# Data communication and networking

# Hw - II

Name: Basil Khowaja

id: BK08432

Q1) a) Given frame Payload:-

| | | | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|
| D | D | E | E | D | D | D | F | D | D | E | D |

⇒ now we will see when a flag byte occurs in the data then we will add Esc byte before it, also in this way if we see that an Esc byte is Present then we will add an additional Esc before it too.

byte-Stuffed frame Payload:-

| | | | | | | | | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| D | D | E | E | E | E | D | D | D | E | F | D | D | D | E | E | D |

b) Given flag bits = 01111110

Given frame:-

000111111100111110100011111111110000111

Ans//:

⇒ now for bit stuff we will see that whenever five consecutive 1's are present in the data, then we will insert a 0 after it to avoid confusion with the flag sequence given = 01111110

So the frame will be:

$\boxed{0001111101100111110100011111011110000111}$

Q2) a) Show the generation of the CRC-32 codeword at the sender site (using binary division) when the dataword is "ABC" in ASCII.

Ans//:- first of all converting ABC in Binary:

A = 01000001 C = 01000011

B = 01000010

⇒ combining:-

$\underbrace{01000001}_{A}\underbrace{01000001}_{B}\underbrace{0010000011}_{C} \rightarrow ①$

⇒ Since this is CRC-32, ⇒ so we will have to add 32 zeros to the ①.

⇒ divisor = 100000100110000010001110110110111

⇒ doing division here in textual format because of space constraint

```
                011
   ____________________________________
  ) 01000001010000010010000110000000000
  - 00000000000000000000000000000000000
  _____________________________________
     1000001010000100100001100000000000
   - 100000100110000010001110110110111
  _____________________________________
          111001000000100011011011100000000
        - 100000100110000010001110110110111
        ___________________________________
          110011001101000010101010101110111 0
        - 100000100110000010001110110110111
        ___________________________________
           1001110101100000100100010001011001
```

011111

100111 01011 000000 1 001 00011 0 11 00 1 0

− 1 100000 100 11 000000 1 0001 11 0110 11 01 1 1

11111 000000000 11 000 11 00000000 1 0 1 000

− 1 000001 0011 00000 1 00011 1 011 01 1 01 1 1

11110 1001 1001 10 1011 111011 00 1 1 1 1 0

− 1 000000 1001 1 0000 0100 0111 011 0110 1 1 1

111 011 0 1 0 1 0 11 0 1 1 1 1 1 0011 0 1 000 1 001

011111111111

111011010101101111100110100010010

− 100000100110000010001110110110111

11011110011101101101000010100101 0

− 10000010011000001000111011011011 1

10111000001011001011110011111101 0

− 100000100110000010001110110110111

111010010011000011001000100110100

− 100000100110000010001110110110111

110101101010000010001100100000110

− 100000100110000010001110110110111

1010100110000000000001001011000100

− 100000100110000010001110110110111

10101111100000100010100110101010

− 100000100110000010001110110110111

01011011110001010100111011100011

⇒ this approach was time consuming
and I got errors in between this
so wrote a basic python code for
calculating the remainder

```python
def remainder(dataword, divisor):
    data=list(map(int, dataword))
    divisor=list(map(int, divisor))
    #adding zeroes in last
    data.extend([0]*(len(divisor)-1)) #this will end zeroes (in our case will append 32 0's)
    for i in range(len(dataword)):    #running the loop for xor of every bit with divisor
        #since we only need to peform xor if the current bit in dataword is 1:
        if data[i]==1:
            for j in range(len(divisor)):
                data[i+j]= data[i+j] ^ divisor[j]

    #remainder will be the last bits after the division
    remainder=data[-(len(divisor)-1):]
    return ''.join(map(str, remainder))

dataword = '010000010100001001000011' #we got this dataword for ABC in asci-II
divisor = '100000100110000010001110110110111' #divisor which was given
remainder=remainder(dataword, divisor)
print("remainder is:", remainder)
```

↳ here I used simple level code for the purpose of calculating remainder.

