CS 640 Introduction to Computer Networks **Fall'25**

Assignment 2

Due at 11:59pm on: Oct 9

Overview

For this assignment, you will implement the forwarding behavior of a switch and a router. Recall that a switch forwards packets based on MAC address, and a router forwards packets based on IP address.

Environment Setup

Option 1: Use a CSL machine for Part 1, and ssh into mininet VM for Part 2.
To run Mininet, you will use the mininet VMs which have already been assigned to you by the course staff. You have one VM per group.
Please connect yourself to the CS departmental VPN. Before connecting to your VM. You and your partner should have already received a password from us, which will be your credential when SSH into the VM. This password is shared in your group only, and works only on the VM you are assigned. When you connect via SSH, you may include the `-X` option to tunnel X11 (i.e., graphics) over SSH, assuming you have a X11 server running locally. For example if you are assigned VM `mininet-85`:

```
ssh -X mininet@mininet-85.cs.wisc.edu
```

Option 2: Download the VM image, and import it into VirtualBox installed on your own PC.
If you have issue accessing your VM or the VPN isn't working out for you, you may also download a pre-configured system image from here. (Warning: 2GB download, 10GB decompressed). The image can be launched locally in VirtualBox - a virtualization software available on Windows, Linux and Mac OS w/intel chip that you may install on your personal computer. For Mac OS with apple chip, please use the image from here and launch it with UTM. The image will have the same setup as the VM with the only addition of a graphical environment for your convenience. The instructions for assignments also work exactly the same on your local VirtualBox as the CS VM assigned to you.

FYI: You will need to have X11 installed on the machine from which you are connecting. On a personal machine, you can install Xming on Windows, XQuartz on Mac OS, or any graphical environment / desktop environment on Linux to get X11. You should only need X11 to run xterms in Mininet. For editing files now is the opportune time to learn a text editor that doesn't require a GUI! Vim and emacs are the most popular ones. (Optional: You can also use VSCode for remote development.)

To transfer files to/from your VM, you may use `scp` or `rsync`. See man pages for `scp` and `rsync` or find a tutorial online for detailed instructions. You could also use a version control system, such as Git or Subversion, and checkout a copy of the repository in your VM. If you opt to use a version control system, please make sure you don't inadvertently allow others (other than your group members) to access your work.

The machine is configured to auto-login and you will not be prompted for password from sudo. The password for the `mininet` user is `user`, and `root` for the root user. Feel free to change them if you wish.

If you want to SSH to the VM in VirtualBox and edit files, you need to change the networking mode to bridge (steps: "Settings" → "Network" → "Adapter 1" → "Attached to: Bridge Adapter") and reboot. You will see the host and guest machine are in the same subnet and you can use the IP of VM to SSH.

Please make sure your deliverables compile and run on either one of these two options. This is the environment where we grade your assignments.

Learning Outcomes

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

1. Construct an Ethernet switch that optimally forwards packets based on link layer headers
2. Determine the matching route table entry for a given IP address
3. Develop a router that updates and forwards packets based on network layer headers

1 Getting Started

You will be using Mininet, POX, and skeleton code for a simple router to complete the assignment. Mininet and POX are already installed in the virtual machine (VM) you used for Assignment 1. You should continue to use this VM for this project. You can always refer back to Part 2 of Assignment 1 if you have questions about using your VM.

Preparing Your Environment

Download the tarball from here: assign2.tgz

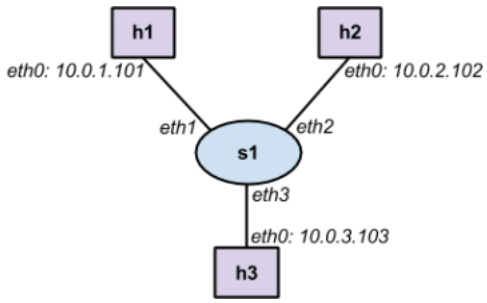
You may use scp or a similar method to transfer the file into your VM. Make sure to put the assignment folder in the directory "home/mininet".

After uncompressing the tarball file inside the VM, symlink POX and configure the POX modules:

```
cd ~/assign2
ln -s ../pox
./config.sh
```

Sample Configuration

The first sample configuration consists of a single switch (s1) and three emulated hosts (h1, h2, h3). The hosts are each running an HTTP server. When you have finished implementing your switch, one host should be able to fetch a web page from any other host (using wget or curl). Additionally, the hosts should be able to ping each other. The topology the below image shows is defined in the configuration file `topos/single_sw.topo`. The implementation details are given in the "Code Overview" Section.



