Computer Network Design

Yalda Edalat – Spring 23

Checksum

- Sum up data in N-bit words
- Usually placed at the end of the message, as the complement of the sum function
- Errors may be detected by summing the entire received codeword
  - If the result comes out to be zero, no error has been detected

- Widely used in, e.g. TCP/IP/UDP

Internet Checksum

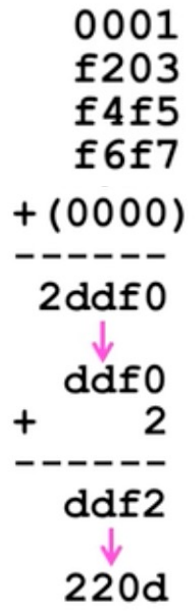
- Sum is defined in 1s complement arithmetic (must add back carries)

- *"The checksum field is the 16 bit one's complement of the one's complement sum of all 16 bit words…"* – RFC 791

Internet Checksum Example

- At sender:

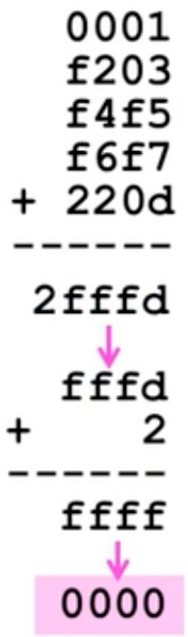
1. Arrange data in 16-bit words
2. Put zero in checksum position, add
3. Add any carryover back to get 16 bits
4. Negate (complement) to get sum



Internet Checksum Example

- At receiver:

1. Arrange data in 16-bit words
2. Checksum will be non zero, add
3. Add any carryover back to get 16 bits
4. Negate the result
   - If 0 -> correct
   - Non zero -> error



CRC (Cyclic Redundancy Check)

- Given n data bits, generate k check bits such that the n+k bits are evenly divisible by a generator C

- Example by numbers:
  - N = 302, k = 1 digit, C = 3

- Protection depends on the generator (C)
  - Standard CRC-32 is 10000010 01100000 10001110 110110111

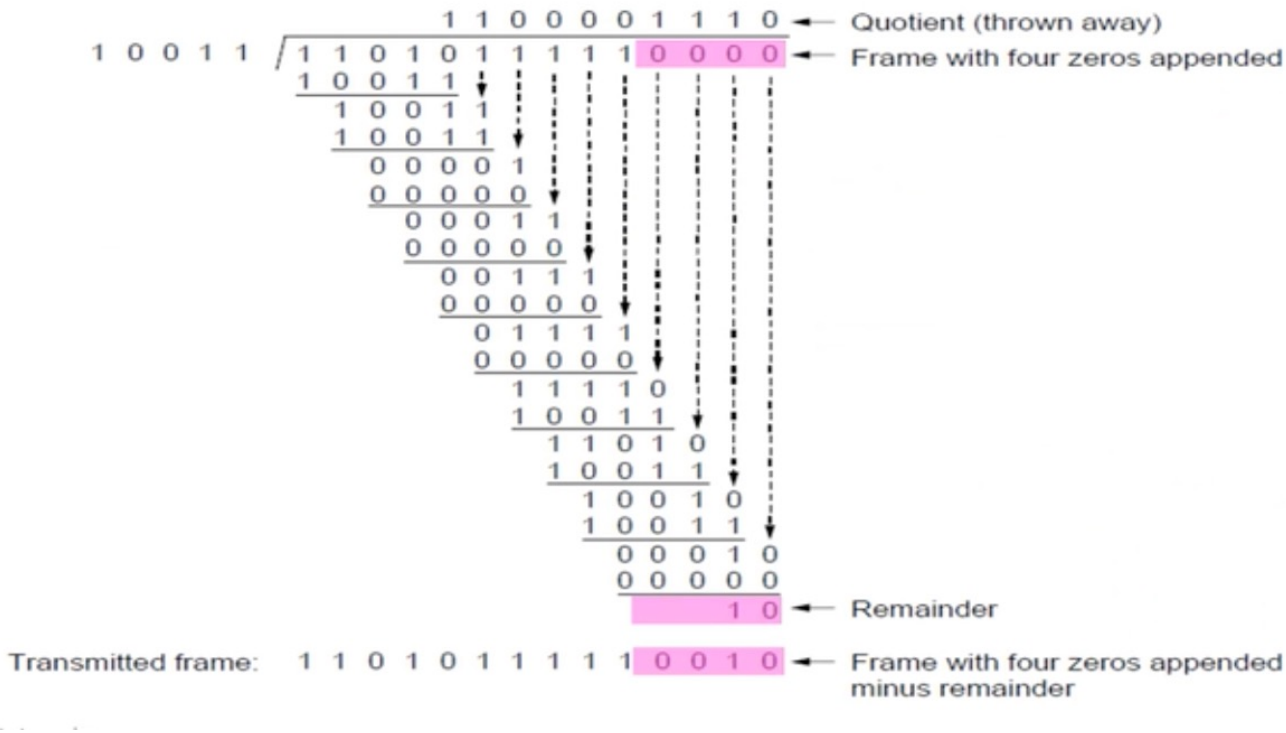
CRC (2)

- At sender:
  1. Extend the n data bits with k zeros
  2. Divide by the generator value C
  3. Keep remainder, ignore quotient
  4. Adjust k check bits by remainder

- At receiver:
  1. Divide and check for zero remainder

CRC Example

Data: 1101011111
C: 10011
K: 4



Error Detection in Practice

- CRCs are widely used on links
  - Ethernet, 802.11, ADSL, cable,…
- Checksum used in Internet
  - IP, TCP, UDP,…. But it is weak
- Parity
  - Is little used!

