CA169: Week 2

Encoding, Transmission Errors and Protocols

Recap from week 1

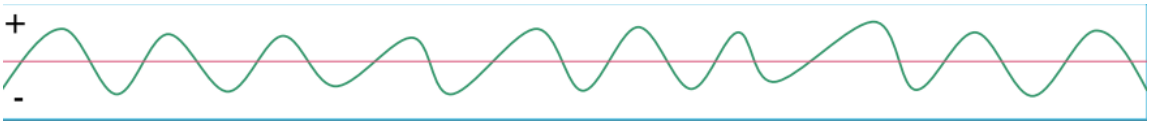
- Last week we looked at the physical ways we can transmit data
    - Unshielded Twisted pair (Phone & Ethernet)
    - Coaxial (Aerial TV, fibre to home)
    - Fibre optics (Fibre internet)
    - Radio
        - Bluetooth
        - Wifi
        - NFC

Transmission & Encoding

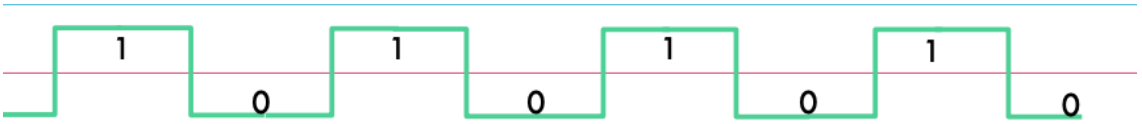
- How do we get 1's and 0's from electrical signals?
    - Encoding
- How do we deal with errors from these signals ?
    - Checksums

Analog v Digital

- Physical cables deal with voltage
- Digital systems prefer binary (1's and 0's)
- Our goal is to turn a signal from a line (analog) into a signal a computer can understand (digital)
- Essentially, turn this:



- Into this:



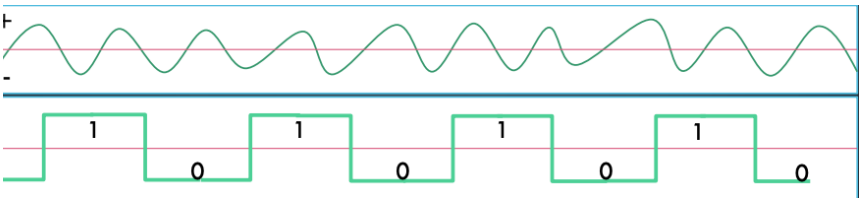
- This problem is harder than it seems
- What about a weak signal

- Or a signal with a lot of interference?

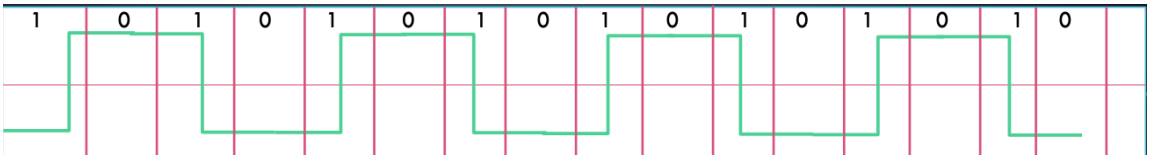
- We use an encoding algorithm
- There is no one size fits all solution
    - But some are better than others
- There are a number of factors to consider when choosing one
    - Physical medium (ethernet, wifi etc…)
    - Distance
    - voltage

Algorithm 1: NRZ-L & NRZ - I

- Non Return Zero Level (NRZ-L) is an encoding scheme where high voltage becomes a 1 and low voltage becomes 0
- This is used for short connections (e.g. to a printer)



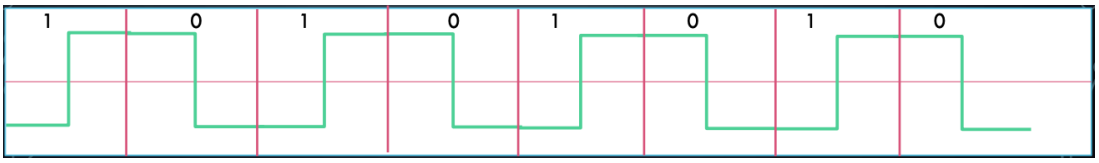
- Non Return Zero Inverted (NRZ-I) doesn't work off high-low voltage
- It looks at the transitions between positive and negative voltage
- If there is a transition, we use 1
- If there is no transition, we use 0



