Programming Assignment #4: Distance Vector Routing

Logistics

- PA discussion needs to be done similar to PA#3 (See KLMS notice for details).
    - Submit 3 questions by 2021.05.16. 10:30 AM, KST (Sunday)
    - Write answers by 2021.05.18. 10:30 AM, KST (Tuesday)
    - Write feedbacks by 2021.05.20. 10:30 AM, KST (Thursday)
    - For who should write answers and feedbacks for whom, go to KLMS
- Due of PA #4: **2021.05.27. 10:30 AM, KST**

Overview

In PA #2 and PA #2 you have implemented parts of TCP in KENSv3 over unreliable and reliable channels. In PA #4 you will use KENSv3 over a reliable channel.

In PA #4 you will implement a simplified version of RIPv1, once one of the most popular routing protocols in the Internet. It is a distance vector routing protocol of which link cost is simply a hop count. It runs on top of UDP and uses a well-known port 520. You must implement UDP. You have already implemented TCP in PA #2 and PA #3. Implementing UDP should be a breeze. Below is the UDP packet header format from RFC 768. The source and destination ports are the same as in TCP. The length field is the length of the UDP header and the UDP data in bytes. The checksum is computed over the header and the data. When the data length is in odd bytes, then a pad byte of 0 is appended at the end just for the computation. The padded byte is not transmitted. As in the case of the TCP checksum computation, 12-byte pseudo-header is included in the checksum computation.

```
 0      7 8     15 16    23 24    31
+--------+--------+--------+--------+
|     Source      |   Destination   |
|      Port       |      Port       |
+--------+--------+--------+--------+
|                 |                 |
|     Length      |    Checksum     |
+--------+--------+--------+--------+
|
|          data octets ...
+---------------- ...

     User Datagram Header Format
```

4-1. Routing Information Protocol Version 1 (RIPv1)

The main task of this assignment is to implement a simplified version of RIPv1 in KENSv3. RFC1058 includes sections on *distance vector algorithms*, *specifications for the protocol*, and *control functions*. You will implement only the very basic parts of each section. The main difference between RIPv1 and the theoretic Bellman-Ford algorithm is the link cost. RIPv1 uses a hop count, while the latter arbitrary values. In the first part of this assignment, you will implement a simple case of RIPv1, where the link cost is uniformly set to 1.

RIPv1 uses the following format for its messages.

```
 0                   1                   2                   3 3
 0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
| command (1)   | version (1)   |       must be zero (2)        |
+---------------+---------------+-------------------------------+
| address family identifier (2) |       must be zero (2)        |
+-------------------------------+-------------------------------+
|                         IP address (4)                        |
+---------------------------------------------------------------+
|                        must be zero (4)                       |
+---------------------------------------------------------------+
|                        must be zero (4)                       |
+---------------------------------------------------------------+
|                          metric (4)                           |
+---------------------------------------------------------------+
                                .
                                .
```

For the command field, you will implement only the following two types:

(1) 1 - request: a request for the responding system to send all or part of its routing table. It is broadcast to 255.255.255.255 only once when the routing protocol starts.
(2) 2 - response: a message containing all or part of the sender's routing table. This message is sent in response to a request or upon a timer expiration in this assignment. In the former case, the response is addressed to the requesting host. In the latter case, the response is broadcast to 255.255.255.255.

Of course, the version is always set to 1 in this assignment.

The address family identifier field must be set to 2 in this assignment, which means the IP address is the usual 4-byte Internet IP address in network order.

The metric field must contain a value between 1 and 15 for reachable destinations and 16 for unreachable ones.

Once initiated, a node must broadcast a request. The first request will have the address family identifier set to 0, IP address set to 0, and the metric to 16. Then the node starts a timer. Upon a timer expiration, the node will send out a response. When the node receives a request, it sends out its distance vector table. When the node receives a response, it updates its distance vector table. All the IP broadcast packets in this assignment reach only immediate neighbors, as they are not routed beyond in KENSv3. Recall that every network interface has an IP address. A node with multiple links must have as many IP addresses as the number of links. The IP address assignments to nodes are in `testrouting.cpp`.

The following packet captures via WireShark will help you understand the above mechanics of RIPv1 in detail.

