Programming Assignment #4: Distance Vector Routing

Logistics

- Due of PA #4: **2022.06.14. 10:30 AM, KST**

Overview

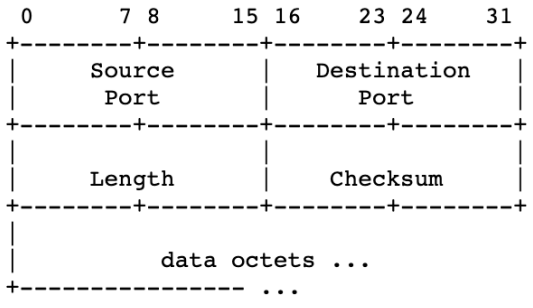
In PA #2 and PA #3 you have implemented parts of TCP in KENSv3 over unreliable and reliable channels. In PA #4 you will use KENSv3 over a reliable channel.

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 the hop count. It runs on top of UDP and uses a well-known port of 520.

For this project you must implement the skeletal UDP protocol KENSv3, but without the socket layer. All you have to implement is encapsulation and decapsulation of RIPv1 messages in the UDP format. You will not implement the socket layer for UDP. The RIPv1 code just encapsulates and decapsulates of RIPv1 messages in the UDP format.

Below is the UDP packet header format from RFC 768. The source and the destination port are fixed to 520 in this assignment. The length field is the length of the UDP header and the UDP data in bytes. The checksum computation in UDP is similar to TCP, using a pseudo-header of the source and destination IP addresses plus the protocol and the UDP length bits. For the sake of simplicity, **we omit the checksum computation in this programming assignment**. We will not check the correctness of checksum computation in your UDP code.

Just as in PA #2 and PA #3, you will use `sendPacket()` to send out a packet. When receiving a packet, instead of `TCPAssingment::packetArrived()`, you will use `RoutingAssignment::packetArrived()` in `RoutingAssignment.cpp`.

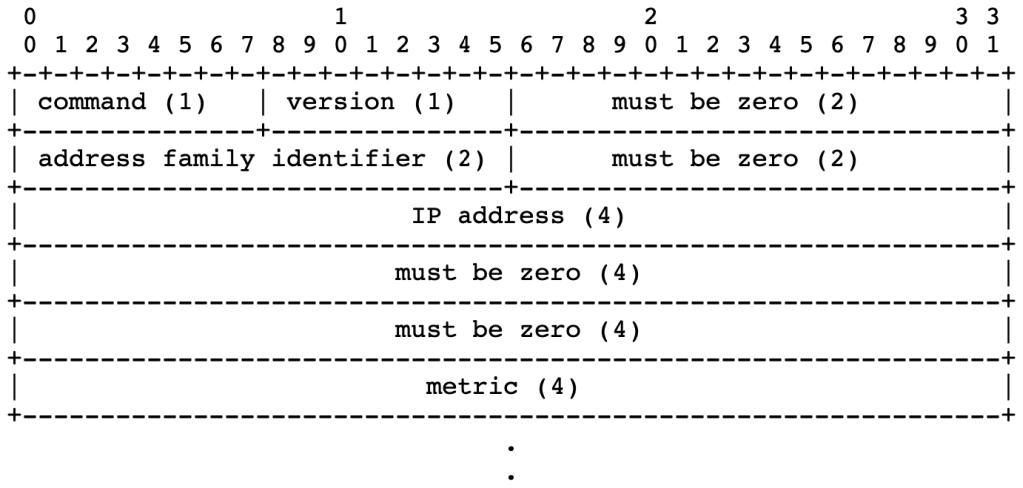


User Datagram Header Format

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 the hop count, while the latter allows 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.



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

(1) 1 - request: a request for the receiving 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 response 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 (except for initial requests), which means the IP address is the usual 4-byte Internet IP address in network order.

The metric field must contain a value between 0 and 15 for reachable destinations and 16 for unreachable ones.

Your RIPv1 implementation should work as follows.

(1) Given a topology during initialization, every node starts with itself in the distance vector table with a metric of 0. If a node has multiple interfaces, it enters all the IP addresses of the interfaces to the distance vector table.

