Assignment 3: ARP, ICMP, and RIP

CS640 Spring 2015

Due: ~~Thursday, March 19~~ Saturday, March 21 at 11pm

Overview

For this assignment, you will modify your virtual router to: (1) generate Internet Control Messaging Protocol (ICMP) messages when error conditions occur; (2) populate the ARP cache by generating and consuming Address Resolution Protocol (ARP) messages; and (3) build a routing table using distance vector routing. With these changes, your virtual router will no longer depend on a static ARP cache or static route table, and it will be pingable and traceable.

Learning Outcomes

After completing this assignment, students should be able to:

- Write code that constructs and deconstructs packets containing multiple layers of protocols
- Explain how the Address Resolution Protocol (ARP) and distance vector (DV) routing work

Clarifications

- 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.

Part 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` and `RouteTable.java` you wrote for Assignment 2, or you can download our solutions for these two files:

```
cd ~/assign3/src/edu/wisc/cs/sdn/vnet/rt/
wget http://cs.wisc.edu/~agember/cs640/s15/files/Router.java
```

```
wget http://cs.wisc.edu/~agember/cs640/s15/files/RouteTable.java
```

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. To run 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
```

You'll need to change the host number included in the interface (`-i`) argument to match the host on which you're running tcpdump.

Part 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 (if the Network Sorcery website is unavailable, you can access our mirror of the relevant pages). There are five different ICMP messages you need to generate.

Time exceeded

A time exceeded message must be sent whenever your virtual router discards a packet because the TTL field is 0. Your router should already have code that decrements the TTL field by 1 and checks if the field (after decrement) equals 0. 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.

When the router generates ICMP messages for time exceeded (type 11, code 0), the packet must contain an an Ethernet header, IP 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 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:

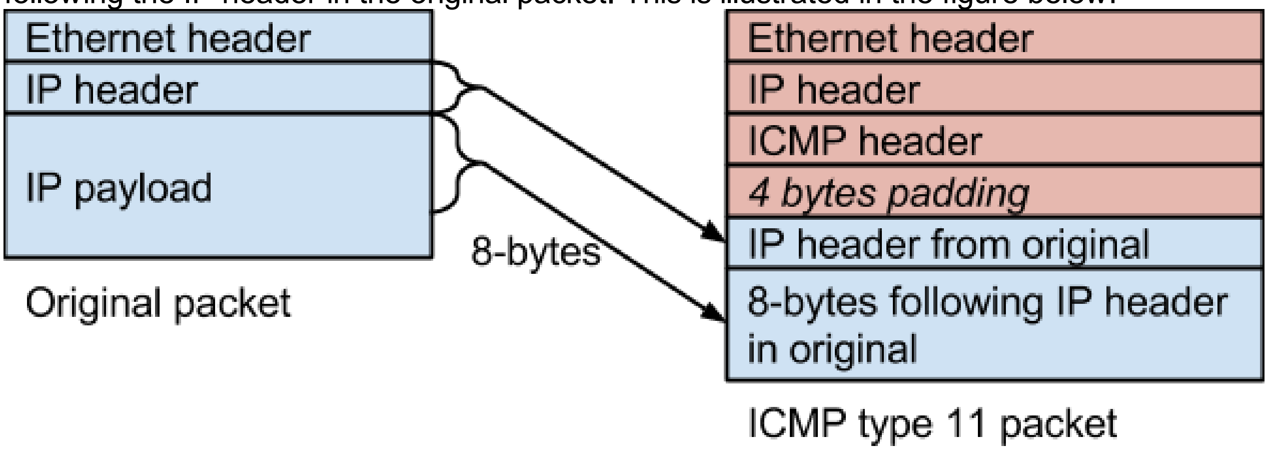
- TTL — set to 64

- 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:



Once the ICMP packet is fully constructed, you should send it on the same 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
```

Destination net 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 net 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 removing a line from the rtable file specified in the command line arguments when you start your router. 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 after the second line has been removed from `rtable.r1`:

```
mininet> h1 ping -c 2 10.0.2.102
PING 10.0.2.102 (10.0.2.102) 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
--- 10.0.2.102 ping statistics ---
2 packets transmitted, 0 received, +2 errors, 100% packet loss,
time 1001ms
```

Destination host unreachable

This message should be sent if the MAC address associated with an IP address cannot be resolved using ARP. You'll implement ARP in the next part of this assignment, but for now 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 removing a line for one of the host IPs from the arp_cache file specified in the command line arguments when you start your router. If you send ping to a host whose IP is not in the ARP cache, then ping should say that it received a host unreachable message. Below is example output produced using the `single_rt` topology after the second line has been removed from `arp_cache`:

```
mininet> h1 ping -c 2 10.0.2.102
PING 10.0.2.102 (10.0.2.102) 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
--- 10.0.2.102 ping statistics ---
2 packets transmitted, 0 received, +2 errors, 100% packet loss,
time 1001ms
```

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:

- If a TCP or UDP header comes after the IP header (i.e., the protocol field in the IP header equals `IPv4.PROTOCOL_UDP` or `IPv4.PROTOCOL_TCP`) then 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.
- If an ICMP header comes after the IP header (i.e., the protocol field in the IP header equals `IPv4.PROTOCOL_ICMP`), then 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.

