

Comp 416 Project 3

Network Layer Analysis and Routing Simulator

Gül Sena Altıntaş – 64284

Deadline: January 11, 2021

Table of Contents

Part 1: ICMP Analysis	2
1.a: Ping Analysis	2
Questions.....	2
1.b: Traceroute Analysis	5
Questions.....	5
Part 2: Routing Implementation	7
NaiveFloodingAlgorithm:	7
FloodingAlgorithm:	7
NaiveMinimumCostAlgorithm:	8
MinimumCostAlgorithm:	8

Part 1: ICMP Analysis

url assigned: <https://www.ucsb.edu/>

First ping request can be seen in Figure 1 Ping 1 in the Command Prompt Figure 1, the rest of the ping requests can be seen in Table 4.

1.a: Ping Analysis

First ping request sent on Jan 9, 2021 03:38 GMT+3

Command Prompt:

```
C:\Users\HEWLETT-PACKARD>ping -4 -n 5 ucsb.edu

Pinging ucsb.edu [23.185.0.2] with 32 bytes of data:
Reply from 23.185.0.2: bytes=32 time=52ms TTL=53
Reply from 23.185.0.2: bytes=32 time=48ms TTL=53
Reply from 23.185.0.2: bytes=32 time=273ms TTL=53
Reply from 23.185.0.2: bytes=32 time=61ms TTL=53
Reply from 23.185.0.2: bytes=32 time=48ms TTL=53

Ping statistics for 23.185.0.2:
    Packets: Sent = 5, Received = 5, Lost = 0 (0% loss),
    Approximate round trip times in milli-seconds:
        Minimum = 48ms, Maximum = 273ms, Average = 96ms
```

Figure 1 Ping 1 in the Command Prompt

Filter used: ICMP

No.	icmp icmpv6	Source	Destination	Protocol	Length	Info
35	1.493080	23.185.0.2	192.168.1.101	ICMP	74	Echo (ping) request id=0x0001, seq=1678/36358, ttl=128 (reply in 35)
47	2.460467	192.168.1.101	23.185.0.2	ICMP	74	Echo (ping) reply id=0x0001, seq=1678/36358, ttl=53 (request in 34)
48	2.508228	23.185.0.2	192.168.1.101	ICMP	74	Echo (ping) request id=0x0001, seq=1679/36614, ttl=128 (reply in 48)
49	2.509406	23.185.0.2	192.168.1.101	ICMP	74	Echo (ping) reply id=0x0001, seq=1679/36614, ttl=53 (request in 47)
54	3.478477	192.168.1.101	23.185.0.2	ICMP	74	Echo (ping) request id=0x0001, seq=1680/36870, ttl=128 (reply in 56)
56	3.751240	23.185.0.2	192.168.1.101	ICMP	74	Echo (ping) reply id=0x0001, seq=1680/36870, ttl=53 (request in 54)
71	4.498485	192.168.1.101	23.185.0.2	ICMP	74	Echo (ping) request id=0x0001, seq=1681/37126, ttl=128 (reply in 72)
72	4.559316	23.185.0.2	192.168.1.101	ICMP	74	Echo (ping) reply id=0x0001, seq=1681/37126, ttl=53 (request in 71)
88	5.516257	192.168.1.101	23.185.0.2	ICMP	74	Echo (ping) request id=0x0001, seq=1682/37382, ttl=128 (reply in 89)
89	5.564442	23.185.0.2	192.168.1.101	ICMP	74	Echo (ping) reply id=0x0001, seq=1682/37382, ttl=53 (request in 88)

Figure 2 Ping 1 Wireshark Capture

Questions

- What are the three layers in the ICMP packet?
 - IP Header (20 bytes in IPv4, 40 bytes in IPv6, the queries in this part are made with IPv4)
 - ICMP Header (8 bytes) consists of Type of the message (1 byte, in the case of ping 0), Code (1 byte), Checksum (2 bytes), Header data (4 bytes, consists of identifier (2 bytes) and sequence numbers (2 bytes))
 - ICMP Payload (size is implementation dependent but the total packet length must be less than or equal to MTU (maximum transmission unit), here it was 32 bytes)
- What is TTL and its significance? Which layer does it reside in and is it constant (format and no. of bits wise) across IPV4 and IPV6 ping commands?
 - TTL (Time-to-live) specifies the lifetime of a packet in the network. TTL is decremented by 1 each time the packet is processed by a router, and the packet is dropped if TTL becomes 0. In IPv4, the unit of TTL is seconds.

