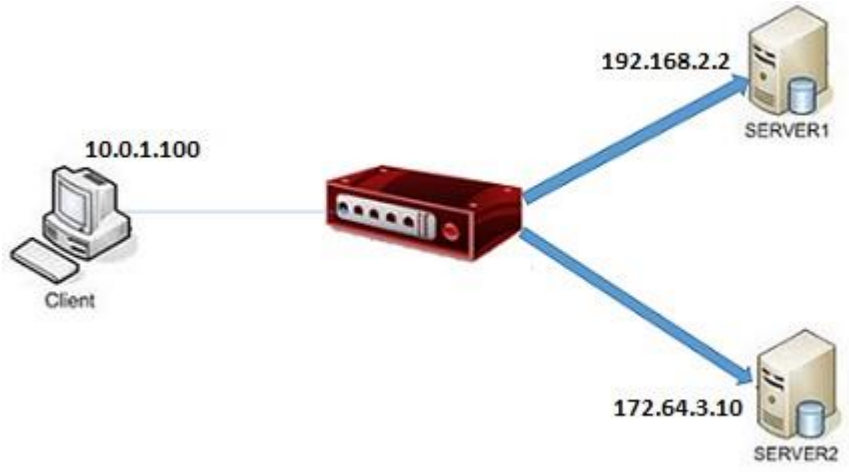
Building a Simple Router:

Basic ARP, ARP caching, ICMP, Switching, Longest Prefix Match, and the other essential features

Overview

In this assignment you will write a simple router given a static network topology and routing table. Your router will receive raw Ethernet frames. It will process these packets just like a real router and then forward them to the correct outgoing interface. You are responsible for implementing the logic for handling the incoming Ethernet frames (forward them, generate ICMP messages, drop them and more).

Your router will route real packets from an emulated host (client) to two emulated application servers (http server 1/2) sitting behind your router. **The application servers are each running an HTTP server. When you have finished the forwarding path of your router, you should be able to access these servers using regular client software. In addition, you should be able to ping and traceroute to and through a functioning Internet router.** A sample routing topology is shown below:



If the router is functioning correctly, you should be able to perform the following operations:

- Ping the router's interfaces (192.168.2.1, 172.64.3.1 , 10.0.1.1) from the client.
- **Traceroute from the client to any of the router's interfaces**
- Ping the servers (192.168.2.2, 172.64.3.10) from the client
- **Traceroute from the client to any of the app servers**
- **Download a file using HTTP from one of the app servers**

Additional requirements are laid out in the 'Required Functionality' section

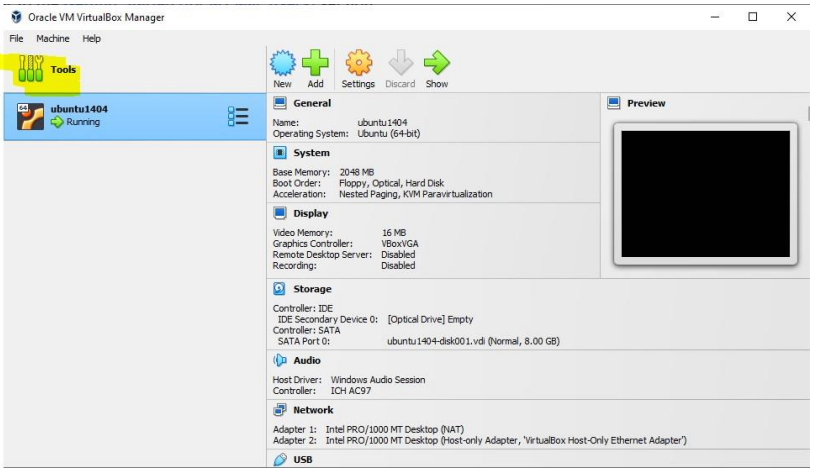
Mininet

This assignment runs on top of Mininet which was developed at Stanford. Mininet allows you toemulate a topology on a single machine. It provides the needed isolation between the emulated nodes so that your router node can process and forward real Ethernet frames between the hosts like a real router. You don't have to know how Mininet works to complete this assignment, but more information about Mininet (if you're curious) is available here.

