

COMP 416: Computer Networks - Project 3

Network Layer Analysis and DV Routing Simulator

Hasan Can Aslan - 60453

Part 1. Network Layer Analysis

Part 1A. Network Interface Analysis

1. We provide 2 parameters to netstat command. First **-a** parameter displays all active TCP connections and the TCP and UDP ports on which the computer is listening, **-i** parameter displays network interface status. So **netstat -ai** command shows the state of the network interfaces and their TCP connections.

Name	Mtu	Network	Address	Ipkts	Ierrs	Opkts	Oerrs	Coll	
lo0	16384	<Link#1>		1042147	0	1042147	0	0	
lo0	16384	127		1042147	-	1042147	-	-	
lo0	16384	localhost	localhost	1042147	-	1042147	-	-	llw0 1500 <Link#13> 1a:43:3d:51:25:33 0 0 0 0 0
			224.0.0.251						llw0 1500 fe09::1843: fe80:d:1843:3dff 0 - 0 - -
			all-systems.mcast.net						ff01::1 (refs: 1)
			::1	1042147	-	1042147	-	-	ff02::1 (refs: 1)
			ff02::fb	(refs: 1)					ff02::1:ff51:2533 (refs: 1)
			ff02::2:ff25:e58e	(refs: 1)					ff02::fb (refs: 1)
			ff01::1	(refs: 1)					ff02::1:fffd:2354 (refs: 3)
lo0	16384	hasan-macbo	ff02::1	(refs: 1)				ff02::1:ff2f:6d75 (refs: 4)	
			ff02::1:ff00:1	(refs: 1)				ff02::2:ff8a:dd10 (refs: 11)	
			ff02::1	(refs: 1)				ff02::1:ffbd:2c9a (refs: 3)	
			ff02::fb	(refs: 1)				utun0 1380 <Link#14> fe80::e457:d006 0 0 20 0 0	
			ff02::2:ff25:e58e	(refs: 1)				utun0 1380 hasan-macbo fe80::e457:d006 0 - 20 - -	
			ff01::1	(refs: 1)				ff01::1 (refs: 1)	
			ff02::1	(refs: 1)				ff02::2:ff8a:dd10 (refs: 1)	
gif0*	1280	<Link#2>	ff02::1:ff00:1	(refs: 1)				ff02::1 (refs: 1)	
			0	0	0	0	0	utun1 2000 <Link#15> fe80::f:60e1:e539 0 0 20 0 0	
			<Link#3>	0	0	0	0	0	utun1 2000 hasan-macbo fe80::f:60e1:e539 0 - 20 - -
			<Link#4>	0	0	0	0	0	ff01::1 (refs: 1)
			f2:18:98:62:13:01	0	0	0	0	0	ff02::2:ff8a:dd10 (refs: 1)
en0	1500	<Link#6>	f0:18:98:62:13:01	14531107	0	28312842	75214	0	ff02::1 (refs: 1)
en0	1500	hasan-macbo	fe80:6:1851:25c0:14531107	-	28312842	-	-	ff02::1 (refs: 1)	
en0	1500	192.168.1	ff02::fb	(refs: 1)				ff02::1:fffd:2c9a (refs: 3)	
			ff01::1	(refs: 1)				utun2 1380 <Link#16> fe80::f:60e1:e539 0 0 20 0 0	
			ff02::2:ff8a:dd10	(refs: 1)				utun2 1380 hasan-macbo fe80::f:60e1:e539 0 - 20 - -	
			ff02::1	(refs: 1)				ff01::1 (refs: 1)	
			ff02::1	(refs: 1)				ff02::2:ff8a:dd10 (refs: 1)	
			ff02::1:ff3a:78a4	(refs: 1)				ff02::1 (refs: 1)	
			192.168.1.13	14531107	-	28312842	-	-	ff02::1:ff3a:78a4 (refs: 1)
en1	1500	<Link#7>	239.255.255.250					ff02::1:ff3a:78a4 (refs: 1)	
			224.0.0.251					utun3 1380 <Link#17> fe80::11:97c5:f07 0 0 20 0 0	
			all-systems.mcast.net					utun3 1380 hasan-macbo fe80::11:97c5:f07 0 - 20 - -	
			82:dd:b9:a2:4c:01	0	0	0	0	0	ff01::1 (refs: 1)
			82:dd:b9:a2:4c:00	0	0	0	0	0	ff02::2:ff8a:dd10 (refs: 1)
			82:dd:b9:a2:4c:05	0	0	0	0	0	ff02::1 (refs: 1)
			82:dd:b9:a2:4c:04	0	0	0	0	0	ff02::1:ff27:e9b7 (refs: 1)
en2	1500	<Link#8>	82:dd:b9:a2:4c:01	0	0	0	0	en5 1500 <Link#4> ac:de:48:00:11:22 41518 0 41198 66 0	
en3	1500	<Link#9>	82:dd:b9:a2:4c:05	0	0	0	0	en5 1500 fe09::aede: fe80:4:aede:48ff 41518 0 41198 - -	
en4	1500	<Link#10>	82:dd:b9:a2:4c:04	0	0	0	0	ff01::1 (refs: 1)	
bridg	1500	<Link#11>	82:dd:b9:a2:4c:01	0	0	0	0	ff02::1 (refs: 1)	
awd10	1500	<Link#12>	1a:43:3d:51:25:33	14761	0	14426	0	ff02::2:ff8a:dd10 (refs: 1)	
awd10	1500	fe09::1843:	fe80:c:1843:3dff	14761	0	14426	-	-	ff02::1:ff00:1122 (refs: 1)
en5	1500	<Link#4>	ff01::1	(refs: 1)					ff01::1 (refs: 1)
			ff02::1	(refs: 1)				ff02::1 (refs: 1)	
			ff02::1:ff51:2533	(refs: 1)				ff02::2:ff8a:dd10 (refs: 1)	
			ff02::2:ff8a:dd10	(refs: 1)				ff02::1:ff00:1122 (refs: 1)	
			ff02::fb	(refs: 1)					