- TTL is specified in the IP Header as a 1 byte-field.
- In IPv6, *Hop Limit* is used instead of TTL to reflect that TTL is decremented by 1 at each hop. Theoretically TTL may range between 0 and 255 (2^8-1) because both in IPv4 and IPv6 TTL is limited to 8 bits. The order it appears in the header changes between v4 and v6.

3. Why is it that an ICMP packet does not have source and destination port numbers?

ICMP packet does not have source and destination port numbers because ICMP is not used for communication between end hosts but to communicate network-layer information. Therefore, the host and source IP addresses specified in the IP header are enough. These hosts or routers handle ICMP messages and do not need to forward messages to an application-layer process.

4. What is the length of the data field of the ICMP part Type -8? Elaborate on the structure of the data field citing any correspondingly common and changing parts across various messages? If part of the data field, what do you think is the reason for that?

Data field is 32 bytes long in ICMP Type -8

Data (32 bytes)

Data: 6162636465666768696a6b6c6d6e6f7071727374757677616263646566676869
[Length: 32]

Figure 3 Data field in ICMP

Data field contains the same 32 characters across different Echo request and replies. This reflects the goal of the ICMP Type-8 messages and their “echo”ing nature. We are not concerned about the data of the packet but whether the packets reach destination and the time it takes. Data field is implementation specific and contents can be seen in Table 1.

Table 1 Contents of the Data Field

abcdefghijklmnopqrstuvwxyz

5. Find the minimum TTL below which the ping messages do not reach your particular URL destination.

⇒ 10. Found the minimum TTL by examining the behavior on the Command Prompt.
The analysis can be seen in Table 2.

Table 2 Determining minimum TTL

	Command	Reply
1.	ping -4 -i 40 -n 5 ucsb.edu	Successful reply
2.	ping -4 -i 30 -n 5 ucsb.edu	Successful reply
3.	ping -4 -i 20 -n 5 ucsb.edu	Successful reply
4.	ping -4 -i 10 -n 5 ucsb.edu	Reply from 62.115.166.183: TTL expired in transit.
5.	ping -4 -i 15 -n 5 ucsb.edu	Successful reply
6.	ping -4 -i 12 -n 5 ucsb.edu	Successful reply
7.	ping -4 -i 11 -n 5 ucsb.edu	Successful reply

