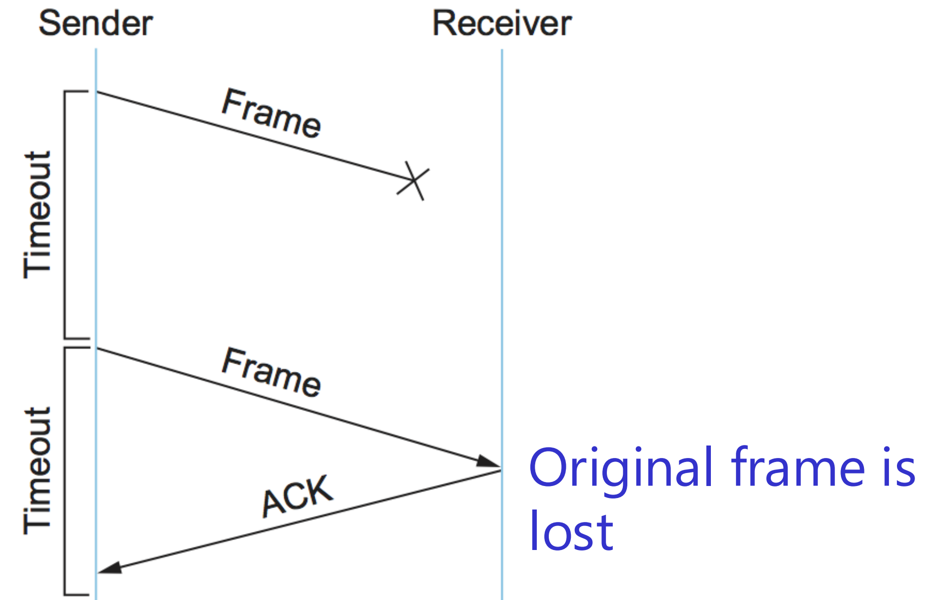
After transmitting one frame, the sender waits for an acknowledgement before transmitting the next frame

If the acknowledgement does not arrive after a certain period of time (“**timeout**”), the sender retransmits the original frame