(2) Once initiated, a node must broadcast a request and the node has multiple interfaces, your code must send as many IP packets as the number of interfaces. 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.
(3) Upon a timer expiration, the node broadcasts out a response with the current distance vector table.
(4) When the node receives a request (regardless of the RIPv1 field values), it sends out its distance vector table to the request sender.
(5) When the node receives a response, it updates its distance vector table.

Recall that all the IP broadcast packets in this assignment reach only immediate neighbors, as their IP addresses belong to the same subnet. 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 destination 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 RIPv1 routers [src]: https://wiki.wireshark.org/SampleCaptures?action=AttachFile&do=get&target=RIP_v1
- A RIPv1 router periodically flooding its database. The pcap log is captured at 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. The pcap log is captured at 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 to assess correctness of your protocol implementation: `ripQuery()`.

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

The `ripQuery()` call receives an IP address as a parameter and should return the cost to reach the IP address. In RIPv1, the cost is just the hop-count.

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`

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 `linkCost()` method in RoutingAssignment.hpp.

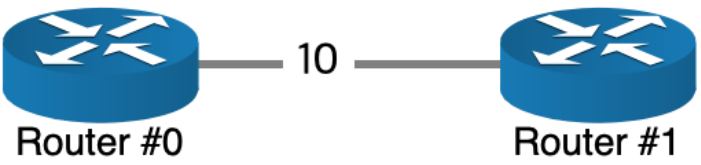
```
Size RoutingAssignment::linkCost(int port_num);
```

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

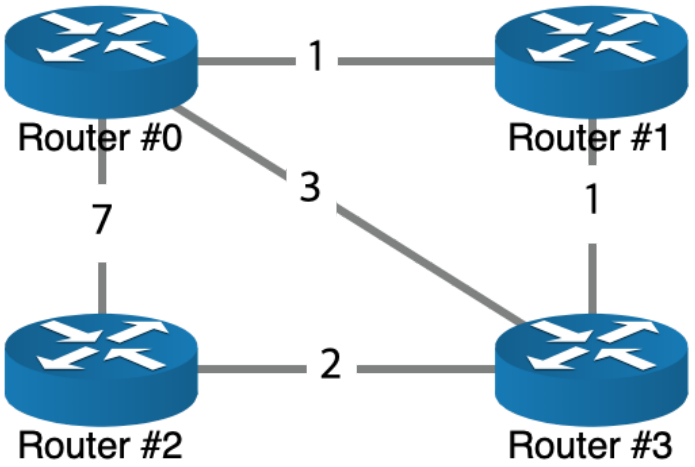
4.3 Testing

The testing script is in `testrouting.cpp`. It includes network topologies in the following figures. For RIPv1, ignore the link cost and use the hop count. Only for the extra credit work, use the link costs in the figures.