2. Destination column: Shows the destination network.

Gateway column: Indicates the router through which packets are forwarded.

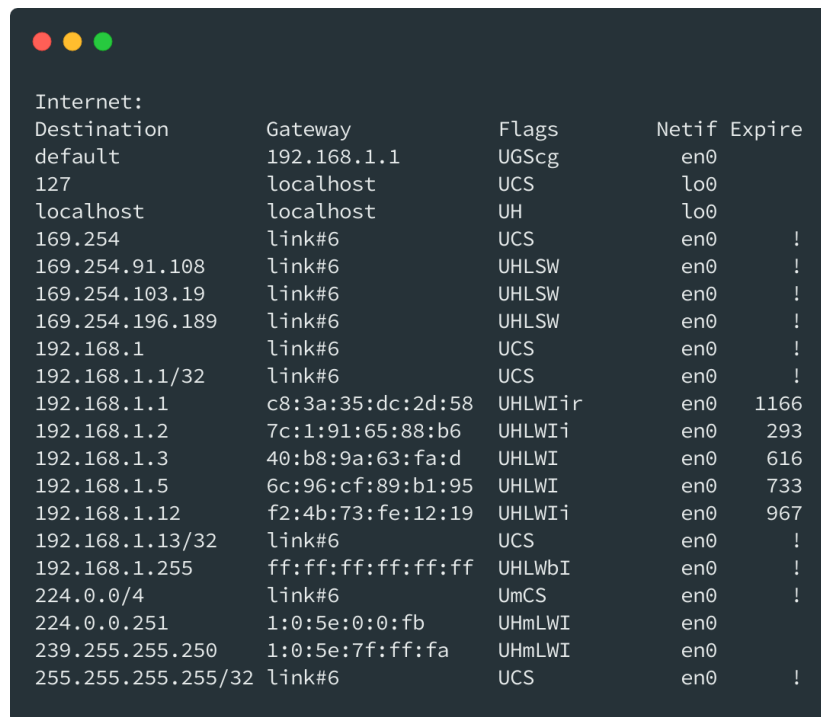
Flags column: The U flag indicates that the route is up route is valid.