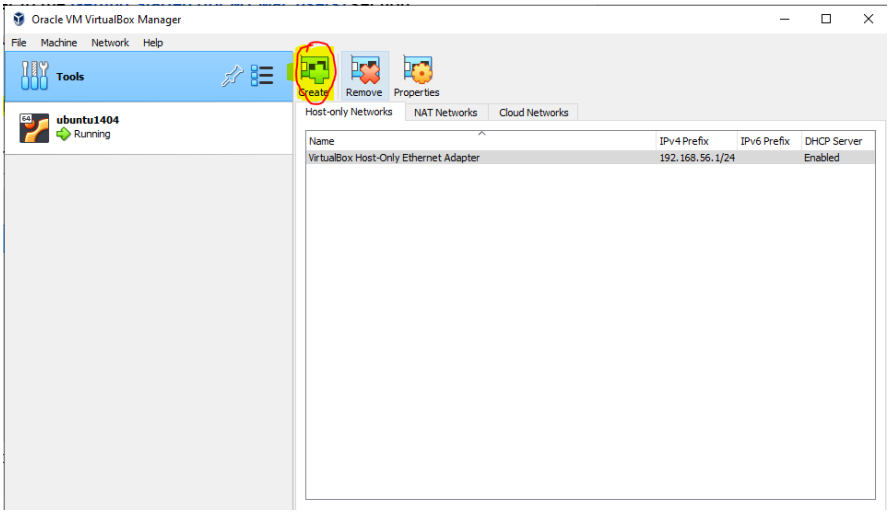
All of the setup instructions were tested on the newest version of VirtualBox. Please make sure that you install the newest version of VirtualBox before setup.

Getting Started (for Windows)

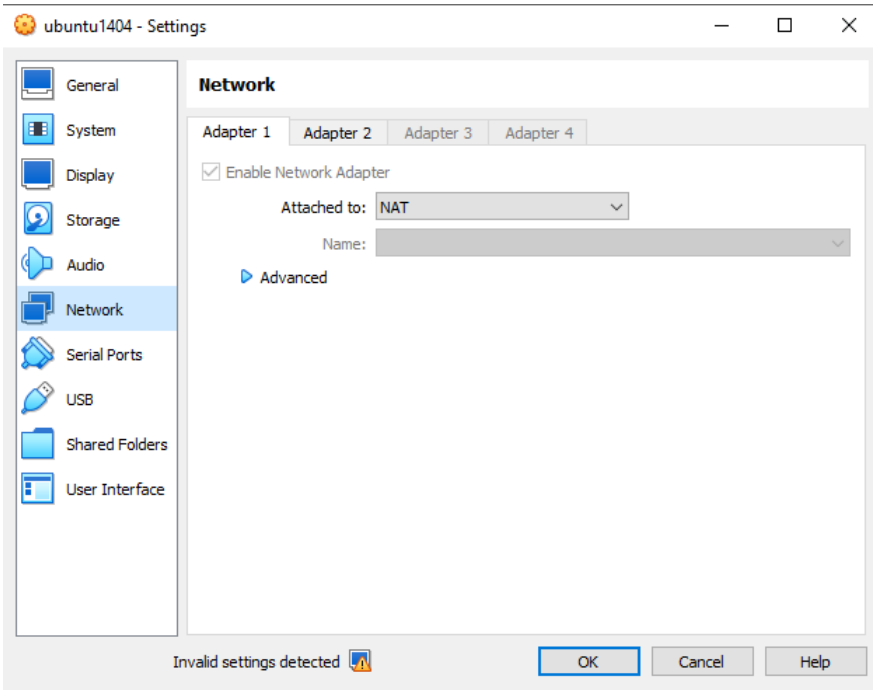
1. Download and install VirtualBox (https://www.virtualbox.org/)
    a. **Windows Users:** Please see the **Virtual Machine FAQ** section if you faceany issues.
2. Download the virtual machine image file ubuntu1404.ova . (The VM file is available upon request)
3. Configure VirtualBox with a **host-only** network:
    a. Open the “Tools” menu. If you do not see this, you can try clicking on the hamburger button in tools and selecting “Network”:



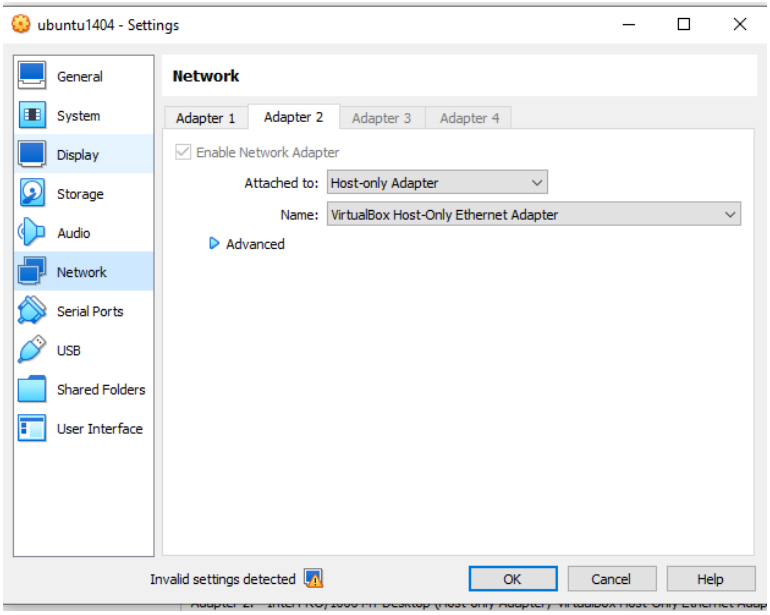
    b. Click on the Create button to create a host-only network.



4. Unzip the VM image and open the .ova file in VirtualBox.
    a. Keep the default settings and click on the Import button.
5. Check the settings for the newly imported virtual machine. The network section should show **two** enabled adapters.
    a. Adapter 1 should be attached to **NAT**.



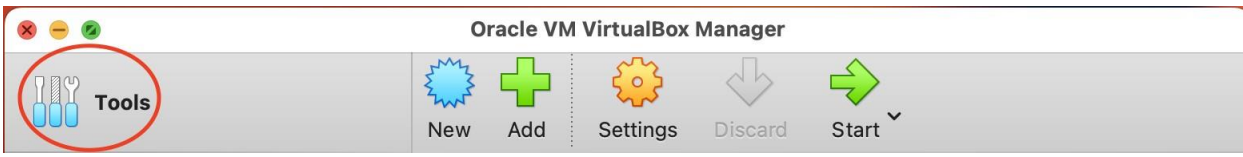
    b. Adapter 2 should be a **host-only** adapter connected to the host-only network you just created.



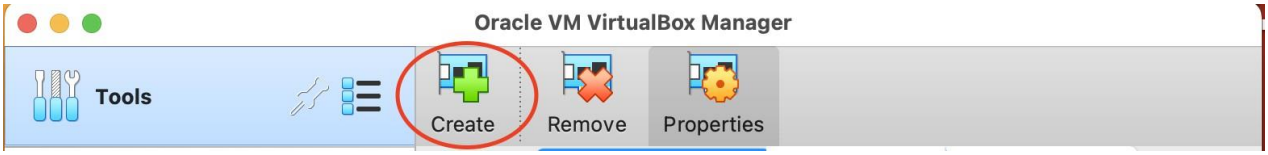
6. Now you should be able to run the virtual machine and log in with the **username mininet** and **password mininet**.
7. **NOTE:** After you log in to the virtual machine, run the command `ifconfig` to get an IP address on the host-only network. Take note of the eth1 IP address. You should only need to do this the first time. Use this IP to ssh into your VM.