Running the Virtual Switch

You'll need to start 3 terminals as described below. Start Mininet emulation of a Ethernet network with a single switch by running the following commands:

```
$ cd ~/assign2/
$ sudo ./run_mininet.py topos/single_sw.topo -a
```

You should see output like the following:

```
*** Loading topology file topos/single_sw.topo
*** Writing IP file ./ip_config
*** Creating network
*** Adding controller
Unable to contact the remote controller at 127.0.0.1:6633
*** Adding hosts:
h1 h2 h3
*** Adding switches:
s1
*** Adding links:
(h1, s1) (h2, s1) (h3, s1)
*** Configuring hosts
h1 h2 h3
*** Starting controller
*** Starting 1 switches
s1
*** Configuring routing for h1
*** Starting SimpleHTTPServer on host h1
*** Configuring routing for h2
*** Starting SimpleHTTPServer on host h2
*** Configuring routing for h3
*** Starting SimpleHTTPServer on host h3
*** Writing ARP cache file ./arp_cache
*** Configuring ARP for h1
*** Configuring ARP for h2
*** Configuring ARP for h3
*** Starting CLI:
mininet>
```

Keep this terminal open, as you will need the mininet command line for debugging. (Don't press ctrl-z.)

Open another terminal. Start the controller, by running the following commands:

```
cd ~/assign2/
./run_pox.sh
```

You should see output like the following:

```
POX 0.5.0 (eel) / Copyright 2011-2018 James McCauley, et al.
INFO:.home.mininet.assign2.pox_module.cs640.ofhandler:Successfully loaded VNet config file
 {'h3-eth0': ['10.0.1.103', '255.255.255.0'], 'h1-eth0': ['10.0.1.101', '255.255.255.0'], 'h2-eth0
     ↪ ': ['10.0.1.102', '255.255.255.0']}

INFO:.home.mininet.assign2.pox_module.cs640.vnethandler:VNet server listening on 127.0.0.1:8888
INFO:core:POX 0.5.0 (eel) is up.
```

You must wait for Mininet to connect to the POX controller before you continue to the next step. Once Mininet has connected, you will see output like the following:

```
INFO:openflow.of_01:[00-00-00-00-00-01 2] connected
INFO:cs640.vnethandler:VNetHandler catch VNetDevInfo(ifaces={'eth3': (None, None, None, 3), 'eth2':
    ↪  (None, None, None, 2), 'eth1': (None, None, None, 1)},swid=s1,dpid=1)
```

Keep POX running. (Don't press ctrl-z.) If you don't see this line being printed out, you may shutdown (i.e. press ctrl-c) pox and mininet and start over. You can try starting POX first and then start mininet. Note that POX is used "under the hood" in this assignment to direct packets between Mininet and your virtual switch and virtual router instances (i.e., Java processes). You do not need to understand, modify, or interact with POX in any way, besides executing the `run_pox.sh` script.

Open a third terminal. Compile your `VirtualNetwork` implementation with ant, and start `VirtualNetwork.jar` on the **only** switch in this topology `s1`, by running the following commands:

```
cd ~/assign2/
ant
java -jar VirtualNetwork.jar -v s1
```

Note: If you are testing on a topology with multiple switches, you should launch your `VirtualNetwork.jar` on **ALL** of the switches.
You should see output like the following:

```
Connecting to server localhost:8888
Device interfaces:
eth3
eth2
eth1
<-- Ready to process packets -->
```

Go back to the terminal where Mininet is running. To issue a command on an emulated host, type the hostname followed by the command in the Mininet console. Only the host on which to run the command should be specified by name; any arguments for the command should use IP addresses. For example, the following command sends 2 ping packets from h1 to h2:

```
mininet> h1 ping -c 2 10.0.1.102
```

The pings will fail because the virtual switch is not fully implemented. However, in the terminal where your virtual switch is running, you should see the following output:

```
*** -> Received packet:
    ip
    dl_vlan: untagged
    dl_vlan_pcp: 0
    dl_src: 00:00:00:00:00:01
    dl_dst: 00:00:00:00:00:02
    nw_src: 10.0.1.101
    nw_dst: 10.0.1.102
    nw_tos: 0
    nw_proto: 1
    icmp_type: 8
    icmp_code: 0
*** -> Received packet:
    ip
    dl_vlan: untagged
    dl_vlan_pcp: 0
    dl_src: 00:00:00:00:00:01
    dl_dst: 00:00:00:00:00:02
    nw_src: 10.0.1.101
    nw_dst: 10.0.1.102
    nw_tos: 0
    nw_proto: 1
    icmp_type: 8
    icmp_code: 0
```

You can stop your virtual switch by pressing ctrl-c in the terminal where it's running. You can restart the simple router without restarting POX and mininet, but it's often useful to restart POX and mininet to ensure the emulated network starts in a clean state.