The G flag indicates that the route is to a gateway.

The S flag indicates route added with the route command.

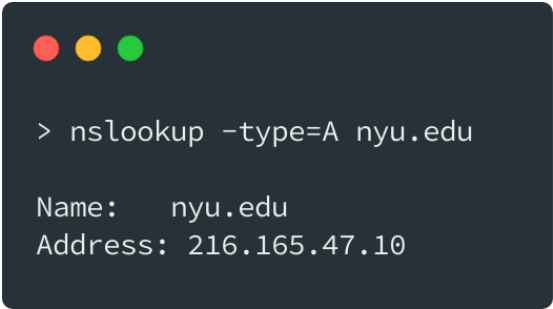
The H flag indicates route is to a host rather than to a network, where the destination address is a complete address.

Netif column: Indicates the interface on the local host that is the source endpoint of the transmission.



Internet:				
Destination	Gateway	Flags	Netif	Expire
default	192.168.1.1	UGScg	en0	
127	localhost	UCS	lo0	
localhost	localhost	UH	lo0	
169.254	link#6	UCS	en0	!
169.254.91.108	link#6	UHLSW	en0	!
169.254.103.19	link#6	UHLSW	en0	!
169.254.196.189	link#6	UHLSW	en0	!
192.168.1	link#6	UCS	en0	!
192.168.1.1/32	link#6	UCS	en0	!
192.168.1.1	c8:3a:35:dc:2d:58	UHLWIir	en0	1166
192.168.1.2	7c:1:91:65:88:b6	UHLWII	en0	293
192.168.1.3	40:b8:9a:63:fa:d	UHLWI	en0	616
192.168.1.5	6c:96:cf:89:b1:95	UHLWI	en0	733
192.168.1.12	f2:4b:73:fe:12:19	UHLWII	en0	967
192.168.1.13/32	link#6	UCS	en0	!
192.168.1.255	ff:ff:ff:ff:ff:ff	UHLWbI	en0	!
224.0.0/4	link#6	UmCS	en0	!
224.0.0.251	1:0:5e:0:0:fb	UHmLWI	en0	
239.255.255.250	1:0:5e:7f:ff:fa	UHmLWI	en0	
255.255.255.255/32	link#6	UCS	en0	!

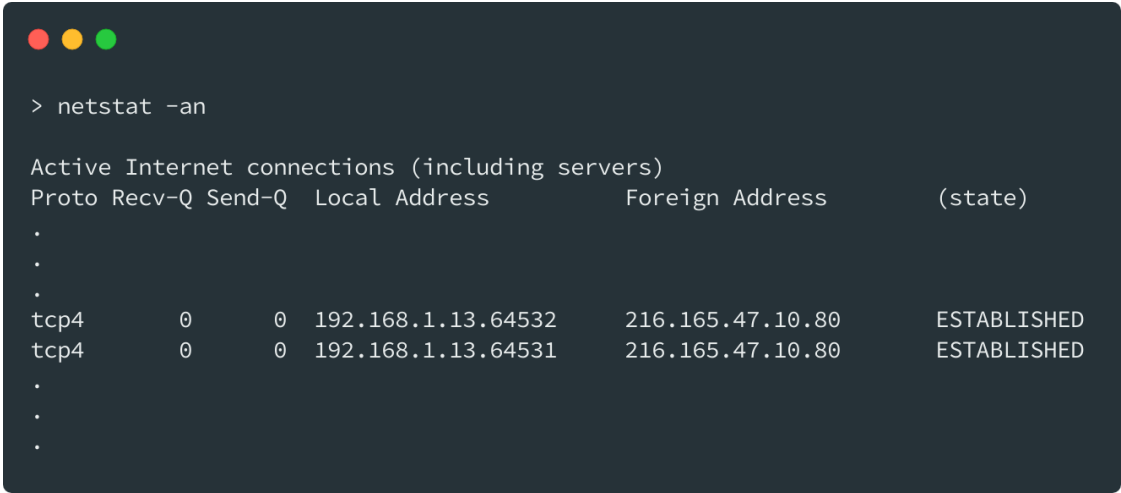
3. My client using ports 64532 and 64531 to connect nyu.edu. First, I get the IP address of the host, using `nslookup`.