- Which to choose?
  - NRZ-I can deal with noise better
  - It is easier to check if voltage went from – to +
  - Than to say "if voltage >5"
  - Problem
  - What if I wanted to send the data 111111111

Clock Information

- So we clearly have to introduce a time component (clock)
- Manchester encoding does this
- We cut our signal into segments (based on time)
- We call this Biphase
- If we go from + to – we use a 0
- IF we go from – to + we use a 1



Why Manchester Encoding

- More difficult to operate
- More resistant to noise
- Requires twice as much bandwidth to send the same data as other methods
- But much more reliable

\4B/5B Coding

- 50% more efficient than Manchester (we use it in 100Mb internet)
- Take four bits from signal
- Encode them as 5 bits (we call this a cell)

Data		4B5B code
(Hex)	(Binary)	
0	0000	11110
1	0001	01001
2	0010	10100
3	0011	10101
4	0100	01010
5	0101	01011
6	0110	01110
7	0111	01111

- By mapping 4 bits onto 5 bits we have better control over the data
- This makes it more resistant to noise
- If there are only 4 bits, there are 16 possible frames (2^4)
- In short, we have a much more reliable signal

Modems

- This process of converting from analog to digital (and back again) is called modulation
- In the real world we use MODEMs to do this
- MODEM is short for:
    - MOdulator
    - DEModulator

Transmission Errors

- We know how to convert our analog signal into a digital one
- But how do we know we converted it right?
- A number of factors can influence our transmission
- We call these noise
- Noise can come in many forms
    - Electrical interference
    - Walls
    - Long distance

How to Tell if Data is Incorrect

- How can we tell if our data is incorrect?
- By adding extra check bits onto the end of our transmission
- And doing a bit of maths

Cyclic Redundancy Code (CRC)

- Error detecting codes
- Not error correcting codes
- Idea: represent binary as a polynomial