Output remainder :-

```
PS C:\Users\Dell> & C:/Python311/python.exe "e:/fall 24/DCN/remainder.py"
remainder is: 10101111100000100010100110101010
PS C:\Users\Dell>
```

Remainder :-

10101111100000100010100110101010

⇒ Since final codeword:

Original dataword + remain

010000010100000100100000111010111110000001000101001
10101010

b) for checking the Syndrome we will check that if we divide the codeword recieved by divisor then we get all 0's or not in remainder

if all 0's → codeword accepted

if not → codeword not accepted by reciever

```
> fall 24 > DCN > remainder.py > ...
def check_codeword(codeword, divisor):
    data=list(map(int, codeword))
    divisor=list(map(int, divisor))
    for i in range(len(codeword)-len(divisor)+1):
        if data[i]==1:
            for j in range(len(divisor)):
                data[i + j]= data[i + j]^divisor[j]
    syndrome=data[-(len(divisor)-1):]
    return ''.join(map(str, syndrome))

#codeword which we got after combining the original dataword and the remainder
codeword = '01000001010000010010000011101011111000000100010100110101010'
divisor = '100000100110000010001110110110111'
syndrome=check_codeword(codeword, divisor)
print("syndrome is:",syndrome)
#checking whether we are getting all 0's as remainder or not
if syndrome == '0' * (len(divisor) - 1):
    print("codeword is accepted")
else:
    print("codeword is rejected")
```

↳ I used the code which I already wrote in the Part (a), just removed the 0's Padding part from the function definition since now we aren't dealing with the dataword which needed 32 0's to be added.

⇒ when I ran the code I got the remainder as all 0's.

calculated Syndrome:

```
PS C:\Users\Dell> & C:/Python311/python.exe "e:/fall 24/DCN/remainder.py"
syndrome is: 00000000000000000000000000000000
codeword is accepted
```

⇒ the recieved codeword is accepted by the reciever because the calculated Syndrome is all zeros which shows us that no errors were detected during the transmission. Also, this zero Syndrome implies that the transmitted dataword when appended with the remainder has not changed the Pattern and the data hence the codeword is accepted.

c) I inserted the given codeword in my Python code which will do the division process and tell whether we have 0 syndrome or not:

```python
> fall 24 > DCN > remainder.py > ...
def check_codeword(codeword, divisor):
    data=list(map(int, codeword))
    divisor=list(map(int, divisor))
    for i in range(len(codeword)-len(divisor)+1):
        if data[i]==1:
            for j in range(len(divisor)):
                data[i + j]= data[i + j]^divisor[j]
    syndrome=data[-(len(divisor)-1):]
    return ''.join(map(str, syndrome))

#codeword which we got after combining the original dataword and the remainder
codeword = '10000100100010101100001001011010010111011010000011001111011'
divisor = '100000100110000010001110110110111'
syndrome=check_codeword(codeword, divisor)
print("syndrome is:",syndrome)
#checking whether we are getting all 0's as remainder or not
if syndrome == '0' * (len(divisor) - 1):
    print("codeword is accepted")
else:
    print("codeword is rejected")
```

```
PS C:\Users\Dell> & C:/Python311/python.exe "e:/fall 24/DCN/remainder.py"
syndrome is: 00110001100111101110010101110100
codeword is rejected
```

=> Since we can see that the syndrome is not zero hence the codeword will be rejected by the reciever

Q3) 1) Pure abha :-

$$S = G \cdot e^{-2G}$$

$\downarrow$

avg number of transmission attempts per frame time

$\Rightarrow$ differentiating S with respect to G and making equal to zero

$$\frac{ds}{dG} = \frac{d}{dG}\left(G \cdot e^{-2G}\right)$$

using Product rule of derivative.