Pcap Packet Capture Resources

Note: *We use host IP addresses in the address field of RFC. Yet the following packet capture pcap files use network numbers or subnet numbers for the destination field. Stick to host IP addresses.*

- A reference to RIPv1 packet format: pcap packet capture log of a basic route exchange between two RIP v1 routers [src]: https://wiki.wireshark.org/SampleCaptures?action=AttachFile&do=get&target=RIP_v1
- A RIPv1 router periodically flooding its database. Capture perspective from R1's 10.0.1.1 interface [src]: https://packetlife.net/media/captures/RIPv1.cap
- RIPv1 routes are being flooded on the R1-R2 link. R2's connection to 192.168.2.0/24 goes down, and the route is advertised as unreachable (metric 16) in packet #5. Capture perspective from R1's 10.0.1.1 interface [src]: https://packetlife.net/media/captures/RIPv1_subnet_down.cap

Section 4 of RFC1058 describes administrative controls. In this assignment, you should implement only one control function used for correctness of your protocol implementation: `ripQuery()`.

```
Size RoutingAssignment::ripQuery(const ipv4_t &ipv4);
```

The `ripQuery()` call receives an IP address from the application layer. It should return the cost to reach the IP address. In RIPv1, the cost is just hop-count.

4-2. Upgrade RIPv1 for Extra Credit (20%)

You have implemented RIPv1 for uniform link costs. Now for extra credit you will modify your implementation such that the link costs are integers in the rage of [1..20]. The upgraded RIPv1 implementation must have the following few modifications from the standard RIPv1:

- The metric for the distance vector routing algorithm is the sum of link costs, not the hop count.
- The cost for a link is in the range from 1 to 20.
- The maximum value for the metric is 300 (15 x 20). The numbers bigger than 300 mean unreachable.

To obtain link costs, you should use the `portCost()` method in RoutingAssignment.hpp.

```
Size RoutingAssignment::portCost(int port_num);
```

The portCost() call receives a port number from your RIP implementation. It should return the link cost for the port number.

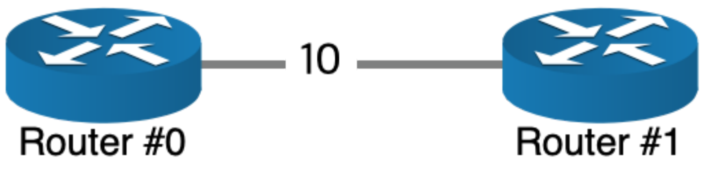
Build and Test

- Build: same as KENS (TCP)
- macOS/Linux (CLI): `./app/routing/routing`
- Windows (CLI): `app\routing\Debug\routing.exe`
- Others (GUI): `routing` target

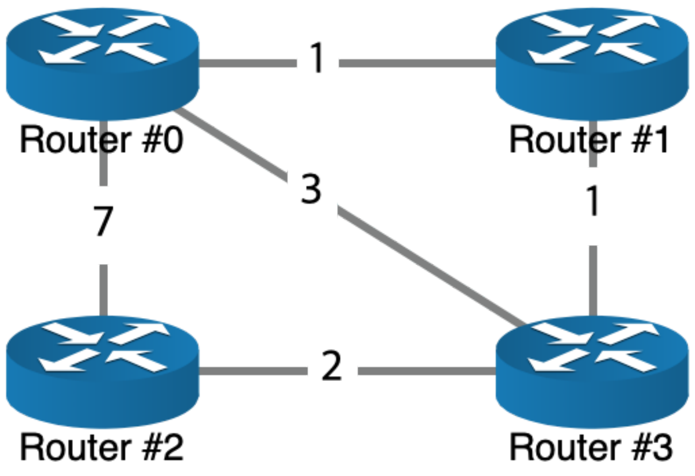
Testing

The testing script is in `testrouting.cpp`. It includes network topologies in the following figures.