Getting Started (for Intel Macs)

1. Download and install VirtualBox (https://www.virtualbox.org/)
2. Download the virtual machine image file ubuntu1404.ova from here. 3. Configure VirtualBox with a **host-only** network:
   a. Open the "Tools" menu:

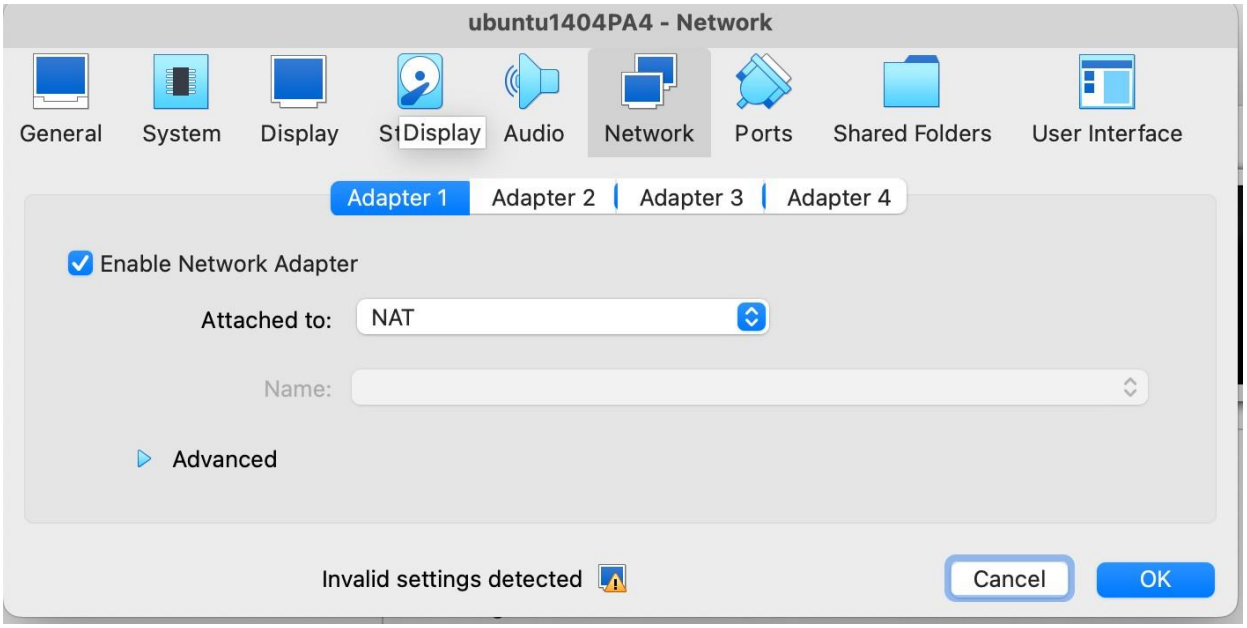


   b. Click on the Create button to create a host-only network.

