CS 640 Introduction to Computer Networks **Spring'22**

Assignment 3: ICMP and RIP

Assigned at 11:00am on Mar 10 **Due at 11:59pm on: Mar 29**

Overview

For this assignment, you will modify your virtual router to build a routing table using distance vector routing. With this change, your virtual router will no longer depend on a static route table.

Learning Outcomes

After completing this programming assignment, students should be able to:

1. Write code that constructs and deconstructs packets containing multiple layers of protocols
2. Explain how the distance vector (DV) routing work

1 Getting Started

You will use the same environment and code base as Assignment 2. You should create a copy of your entire `assign2` and name it `assign3`:

```
cp -r ~/assign2 ~/assign3
```

You can use the version of `Router.java`, `RouteTable.java` and `Switch.java` you wrote for Assignment 2, or you can download our solutions for these files from this GDrive share (requires UW login). If you've forgotten the commands to start Mininet, POX, or your virtual router, you should refer back to Assignment 2.

As you complete this assignment, you may want to use `tcpdump` to examine the headers of packets sent/received by hosts. This will help you identify potential issues in your implementation. To record network traffic with `tcpdump` on a specific host, open an xterm window:

```
mininet> xterm h1
```

Then start tcpdump in that xterm:

```
sudo tcpdump -n -vv -e -i h1-eth0
```

This will cause tcpdump to print brief messages on the packets sent/received on the interface `h1-eth0`. You'll need to change the host number included in the interface (`-i`) argument to match the host on which you're running tcpdump. tcpdump is also capable of storing the packet contents into a capture dump for offline analysis with `-w`. Consult the manual of tcpdump (with `man tcpdump`) for advanced usage. Come to office hours if you have issue debugging with `tcpdump`.

2 Implement ICMP

For this part of the assignment, you will add support for generating and responding to Internet Control Message Protocol (ICMP) messages. Details on the ICMP protocol are available from Network Sorcery's RFC Sourcebook. There are five different ICMP messages you need to generate. These ICMP messages may also help you when testing and debugging your RIP implementation in the next section.

2.1 Time exceeded

Recall that in Assignment 2, your router is programmed to drop an incoming IP packet if TTL decrements to zero. You should update this portion of the code to generate an ICMP time exceeded message prior to dropping the packet whose TTL field is 0. ICMP type and code for this message is 11 and 0, respectively.

When the router generates ICMP messages for time exceeded, the packet must contain an Ethernet header, IPv4 header, ICMP header, and ICMP payload. You can construct these headers, by creating Ethernet, IPv4, ICMP, and Data objects using the classes in the `net.floodlightcontroller.packet` package. To link the headers together, you should call the `setPayload(...)` method defined in the `BasePacket` class, which is the superclass for all of the header classes. Below is a snippet of code to get you started:

```
Ethernet ether = new Ethernet();
IPv4 ip = new IPv4();
ICMP icmp = new ICMP();
Data data = new Data();
ether.setPayload(ip);
ip.setPayload(icmp);
icmp.setPayload(data);
```

In the Ethernet header, you must populate the following fields:

- EtherType: set to `Ethernet.TYPE_IPV4`
- Source MAC: set to the MAC address of the out interface obtained by performing a lookup in the route table (invariably this will be the interface on which the original packet arrived)
- Destination MAC: set to the MAC address of the next hop, determined by performing a lookup in the route table followed by a lookup in the ARP cache

In the IP header, you must populate the following fields:

- TTL: set to 64
- Protocol: set to `IPv4.PROTOCOL_ICMP`
- Source IP: set to the IP address of the interface on which the original packet arrived
- Destination IP — set to the source IP of the original packet

In the ICMP header you must populate the following fields:

- Type: set to 11
- Code: set to 0

The payload that follows the ICMP header must contain: (1) 4 bytes of padding, (2) the original IP header from the packet that triggered the error message, and (3) the 8 bytes following the IP header in the original packet. This is illustrated in the figure below:

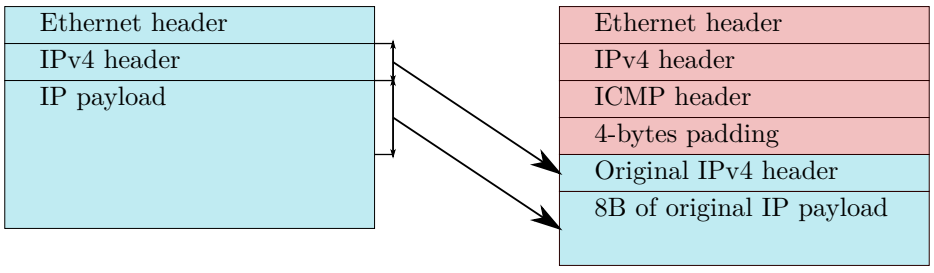

Figure 1: ICMP response structure

Once the ICMP packet is fully constructed, you should send it on the interface that is obtained from the longest prefix match in the route table for the source IP of original packet (invariably this will be the interface

on which the original packet arrived). You should drop the original packet after sending the time exceeded message.