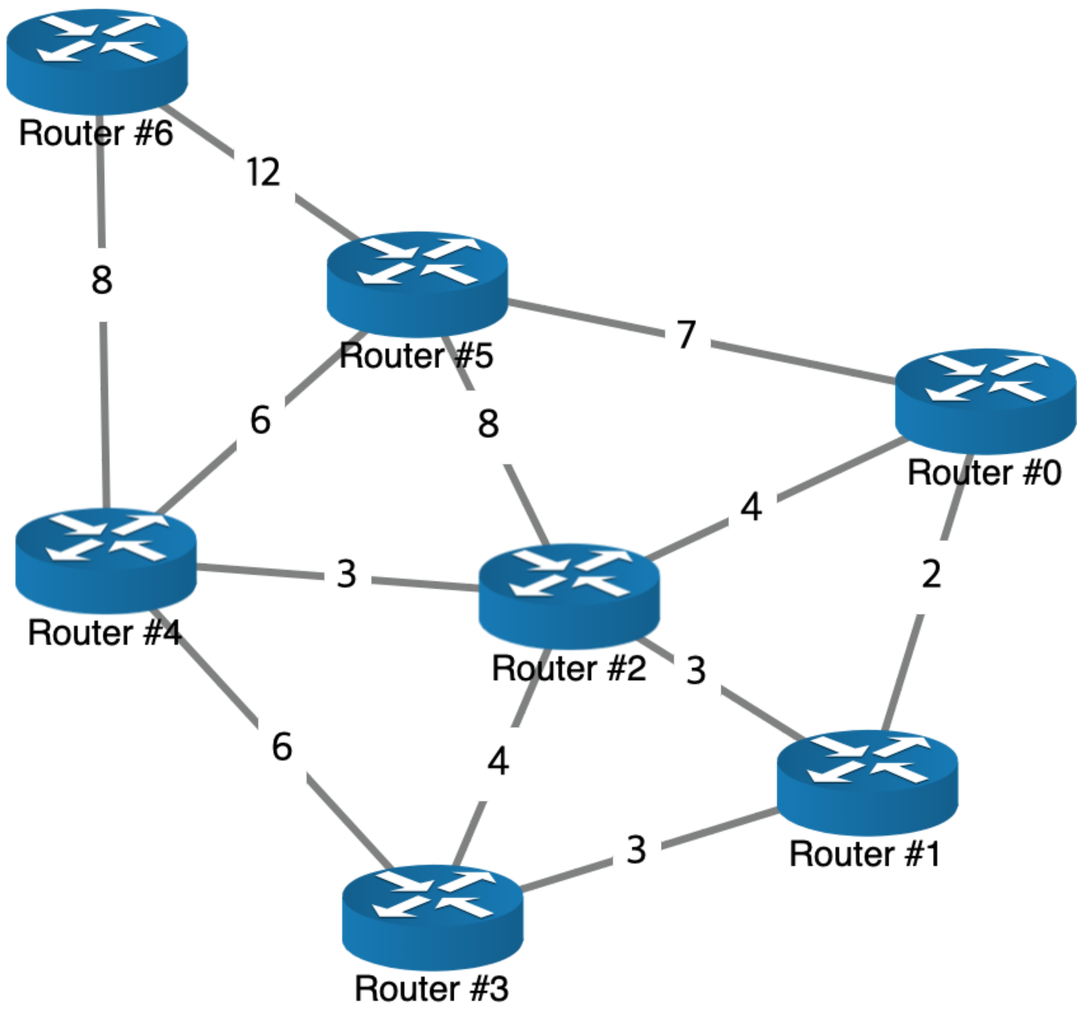
RoutingEnvRipTwoNodes / RoutingEnvCustomTwoNodes