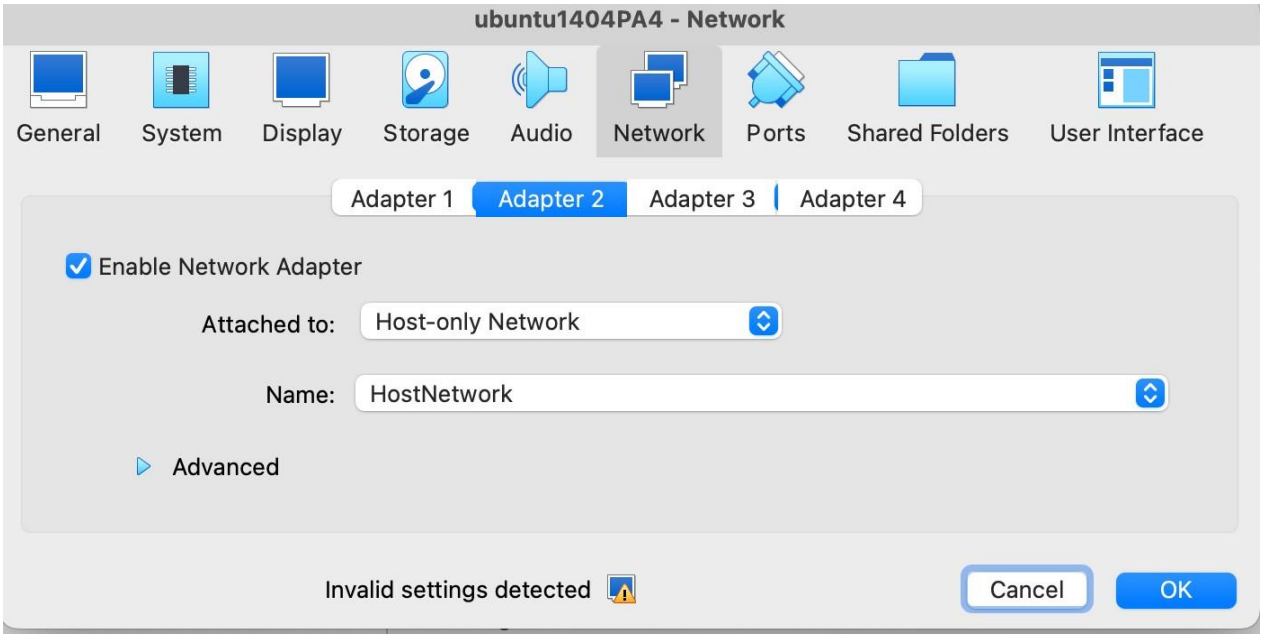


4. Unzip the VM image and open the .ova file in VirtualBox.
   a. Keep the default settings and click on the Import button.
5. Check the settings for the newly imported virtual machine. The network section should show **two** enabled adapters.

a. Adapter 1 should be attached to **NAT**.



b. Adapter 2 should be a **host-only network** connected to the host-only network you just created.



6. Now you should be able to run the virtual machine and log in with the **username: mininet** and **password: mininet**.
7. **NOTE:** After you log in to the virtual machine, run the command `ifconfig` to get an IP address on the host-only network. Take note of the eth1 IP address. You should only need to do this the first time. Use this IP to ssh into your VM.

Virtual Machine FAQ (skip this if you did not face any issue)

Cannot Start VM On Windows VirtualBox

- This may be related to having **Hyper-V** installed and enabled. Hyper-V is useful for many other applications such as Docker, but it may cause you some issues with this assignment. We recommend disabling Hyper-V while you work on the assignment. Todo this, try the following:
    - In the Control Panel, select Programs and Features.
    - Select Turn Windows features on or off.
    - Clear the **Hyper-V** check box (if you are on **Windows 10**) or the **Windows Hypervisor Platform** check-box (if you are on **Windows 11**).
    - Restart your computer when prompted. You **may have to restart it one more time** after this if it still doesn't work (make sure you select restart and not shutdown).

VERR_NEM_VM_CREATE_FAILED On Windows 11

- You might encounter this error in Windows 11 if **you had VirtualBox installed and working on Windows 10 and then upgraded your OS to Windows 11**. The VirtualBoxapplication will work fine, but when you start any VM, a window will pop up and then shortly after, it will be destroyed and the above error will be shown. To fix this, from an elevated (Run as administrator) Command Prompt window, execute the command `bcdedit /set`
- `hypervisorlaunchtype off` and then reboot your computer.

Ensure Hardware Virtualization is Enabled on BIOS Settings

- VirtualBox virtualization requires that the host machine has hardware virtualization enabled. While most computers have this feature, it might be disabled in the BIOS. You may have to reboot into the **system BIOS** and enable Virtualization Technology(VTx) in the system BIOS settings if it is not enabled on your computer. Please Google how to do this (update BIOS settings) depending on your computer's make and model.

Issue with Host-Only Adapter

- If you are on **Windows** (as of now, I haven't seen this issue on Linux) and face anissue with Adapter 2 set to "Host-only adapter", you can change that to a "BridgedAdapter".