Error Detecting Codes

- Three different error detecting codes:
  1. Parity
  2. Checksums
  3. Cyclic Redundancy Checks (CRCs)

- Detection let us fix the error, for example, by retransmission (later)

CRC (Cyclic Redundancy Check)

- Given n data bits, generate k check bits such that the n+k bits are evenly divisible by a generator C

- Example by numbers:
  - N = 302, k = 1 digit, C = 3

- Protection depends on the generator (C)
  - Standard CRC-32 is 10000010 01100000 10001110 110110111

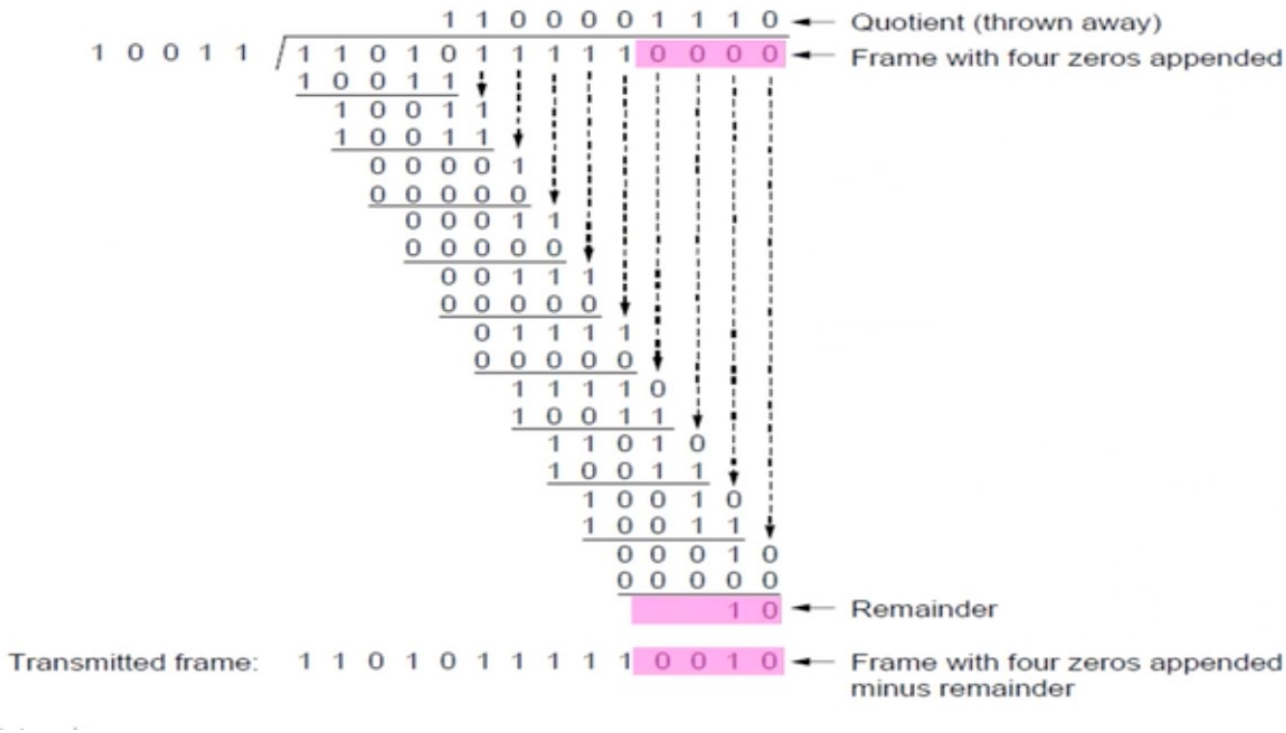
CRC (2)

- At sender:
  1. Extend the n data bits with k zeros
  2. Divide by the generator value C
  3. Keep remainder, ignore quotient
  4. Adjust k check bits by remainder

- At receiver:
  1. Divide and check for zero remainder

CRC Example

Data: 1101011111
C: 10011
K: 4



Error Detection in Practice

- CRCs are widely used on links
  - Ethernet, 802.11, ADSL, cable,…
- Checksum used in Internet
  - IP, TCP, UDP,…. But it is weak
- Parity
  - Is little used!

Why Error Correction is Hard?