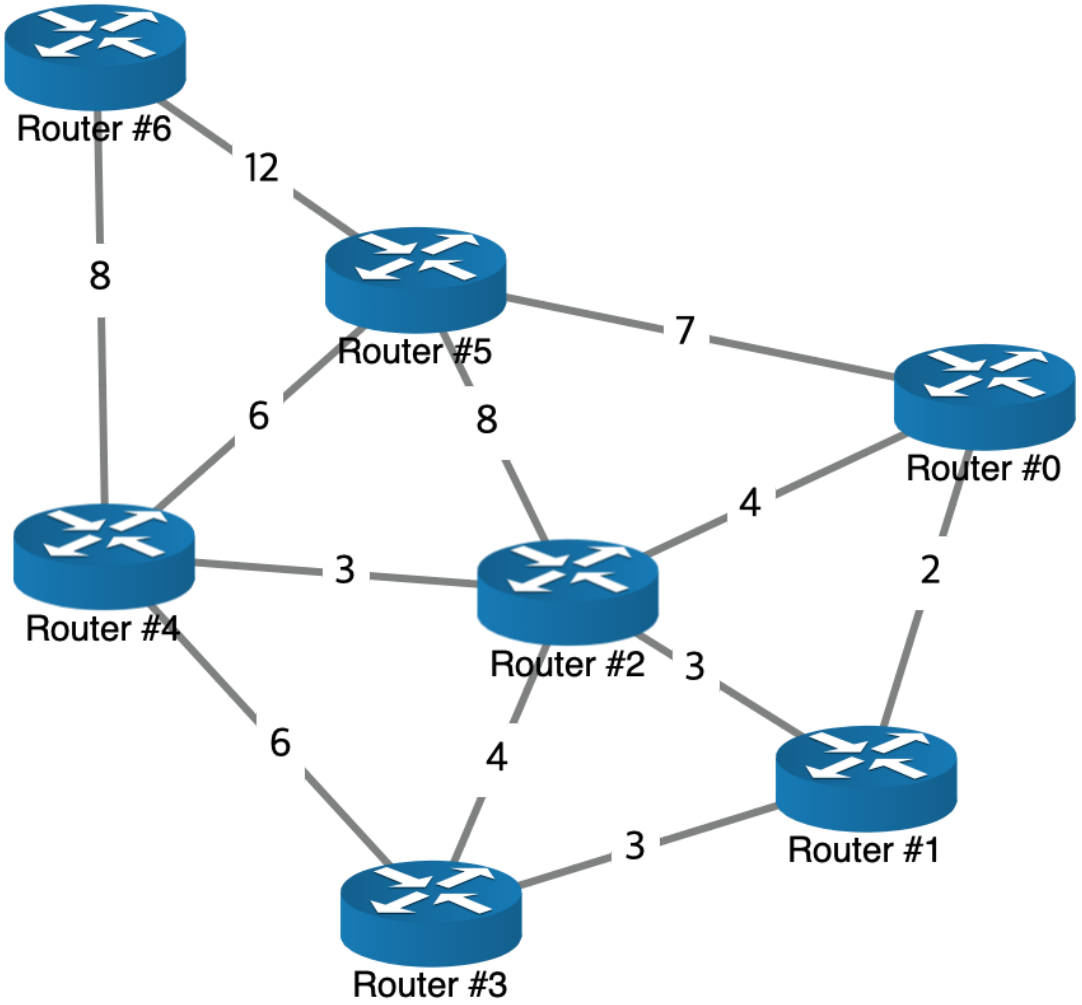
RoutingEnvRipTwoNodes / RoutingEnvCustomTwoNodes