$u = G \quad v = e^{-2G}$

$$\frac{ds}{dG} = u' \cdot v + v' \cdot u$$

$$u' = \frac{d}{dG}(G) = 1$$

$$v' = \frac{d}{dG} e^{-2G} = -2 \cdot e^{-2G}$$

$$\frac{ds}{dG} = 1 \cdot e^{-2G} + G \cdot (-2) \cdot e^{-2G}$$

$$\frac{ds}{dG} = e^{-2G} \cdot (1-2G)$$

$$e^{-2G} \cdot (1-2G) = 0$$

$$1-2G = 0$$

$$\boxed{G = 1/2} \longrightarrow \text{condition for max throughput}$$

2) Slotted aloha :-

$$S = G \cdot e^{-G}$$

$$\frac{ds}{dG} = \frac{d}{dG}\left(G \cdot e^{-G}\right)$$

$$u = G \quad , \quad v = e^{-G}$$

$$u' = \frac{d}{dG}(G) = 1$$

$$v' = \frac{d}{dG}\left(e^{-G}\right) = -e^{-G}$$

$$\frac{ds}{dG} = 1 \cdot e^{-G} + G \cdot (-1) \cdot e^{-G}$$

$$\frac{ds}{dG} = e^{-G} \cdot (1-G) = 0$$

$$1 - G = 0$$

$$\boxed{G = 1}$$

$$\Downarrow$$

condition for max throughput

⇒ Pure aloha is random access Protocol where the device transmits data whenever it has information to Send and it doesn't waits for a Specific time Slot, if for example lets Say two devices transmit at the Same time a collision will occur and both messages will be lost and then each device will retransmit after the random delay. while the Slotted aloha improves on the Pure aloha by dividing time into Slots, the devices are only allowed to transmit the data at the beginning of each time Slot.

Pure aloha → max throughput → 18.4%

Slotted aloha → " → 36.8%

**Q4)**

In a CSMA/CD network, devices begin transmitting as soon as they sense that the channel is clear. However, if two devices start transmitting nearly simultaneously, a collision can occur. For effective collision detection, the transmitting device must remain active on the network long enough to detect any collisions that might occur as the signal propagates to the farthest point on the network and back, known as the round-trip time requirement. If the frame is too short, a device could finish transmitting before the collision signal has time to return, resulting in undetected collisions and errors on the network. Ensuring a sufficient frame length allows each station to detect collisions before completing its transmission, thereby maintaining network reliability.

- **Propagation Delay**: In a CSMA/CD network, it takes time Tp for the signal to propagate from one end of the network to the other. If a collision occurs at the far end, it will take another Tp for the collision signal to propagate back to the transmitting station.
- **Frame Transmission Time**: If Tfr is shorter than 2·TP a station could finish sending its frame before it even realizes that a collision occurred. This would defeat the purpose of collision detection.
- **Ensuring Collision Detection**: By ensuring Tfr≥2·TP
 the rule guarantees that:

  - The transmitting station will still be actively sending data when the collision signal reaches it.
  - This enables the station to detect the collision and retransmit the frame later, following the CSMA/CD protocol.

Given:

Tfr=40us

Tp=25us

- Station A starts sending a frame at t=0 μs

- Station B starts sending a frame at t=20 μs

**Does Station A Detect Collision?**

Station A starts transmitting a frame at t=0 μs and will complete its transmission by t=40 μs. If a collision occurs, the collision signal takes Tp=25 μs to reach Station A. By the time this signal arrives at t=25 μs, Station A is still transmitting. Since A continues to transmit until t=40 μs it will ultimately detect the collision. Therefore, Station A will detect the collision.

**Does station B detect collision?**

Station B starts sending its frame at t=20 μs. If a collision happens, it means both Station A and Station B are trying to send data at the same time. Since Station B has already begun sending its data when the collision occurs, it will be able to notice that something went wrong. When a collision happens, the signals from both stations interfere with each other. This interference creates a collision signal that travels back through the network. Because Station B started sending its frame at t=20 μs and the collision occurs shortly after that, the collision signal will reach Station B while it is still transmitting. As a result, Station B can detect the collision during its transmission. It will recognize that its data did not go through correctly, prompting it to stop transmitting and try again later. Therefore, Station B will detect the collision.