- If we have reliable check bits, we could use them to narrow down the position of error
  - Then the correction would be easy
- Problem: error could be in check bits as well as data bits!
  - Data might even be correct

Hamming Distance

- The number of bit positions in which two codewords differ is called the Hamming distance
- Example: 10001001 and 10110001
  - d = 3

```
10001001
10110001
--------
00111000
```

- Hamming distance of a code is the minimum distance between any pair of codewords

Hamming Distance (2)

- Error detection:
  - For a code of distance m+1, up to m errors will always be detected

  - Example:

  valid codes: 000, 111

  m+1=3 => up to m=2 errors are guaranteed to be detected

  Codes with error: 001, 010, 100, 100, 101, 011

Hamming Distance (3)

- Error correction:
  - For a code of distance 2m+1, up to m errors can always be corrected by mapping to the closest codeword

  - Example:

  valid codes: 000, 111

  2m+1=3 => up to m=1 errors are guaranteed to be corrected

  If received 001, mapped to 000

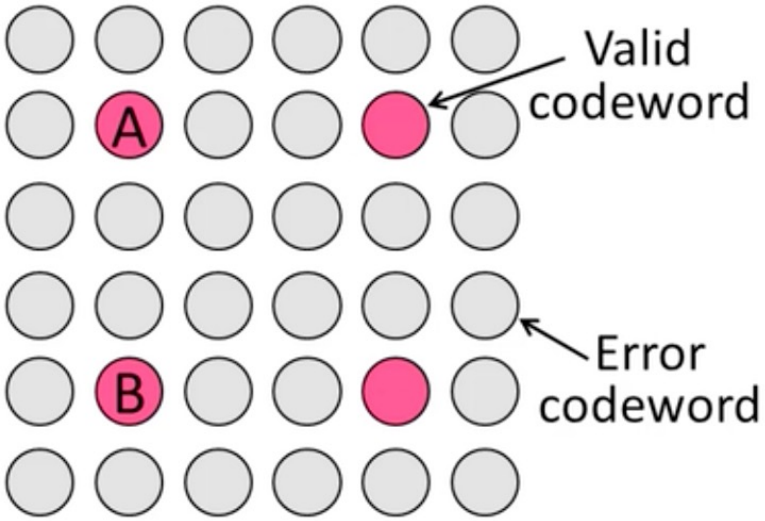
Intuition For Error Correcting Code

- Suppose we construct a code with a Hamming distance of at least 3
  - Need more than or equal to 3 bit errors to change one valid codeword to another
  - Single bit errors will be closest to a unique valid codeword

- If we assume errors are only 1 bit, we can correct them by mapping an error to the closest valid codeword
  - Works for d errors if HD> 2m+1

Intuition

- Visualization of code:



Hamming Code

- Gives a method for constructing a code with a distance of 3
  - Bits of the codeword are numbered consecutively, starting with bit 1
  - Check bits are in positions *p* that are power of 2 (1, 2, 4, 8, 16, etc.)
  - The rest positions are filled with data
  - Check bit in position *p* is parity of positions with a *p* term in their values
    - For example, 11 = 1 + 2 + 8 -> (e.g., bit 11 is checked by bits 1, 2, and 8)

- Given data of length *k,* number of check bits (*r*) needed to correct single errors are computed by: $(k + r + 1) =< 2^r$

Hamming Code Example

- Data = 0101
  - k = 4, r = 3 -> codeword is 7 bits
  - Check bit positions: 1, 2, 4
  - Check bit 1 covers 1, 3, 5, 7
  - Check bit 2 covers 2, 3, 6, 7
  - Check bit 4 covers 4, 5, 6, 7

0	1	0	0	1	0	1
1	2	3	4	5	6	7

Hamming Code (Decoding)

- At receiver:
  - Recompute check bits
  - Arrange as binary number
  - Value (syndrome) tells the error position
  - Value of 0 means no error
  - Otherwise, flip bit to correct

Hamming Code (Decoding) Example

- Data sent: 0100101 Data received: 0110101

0	1	1	0	1	0	1
1	2	3	4	5	6	7

P1= 0+1+1+1=1 P2= 1+1+0+1=1 P4= 0+1+0+1=0

syndrome= 011 -> error is at position 3
change it from 1 to 0

0	1	0	0	1	0	1
1	2	3	4	5	6	7

Link Layer Protocol Example 1

- Assumption: A is sending infinite stream of data (non stop) to B on not noisy link
- Problem: sender floods the receiver with frames faster than it is able to process them (flow control problem)

- Solution: Receiver provides some kind of feedback to sender
  - Can be dummy frame, acknowledgement,…
- Protocols in which the sender sends one frame and then waits for an acknowledgement before proceeding are called **stop-and-wait**

Link Layer Protocol Example 2

- Assumption: A is sending infinite stream of data to B on noisy link
- Problem: Data will be lost or damaged

- Solution: Adding timer for each frame and need to receive acknowledgment (ACK)

- Question: What should be the timer value?
- Answer: Allow enough time for the frame to get to the receiver, for the receiver to process it in the worst case, and for the acknowledgement frame to propagate back to the sender.