6. How do the Identifier and Sequence Number compare for successive echo request packets?

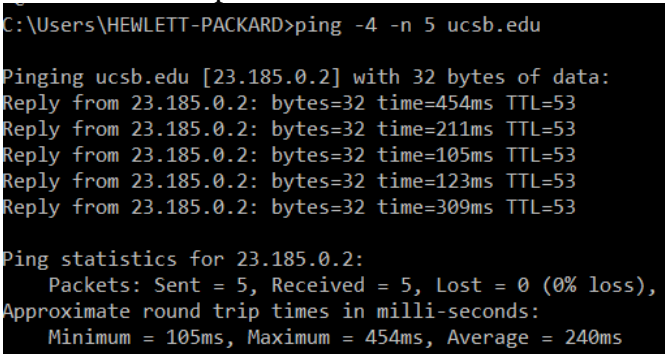
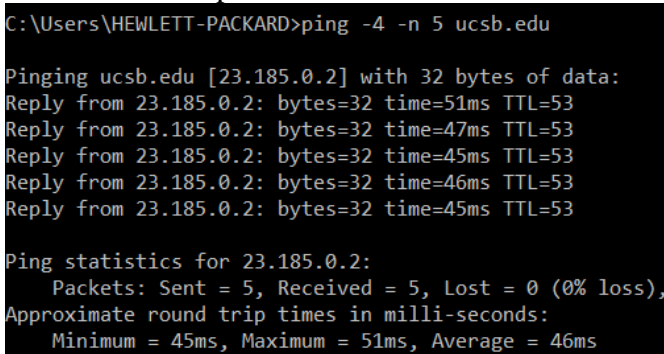
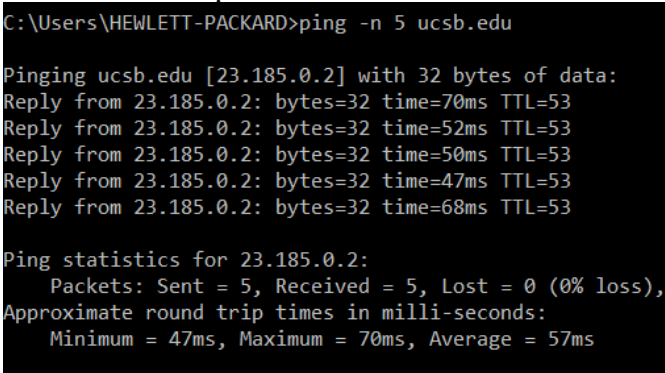
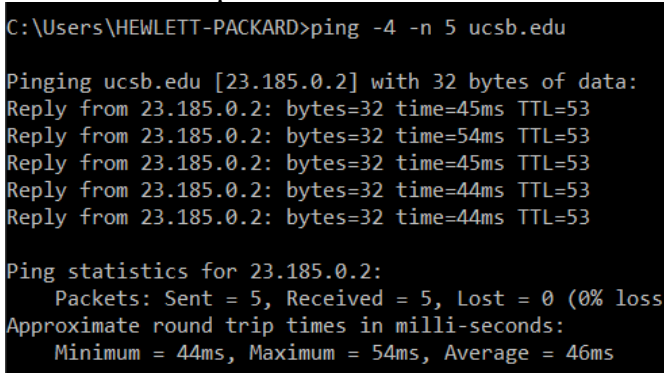
Since I am using a Windows computer, Identifier number is fixed. Sequence number is used to match echo requests with responses as seen in Table 3. Therefore, each time a reply is received with the last sequence number, it is increased, see Big Endian (BE) representation for easier follow.

Table 3 Sequence and Identifier Numbers in the first two echo requests and replies

Identifier (BE): 1 (0x0001) Identifier (LE): 256 (0x0100) Sequence Number (BE): 1678 (0x068e) Sequence Number (LE): 36358 (0x8e06) <i>Figure 4 Sequence Numbers in First Echo Request</i>	Identifier (BE): 1 (0x0001) Identifier (LE): 256 (0x0100) Sequence Number (BE): 1678 (0x068e) Sequence Number (LE): 36358 (0x8e06) <i>Figure 5 Sequence Numbers in First Echo Reply</i>
Identifier (BE): 1 (0x0001) Identifier (LE): 256 (0x0100) Sequence Number (BE): 1679 (0x068f) Sequence Number (LE): 36614 (0x8f06) <i>Figure 6 Sequence Numbers in Second Echo Request</i>	Identifier (BE): 1 (0x0001) Identifier (LE): 256 (0x0100) Sequence Number (BE): 1679 (0x068f) Sequence Number (LE): 36614 (0x8f06) <i>Figure 7 Sequence Numbers in Second Echo Reply</i>

Finally, the rest of the Ping requests can be seen in Table 4.