Getting Started (for M chip Mac users)

1. Download and install UTM (https://mac.getutm.app/)
2. Download the virtual machine image zip file - **PA3_VM_UTM.zip** fromhere.
3. Unzip the **cse-123-p2.utm** image.
4. Press the "**+**" icon on the top of the screen to add a new VM.
5. Then under "**Existing**", click "**Open**" and open the **cse-123-p2.utm** image.
6. Press "**play**" to launch the VM. **Wait a bit and the operating system shouldboot.**
7. Log in with the **username mininet** and **password mininet**.

NOTE: From here, you might find it more convenient to **ssh** into your VM instead of working inside of it directly (you'll be able to copy and paste, among other luxuries). ● You need to discover the IP address of the mininet host so you can SSH into. This can be done by running `ifconfig eth0` and noting the IP address (which is likely to be `192.168.128.2`by default.

You can SSH into the mininet host from your regular machine by running `ssh mininet@192.168.128.2.`

Configuration Files

Use the same code repository that you used for project 3.

There are two configuration files.

- ~/cse123-p3/IP_CONFIG: Lists the IP addresses assigned to the emulated hosts. It is used by mininet to emulate the topology shown above.
- ~/cse123-p3/rtable: The static routing table used for the simple router.

The default *IP_CONFIG* and *rtable* should look like the following:

```
> cat ~/cse123-p3/IP_CONFIG
server1 192.168.2.2
server2 172.64.3.10
client   10.0.1.100
sw0-eth1 192.168.2.1
sw0-eth2 172.64.3.1
sw0-eth3 10.0.1.1
> cd ~/cse123-p3/
> cat ~/cse123-p3/rtable
192.168.2.2   192.168.2.2    255.255.255.255    eth1
172.64.3.10   172.64.3.10    255.255.255.255    eth2
10.0.1.100    10.0.1.100     255.255.255.255    eth3
```

If nothing appears when you cat these files, you may have missed the 'router' at the end of the git clone command. These files are symlinks to files that exist in the git repository which should be cloned to ~/cse123-p3/router

Test Connectivity of Your Emulated Topology

Mininet requires a controller, which is implemented in POX. To run the controller, use the following command:

```
> cd ~/cse123-p3/
> ./run_pox.sh
```

You should be able to see some output like the following:

```
POX 0.0.0 / Copyright 2011 James McCauley
DEBUG:.usr.local.lib.python2.7.dist-packages.cs144-0.0.0-py2.7.egg.cs
144.ofhandler:*** ofhandler: Successfully loaded ip settings for
hosts
 {'server1': '192.168.2.2', 'sw0-eth3': '10.0.1.1', 'sw0-eth1':
'192.168.2.1', 'sw0-eth2': '172.64.3.1', 'client': '10.0.1.100',
'server2': '172.64.3.10'}
INFO:.usr.local.lib.python2.7.dist-packages.cs144-0.0.0-py2.7.egg.cs1
44.srhandler:created server
DEBUG:.usr.local.lib.python2.7.dist-packages.cs144-0.0.0-py2.7.egg.cs
144.srhandler:SRServerListener listening on 8888
DEBUG:core:POX 0.0.0 going up...
DEBUG:core:Running on CPython (2.7.4/Apr 19 2013 18:32:33)
INFO:core:POX 0.0.0 is up.
This program comes with ABSOLUTELY NO WARRANTY. This program is free
software,
and you are welcome to redistribute it under certain conditions.
Type 'help(pox.license)' for details.
DEBUG:openflow.of_01:Listening for connections on 0.0.0.0:6633
INFO:openflow.of_01:[Con 1/139234599119694] Connected to
7e-a2-14-d2-2b-4e
DEBUG:.usr.local.lib.python2.7.dist-packages.cs144-0.0.0-py2.7.egg.cs
144.ofhandler:Connection [Con 1/139234599119694]
DEBUG:.usr.local.lib.python2.7.dist-packages.cs144-0.0.0-py2.7.egg.cs
144.srhandler:SRServerListener catch RouterInfo even, info={'eth3':
('10.0.1.1', 'c2:e3:78:bf:71:f9', '10Gbps', 3), 'eth2':
('172.64.3.1', '26:e4:1f:26:49:ab', '10Gbps', 2), 'eth1':
('192.168.2.1', '56:01:ec:fb:34:09', '10Gbps', 1)},
rtable=[('10.0.1.100', '10.0.1.100', '255.255.255.255', 'eth3'),
('192.168.2.2', '192.168.2.2', '255.255.255.255', 'eth1'),
('172.64.3.10', '172.64.3.10', '255.255.255.255', 'eth2')]
Ready.
POX>
```