Link Layer Protocol Example 3

- Problem: What if the ACK get lost? A will send again -> duplicate packets at receiver

- Solution: Adding **sequence number** to each frame's header
  - The acknowledgement field contains the number of the last frame received without error

- Question: How many bits should be in sequence number field?
  - Sliding window size

Link Layer Protocol Example 4

- Problem: reverse channel normally has the same capacity as the forward channel
  - The capacity of the reverse channel is wasted

- Solution: Piggybacking
  - The acknowledgement is attached to the outgoing data frame (using the ack field in the frame header)
  - The acknowledgement gets a free ride on the next outgoing data frame

Sender's Possible Scenarios

- ACK frame arrives undamaged
  - Checks if the sequence number in ACK matches the frame it is next to send
  - Get the next frame from Network layer
  - Advance the sequence number and send the frame

- Damaged ACK frame arrives or timer expires
  - Frame will be sent again
  - Sequence number will not be changed

Receiver's Possible Scenarios

- Upon reception of a frame, frame and it's sequence number is checked:
  - If duplicate, or damaged, it will be dropped
  - Otherwise, passes it to network layer and generates an ACK

Motivation on Pipelining

- Example: Consider a 50-kbps satellite channel with a 500-msec round-trip propagation delay and 1000-bit frames
  - At t = 0 the sender starts sending the first frame
  - At t = 20 msec the frame has been completely sent
  - At t = 270 msec the frame fully arrived at the receiver
  - At t = 520 msec the acknowledgement arrived back at the sender

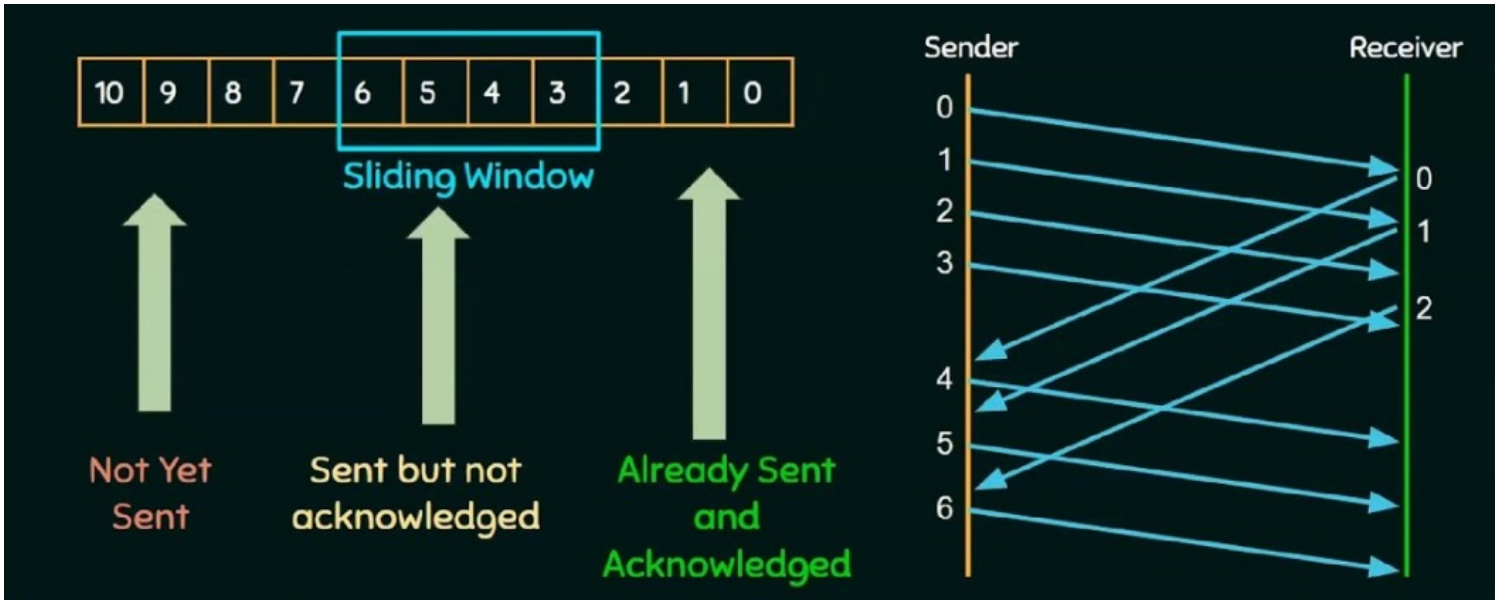
- Sender is blocked 500/520 or 96% of the time (only 4% of the available bandwidth was used)
- Long transit time + high bandwidth + short frame length = disastrous in terms of efficiency
- Solution: Using sliding window (allowing the sender to transmit up to *w* frames before blocking, instead of just 1)

Sliding Window Protocols

- Sending window corresponding to frames sender is permitted to send
  - "window" on the transmitter side represents the logical boundary of the total number of packets yet to be acknowledged by the receiver