Table 4 Four Ping Requests sent at different times

Second ping request sent on Jan 9, 2021 03:34 GMT+3 Command Prompt:  <i>Figure 8 Ping 2 in the Command Prompt</i>	Fourth ping request sent on Jan 9, 2021 16:35 GMT+3 Command Prompt:  <i>Figure 9 Ping 4 in the Command Prompt</i>
Third ping request sent on Jan 9, 2021 11:27 GMT+3 Command Prompt:  <i>Figure 10 Ping 3 in the Command Prompt</i>	Fifth ping request sent on Jan 11, 2021 12:42 GMT+3 Command Prompt:  <i>Figure 11 Ping 5 in the Command Prompt</i>

1.b: Traceroute Analysis

Traceroute on Windows on Jan 8, 2021 13:22 GMT+3

```
C:\Users\HEWLETT-PACKARD>tracert ucsb.edu

Tracing route to ucsb.edu [23.185.0.2]
over a maximum of 30 hops:

  1  *      11 ms    3 ms  MitraStar.Home [192.168.1.1]
  2  18 ms   17 ms    8 ms  212.156.201.61.static.turktelekom.com.tr [212.156.201.61]
  3  21 ms   12 ms   11 ms  81.212.2.41.static.turktelekom.com.tr [81.212.2.41]
  4  18 ms   22 ms   16 ms  35-izmir-xrs-t2-2---10-balikesir-t3-4.statik.turktelekom.com.tr [212.156.109.252]
  5  12 ms   10 ms   11 ms  46-kahramanmaras-t3-1---46-kahramanmaras-t3-2.statik.turktelekom.com.tr [81.212.30.5]
  6  30 ms   31 ms   35 ms  308-buk-col-1---34-inkilap-xrs-t2-1.statik.turktelekom.com.tr [212.156.139.66]
  7  52 ms   48 ms   37 ms  buca-b2-link.telial.net [62.115.37.72]
  8  52 ms   49 ms   48 ms  win-bb3-link.ip.twelve99.net [62.115.118.48]
  9  48 ms   47 ms   49 ms  win-b2-link.ip.twelve99.net [62.115.114.185]
 10  51 ms   50 ms   49 ms  fastly-ic-340351-win-b4.c.telial.net [62.115.166.183]
 11  48 ms   48 ms   47 ms  23.185.0.2

Trace complete.
```

Figure 12 Traceroute Analysis on Command Line

Questions

- How long is the ICMP header of a TTL Exceeded packet? Select different parts of the header in Wireshark to see how they correspond to the bytes in the packet.
Header is 8 bytes long. The breakdown of the fields can be seen in Table 5 and Figure 13. Note that ICMP is encapsulated in an IP packet and ICMP Payload includes the ICMP Header and Data as seen in Table 6.

Table 5 ICMP Packet Header

IP Header	Type = 11	Code = 0	Checksum	Unused = 0	ICMP Payload		
					IP Header	ICMP Header	ICMP Data
20 bytes	1 byte	1 byte	2 bytes	4 bytes	20 bytes	8 bytes	64 bytes

Wireshark · Packet 533 · part2-tracert.pcapng

```
> Frame 533: 134 bytes on wire (1072 bits), 134 bytes captured (1072 bits) on interface \Device\NPF_{492166AC-832F-4DA9-957B-1DAD10908562}, id 0
> Ethernet II, Src: MitraStar.Home (c8:54:4b:62:aa:06), Dst: IntelCor_28:a9:47 (10:02:b5:28:a9:47)
> Internet Protocol Version 4, Src: MitraStar.Home (192.168.1.1), Dst: 192.168.1.101 (192.168.1.101)
  > Internet Control Message Protocol
    Type: 11 (Time-to-live exceeded)
    Code: 0 (Time to live exceeded in transit)
    Checksum: 0xf4ff [correct]
    [Checksum Status: Good]
    Unused: 00000000
  > Internet Protocol Version 4, Src: 192.168.1.101 (192.168.1.101), Dst: 23.185.0.2 (23.185.0.2)
    > Internet Control Message Protocol
      Type: 8 (Echo (ping) request)
      Code: 0
      Checksum: 0xf15a [unverified] [in ICMP error packet]
      [Checksum Status: Unverified]
```

```
0000  00010000 00000010 10110101 00101000 10101001 01000111 11001000 01010100  ... (.G.T
0008  01001011 01100010 10101010 00000110 00001000 00000000 01000101 11000000  Kb...E-
0010  00000000 01111000 01100110 00100011 00000000 00000000 01000000 00000001  -xf#...@.
0018  10001111 11101011 11000000 10101000 00000001 00000001 11000000 10101000  ....
0020  00000001 01100101 00001011 00000000 11110100 11111111 00000000 00000000  -e....
0028  00000000 00000000 01000101 00000000 00000000 01011100 11101010 01101000  E...\h
```