**c) For the given network Tp = 25 µs and data rate of 10Mbps, suggest the appropriate value of Tfr and minimum frame size so that both station can detect collision.**

Using the rule;

Tfr>=2.Tp

Tfr>=2x25us (given Tp=25us)

**Tfr>=50us**

Minimum frame size=Tfr x Data rate

Data rate=10mbps (given);

Minimum frame size= 50us (found earlier) x 10mbps

Minimum frame size=50x10^(-6)s x 10 x 10^(6)

**Minimum frame size= 500bits**

**Q5) Contrast different types of MAC addresses and state one use case for each type of MAC address in different LAN settings.**

**1. Unicast MAC Address**

- A unicast MAC address is a unique identifier assigned to a network interface card (NIC) that allows for one-to-one communication. This address is used to send data to a specific device on a network.
- uses:
    - **File Sharing**: Transmitting files directly between two computers in a local network.
    - **VoIP Calls**: Establishing a one-on-one voice call between two IP phones, ensuring that audio packets are sent to the intended recipient.
    - **Secure Connections**: Creating a direct encrypted connection between a user's device and a secure server for data transfer, such as in VPN connections.

**2. Broadcast MAC Address**

- A broadcast MAC address allows data packets to be sent to all devices on a network segment. The standard broadcast address in Ethernet networks is `FF:FF:FF:FF:FF:FF`.
- **Uses**:
    - **ARP Requests**: Address Resolution Protocol (ARP) requests use the broadcast MAC address to map IP addresses to MAC addresses within a local network, allowing devices to discover each other's addresses.
    - **Network Time Protocol (NTP)**: NTP servers may use broadcast messages to synchronize time across all devices in a network simultaneously.
    - **Software Updates**: Sending broadcast messages to notify all devices on a network about available software updates or maintenance tasks.

**3. Multicast MAC Address**

- A multicast MAC address is used to send data packets to a specific group of devices rather than all devices (as with broadcast) or a single device (as with unicast). Multicast addresses are typically in the range of `01:00:5E:00:00:00` to `01:00:5E:7F:FF:FF`.
- **Uses**:
    - **Video Conferencing**: Sending video and audio streams to multiple participants in a video conference using multicast addresses to optimize bandwidth.
    - **Online Gaming**: Multiplayer online games often use multicast to send game state updates to multiple players connected in the same local network.
    - **Sensor Networks**: In IoT applications, multicast can be used to send sensor data to multiple applications or devices that need to receive updates simultaneously.

**Difference between the Different Types of MAC Addresses**

- 
    - **Unicast**: One-to-one communication, where data is sent to a specific device only.
    - **Broadcast**: One-to-all communication, where data is sent to every device on the network.
    - **Multicast**: One-to-many communication, where data is sent to a specific group of devices that have joined a multicast group.
- **In terms of efficiency;**
    - **Unicast**: Efficient for direct communication but can lead to high traffic if many devices are sending data to a single device simultaneously.
    - **Broadcast**: Efficient for sending the same message to multiple devices, but can lead to network congestion if overused, as all devices must process the broadcast frame.
    - **Multicast**: More efficient than broadcast when sending data to multiple specific devices, as only those devices in the multicast group process the data, reducing overall network load.

## Q6)

## A)

The IEEE 802.3 standard for 40Gbps and 100Gbps Ethernet introduces several advancements that differentiate these high-speed Ethernet technologies from earlier standards, such as 10Gbps Ethernet.