- Receiving window corresponding to the set of frames it is permitted to accept

Sliding Window Example



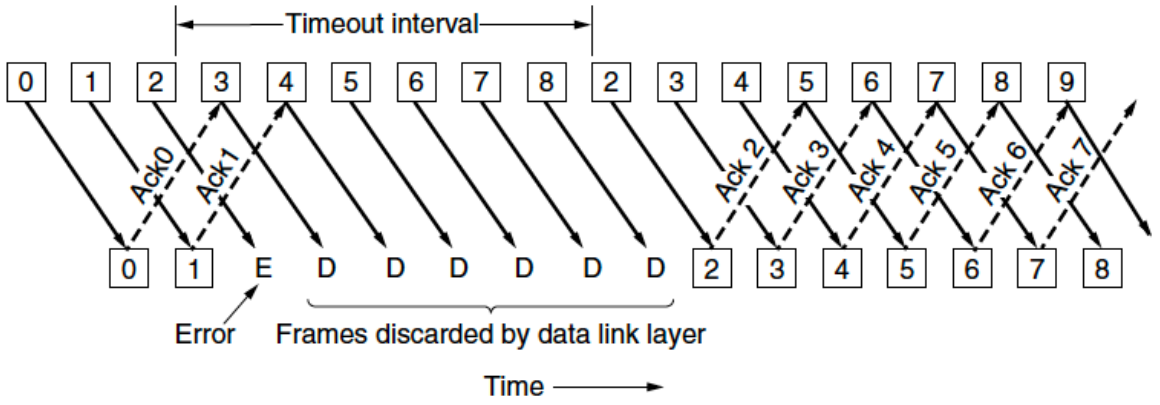
Issues of Pipelining

- What happens if a frame in the middle of a long stream is damaged or lost?
- What should the receiver do with all the correct frames following it?

- Two techniques:
  - Go-back-n
  - Selective repeat

Go-Back-N

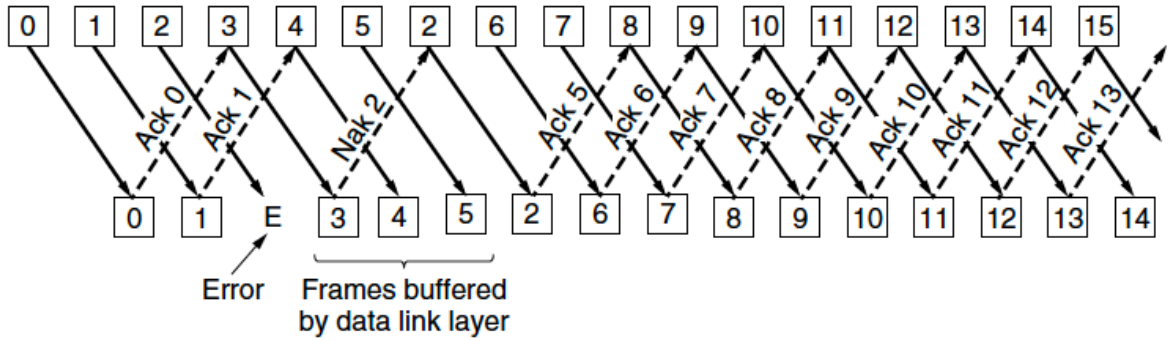
- If a sent frame is found suspected or damaged then all the frames are retransmitted till the last packet
- Receiver Window Size is 1



Selective Repeat

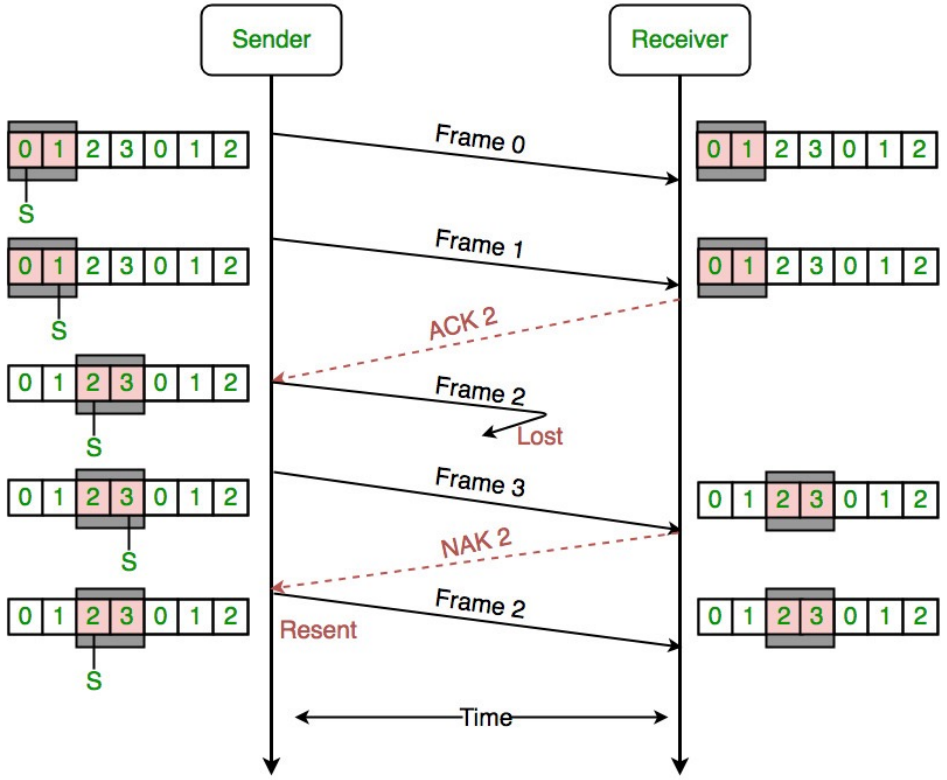
- Only the suspected or damaged frames are retransmitted
- A bad frame that is received is discarded, but any good frames received after it are accepted and buffered
- Receiver Window Size is N
- Receiver send a negative acknowledgement (NAK) when it detects an error
- Receiver window needs to sort the frames