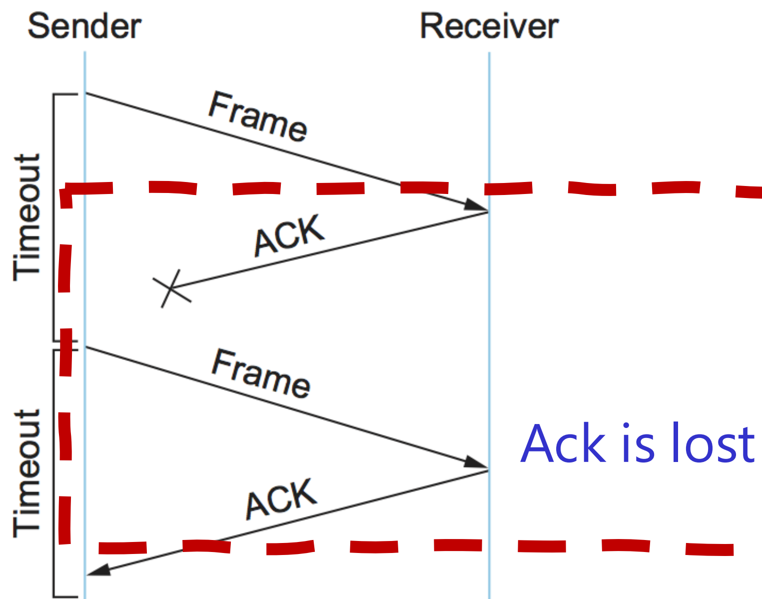
Example



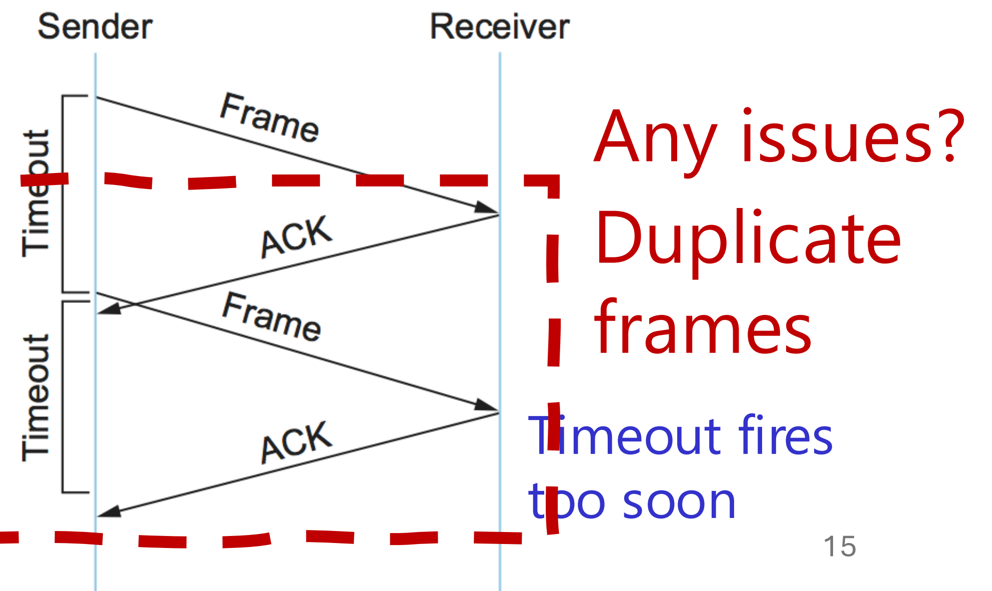
Ack arrives before timeout



Original frame is lost



Ack is lost



Timeout fires too soon

Any issues?

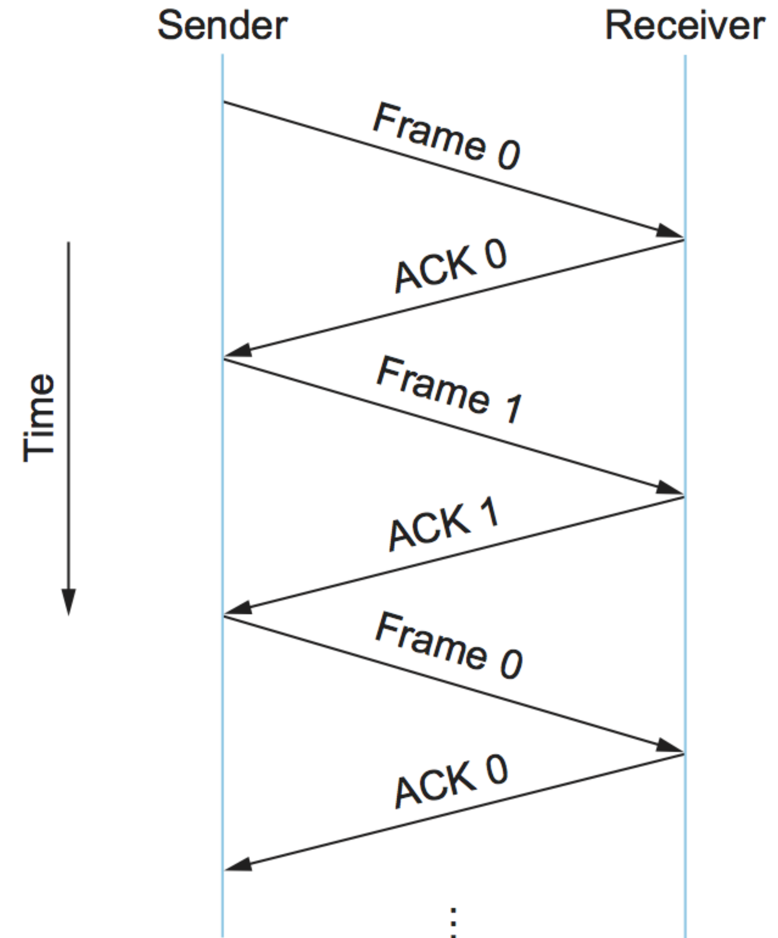
Duplicate frames

1-bit sequence number

Use 1 bit sequence number (0 or 1) in a frame header

When the sender retransmits frame 0, the receiver can determine that it is seeing a second copy of frame 0