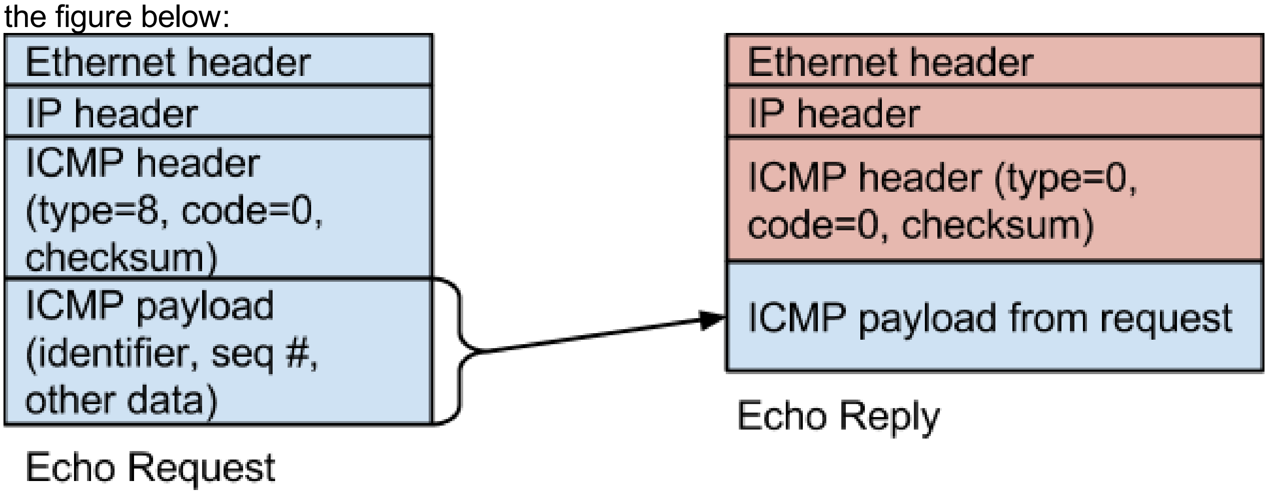
You can verify you have implemented this correctly by attempting to wget a file from an IP assigned to one of the router's interface. Below is example output produced using the `single_rt` topology:

```
mininet> h1 wget 10.0.1.1
--2015-03-15 05:55:09--  http://10.0.1.1/
Connecting to 10.0.1.1:80... failed: Connection refused.
```

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.

Part 3: Implement ARP

For this part of the assignment, you will implement the Address Resolution Protocol (ARP) such that your router does not need to be provided with a static ARP cache. Details on the ARP protocol are available from Network Sorcery's RFC Sourcebook (if the Network Sorcery website is unavailable, you can access our mirror of the relevant pages).

Generating ARP Replies

You should first update your router to generate ARP replies in response to received ARP requests. Your router should currently drop all non-IP packets. You need to modify this part of your router's code to accept ARP packets (i.e., the EtherType field in the Ethernet header equals `Ethernet.TYPE_ARP`) in addition to IP packets.

If an ARP packet is an ARP request (i.e., the opcode field in the ARP header equals `ARP.OP_REQUEST`), then your router **may** need to send an ARP reply. Your router must only respond to ARP requests whose target IP protocol address equals the IP address of the interface on which the ARP request was received. You can obtain the target protocol address from an ARP packet as follows:

```
ARP arpPacket = (ARP)etherPacket.getPayload();
int targetIp =
ByteBuffer.wrap(arpPacket.getTargetProtocolAddress()).getInt();
```

An ARP reply packet must contain an Ethernet header and an ARP header. You can construct these headers, by creating `Ethernet` and `ARP` objects using the classes in the `net.floodlightcontroller.packet` package. Remember to link the headers together using the setPayload(..) method defined in the `BasePacket` class.