Selective Repeat Example



Cumulative Acknowledgement

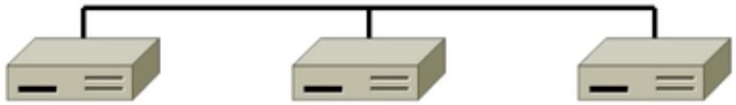
- When an acknowledgement comes in for frame n, frames n − 1, n − 2, and so on are also automatically acknowledged. This type of acknowledgement is called a **cumulative acknowledgement**



MAC Sublayer

Overview

- Network links:
  - Point to point
  - Broadcast channels (multiaccess channels or random access channels)



  - How do nodes share a single link? Who sends when, e.g., in Wifi?
    - Assume no one is in charge. It is a distributed system.

Static Channel Allocation

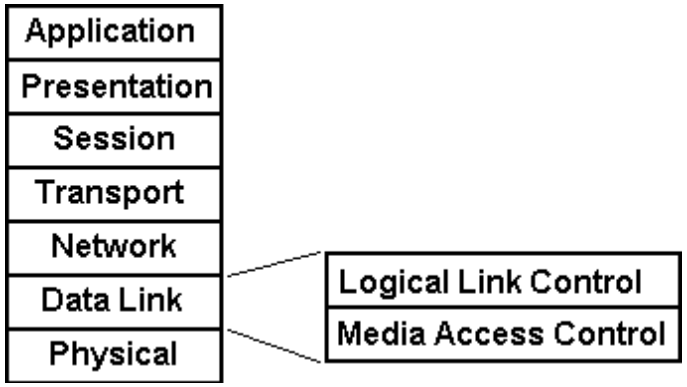
- Channel's capacity is chop up by using one of the multiplexing schemes (FDM, TDM,…)
- Problem: When the number of senders is large and varying or the traffic is bursty, a large piece of valuable spectrum may be wasted

- Solution: Multiple access protocols

Medium Access Control (MAC) Layer

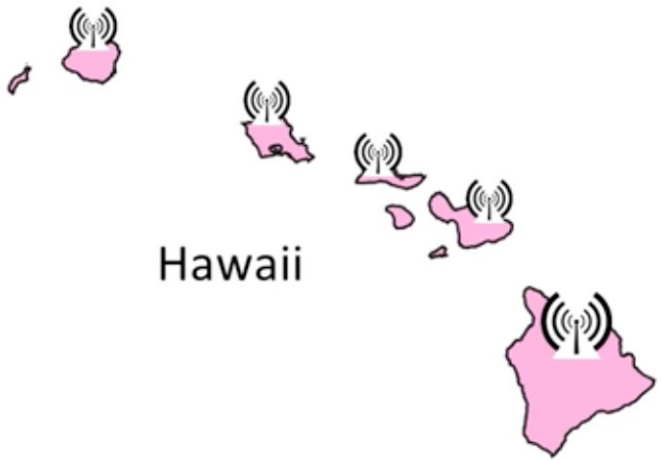
- The protocols used to determine who goes next on a multiaccess channel belong to a sublayer of the data link layer called the **MAC (Medium Access Control) sublayer**

- The MAC sublayer is the bottom part of the data link layer



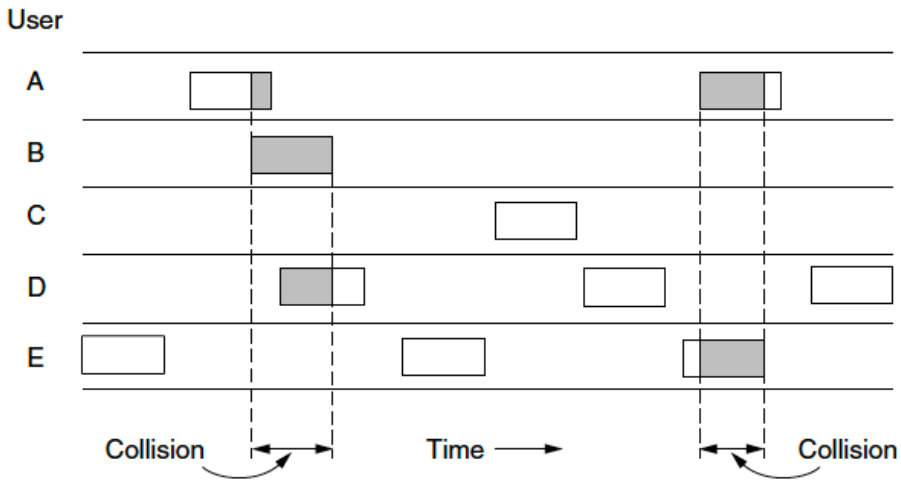
ALOHA Network

- Seminal computer network connecting the Hawaiian islands in the late 1960s
  - When should nodes send?
  - A new protocol was devised by Norm Abramson