You can verify you have implemented this correctly by running the traceroute command. You should traceroute from one host to another, and each router on the path between the hosts should show up in the traceroute once. Include the `-n` argument when you run traceroute, otherwise traceroute will try to convert IPs to hostnames using DNS which will generate spurious traffic and make traceroute slow. Below is example output produced using the `single_rt` topology:

```
mininet> h1 traceroute -n 10.0.2.102
traceroute to 10.0.2.102 (10.0.2.102), 30 hops max, 60 byte packets
 1 10.0.1.1 153.720 ms 229.000 ms 228.789 ms
 2 10.0.2.102 427.040 ms 426.764 ms 426.554 ms
 ....
```

Figure 2: Traceroute sample output on a `single_rt` topology.

2.2 Destination network unreachable

This message must be sent if there is no matching entry in the route table when forwarding an IP packet. Your router should already have code that drops a packet if there is no matching route entry. You should update this portion of the code to generate an ICMP destination network unreachable message prior to dropping the packet.

The destination net unreachable message should be constructed similar to the time exceeded message. However, in the ICMP header the type field should be 3 and the code field should be 0. You should drop the original packet after sending the destination net unreachable message.

You can verify you have implemented this correctly by ping-ing a host in a network that is not in the mininet topology. If you send ping a host the router does not know how to reach, then ping should say that it received a net unreachable message. Below is example output produced using the `single_rt` topology when pinging a host in a nonexistent network:

```
mininet> h1 ping -c 2 10.1.1.1
PING 10.1.1.1 (10.1.1.1) 56(84) bytes of data.
From 10.0.1.1 icmp_seq=1 Destination Net Unreachable
From 10.0.1.1 icmp_seq=2 Destination Net Unreachable
--- 10.1.1.1 ping statistics ---
2 packets transmitted, 0 received, +2 errors, 100% packet loss, time 1001ms
```

Figure 3: Ping-ing nonexistent subnet on a `single_rt` topology.

2.3 Destination host unreachable

This message should be sent if the MAC address associated with an IP address cannot be resolved using ARP. You should update the portion of your router that drops a packet if the `lookup(...)` function in the ARPCache class returns `null`.

The destination host unreachable message should be constructed similar to the destination net unreachable message, except the code field in the ICMP header should be 1. You should drop the original packet after sending the destination host unreachable message.

You can verify you have implemented this correctly by ping-ing a nonexistent host in an existing subnet. If you send ping a host the router couldn't obtain MAC address, then ping should say that it received a host

unreachable message. Below is example output produced using the `single_rt` topology when pinging an nonexistent IP in a known network:

```
mininet> h1 ping -c 2 10.0.2.33
PING 10.0.2.33 (10.0.2.33) 56(84) bytes of data.
From 10.0.1.1 icmp_seq=1 Destination Host Unreachable
From 10.0.1.1 icmp_seq=2 Destination Host Unreachable
--- 10.0.2.33 ping statistics ---
2 packets transmitted, 0 received, +2 errors, 100% packet loss, time 1001ms
```

Figure 4: Ping-ing nonexistent host on a `single_rt` topology.

2.4 Destination port unreachable

This message should be sent if your router receives a TCP or UDP packet destined for one of its interfaces. Your router should already check if an incoming IP packet is destined for one of its interfaces, and drop the packet if this is the case. However, you need to modify this code to check what header comes after the IP header:

- For a TCP/UDP messages (i.e. the protocol field in the IP header equals `IPv4.PROTOCOL_UDP` or `IPv4.PROTOCOL_TCP`), you should construct and send destination port unreachable message. This message should be constructed similar to the destination net unreachable message, except the code field in the ICMP header should be 3. You should drop the original packet after sending the destination port unreachable message.
- For ICMP messages (i.e. the protocol field in the IP header equals `IPv4.PROTOCOL_ICMP`), you should check if the ICMP message is an echo request (i.e., the type field in the ICMP header equals 8). If the ICMP message is an echo request (used by ping), then you should construct and send an echo reply message, described below. Otherwise, you should drop the packet.

You can verify you have implemented this correctly by attempting to `wget` a file from an IP assigned to one of the router's interface. Below is example output produced using the `single_rt` topology:

```
mininet> h1 wget 10.0.1.1
--2022-03-01 12:55:09-- http://10.0.1.1/
Connecting to 10.0.1.1:80... failed: Connection refused.
```

Figure 5: Initiating a TCP connection to the router should give you "Connection refused"

2.5 Echo reply

This message should be sent when your router receives an ICMP echo request destined for one of its interfaces. You **should not** send an ICMP echo reply if you receive an echo request whose destination IP address does not match any of the IP addresses assigned to the router's interfaces; these packets should be forwarded as they were before, since they are destined for hosts (or other routers).