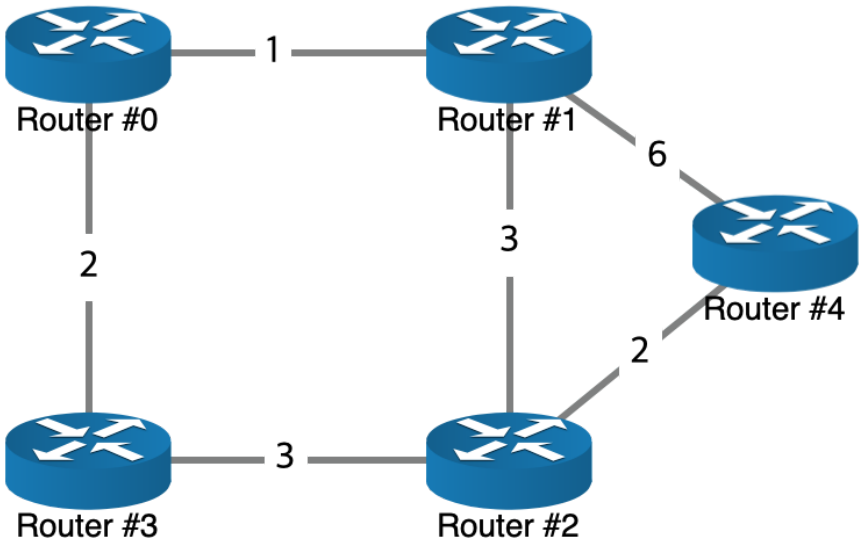
RoutingEnvRip1 / RoutingEnvCustom1



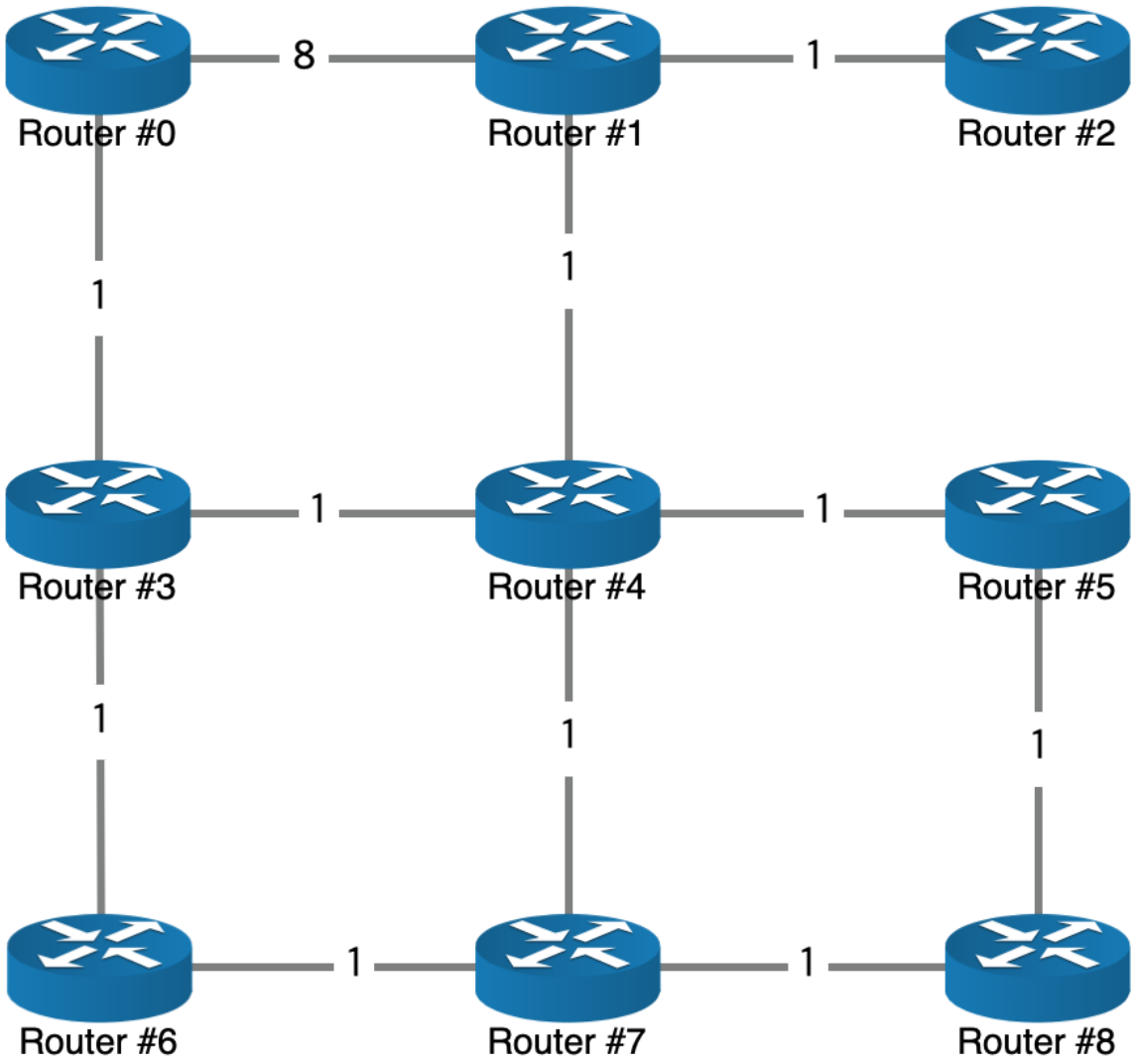
RoutingEnvRip2 / RoutingEnvCustom2



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.