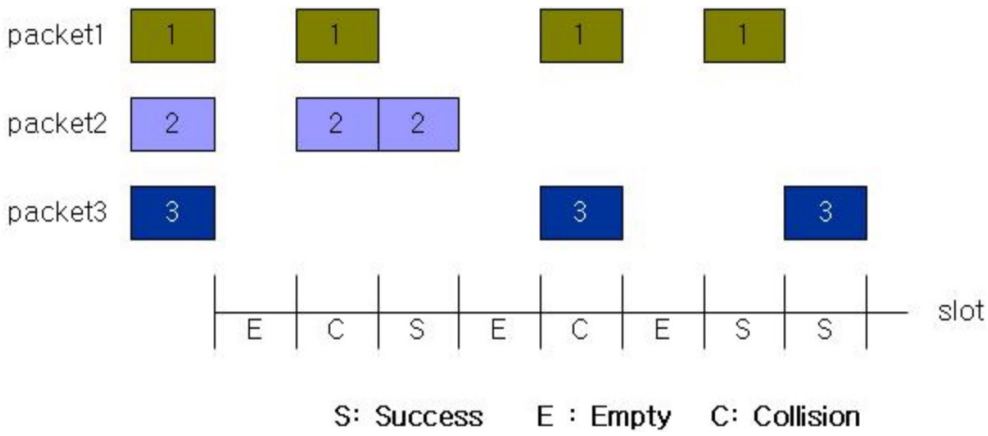
ALOHA Protocol

- Simple idea:
  - Node just sends when it has traffic
  - If there was a collision (no ACK), then wait a random time and resend
- That's it!
- Some frames will be lost but many may get through…
- Good idea??



ALOHA Protocol (2)

- Simple, decentralized protocol that works well under low load!
- Not efficient under high load
  - Analysis shows at most 18% efficiency
  - Improvement: Slotted ALOHA
- Slotted ALOHA: Time is divided into equal size slots and nodes can send at the beginning of slot
  - Efficiency goes up to 36%
- Let's look at other improvements



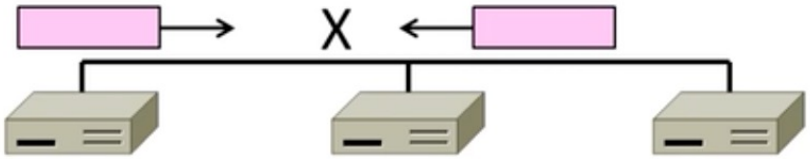
CSMA (Carrier Sense Multiple Access)

- Improve ALOHA by listening for activity before we send
  - Can do easily with wires, not wireless

- Does this eliminate collisions completely?
  - Why? Or why not?

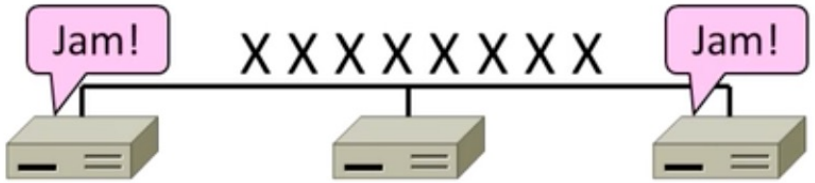
CSMA (2)

1. Sending simultaneously
2. Still possible to listen and hear nothing when another node is sending because of delay



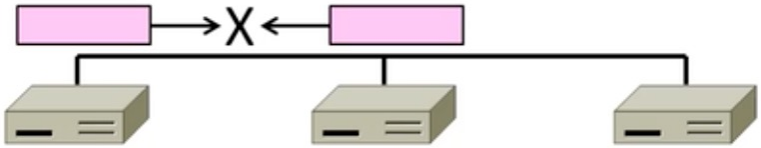
CSMA/CD (with Collision Detection)

- Can Reduce the cost of collisions by detecting them and aborting (Jam) the rest of the frame time
  - Again, we can do it with wires

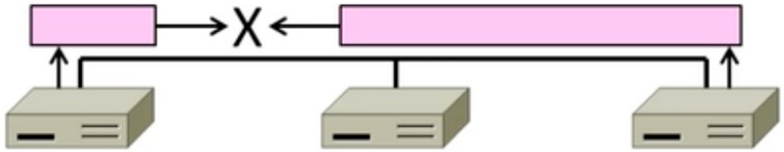


CSMA/CD Complications

- Want everyone who collides to know that it happened
  - Time window in which a node may hear of a collision is 2D seconds

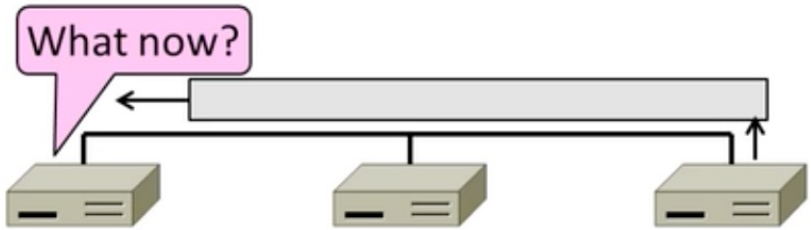


- Impose a minimum frame size that lasts for 2D seconds
  - So, node can't finish before collision
  - Ethernet minimum frame size is 64 bytes



CSMA “Persistence”

- What should a node do if another node is sending?



