Homework 2
CSCE5580 Computer Network (Fall 2024)

Uday Bhaskar Valapadasu
11696364

1Ans:

a)
When sending an application message using TCP, the application writes data to the connection's send buffer. TCP then extracts bytes from this buffer, which means it can include more or fewer bytes than a single message in a TCP segment. In contrast, UDP encapsulates exactly what the application provides in a segment; if the application sends a message to UDP, that message becomes the payload of the UDP segment. As a result, UDP allows the application more direct control over the data included in a segment.

b)
With TCP, the presence of flow control and congestion control can lead to notable delays between when an application writes data to its send buffer and when that data is passed to the network layer. On the other hand, UDP does not experience such delays, as it does not implement flow control or congestion control mechanisms.

2Ans:

SYN Flood Attacks
A SYN flood attack targets the TCP connection establishment process, specifically exploiting the three-way handshake that TCP uses. In this handshake, a client sends a SYN (synchronize) message to initiate a connection with a server. The server then allocates resources and responds with a SYN-ACK (synchronize-acknowledge) message. Finally, the client completes the process by sending back an ACK (acknowledge). During a SYN flood attack, an attacker sends many SYN segments to the server without completing the handshake by sending the final ACK. This results in the server allocating resources for many half-open connections that never fully establish. Consequently, legitimate clients may be unable to connect because the server's resources become exhausted.

SYN Cookies as a Defense
SYN cookies provide an effective defense against SYN flood attacks. When a server receives a SYN segment, it does not immediately allocate resources for the connection. Instead, it creates an initial TCP sequence number based on a hash function that includes the source and destination IP addresses, port numbers, and a secret number known only to the server. This sequence number acts as a "cookie."
The server then sends a SYN-ACK message to the client, including this cookie. If the client is legitimate, it will respond with an ACK that includes the cookie value. Upon receiving this ACK, the server can verify the legitimacy of the request by recomputing the cookie using the same hash function. If the calculated value matches the one in the ACK, the server can confidently establish a full connection and allocate resources accordingly.
If the original SYN was part of an attack, the server will not receive a valid ACK response, meaning it hasn't wasted resources on half-open connections. This method effectively protects the server from SYN flood attacks while still allowing legitimate connections to be established, ensuring that the server remains responsive to valid requests.

3Ans:

Issues with Not Using Sequence Numbers in TCP/IP Protocol

1. **Data Integrity and Ordering**: Without sequence numbers, packets may arrive out of order, making it difficult for the receiving application to reconstruct messages correctly. For example, if packets 1, 3, and 2 are received, the application cannot process them in the intended sequence.
2. **Duplicate Packet Handling**: Sequence numbers help identify and discard duplicate packets. If a packet is resent due to a timeout and lacks a sequence number, the receiver may process it multiple times, leading to inconsistent application states.
3. **Connection Management Vulnerabilities**: Mechanisms like SYN cookies, used to defend against SYN flood attacks, rely on sequence numbers to verify incoming ACKs. Without them, the server cannot distinguish between legitimate and malicious connection attempts, risking resource exhaustion.

Reliable Data Transfer at Other Layers

1. **Application Layer Reliability**: Implementing reliability at the application layer adds complexity, as applications must manage their own retransmissions and error handling. This can lead to inconsistencies and increased development effort.
2. **Link Layer Protocols**: Link-layer protocols like Ethernet provide some reliability, but this can result in redundancy if TCP also implements reliability. This overlap may reduce throughput and increase latency, especially in high-speed networks.
3. **Network Layer Reliability**: Adding reliability features to the network layer (e.g., in IP) would complicate the protocol's design and increase processing overhead for routers. This contradicts IP's goal of being simple and efficient.

Conclusion

While reliable data transfer can be implemented at different layers of the TCP/IP stack, each approach presents challenges that can complicate implementation and reduce efficiency. TCP effectively manages reliability at the transport layer, using sequence numbers to ensure data integrity and proper connection management.

4Ans:

Step 1: Determine New Subnet Mask

1. The original subnet mask is /19, which means the first 19 bits are for the network portion.
2. To create four equally sized subnets, we need to borrow 2 bits from the host portion (since 2^2=4).
3. This gives us a new subnet mask of /21 (19 + 2).

Step 2: Calculate Subnets

With a new subnet mask of /21, the subnets will be:

1. **First Subnet**: 138.90.160.0/21
2. **Second Subnet:** 138.90.168.0/21
3. **Third Subnet:** 138.90.176.0/21
4. **Fourth Subnet**: 138.90.184.0/21

Step 3: Calculate Hosts per Subnet

With a /21 subnet mask:

- Total addresses per subnet: 2^{11}=2048
- Usable addresses: 2048−2=2046 (subtracting 1 for the network address and 1 for the broadcast address).