Figure 13 Header in TTL exceeded

Table 6 ICMP Packet Fields Breakdown

IP Header 20 bytes	Type 1 byte	Code 1 byte	Checksum 2 bytes
-----------------------	----------------	----------------	---------------------

Unused 4 bytes	IP Header 20 bytes
ICMP Header 8 bytes	ICMP Data 64 bytes

8. How does your computer (the source) learn the IP address of a router along the path from a TTL exceeded packet?

tracert first sends an ICMP message with TTL=1. When the packet reaches the first router, it is dropped and TTL exceeded packet is transmitted back to the source. Then, ICMP message with TTL=2 is sent. One by one, the program learns the IP addresses of the routers along the path.

9. How many times is each router along the path probed by traceroute?

Each router is probed three times as seen in Figure 14.

2103 7.147512	212.156.201.61.stat...	192.168.1.101	ICMP	94 Time-to-live exceeded
Number 51190	192.168.1.101	23.185.0.2	ICMP	106 Echo (ping) request
2164 7.171729	212.156.201.61.stat...	192.168.1.101	ICMP	94 Time-to-live exceeded
2165 7.172152	212.156.201.61.stat...	192.168.1.101	ICMP	94 Time-to-live exceeded
2166 7.174683	192.168.1.101	23.185.0.2	ICMP	106 Echo (ping) request
2168 7.181687	212.156.201.61.stat...	192.168.1.101	ICMP	94 Time-to-live exceeded
2556 8.217380	192.168.1.101	23.185.0.2	ICMP	106 Echo (ping) request
2595 8.310915	81.212.2.41.static...	192.168.1.101	ICMP	134 Time-to-live exceeded
2596 8.314633	192.168.1.101	23.185.0.2	ICMP	106 Echo (ping) request
2628 8.374889	81.212.2.41.static...	192.168.1.101	ICMP	134 Time-to-live exceeded
2629 8.376099	81.212.2.41.static...	192.168.1.101	ICMP	134 Time-to-live exceeded
2630 8.378782	192.168.1.101	23.185.0.2	ICMP	106 Echo (ping) request
2640 8.405684	81.212.2.41.static...	192.168.1.101	ICMP	134 Time-to-live exceeded
4324 13.953332	192.168.1.101	23.185.0.2	ICMP	106 Echo (ping) request

Figure 14 tracert probes

10. Within the tracert measurements, is there a link whose delay is significantly longer than others? The echo request packets sent by traceroute are probing successively more distant routers along the path. You can look at these packets and see how they differ when they elicit responses from different routers.

Yes, 4-5-6-7 are significantly longer than others.

Part 2: Routing Implementation

NaiveFloodingAlgorithm:

11. See that this algorithm succeeds in topology 1. What does the total communication cost represent, and why is it different from the path cost?

In order to choose the path, the packets are sent through every possible route from source to destination. Total communication cost represents the accumulated costs of all tried paths while the path cost represents the cost of the final packet.

12. See that this algorithm fails in topology 2, and the simulator notes that the protocol does not converge. Why is this the case.

As the Naïve Flooding Algorithm chooses to investigate every possible path, the algorithm goes into an infinite loop and cannot terminate.

FloodingAlgorithm:

Implementation

I added a Boolean field `-routed-` to keep track of whether the neighbor selection algorithm is executed in this node. Since each algorithm is copied as a new instance to each node in the topology (see `Main` – line 44), `routed` is initialized as `false`, and set `true` at the end of the algorithm. The algorithm wraps the list streaming in `NaiveFlooding` so that it is executed only if the node is not routed before.