Note: In order to run mininet, POX and the router/switch simultaneously, use the screen command, you can find more detailed information from this link. The key bindings for switching between screens and other actions can be found in the man page linked above.

Code Overview

The virtual network code consists of the following important packages and classes:

- `edu.wisc.cs.sdn.vnet` - **no need** to modify in this package
  - The main method (`Main`)
  - Classes representing a network device and interfaces (`Device`, `Iface`)
  - Code for creating a PCAP file containing all packets sent/received by a network device (`DumpFile`)
- `edu.wisc.cs.sdn.vnet.sw` - **add/modify** code in this package to complete Part 2
  - Skeleton code for a virtual switch (Switch)
- `edu.wisc.cs.sdn.vnet.rt` - **add/modify** code in this package to complete Part 3
  - Skeleton code for a virtual router (`Router`)
  - A complete implementation of an ARP cache (`ArpCache`, `ArpEntry`)
  - A partial implementation of a route table (`RouteTable`, `RouteEntry`)
- `net.floodlightcontroller.packet` - no need need to modify code in this package Code for parsing and manipulating packet headers

There are also several supporting packages and classes, which for the purpose of the assignment you `do not need to modify or understand`:

- `edu.wisc.cs.sdn.vnet.vns` - code to communicate with POX
- `org.openflow.util` - code for manipulating special types

When your virtual switch or router receives a packet, the `handlePacket()` function in the `Switch` or `Router` class is called. When you want to send a packet, call the `sendPacket()` function in the `Device` class (which is a superclass of the `Switch` and `Router` classes).

2 Implement Virtual Switch

For this part of the assignment, you will implement a learning switch which forwards packets at the link layer based on destination MAC addresses. If you're not sure how Ethernet switches work and discover on-link devices, you should read Section 3.1.4 of the textbook or review your notes from class.

Forwarding Packets

You should complete the `handlePacket(...)` method in the `edu.wisc.cs.sdn.vnet.sw.Switch` class to send a received packet out the **appropriate interface(s)** of the switch. You can use the `getSourceMAC()` and `getDestinationMAC()` methods in the `net.floodlightcontroller.packet.Ethernet` class to determine the source and destination MAC addresses of the received packet.

You should call the `sendPacket(...)` function inherited from the `edu.wisc.cs.sdn.vnet.Device` class to send a packet out a specific interface. To broadcast/flood a packet, you can call this method multiple times with a different interface specified each time. The `interfaces` variable inherited from the `Device` class contains all interfaces on the switch. The interfaces on a switch only have names; they do not have MAC addresses, IP addresses, or subnet masks.

You will need to add structures and/or classes to track the MAC addresses, and associated interfaces, learned by your switch. You should timeout learned MAC addresses after 15 seconds. The timeout does not need to be exact; a granularity of 1 second is fine. The timeout for a MAC address should be reset whenever the switch receives a new packet originating from that address.

Testing

You can test your learning switch by following the directions from Part 1. You can use any of the following topologies (in the `/assign2/topos` directory):

- `single_sw.topo`
- `linear5_sw.topo`
- `inclass_sw.topo`

If you are testing on a topology with multiple switches, you should launch your `VirtualNetwork.jar` on **ALL** of the switches (`java -jar VirtualNetwork.jar -v sX` where `sX` is a switch in your topology). You may test your implementation with any software that initiates a network transfer (e.g. your Iperfer from assignment 1). For your convenience, each virtual host has a simple web server running on them. To initiate a connection to the provided web servers, use `hX curl hY` to connect to `hY` from `hX` where `hX` and `hY` are different hosts on your mininet network. You should see something like this if your implementation is forwarding packets correctly:

```
<html>
<head><title>Test Page</title></head>
<body>
Congratuations! <br/>
Your router successfully route your packets. <br/>
</body>
</html>
```

You can also create your own topologies based on these examples, but do not create topologies with loops. Your virtual switch cannot handle loops in the topology because it does not implement spanning tree.