The receiver still acknowledges it, in case the first acknowledgement was lost



Limitation of the stop and wait protocol

The sender has only one outstanding frame on the link at a time

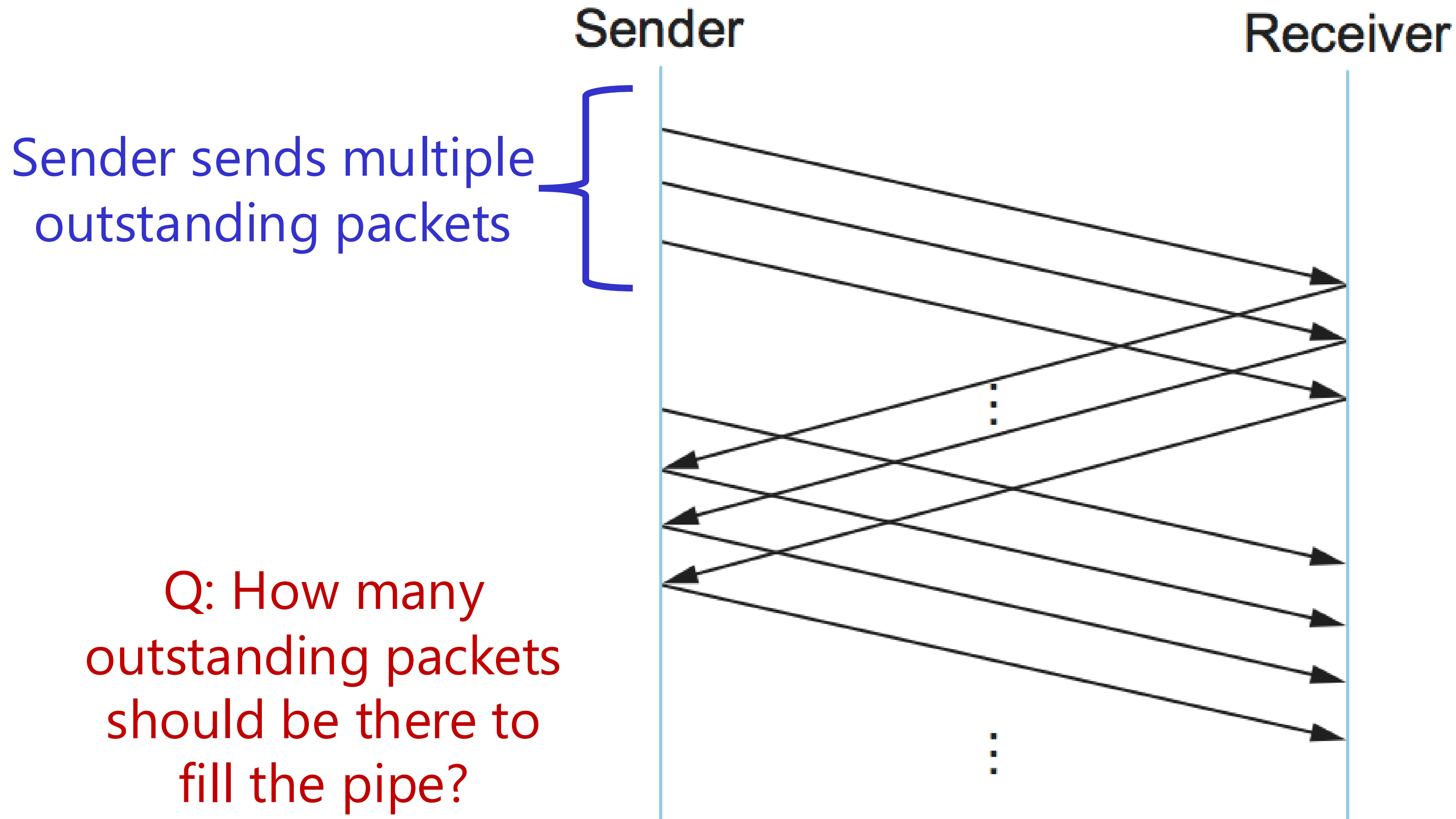
Consider a 1.5 Mbps link with a 45 ms RTT, a frame size of 1 KB