Table 7 FloodingAlgorithm Implementation

```
private boolean routed = false;
@Override
public List<NeighborInfo> selectNeighbors(String origin, String destination,
String previousHop, List<NeighborInfo> neighbors) {
    List<NeighborInfo> chosen = new ArrayList<>();
    if (!routed) {
        chosen = neighbors.stream()
            // Make sure that we do not route back to the previous hop.
            .filter(n -> !n.address.equals(previousHop))
            .collect(Collectors.toList());
        // once the algorithm is called on the node, set router true to avoid
        further executions
        routed = true;
    }
    return chosen;
}
```

13. Consider the path taken by the packet in topology 2 with this algorithm. Is this what you have expected? Why or why not?

Yes, because the edges between nodes and the nodes in chosen list are evaluated the justification can be seen in Table 8.

Table 8 Execution of the flooding algorithm in topology 2

Node	flag	chosen	cost
1	false → true	{2, 3}	+ 5, + 2
3	false → true	{4, 2}	+ 7, + 1
2	false → true	{1, 4}	+ 5, + 8
			→ 28

NaiveMinimumCostAlgorithm:*Implementation*

Similar to the NaiveFlooding I first find the appropriate next hops and then find the minimum cost edge and return it. If there are no next hops, I return an empty list as seen in Table 9.

Table 9 NaiveMinimumCostAlgorithm Implementation

```
public List<NeighborInfo> selectNeighbors(String origin, String destination,
String previousHop, List<NeighborInfo> neighbors) {
    List<NeighborInfo> chosen = neighbors.stream()
        // Make sure that we do not route back to the previous hop.
        .filter(n -> !n.address.equals(previousHop))
        .collect(Collectors.toList());
    NeighborInfo minNeighbor;
    // if the chosen list is non empty, pick minimum element
    if (!chosen.isEmpty()) {
        minNeighbor = chosen.stream().min(Comparator.comparingInt(n ->
n.cost)).get();
        chosen.clear();
        chosen.add(minNeighbor);
    }
    return chosen;
}
```

14. If you implemented the algorithm as specified, it should succeed in topology 1 but fail in topology 2. Why does it fail in topology 2?

As the algorithm chooses the minimum cost edge at every hop, it enters an infinite loop of the path $1 \rightarrow 3 \rightarrow 2 \rightarrow 1$.

MinimumCostAlgorithm:*Implementation*

I keep track of the `exclusionSet`. When choosing the next potential hops, I filter out the previous hop and the neighbors that are already in the `exclusionSet`. Then, if there are no appropriate next hops satisfying these criteria, I choose a random neighbor; otherwise, I choose the minimum of those neighbors. I then add the previous hop and the chosen neighbor to the `exclusionSet` and return a list containing the chosen neighbor.

Table 10 MinimumCostAlgorithm Implementation

```
Set<String> exclusionSet = new HashSet<>();

@Override
public List<NeighborInfo> selectNeighbors(String origin, String destination,
String previousHop,
                                List<NeighborInfo> neighbors) {
    List<NeighborInfo> chosen = neighbors.stream()
        // Make sure that we do not route back to the previous hop.
        .filter(n -> !n.address.equals(previousHop) &&
!exclusionSet.contains(n.address))
        .collect(Collectors.toList());
    NeighborInfo minNeighbor;
    // if every neighbor is in the exclusion set, pick a random neighbor
    if (chosen.isEmpty()) {
        // if all neighbors are in the exclusion set, select randomly
        int randomInd = rand.nextInt(neighbors.size());
        minNeighbor = neighbors.get(randomInd);
    }
}
```



```

    } else {
        // chose the minimum neighbor according to their costs
        minNeighbor = chosen.stream().min(Comparator.comparingInt(n ->
n.cost)).get();
    }
    // empty chosen array list
    chosen.clear();
    chosen.add(minNeighbor);
    // add the next and previous hops to the exclusion set
    exclusionSet.add(minNeighbor.address);
    exclusionSet.add(previousHop);
    return chosen;
}

```

15. List the nodes in the exclusion set of each node at the end of the simulation of network topology 2.

Table 11 Exclusion Set of topology 2

Node	Exclusion Set
1	{2, 3}
2	{1, 3}
3	{1, 2, 4}
4	-