A terminal window with a dark background and three colored window control buttons (red, yellow, green) at the top left. The terminal shows the command `> nslookup -type=A nyu.edu` and its output: `Name: nyu.edu` and `Address: 216.165.47.10`.

```
> nslookup -type=A nyu.edu

Name: nyu.edu
Address: 216.165.47.10
```

Then using `netstat -P tcp` command, we can find nyu.edu's IP address in the list.

A terminal window with a dark background and three colored window control buttons (red, yellow, green) at the top left. The terminal shows the command `> netstat -an` and its output, which includes a table of active internet connections. The table has columns for Proto, Recv-Q, Send-Q, Local Address, Foreign Address, and (state). Two entries for tcp4 are shown, both with state ESTABLISHED, connecting to 216.165.47.10.80.

```
> netstat -an

Active Internet connections (including servers)
Proto Recv-Q Send-Q Local Address           Foreign Address         (state)
.
.
.
tcp4    0      0 192.168.1.13.64532     216.165.47.10.80      ESTABLISHED
tcp4    0      0 192.168.1.13.64531     216.165.47.10.80      ESTABLISHED
.
.
.
```

Part 1B. ICMP Analysis

4. TTL, time to live is the number of hops or amount of time that a packet exists inside a network. When TTL exceeded, packet is discarded. Time to live is important for caching information. It determines how long the data can be cached and when the information should be updated. We can find TTL under the network layer, for example under the Internet Protocol.

5. ICMP protocol is designed to communicate between network layers. However, port numbers are used to communicate between application layers, so ICMP has neither destination nor source port number.

6. Minimum TTL is 1.

```
> Frame 126: 182 bytes on wire (1456 bits), 182 bytes captured (1456 bits) on interface en0, id 0
> Ethernet II, Src: TendaTec_dc:2d:58 (c8:3a:35:dc:2d:58), Dst: Apple_62:13:01 (f0:18:98:62:13:01)
> Internet Protocol Version 4, Src: 64.125.26.172, Dst: 192.168.1.13
< Internet Control Message Protocol
  Type: 11 (Time-to-live exceeded)
  Code: 0 (Time to live exceeded in transit)
  Checksum: 0xbe96 [correct]
  [Checksum Status: Good]
  Unused: 00000000
< Internet Protocol Version 4, Src: 192.168.1.13, Dst: 216.165.47.10
  0100 .... = Version: 4
  .... 0101 = Header Length: 20 bytes (5)
  > Differentiated Services Field: 0x00 (DSCP: CS0, ECN: Not-ECT)
  Total Length: 52
  Identification: 0x9507 (38151)
  > Flags: 0x00
  Fragment Offset: 0
  > Time to Live: 1
  Protocol: UDP (17)
  Header Checksum: 0x5b4d [validation disabled]
  [Header checksum status: Unverified]
  Source Address: 192.168.1.13
  Destination Address: 216.165.47.10
  > User Datagram Protocol, Src Port: 38120, Dst Port: 33465
  > ICMP Multi-Part Extensions
  > Data (24 bytes)
```

7. My computer sends first a packet with $TTL = 1$, the second packet has $TTL = 2$ so and so on. A router will decrement the packet's TTL when the packet passes through. If the packet arrives the router with TTL value equals to 1, then router sends an ICMP packet to the source. In this way we can count the router along the path to the destination.