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

- EtherType — set to `Ethernet.TYPE_ARP`
- Source MAC — set to the MAC address of the interface on which the original packet arrived
- Destination MAC — set to the source MAC address of the original packet

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

- Hardware type — set to `ARP.HW_TYPE_ETHERNET`
- Protocol type — set to `ARP.PROTO_TYPE_IP`
- Hardware address length — set to `Ethernet.DATALAYER_ADDRESS_LENGTH`
- Protocol address length — set to 4
- Opcode — set to `ARP.OP_REPLY`
- Sender hardware address — set to the MAC address of the interface on which the original packet arrived
- Sender protocol address — set to the IP address of the interface on which the original packet arrived
- Target hardware address — set to the sender hardware address from the original packet
- Target protocol address — set to the sender protocol address from the original packet

Once the ARP reply is fully constructed, you should send it on the same interface on which the original packet arrived.

You can verify you have implemented this correctly by **excluding** the `-a` argument when you call the `run_mininet.py` script. You **should still** specify the `-a arp_cache` argument when you start your virtual router. If you can successfully ping between hosts, and running `arp -n` on the hosts after running ping shows some ARP cache entries on the host, then you have successfully implemented ARP replies.

Generating ARP Requests

ARP requests are more challenging to implement because your router must queue packets while waiting for a reply. Your router should currently call the `lookup(...)` function the `ARPCache` class when forwarding packets, and if no entry is found, generate a destination host unreachable ICMP message and drop the packet. You should modify your router to instead enqueue the packet and generate an ARP request if no matching entry is found in the ARP cache. Also, you should **no longer** generate a destination host unreachable message when a call to lookup returns `null`.

You should maintain a separate queue of packets for each IP address for which we are waiting for the corresponding MAC address. All packet headers should be complete, except the destination MAC address in the Ethernet header, before enqueueing the packet.

An ARP request should be construct similar to an ARP reply, except for the following fields:

- Destination MAC address — set to the broadcast MAC address `FF:FF:FF:FF:FF:FF`
- Opcode — set to `ARP.OP_REQUEST`
- Target hardware address — set to `0`
- Target protocol address — set to the IP address whose associated MAC address we want

If a corresponding ARP reply does not arrive within 1 second of issuing the ARP request, then you should send another ARP request that's exactly the same as the original ARP request. You should continue to send ARP requests every second until your router either receives a corresponding ARP reply or you have sent the same ARP request 3 times. (It's not important that the 1 second is exact. However, you should send a duplicate ARP request no earlier than 1 second and no later than 2 seconds after the last ARP request was sent.)

If you have sent an ARP request **3 times** and no corresponding ARP reply has been received 1 second after sending the third request, then you should drop any packets that were queued waiting for a reply. However, before dropping each packet, you should generate a destination host unreachable message.

If the router wants to forward other IP packets to the same IP that your router is already trying to resolve, you should simply add those packets to the queue for the corresponding IP address.

Receiving ARP Replies

When your router receives an ARP reply, it should add an entry to the ARP cache (populated from the sender hardware address and sender protocol address fields). It should also dequeue any waiting packets, fill in the correct destination MAC address (from the sender hardware address field on the ARP reply), and send those packets out the interface on which the ARP reply arrived.

You can verify you have implemented ARP requests correctly by **excluding** the `-a arp_cache` argument when you start your virtual router. If your unsure if your code for generating ARP replies is working correctly, then you **should still** specify the `-a` argument when you call the `run_mininet.py` script. If you can successfully ping between hosts, and running `tcpdump -n -i hN-eth0` on `hN` (replace N with the number of the host you are pinging) shows some ARP request and reply packets, then you have successfully implemented ARP requests.

You should test that unsuccessful ARP requests are handled correctly by pinging a **non-existent IP** address in one of the subnets (e.g., ping 10.0.2.105, which is a non-existent IP in the subnet 10.0.2.0/24, from h1).

Part 4: Implement RIP

For the last 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 Network Sorcery's RFC Sourcebook (if the Network Sorcery website is unavailable, you can access our mirror of the relevant pages). If you're not sure how distance vector routing works, you should read Section 3.3.2 of the textbook or the notes from February 24th.

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 update `Main.java` and/or `Router.java` to appropriately start RIP.

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 has a destination 224.0.0.9, a protocol type of 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 ~~(i.e., the route table entries added by the `init()` function in the `RIP` class we provide)~~.

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.

Submission Instructions

You must submit a single tar file of the `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 Learn @ UW. Please submit only one tar file per group.

Acknowledgements

This programming assignment borrows from the *Simple Router* assignment from Stanford CS144: An Introduction to Computer Networks and Rodrigo Fonseca's *IP Project* from Brown University CSCI-1680: Computer Networks.