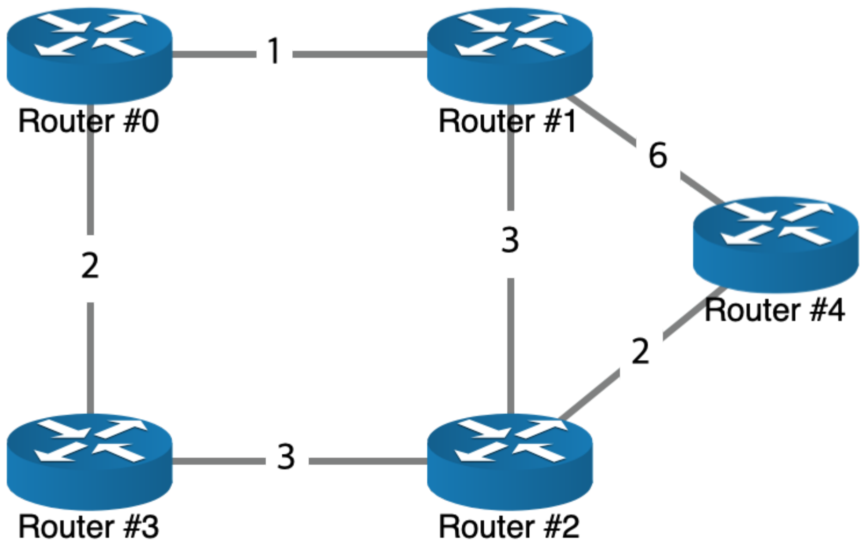
RoutingEnvRip1 / RoutingEnvCustom1



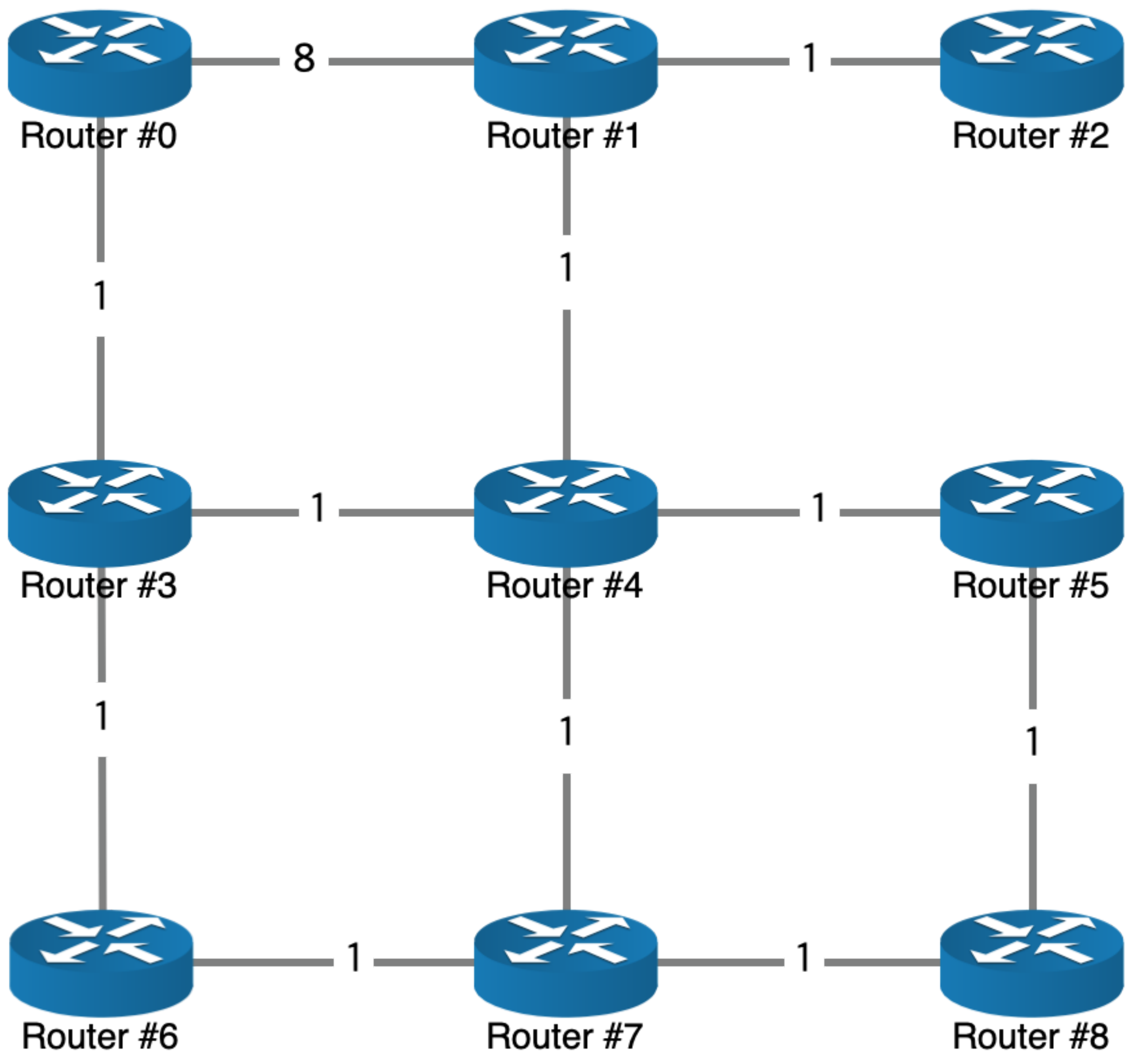
RoutingEnvRip2 / RoutingEnvCustom2



RoutingEnvRip3 / RoutingEnvCustom3



RoutingEnvRip4 / RoutingEnvCustom4



Submission

- You should submit only three files: **readme.txt**, **RoutingAssignment.cpp**, **RoutingAssignment.hpp**. RoutingAssignment.cpp and RoutingAssignment.hpp should contain your implementation.
- Upload the files on KLMS.
- There is no designated template for readme.txt; just briefly describe how you have progressed to complete this assignment. It does not have to be long and detailed. A brief summary will suffice.
- If you wish to use token(s) for this assignment, please state clearly **the number of tokens** to use **for each member** in readme.txt.