The ICMP echo reply message should be constructed similar to the time exceeded message. However, the source IP in the IP header should be set to the destination IP from the IP header in the echo request. Additionally, the type field in the ICMP header should be set to 0. Lastly, the payload that follows the ICMP header in the echo reply must contain the entire payload that follows the ICMP header in the echo request. This is illustrated in the figure below:

You can verify you have implemented this correctly by pinging the IP assigned to one of the router's interfaces.

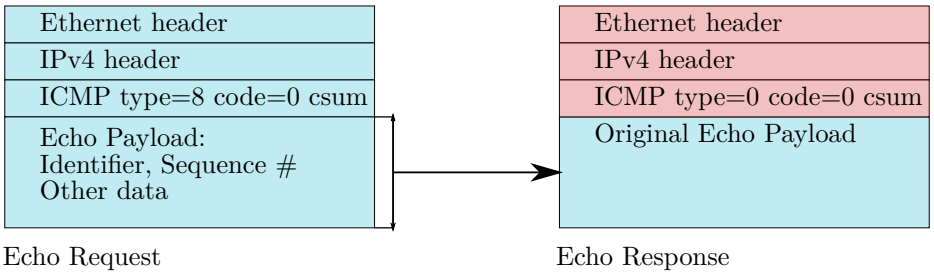

Figure 6: ICMP ECHO response structure

3 Implement RIP

For this part of the assignment, you will implement distance vector routing to build, and update, your router's route table. Specifically, you will implement a simplified version of the Routing Information Protocol v2 (RIPv2). Details on the RIPv2 protocol are available from RFC2453 and Network Sorcery RFC Sourcebook. If you're not sure how distance vector routing works, you should read Section 3.3.2 of the textbook or the lecture slides on distance vector routing.

Starting RIP

Your router should only run RIP when a static route table is not provided (via the `-r` argument when running `VirtualNetwork.jar`). You should modify `Main.java` and/or `Router.java` to appropriately start RIP when required.

When your router starts, you should add entries to the route table for the subnets that are directly reachable via the router's interfaces. This subnets can be determined based on the IP address and netmask associated with each of the router's interfaces. These entries should have no gateway.

RIP Packets

The `RIPv2` and `RIPv2Entry` classes in the `net.floodlightcontroller.packet` package define the format for RIPv2 packets. All RIPv2 packets should be encapsulated in UDP packets whose source and destination ports are 520 (defined as a constant `RIP_PORT` in the UDP class). When sending RIP requests and unsolicited RIP responses, the destination IP address should be the multicast IP address reserved for RIP (`224.0.0.9`) and the destination Ethernet address should be the broadcast MAC address (`FF:FF:FF:FF:FF:FF`). When sending a RIP response for a specific RIP request, the destination IP address and destination Ethernet address should be the IP address and MAC address of the router interface that sent the request.

RIP Operation

Your router should send a RIP request out all of the router's interfaces when RIP is initialized. Your router should send an unsolicited RIP response out all of the router's interfaces every 10 seconds thereafter.

You should update the `handlePacket(...)` method in the `Router` class to check if an arriving IP packet is of protocol type UDP, and a UDP destination port of 520. Packets that match this criteria are RIP requests or responses. Your router should update its route table based on these packets, and send any necessary RIP response packets.

Your router should time out route table entries for which an update has not been received for more than 30 seconds. You should never remove route entries for the subnets that are directly reachable via the router's interfaces.

Your implementation does not need to be a complete standards-compliant implementation of RIPv2. You should implement basic distance vector routing as discussed in the textbook and in-class, using RIPv2 packets as the format for messages exchanged between routers.

Testing RIP

To test your router's control plane, you will need a topology with more than one router: `pair_rt.topo`, `triangle_rt.topo`, `triangle_with_sw.topo`, or `linear5.topo`. You should not include the `-r` argument when starting your routers, since your router should construct its route table using RIP, rather than using a statically provided route table. Start your implementation as you would in Assignment 2 Part 3, but omit the static routing table in the parameters `-r rtable.rX` so that your router knows it should use RIP to find out on its own:

```
java -jar VirtualNetwork.jar -v r1 -a arp_cache
```

You may use the same method you tested with your implementation in Assignment 2 Part 3.

To test whether your RIP implementation can safely fail-over in a redundant topology (i.e. with loop), `triangle_rt.topo` is a good starting point. This topology has three routers connected in a ring. You might want to bring down one of the routers (say `r1`) by pressing Ctrl-C on the VirtualNetwork instance running on r1, then verify that h2 can still reach h3 even if r1 has failed. Feel free to create other topologies to test your implementation.

Submission Instructions

You must submit a single gzipped tar of your `src` directory containing the Java source files for your virtual switch and router. Please submit the entire `src` directory; do not submit any other files or directories. To create the tar file, run the following command, replacing `username1` and `username2` with the CS username of each group member:

```
tar -czvf username1_username2.tgz src
```

Upload the tar file to the Assignment 3 dropbox on canvas. Please submit only one tar file per group.