8. For each router along the path, **traceroute** sends 3 probes.

```

tracert to nyu.edu (216.165.47.10), 64 hops max, 52 byte packets
 1 192.168.1.1 (192.168.1.1) 9.176 ms 2.175 ms 3.090 ms
 2 212.156.201.180.static.turktelekom.com.tr (212.156.201.180) 21.905 ms 19.413 ms
19.555 ms
 3 81.212.71.241.static.turktelekom.com.tr (81.212.71.241) 13.059 ms 13.178 ms 13.232
ms
 4 01-adana-xrs-t2-1---31-hatay-t3-1.statik.turktelekom.com.tr (195.175.167.56) 19.527
ms 17.302 ms 17.085 ms
 5 35-izmir-xrs-t2-1---01-adana-xrs-t2-1.statik.turktelekom.com.tr (195.175.166.0)
28.367 ms 30.671 ms 31.680 ms
 6 * 35-ebgp-izmir-sr12e-k---35-izmir-xrs-t2-1.statik.turktelekom.com.tr (81.212.30.5)
31.320 ms 29.615 ms
 7 301-fra-col-1---06-ebgp-ulus-sr12e-k.statik.turktelekom.com.tr (212.156.101.32)
73.037 ms 69.714 ms 68.996 ms
 8 62.157.248.1 (62.157.248.1) 868.168 ms * *
 9 pd900cb02.dip0.t-ipconnect.de (217.0.203.2) 85.037 ms 79.513 ms *
10 new-york-un.ear3.newyork1.level3.net (4.28.130.118) 160.253 ms
80.150.170.214 (80.150.170.214) 91.782 ms
new-york-un.ear3.newyork1.level3.net (4.28.130.118) 153.203 ms
11 ae12.cs1.fra6.de.eth.zayo.com (64.125.26.172) 150.142 ms 146.613 ms
dmzgw-b-p2p-extgwc.net.nyu.edu (128.122.254.75) 160.302 ms
12 nyugwa-ntp-dmzgw-vl3082.net.nyu.edu (128.122.254.110) 157.920 ms
ae2.cs1.ams17.nl.eth.zayo.com (64.125.29.59) 167.329 ms 167.260 ms
13 nyufw-outside-ngfw-vl3080.net.nyu.edu (128.122.254.116) 157.550 ms 160.002 ms *
14 * * *
15 * * wsqdcgwa-vl902.net.nyu.edu (128.122.1.38) 164.298 ms
16 ae5.cs3.lga5.us.eth.zayo.com (64.125.29.126) 148.268 ms * *
17 * * *
18 * 209.66.118.177.idia-282827-zyo.zip.zayo.com (209.66.118.177) 173.936 ms 167.051 ms
19 * * *
20 * * *
21 nyufw-outside-ngfw-vl3080.net.nyu.edu (128.122.254.116) 163.018 ms * 154.461 ms
22 * * *
23 wsqdcgwa-vl902.net.nyu.edu (128.122.1.38) 157.344 ms * *