Step 4: Range of Host IP Addresses for the First Subnet

For the first subnet 138.90.160.0/21:

- **Network Address**: 138.90.160.0
- **Broadcast Address**: 138.90.167.255
- **Range of Usable Host IP Addresses**:

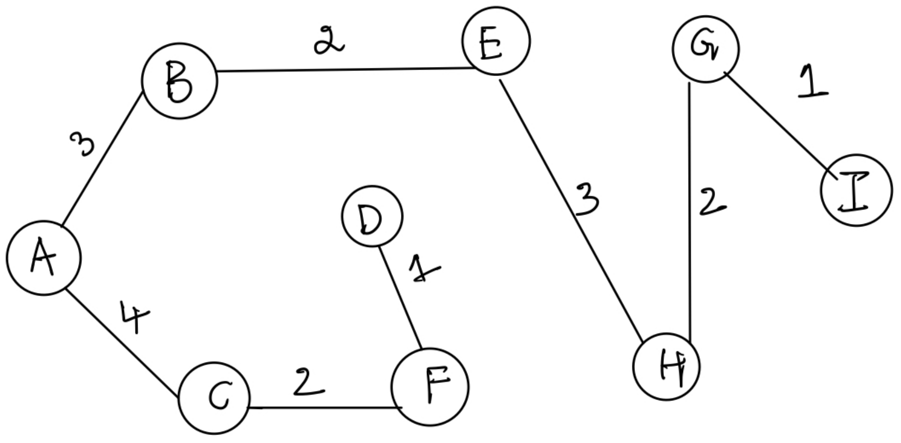
From First usable IP: 138.90.160.1 – To Last usable IP: 138.90.167.254

5Ans:

5.1)
Work/Table:

	A	B	C	D	E	F	G	H	I
A	0	③	4	∞	6	∞	∞	∞	∞
B			④	∞	5	∞	∞	∞	∞
C				8	⑤	6	∞	∞	∞
E				8		⑥	11	8	∞
F				⑦			11	8	∞
D							11	⑧	∞
H							⑩		12
G									⑪
I									

Destination	Distance from A	Predecessor
A	0	—
B	3	A
C	4	A
D	7	F
E	5	B
F	6	C
G	10	H
H	8	E
I	11	G



Shortest Path from A to all network nodes

Final Next-Hop (Forwarding) Table for A :-

Shortest path to Destination	Next Hop
B : A → B	B
C : A → C	C
D : A → C → F → D	C
E : A → B → E	B
F : A → C → F	C
G : A → B → E → H → G	B
H : A → B → E → H	B
I : A → B → E → H → G → I	B

5.2)

Bellman-Ford Algorithm:

1. Initialize:
- Source node A = 0
- All other nodes = ∞
- Each node only knows costs to immediate neighbors

Initial State (Distance to A):

A = 0
B = ∞
C = ∞
D = ∞
E = ∞
F = ∞
G = ∞
H = ∞
I = ∞

Solving it iteratively:

Iteration 1:

- A tells neighbors: "I'm 0 from A"
    - B updates: min(∞, 0+3) = 3
    - C updates: min(∞, 0+4) = 4
    - E updates: min(∞, 0+6) = 6

After Iteration 1:

A = 0
B = 3
C = 4
D = ∞
E = 6
F = ∞
G = ∞
H = ∞
I = ∞

Iteration 2:

- B tells neighbors: "I'm 3 from A"
    - E updates: min(6, 3+2) = 5
- C tells neighbors: "I'm 4 from A"
    - D updates: min(∞, 4+4) = 8
    - F updates: min(∞, 4+2) = 6
- E tells neighbors: "I'm 5 from A"
    - D updates: min(8, 5+6) = 8
    - G updates: min(∞, 5+6) = 11
    - H updates: min(∞, 5+3) = 8

After Iteration 2:

A = 0
B = 3
C = 4
D = 8
E = 5
F = 6
G = 11
H = 8
I = ∞

Iteration 3:

- F tells neighbors: "I'm 6 from A"
    - D updates: min(8, 6+1) = 7
- G tells neighbors: "I'm 11 from A"
    - I updates: min(∞, 11+1) = 12
    - H updates: min(8, 11+2) = 8

- H tells neighbors: "I'm 8 from A"
    - G updates: min(11, 8+2) = 10

After Iteration 3:

A = 0
B = 3
C = 4
D = 7
E = 5
F = 6
G = 10
H = 8
I = 12

Iteration 4:

- G tells neighbors: "I'm 10 from A"
    - I updates: min(12, 10+1) = 11

Final distances to A:

A = 0
B = 3
C = 4
D = 7
E = 5
F = 6
G = 10
H = 8
I = 11

Final Next-Hop Routing Table:

Node	Next Hop	Distance to A
A	-	0
B	A	3
C	A	4
D	F	7
E	B	5
F	C	6
G	H	10
H	E	8
I	G	11

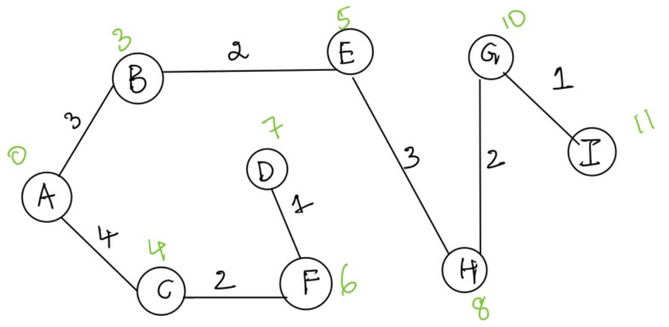


Fig :- Shortest Distance from each node to A

The algorithm converges because:
1. Each node updates its distance when it finds a better path
2. Updates continue until no better paths can be found
3. The process stops when no changes occur in an iteration