- Maximum Sending rate: Bits per frame / Time per frame
- 1024 x 8 / 0.045 / 1024= 178 Kbps
- Bandwidth x delay product: 69.1 Kb or approximately 8 KB

1/8th of the link capacity ➔ Link is underutilized!

To use the link fully, then sender should transmit up to eight frames before having to wait for an acknowledgement

Alternative: Sliding window protocol

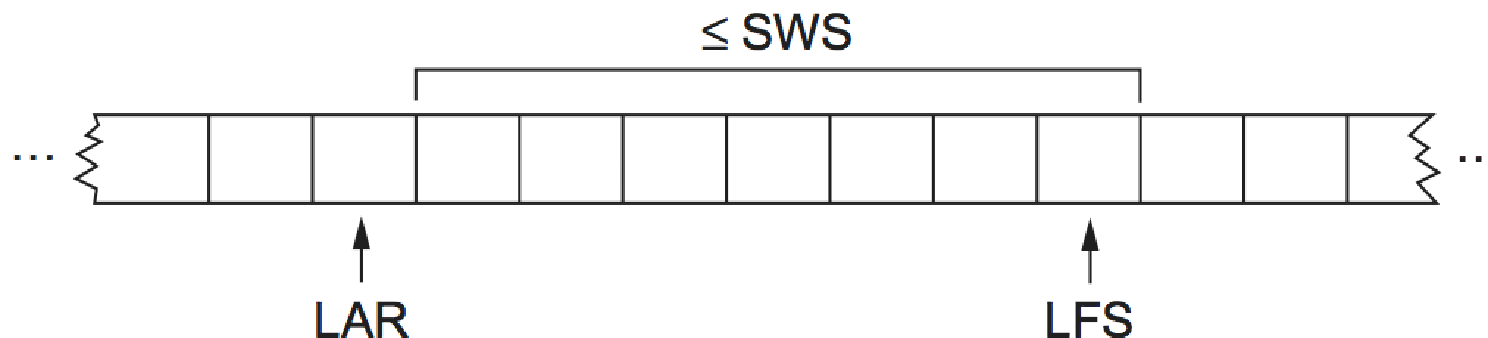


Sender-side variables

Sender assigns a sequence number (SeqNum) to each frame

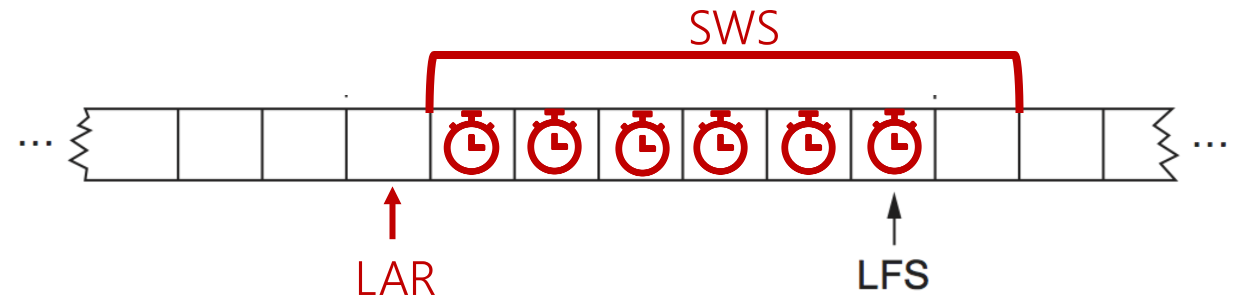
Sender maintains three variables:

- Sending Window Size (SWS)
  - Upper bound on the number of outstanding (unacknowledged) frames that the sender can transmit
- Last Acknowledgement Received (LAR)
  - Sequence number of the last acknowledgement received
- Last Frame Sent (LFS)
  - Sequence number of the last frame sent



★Sender-side invariant:
LFS – LAR ≤ SWS

Sender-side actions



When an acknowledgement arrives

- The sender moves LAR to right, thereby allowing the sender to transmit another frame

The sender associates a timer with each frame it transmits

- It retransmits the frame if the timer expires before the ACK is received

The sender must buffer up to SWS frames. Why?