F = 110001

F(x) =$(x^5 * 1) + (x^4 * 1) + (x^3 * 0) + (x^2 * 0) + (x^1 * 0) + (x 0 * 1)$

F(x) = $x\ 5^+\ x^4 + 0 + 0 + 0 + x^0$

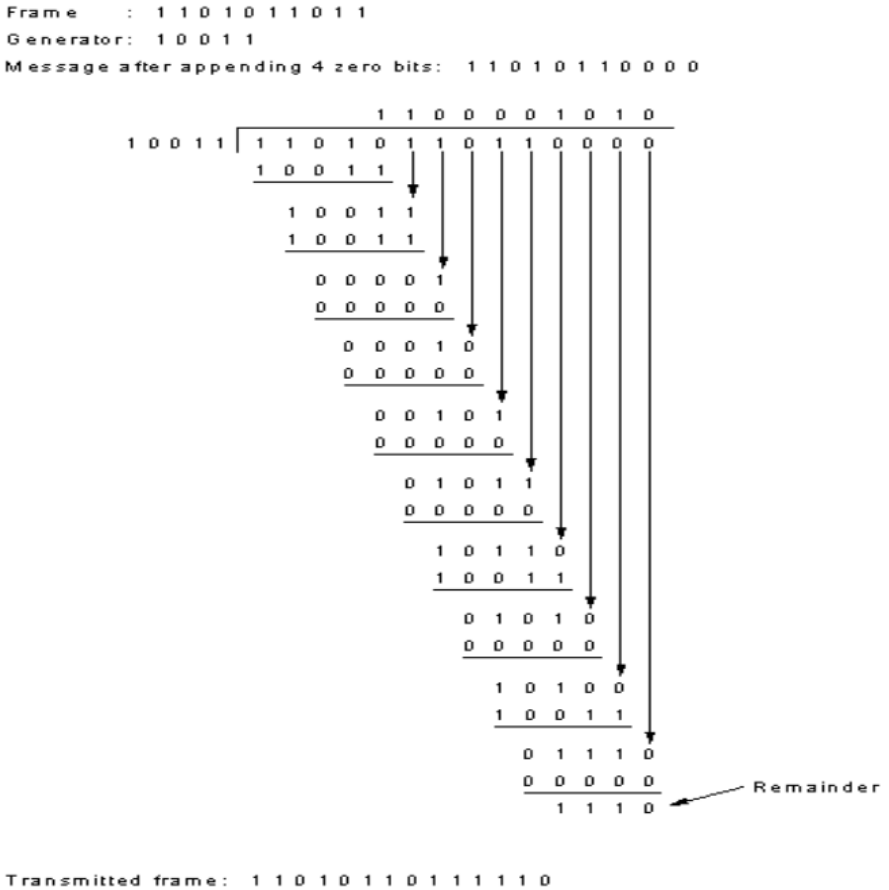
F(x) = $x\ 5^+\ x^4 + x 0$

F(x) = $x\ 5^+\ x^4 + x^0$

CRC Algorithm

- Compare using Modulo 2 arithmetic (x%2)
    - This is the same as the XOR operation ⊕
- Sender & receiver agree on G(x) generator polynomial.
- Append R 0 bits to M(x), the message, where R is the degree of G(x), this yields
- $x^R * M(x)$
- Divide G(x) into $x^R * M(x)$.
- Add reminder to $x^R * M(x)$, result T(x)
- Example

    M(x) = 1101 0110 11
    $x^R * M(x)$ = 1101 0110 1100 00
    G(x) = 10011 or x 4 + x1 + x0
    Remainder should be 1110
    T(x) transmitted message should be
    1101 0110 11 1110
    This will be evenly divisible by G(x) !



- Standard CRCS
    - CRC-12
        - $x^{12} + x^{11} + x^3 + x^2 + x + 1$
    - CRC-16
        - $x^{16} + x^{15} + x^2 + 1$
    - CRC-CCITT
        - $x^{16} + x^{12} + x^5 + 1$
    - CRC-32
        - $x^{32} + x^{26} + x^{23} + x^{22} + x^{16} + x^{12} + x^{11} + x^{10} + x^8 + x^7 + x^5 + x^4 + x^2 + x + 1$

How Data is Transmitted

- I want to send a document to another device
- This document is converted into 1's and 0's
- It is sent to the other device
- The other device then converts the 1's and 0's back into the documen

IRL

- In reality, this is much harder
- The document isn't converted into one long stream of 1's and 0's
- It is broken down into packets/frames of information
- These packets are then sent to the other device
- Each packet contains only one part of the document
- Why do we do this?
    - If we sent the entire document in one go
    - We can't handle sending multiple things at once (multiplexing)
    - Your computer sends a lot of information at once all the time
    - See netstat
- What's the problem?
    - How do we tell when one packet/frame ends and another begins?
    - We call this the framing problem

Possible Solutions to the Framing Problem

- Time delay:
    - Every 1 second send a new packet
    - Data may be lost if signal is interrupted, or a delay in transmission
- What if we include the length of the packet at the end
    - Add on another bit to the CRC
    - If the count is corrupted, we would never know how long our data is

Actual Solution

- At the start and end of our packet we add in special markers
    - We call this character stuffing
- DLE STX (meaning start of packet)
    - Lets pretend this is: 1001 in binary
- DLE ETX (end of packet)
    - Lets pretend this is 0110 in binary

Examples

- If I want to send 11010101 to another computer
    - We add on our DLE STX to the start
        - 1001 11010101
    - Then we add on our DLE ETX to the end
        - 1001 11010101 0110
- The computer we are sending the data to knows that a start packet is 1001 and an end packet is 0110
- When it gets this data it removes them
- This leaves us with our original data

Double Stuffing Example

- If I want to send the data 110101011001010
- Find bits that match STX or ETX 110101011001010
- Duplicate them 1101010110011001010
- Now add on the STX and ETX to the packet
- 100111010101100110010100110
- If we see a duplicated pattern, only remove one of them!

What Does Protocol Mean

- We now know how to change data from signals into nice 1's and 0's
- We can also be sure that the data we sent is correct
- But there are a number of questions we still have to answer
- What do we do if we do get an error?
- How do we signal that we have finished sending data?