```

9. I reach to the destination much faster, at 11 hops, when tracing 18.31.0.200.

```

tracert to 18.31.0.200 (18.31.0.200), 64 hops max, 52 byte packets
 1 192.168.1.1 (192.168.1.1) 7.753 ms 21.956 ms 13.085 ms
 2 212.156.201.180.static.turktelekom.com.tr (212.156.201.180) 240.803 ms 148.522 ms
238.018 ms
 3 81.212.71.241.static.turktelekom.com.tr (81.212.71.241) 243.378 ms 67.492 ms 24.846
ms
 4 01-adana-xrs-t2-2---31-hatay-t3-1.statik.turktelekom.com.tr (212.156.119.103) 21.969
ms 28.330 ms 58.632 ms
 5 06-ulus-xrs-t2-2---01-adana-xrs-t2-2.statik.turktelekom.com.tr (81.212.216.58) 34.451
ms 103.286 ms 115.133 ms
 6 06-ebgp-ulus-sr12e-k---06-ulus-xrs-t2-2.statik.turktelekom.com.tr (81.212.217.121)
154.806 ms 149.470 ms 129.796 ms
 7 305-vie-col-3---06-ebgp-ulus-sr12e-k.statik.turktelekom.com.tr (212.156.140.170)
217.891 ms 162.312 ms *
 8 80.157.206.133 (80.157.206.133) 275.066 ms
185.84.16.29 (185.84.16.29) 213.191 ms 266.284 ms
 9 80.156.160.134 (80.156.160.134) 152.580 ms
4.68.63.201 (4.68.63.201) 180.306 ms
pd9ef317d.dip0.t-ipconnect.de (217.239.49.125) 147.381 ms
10 80.156.160.134 (80.156.160.134) 198.457 ms
ae-2-3.bear1.boston1.level3.net (4.69.159.249) 222.319 ms
80.156.160.134 (80.156.160.134) 145.162 ms
11 massachuset.bear1.boston1.level3.net (4.53.48.98) 236.803 ms 283.898 ms
ae-2-3.bear1.boston1.level3.net (4.69.159.249) 474.872 ms


```

10. Routing Blackhole is the routing the all network traffic into blackhole and lost it.

We can use such a system to prevent attacks to our networks like DDoS attack.

However, normal traffic is also affected such prevention system.

11. We can arrange the output fields by using the -o flag. I use following command:



```
sudo mtr -o "LSDR NBAW JMXI" nyu.edu
```

Host	Packets				Pings							
	Loss%	Snt	Drop	Rcv	Last	Best	Avg	Wrst	Jttr	Javg	Jmax	Jint
1. 192.168.1.1	3.4%	89	3	86	12.5	2.0	23.7	317.8	7.4	17.6	295.	156.
2. 212.156.201.180.static.turktelekom.com.tr	1.1%	89	1	87	19.6	15.2	50.2	355.5	21.2	23.7	183.	307.
3. 81.212.71.241.static.turktelekom.com.tr	0.0%	88	0	88	13.3	13.1	37.5	301.5	4.6	17.7	144.	188.
4. 01-adana-xrs-t2-1---31-hatay-t3-1.statik.turktelekom.com.tr	0.0%	88	0	88	18.4	15.3	37.5	276.3	7.6	18.7	164.	162.
5. 35-izmir-xrs-t2-1---01-adana-xrs-t2-1.statik.turktelekom.com.tr	0.0%	88	0	88	33.0	29.7	49.1	253.2	6.4	18.2	183.	129.
6. 35-ebgp-izmir-sr12e-k---35-izmir-xrs-t2-1.statik.turktelekom.com.tr	2.3%	88	2	86	35.9	28.9	46.9	210.7	16.6	16.4	167.	218.
7. 301-fra-col-1---06-ebgp-ulus-sr12e-k.statik.turktelekom.com.tr	1.1%	88	1	87	73.4	68.9	85.8	403.2	2.9	20.0	330.	179.
8. 62.67.19.245	96.6%	88	84	3	80.8	77.3	79.2	80.8	3.5	1.8	3.5	5.4
9. ae-1-3501.ear3.newyork1.level3.net	84.1%	88	74	14	164.3	156.6	176.4	354.0	6.5	33.1	197.	252.
10. new-york-un.ear3.newyork1.level3.net	1.1%	88	1	87	160.1	153.3	191.9	549.2	22.5	25.2	302.	245.
11. dmzgw-b-p2p-extgwc.net.nyu.edu	81.6%	88	71	16	157.9	157.0	215.1	479.1	22.9	52.3	322.	443.
12. nyugwa-ntp-dmzgw-b-vl3082.net.nyu.edu	1.1%	88	1	87	159.5	155.5	194.7	487.4	1.1	34.8	250.	460.
13. nyufw-outside-ngfw-vl3080.net.nyu.edu	79.3%	88	69	18	158.3	155.7	169.2	208.5	50.2	17.3	50.2	208.
14. (waiting for reply)												
15. wsqdcgw-vl902.net.nyu.edu	95.4%	88	83	4	185.2	159.3	271.1	468.2	86.6	148.	308.	542.
16. (waiting for reply)												