3 Implement Virtual Router

For this part of the assignment, you will implement a router which forwards packets at the network layer based on destination IP addresses. If you're not sure how IP packet forwarding works, you should read Section 3.2 of the textbook or review your notes from class.

For simplicity, your router will use a statically provided route table and a statically provided ARP cache. Furthermore, when your router encounters an error (e.g., no matching route entry), it will silently drop a packet, rather than sending an ICMP packet with the appropriate error message.

Route Lookups

Your first task is to complete the `lookup(...)` function in the `edu.wisc.cs.sdn.vnet.rt.RouteTable` class. Given an IP address, this function should return the RouteEntry object that has the longest prefix match with the given IP address. If no entry matches, then the function should return null.

Checking Packets

Your second task is to complete the `handlePacket(...)` method in the `edu.wisc.cs.sdn.vnet.rt.Router` class to update and send a received packet out the appropriate interface of the router.

When an Ethernet frame is received, you should first check if it contains an IPv4 packet. You can use the `getEtherType()` method in the `net.floodlightcontroller.packet.Ethernet` class to determine the type of packet contained in the payload of the Ethernet frame. If the packet is not IPv4, you do not need to do any further processing - i.e., your router should drop the packet.

If the frame contains an IPv4 packet, then you should verify the checksum and TTL of the IPv4 packet. You use the `getPayload()` method of the Ethernet class to get the IPv4 header; you will need to cast the result to `net.floodlightcontroller.packet.IPv4`.

The IP checksum should only be computed over the IP header. The length of the IP header can be determined from the header length field in the IP header, which specifies the length of the IP header in 4-byte words (i.e., multiple the header length field by 4 to get the length of the IP header in bytes). The checksum field in the

IP header should be zeroed before calculating the IP checksum. You can borrow code from the `serialize()` method in the IPv4 class to compute the checksum. If the checksum is incorrect, then you do not need to do any further processing - i.e., your router should drop the packet.

After verifying the checksum, you should decrement the IPv4 packet's TTL by 1. If the resulting TTL is 0, then you do not need to do any further processing - i.e., your router should drop the packet.

Now, you should determine whether the packet is destined for one of the router's interfaces. The `interfaces` variable inherited from the Device class contains all interfaces on the router. Each interface has a name, MAC address, IP address, and subnet mask. If the packet's destination IP address exactly matches one of the interface's IP addresses (not necessarily the incoming interface), then you do not need to do any further processing - i.e., your router should drop the packet.

Forwarding Packet

IPv4 packets with a correct checksum, TTL $>$ 1 (pre decrement), and a destination other than one of the router's interfaces should be forwarded. You should use the `lookup(...)` method in the `RouteTable` class, which you implemented earlier, to obtain the `RouteEntry` that has the longest prefix match with the destination IP address. If no entry matches, then you do not need to do any further processing - i.e., your router should drop the packet.

If an entry matches, then you should determine the next-hop IP address and lookup the MAC address corresponding to that IP address. You should call the `lookup(...)` method in the `edu.wisc.cs.sdn.vnet.rt.ArpCache` class to obtain the MAC address from the statically populated ARP cache. This address should be the new destination MAC address for the Ethernet frame. The MAC address of the outgoing interface should be the new source MAC address for the Ethernet frame.

After you have correctly updated the Ethernet header, you should call the `sendPacket(...)` function inherited from the `edu.wisc.cs.sdn.vnet.Device` class to send the frame out the correct interface.

Testing

You can test your learning switch by following the directions from Part 1. However, when starting your virtual router, you must include the appropriate static route table and static ARP cache as arguments. For example:

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

Make sure you already started mininet before hand so that it could generate the routing tables `rtable.rX` and ARP cache `arp_cache` for you.

You can use any of the following topologies (in the `/assign2/topos` directory) to test your router:

- `single_rt.topo`
- `pair_rt.topo`
- `triangle_rt.topo`
- `linear5_rt.topo`

To test your switch and router implementations together, use any of the following topologies:

- `single_each.topo`
- `triangle_with_sw.topo`

You can also create your own topologies based on these examples. As is in Part 2, you'll want to launch your `VirtualNetwork.jar` on **ALL** of the routers **AND** switches in the topology you are testing. Be sure you use the correct routing table (`rtable.rX`) on the correct virtual router `rX`.

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 2 dropbox on canvas. 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.