- Idea: Wait until it is done, and send (The protocol is called **1-persistent** because the station transmits with a probability of 1 when it finds the channel idle)

CSMA “Persistence” (2)

- Problem is that multiple waiting nodes will queue up then collide
  - More load, more of a problem

CSMA “Persistence” (3)

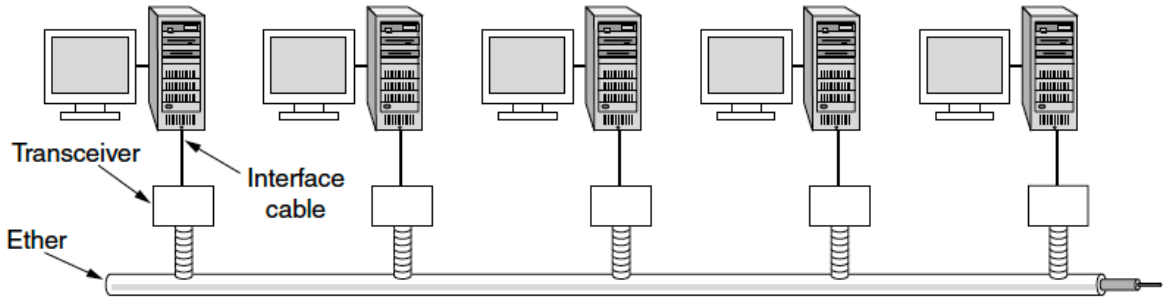
- Intuition for a better solution
  - If there are N queued senders, we want each to send next with probability 1/N

Binary Exponential Backoff (BEB)

- Cleverly estimates the probability
  - 1st collision, wait 0 or 1 frame times
  - 2nd collision, wait 0 or 3 frame times
  - 3rd collision, wait 0 or 7 frame times …
- BEB doubles interval for each successive collision
  - Quickly gets large enough to work
  - Very efficient in practice

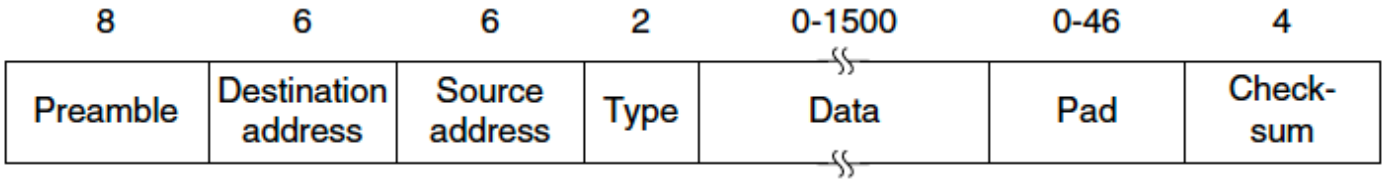
Classic Ethernet, or IEEE 802.3

- Most popular LAN of the 1980s, 1990s
  - 10 Mbps over shared coaxial cable, with baseband signals
  - Multiple access with “1-persistent CSMA/CD with BEB”



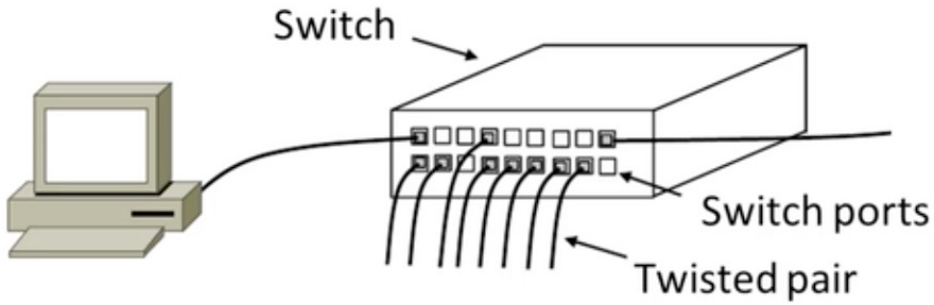
Ethernet Frame Format

- Has Addresses to identify the sender and receiver
  - Globally unique
  - 48 bit number (shown in Hex)
- CRC-32 for error detection, no ACK or retransmission
- Start of frame identified with physical layer preamble



Switched (Modern) Ethernet

- Based on switches, not multiple access, but still called Ethernet
  - We will get to it later



To-do

- Research topic due is on Feb 9th
- Quiz on this lecture next week