6Ans:

1. First, let's understand what we're given:
    - Base network: 211.1.16.0/24
    - Subnet 1 needs: 62 interfaces
    - Subnet 2 needs: 95 interfaces
    - Subnet 3 needs: 16 interfaces
2. Let's calculate the required subnet sizes:
    - Subnet 1: 62 hosts → needs 6 bits (2^6 = 64 addresses)
    - Subnet 2: 95 hosts → needs 7 bits (2^7 = 128 addresses)
    - Subnet 3: 16 hosts → needs 5 bits (2^5 = 32 addresses)
3. This means we need:
    - Subnet 1: /26 subnet (32-26 = 6 bits for hosts)
    - Subnet 2: /25 subnet (32-25 = 7 bits for hosts)
    - Subnet 3: /27 subnet (32-27 = 5 bits for hosts)
4. Starting from 211.1.16.0, we can allocate:
    - Subnet 2 (largest) first: 211.1.16.0/25 (0-127)
    - Subnet 1 next: 211.1.16.128/26 (128-191)
    - Subnet 3 last: 211.1.16.192/27 (192-223)

Therefore, the three network addresses that satisfy these constraints are:

- Subnet 2: 211.1.16.0/25
- Subnet 1: 211.1.16.128/26
- Subnet 3: 211.1.16.192/27

7Ans:

No, Dijkstra's algorithm cannot be directly modified to find the longest path in a graph. Here's why:

1. **Fundamental Issue**:

- Dijkstra's algorithm works for shortest paths because it relies on the principle that sub-paths of shortest paths are also shortest paths
- This principle doesn't hold for longest paths
- If the graph contains cycles, the longest path could become infinite by repeatedly traversing the cycle

2. **Alternative Solutions**:

- For acyclic graphs (DAGs): You can negate all edge weights and use Dijkstra's to find shortest path
- For general graphs: Must use different algorithms like:
    - Dynamic programming with path length constraints
    - Branch and bound methods
    - Depth-first search with cycle detection

The longest path problem is actually NP-hard in general graphs, unlike the shortest path problem which can be solved efficiently with Dijkstra's algorithm.

8Ans:

In a synchronous version of the distance-vector algorithm, the maximum number of iterations required for the algorithm to converge is at most d−1, where d represents the diameter of the network. The diameter is defined as the length of the longest path without loops between any two nodes in the network.

During each iteration, all nodes simultaneously exchange their distance vectors with their immediate neighbors. Initially, each node knows only the costs to its direct neighbors. After the first iteration, a node can determine the shortest paths to nodes that are one hop away, and with each subsequent iteration, it can learn about paths that are progressively longer.

By the d-1iteration, all nodes will have updated their distance tables to reflect the shortest paths that are d or fewer hops away. This is because any path that exceeds d hops will necessarily involve loops and thus cannot represent a shorter path than those identified within d hops.

It's important to note that if the algorithm is triggered by changes in link costs, there is no predetermined limit on the number of iterations required for convergence, as this depends on the specifics of those changes. Therefore, while the algorithm converges within d-1iterations under stable conditions, dynamic changes can complicate convergence.

9Ans:

1) **DHCP Server in Router**
    a. The wireless router includes a built-in DHCP (Dynamic Host Configuration Protocol) server
    b. This DHCP server automatically handles IP address management for your internal network
    c. When a PC connects to the wireless network, it sends a DHCP request
    d. The router's DHCP server responds by assigning:
        i. A private IP address (typically in ranges like 192.168.1.x)
        ii. Subnet mask
        iii. Default gateway (router's internal IP)
        iv. DNS server information
2) **NAT (Network Address Translation)**
    a. The router uses NAT because it only receives one public IP address from the ISP
    b. NAT is necessary to allow multiple devices (5 PCs) to share this single public IP
    c. How NAT works in this setup:
        i. Internal devices use private IP addresses assigned by DHCP
        ii. When a PC sends traffic to the Internet:
            1. Router records the PC's private IP and port in its NAT table
            2. Router replaces private IP with its public IP
            3. When responses return, router uses NAT table to forward traffic back to correct PC
    d. This allows all 5 PCs to simultaneously access the Internet using one public IP
3) **Complete Process Flow**
    a. ISP assigns one public IP to router's WAN interface
    b. Router's DHCP server creates private network for PCs
    c. Each PC connects wirelessly and gets private IP configuration
    d. NAT handles translation between private and public addressing
    e. All internal devices can access Internet through shared public IP

This setup provides:

- Efficient IP address management through DHCP
- Internet access for multiple devices using one public IP through NAT
- Basic network security by hiding internal IP addresses