Need for Protocols

- Protocols dictate how we communicate
- They control
    - Data format
    - Data content Timing/synchronisation of communication
    - How the communication is managed

Why Use Protocols

- In networking a protocol is like a language
- It provides a common way of devices to understand each other
- There is no magic
- They are designed by programmers
- They are only useful if they are widely adopted
- They are refined through trial and error

Making a Protocol

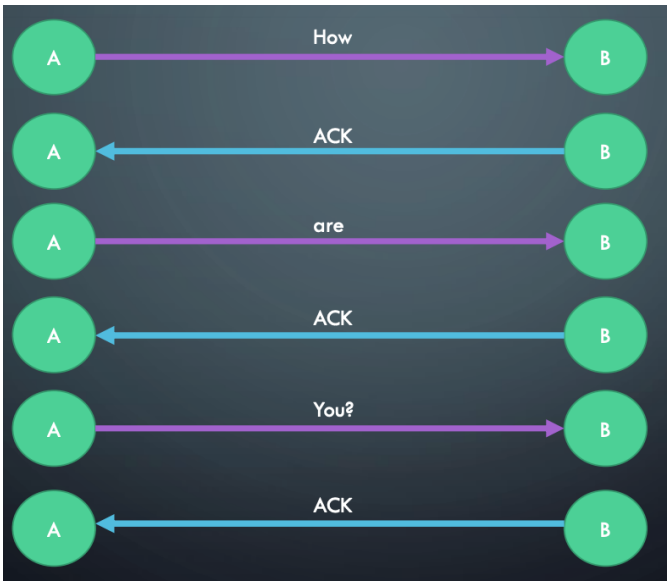
- Lets first make some assumptions:
    - Transmission will be simplex (only one computer can speak at a time)
    - Computers are always ready to receive data
    - Processing time is negligible and we have unlimited space
    - Our physical medium is error free!

Utopia Protocol

- Make all assumptions, a, b, c, d.
- For this protocol the control header and checksum are unnecessary.
- The transmitting host simply takes packets from host A (which always has one ready) and pumps them as fast as it can onto the physical link.
- The receiver accepts the frames and passes them straight to host B

Stop & Wait

- The UTOPIA protocol is nice but…
- Processing time is not negligible, and we do not have infinite space
- So we drop assumption c
- In practice the receiving computer needs time to process the packets
- Only has a fixed amount of space to store incoming packets for processing
- The receiver needs a way to prevent it from getting more data then it can handle
- When the receiver processes a packet, it sends one back requesting more data (we'll call this a control frame)
- Only after receiving this control frame will the sender fetch and transmit the next packet
- E.g Let's send the data "How are you"?



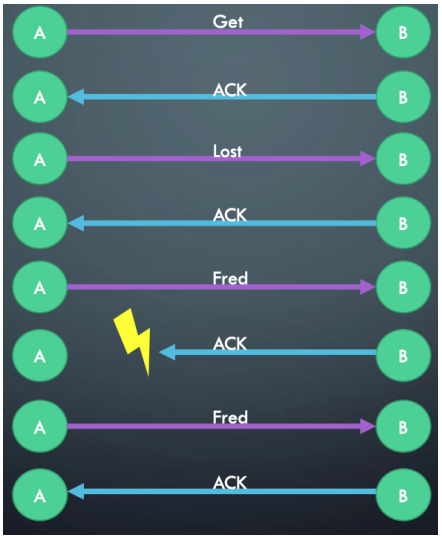
- Issues?
    - Can anyone see any problem with this method?
    - What about a failed transmission?