1. **Physical Layer Specifications**

- Cabling: 40GbE and 100GbE support similar cabling options, including multimode fiber (MMF) using QSFP+ SR4 optics. 100GBASE-SR10 uses 10 lanes over multimode fiber, and, in contrast, the single-mode version of this module achieves a distance of up to two miles while employing four wavelengths.
- Transceivers & Connectors: Your information will allow any connection, the standards defines different types of transceiver mostly CFP (C Form-factor Pluggable) for 100Gbps connections that operates at various implementations such as 100GBASE-SR10 and 100GBASE-LR4.

2. **Modulation Techniques and Encoding Schemes**

- Encoding Schemes: Different encoding schemes are crucial to an effective apllication of data over communication steams. This includes, for instance, the definition of a 40GbE **that can** take advantage of a 64b/66b encoding scheme instead of the 8b/10 used in legacy Ethernet implementations to provide lower overhead and increased efficiency than its older sibling at >100 Gbps data rates.

- Baseband Modulation Techniques: Progress in modulation techniques allows for faster data rates on the same cabling. Wavelength division multiplexing (WDM) is used in 100GBASE-LR4 for example to have signals on different wavelengths over the same fiber.

3. **Maximum Transmission Distances**

- **Transmission Distances**: 40GbE can achieve a maximum distance of 150 meters over OM3 multimode fiber and up to 400 meters over OM4 fiber. In contrast, 100GbE can reach up to 100 meters on OM3 and 150 meters on OM4 fiber. For single-mode fiber, both standards support longer distances, with 100GBASE-LR4 achieving distances of up to 10 kilometers.

4. **Data Rate**

- **Data Rates**: The main distinguishing feature is the significant increase in data rates. While 10Gbps Ethernet allows for speeds of up to 10 Gbps, 40GbE operates at 40 Gbps, and 100GbE operates at 100 Gbps. This increase allows for faster data transmission and the ability to handle larger amounts of network traffic

5**. Signal Types**

- **Signal Types**: 40GbE and 100GbE utilize multiple signal types, including:
  - **10GBASE-R** for 10 Gbps links,
  - **40GBASE-SR4** for short-range multimode applications,
  - **100GBASE-SR10** for short-range multimode applications,
  - **100GBASE-LR4** for long-range single-mode applications

These signal types allow for flexibility in deployment and compatibility with existing infrastructure while providing options for both short-range and long-range connectivity. The IEEE 802.3 standard for 40Gbps and 100Gbps Ethernet introduces critical advancements in physical layer specifications, modulation techniques, maximum transmission distances, data rates, and signal types, significantly enhancing network performance compared to earlier standards like 10Gbps Ethernet. These improvements cater to the growing demand for higher bandwidth and faster data transmission in modern network environments.

**Q7)**

Q7) a) Sum of all words

$= 4500 + 0073 + 0000 + 4000$

$$\begin{array}{r} 4500 \\ +\ 0073 \\ \hline 4573 \end{array}$$

$$\begin{array}{r} 4573 \\ +\ 0000 \\ \hline 4573 \end{array}$$

$$\begin{array}{r} 4573 \\ +\ 4000 \\ \hline 8573 \end{array}$$

One's complement:-

$(8573)_{16}$

1000 0101 0111 0011

$(7ABC)_{16} \leftarrow$ 0111 1010 1000 1100

Checksum $= (7ABC)_{16}$

**b)**

b) message that the reciever gets :-

words + checksum

4500 0073 0000 4000 7A8C

↓

Sum = 8573 (done earlier)

$$\begin{array}{r} 8573 \\ +\ 7A8C \\ \hline ffff \end{array}$$

↓

1111 1111 1111 1111

↓

0000 0000 0000 0000

↓

$(0000)_{16}$

hence the recieved packet will be error free

c) the recalculated Syndrome at reciever's end is $(8573)_{16}$

adding it to the checksum sent by the sender → $(7A8C)_{16}$

we get,

$(ffff)_{16}$

⇒ the Syndrome value has all bits equal to 1 which means there is no error and the message is perfect match.