- To retransmit them until they are acknowledged

Receiver-side variables

Receiving Window Size (RWS)

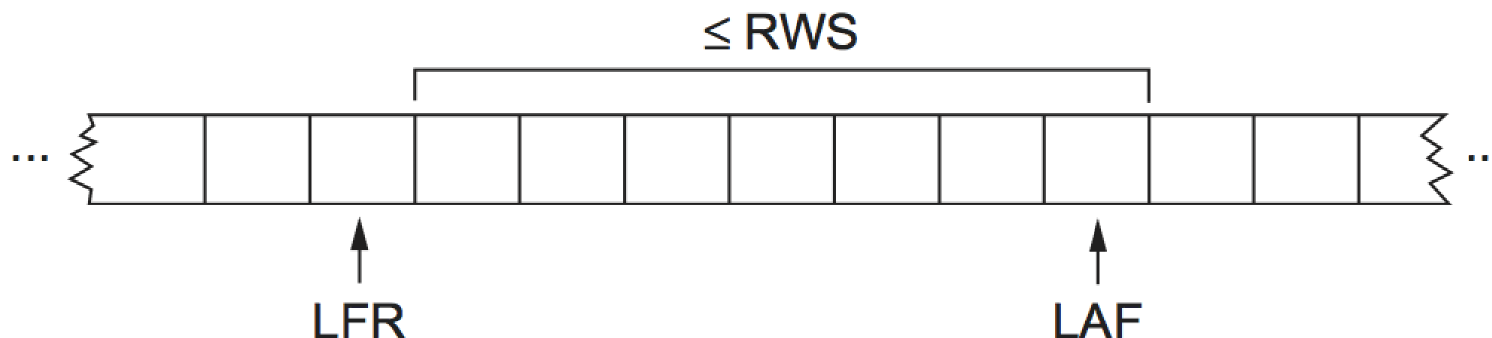
- Upper bound on the number of out-of-order frames that the receiver is willing to accept

Largest Acceptable Frame (LAF)

- Sequence number of the largest acceptable frame

Last Frame Received (LFR)

- Sequence number of the last frame received



★Receiver-side invariant:
LAF – LFR ≤ RWS

Receiver-side actions

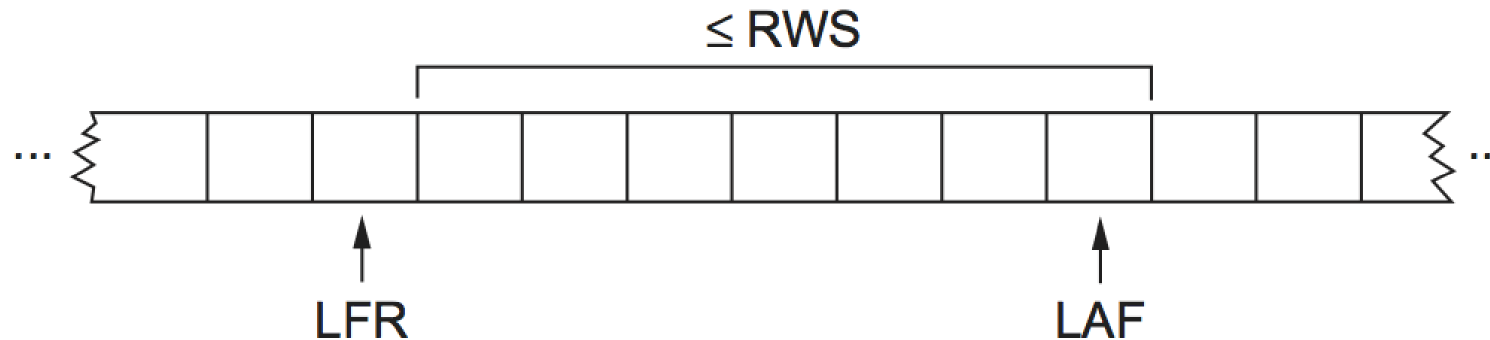
When a frame with sequence number SeqNum arrives:

If SeqNum ≤ LFR or SeqNum > LAF

- Discard it (the frame is outside the receiver window)

If LFR < SeqNum ≤ LAF

- Accept it
- Now the receiver needs to decide whether or not to send an ACK



When to send an ACK?

SeqNumToAck: the largest sequence number not yet acknowledged, such that all frames with sequence number less than or equal to SeqNumToAck have been received

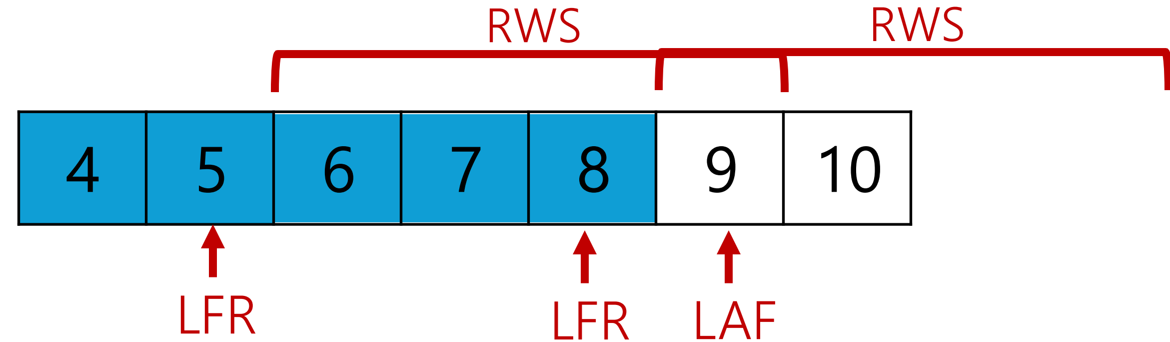
The receiver acknowledges the receipt of SeqNumToAck even if high-numbered packets have been received

- It is called a cumulative ACK

The receiver then sets

- LFR = SeqNumToAck
- LAF = LFR + RWS

Example scenario



Suppose LFR = 5 and RWS = 4 ➔LAF = 9

- i.e., the last ACK that the receiver sent was for SeqNo 5

If frames 7 and 8 arrive, they will be buffered because they are within the receiver window

But no ACK will be sent since frame 6 is yet to arrive

- Frames 7 and 8 are out of order

Now, frame 6 arrives (e.g., lost first time and retransmitted)

The receiver acknowledges frame 8 and sets LFR to 8 and LAF to 12

Issues with Sliding window protocol

When timeout occurs (i.e., packet loss), the amount of data in transit decreases

- The sender is unable to advance its window
- The longer it takes to notice that a packet loss has occurred, the more severe the problem becomes

How to improve this: Giving more (early) information to the sender!

- Negative Acknowledgement
- Additional Acknowledgement
- Selective Acknowledgement

More informative ACKs

Negative Acknowledgement (NAK)

- Receiver sends NAK for frame 6 when frame 7 arrive (in the previous example)

Additional Acknowledgement

- Receiver sends additional ACK for frame 5 when frame 7 arrives
- Sender uses duplicate ACK (DupACK) as a clue for frame loss

Selective Acknowledgement

- Receiver will acknowledge frames 7 and 8
- Sender knows frame 6 is lost
- Sender can keep the pipe full (additional complexity)

Sequence numbers are finite

Sequence numbers are integers with a finite number of bits

We need to be careful when they “roll over” to zero. While important, the exact mechanism used is boring, so we will not discuss it in this course