Positive Acknowledgement With Retransmission (PAR)

- Noisy channel simplex protocol: drop some assumptions
    - c) [processing time, buffers pace] and
    - d) [error free link]
- With error prone physical link, frames may be either damaged or lost completely.
- However,
    - Damaged frames can be detected by the checksum.
    - Lost frames will not be acknowledged.
    - Eventually the sender will tire of waiting for an acknowledgement, timeout, and retransmit the frame.
- Same as Stop-and-Wait, except that
    - damaged frames are not acknowledged,
    - causing a timeout and subsequent retransmission

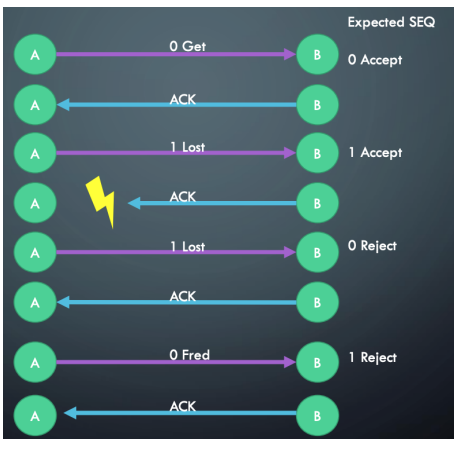
Examining Another Protocol

- Suppose the message “Get Lost Fred” is being sent from A to B, one word per packet



- What happened?
- B receives “Get lost Fred Fred” and the protocol has failed !
- Basically what has happened is that the receiver has accepted a duplicate frame.
- The solution is to use SEQ, a sequence number in the control header to differentiate between frames and allow duplicates to be discarded.
- As a frame must be acknowledged before the next one is sent, a one-bit sequence number [0,1] is sufficient.
- The receiver will expect alternatively numbered frames (0 1 0 1 0 1 ... etc.).
- Any frame with the wrong sequence number is rejected as a duplicate
    - (but still acknowledged).

- What about time outs



Timesouts

- The message “Get Lost” is received and the protocol has failed

Solution

- Include in the ACK field of the acknowledgement control frame the SEQ number of the last frame received without error.
- Then if this number (0 or 1) differs from the transmitted frame, the sender transmits it again.
- The resulting PAR simplex protocol will now work in the face of any combination of
    - garbled frames,
    - lost frames and
    - premature timeouts.

Summary

- After transmitting a frame and starting the timer, Host A waits for a response.
- There are three possibilities:
    - 1. an acknowledgement frame arrives undamaged
    - 2. a damaged acknowledgement arrives, or
    - 3. the timer goes off.
- If a valid ACK comes in, A fetches the next packet and puts it in the buffer
    - overwriting the previous packet, and advancing the sequence number.
- If a damaged frame arrives or no frame at all arrives,
    - neither a buffer nor the sequence number are changed, so that a duplicate can be sent

Bi-Directional PAR

- The control header of all frames contains the three fields KIND, SEQ and ACK.
- Sending data packets/acknowledgements in both directions is no problem –
    - by looking at the KIND bit in the header, the receiver knows which it is dealing with.
- However, this would be inefficient.
- Consider an Host B which is about to acknowledge a data frame received from
- Host A, and also about to send off a data frame to A.
- Instead of sending two frames, the acknowledgement can hitch a lift on the data frame, using the ACK field in the header.
- This is called piggybacking.
- As we are still making assumption b)
- [Transmitting and receiving hosts always ready to transmit & receive data]
- All acknowledgements can be piggybacked. Thus, data packets are bounced back and forth between A and B.
- Notes:
    - For the protocols considered so far, only one frame is in the wire at any one time.
    - The sender needs to keep a copy of each frame in a buffer for possible retransmission until the frame has been successfully acknowledged.
- Assumption b)
- [Transmitting and receiving hosts always ready to transmit & receive data]
- Easily dealt with.
    - If there is no outgoing data frame, the host will wait a short while to see if one comes along to provide a piggyback.
    - If not, a separate acknowledgement frame will be sent.
    - It must not wait too long to avoid unnecessary duplicates being sent due to the sender timing-out.
- Up until now, lost and damaged frames have been dealt with in the same way.
- No ACK is sent, leading to timeout and retransmission.
- The timeout period is usually set quite long in order to avoid complications caused by premature timeout.
- This is inefficient, as while the timeout is expiring, the link is not being used.
- A partial solution is NAK, a negative acknowledgement.
- This is sent immediately a damaged frame is received and elicits immediate retransmission.
- The NAK may also be piggybacked.
- If the NAK is damaged, no harm is done as the sender will eventually timeout and retransmit as before.
- A damaged frame is Nak'd only once.

Pipelining

- When propagation delay is not negligible, these previous methods are wasteful of bandwidth.
- The solution is to 'fill up the pipe'.
- However, doing this entails sending off frames before ACKs for previous frames have arrived.