These fields are described in man page of **mtr** as follows:



```

+---+
|L | Loss ratio      |
+---+
|D | Dropped packets |
+---+
|R | Received packets |
+---+
|S | Sent Packets     |
+---+
|N | Newest RTT(ms)   |
+---+
|B | Min/Best RTT(ms) |
+---+
|A | Average RTT(ms)  |
+---+
|W | Max/Worst RTT(ms)|
+---+
|V | Standard Deviation|
+---+
|G | Geometric Mean   |
+---+
|J | Current Jitter    |
+---+
|M | Jitter Mean/Avg.  |
+---+
|X | Worst Jitter      |
+---+
|I | Interarrival Jitter|
+---+

```

12. mtr continuously updates times by polling a remote server and allows us to see how the latency and performance changes over time. On the other hand, **traceroute** allows us to discover the pathway to a host. When we compare two Wireshark record, we can see that **mtr** continuously sends probes whilst **traceroute** sends predetermined number of probes.

Part 2. DV Routing Simulator

Part 2A. Implementation

When constructing Node class, I initialize id from static counter, get cost and neighbors from static arrays of `DVSimulator` class. `myDV` is initially equals to cost of that node. I initialize `bestPath` array based on following conditions: If nodes id or neighbor's id is equal to destination id, then use that id, otherwise get a random node from `randomNeighbor` method.

```
public Node() {
    this.id = count++;
    this.cost = DVSimulator.cost[this.id];
    this.neighbors = DVSimulator.neighbors[this.id];
    this.myDV = this.cost;

    for (int i = 0; i < DVSimulator.NUMNODES; i++) {
        if (i == this.id) {
            this.bestPath[i] = this.id;
        } else if (Node.contains(neighbors, i)) {
            this.bestPath[i] = i;
        } else {
            this.bestPath[i] = randomNeighbor();
        }
    }

    notifyNeighbors();
}
```


For the `notifyNeighbors` method, I create a new packet for each neighbor that created with current node `id` as source, neighbor id as destination and current node's `myDV` array as the dv. Then, I send the packet.

```
public void notifyNeighbors() {  
    for (int neighbor : neighbors) {  
        Packet packet = new Packet(this.id, neighbor, this.myDV);  
        DVSimulator.sendPacket(packet);  
    }  
}
```

Citation: I get the following method from <https://stackoverflow.com/a/34541755>. It is simple method for ease of use when searching a value in array.

```
public static boolean contains(final int[] arr, final int key) {  
    return Arrays.stream(arr).anyMatch(i -> i == key);  
}
```

Part 2B. Optional (Bonus)

I compare for each value in the DV received from neighbor as **new_cost** and current DV as **current_cost**. If there is a cheaper path, I update **myDV** with corresponding cost and update **bestPath** with corresponding neighbor's id. If there is such a change in current Node's DV I update **hasDVChanged** value as true. If **hasDVChanged** value is true, then I **notifyNeighbors** and increment **numUpdates**. By doing that, I increment and notify neighbors just once for each packet.

```

public void updateDV(Packet p) {
    int neighbor_id = p.getSource();
    neighborDV[neighbor_id] = p.getDV();
    boolean hasDVChanged = false;

    for (int i = 0; i < myDV.length; i++) {
        int new_cost = cost[neighbor_id] + neighborDV[neighbor_id][i];
        int current_cost = myDV[i];

        if (new_cost < current_cost) {
            myDV[i] = new_cost;
            bestPath[i] = neighbor_id;
            hasDVChanged = true;
        }
    }

    if (hasDVChanged) {
        notifyNeighbors();
        numUpdates++;
    }
}

```