Sliding Window Protocol

- Each outbound frame is given a sequence number in the range of 2^n-1 using an n-bit field, e.g. if n=1, then range is 0.....1 as in ABP or PAR protocols.
- Both sender and receiver keep windows informing them of which frames can be validly sent and which validly received.

Rules

- Sender :- The upper edge of sender is advanced when a frame is sent (up to max. window size).
- The lower edge advanced when ACK received for lowest numbered frame in the window.
- Receiver :- Both edges are advanced when the lowest numbered frame in window is correctly received and ACK sent.

Notes

- Buffering requirements at both sender and receiver depend on the size of the sending and receiving windows respectively.
- Each transmitted frame has its own separate timeout clock.
- In these protocols, an acknowledgement for frame N is accepted as acknowledging all transmitted frames numbered up to N (counting circularly).
- Thus, if ACK(0) and ACK(1) were both destroyed, but ACK(2) now arrives, it implicitly acknowledges 0 and 1 also.

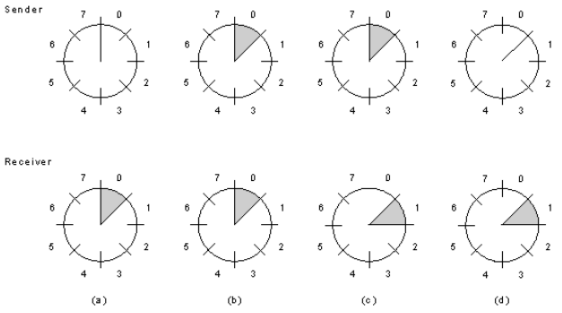
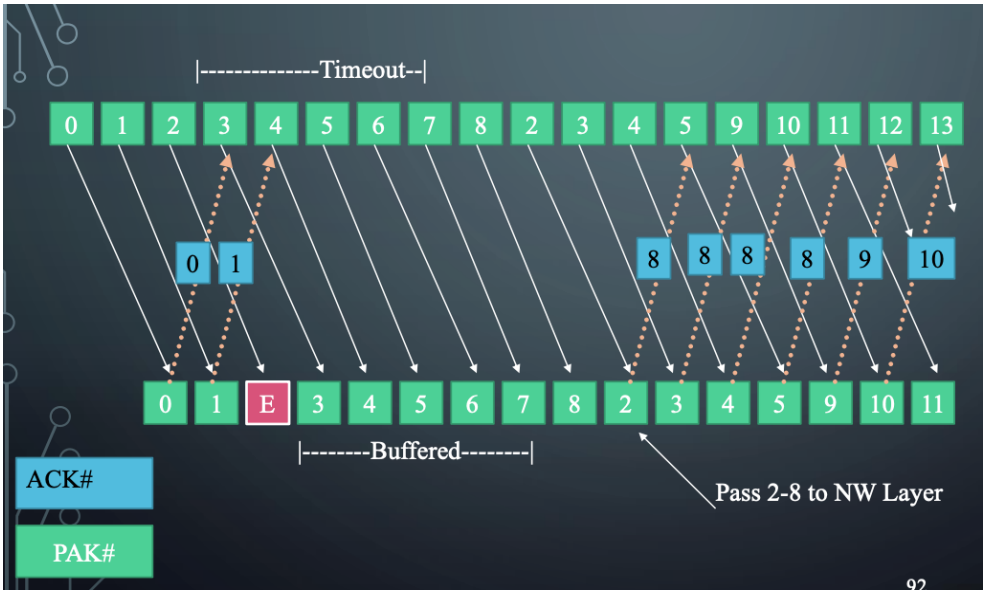


Fig. 3-12. A sliding window of size 1, with a 3-bit sequence number. (a) Initially. (b) After the first frame has been sent. (c) After the first frame has been received. (d) After the first acknowledgement has been received.

Session With Recovery



Stopping the Problem

- A Data-Link cannot be stopped.
- Consider a session termination.
- Neither terminal knows that other has sent last packet, the last packet must be ACK'd.
- In practice the data-link is dropped after the link is sensed as being dead for a prolonged period