Please note that you have to wait for a few seconds till Mininet is able to connect to the POX controller before you continue to the next step

Keep POX running. Now, open another terminal to continue to the next step.

You may want to use screen to run multiple shell sessions on the same window. For a cheat sheet of keyboard shortcuts available, consult the link or use manpages.

Start Mininet emulation by using the following command

```
> cd ~/cse123-p3/
> ./run_mininet.sh
```

You should be able to see some output like the following:

```
*** Shutting down stale SimpleHTTPServers
*** Shutting down stale webservers
*** Successfully loaded ip settings for hosts
 {'server1': '192.168.2.2', 'sw0-eth3': '10.0.1.1', 'sw0-eth1':
'192.168.2.1', 'sw0-eth2': '172.64.3.1', 'client': '10.0.1.100',
'server2': '172.64.3.10'}
*** Creating network
*** Creating network
*** Adding controller
*** Adding hosts:
client server1 server2
*** Adding switches:
sw0
*** Adding links:
(client, sw0) (server1, sw0) (server2, sw0)
*** Configuring hosts
client server1 server2
*** Starting controller
*** Starting 1 switches
sw0
*** setting default gateway of host server1
server1 192.168.2.1
*** setting default gateway of host server2
server2 172.64.3.1
*** setting default gateway of host client
client 10.0.1.1
*** Starting SimpleHTTPServer on host server1
*** Starting SimpleHTTPServer on host server2
*** Starting CLI:
mininet>
```

Keep this terminal open, as you will need the mininet command line for debugging. Now, use another terminal to continue to the next step.

If you're using screen (type screen to open new terminal). Use crtl+a+w to list different screens open and use crtl+a+"screen-id-number" to switch to needed screen.

Now you are ready to test out the connectivity of the environment setup. To do so, do the following:

```
> cd ~/cse123-p3
> ./sr_solution
```

You should be able to see some output like the following:

```
Using VNS sr stub code revised 2009-10-14 (rev 0.20)
Loading routing table from server, clear local routing table.
Loading routing table
---------------------------------------------
Destination     Gateway         Mask    Iface
192.168.2.2             192.168.2.2     255.255.255.255 eth1
172.64.3.10             172.64.3.10     255.255.255.255 eth2
10.0.1.100              10.0.1.100      255.255.255.255 eth3
---------------------------------------------
Client mininet connecting to Server localhost:8888
Requesting topology 0
successfully authenticated as mininet
Loading routing table from server, clear local routing table.
Loading routing table
---------------------------------------------
Destination     Gateway         Mask    Iface
192.168.2.2             192.168.2.2     255.255.255.255 eth1
172.64.3.10             172.64.3.10     255.255.255.255 eth2
10.0.1.100              10.0.1.100      255.255.255.255 eth3
---------------------------------------------
Router interfaces:
eth3    HWaddrc2:e0:f7:04:02:0c
        inet addr 10.0.1.1
eth2    HWaddr9a:25:38:88:a3:24
        inet addr 172.64.3.1
eth1    HWaddr1a:4c:9c:2b:b2:d9
        inet addr 192.168.2.1
<-- Ready to process packets -->
```

In this particular setup, 192.168.2.2 is the IP for server1, and 172.64.3.10 is the IP for server2. You can find the IP addresses in your IP_CONFIG file.

Now, back to the terminal where Mininet is running. To issue a command on the emulated host, type the host name followed by the command in the Mininet console. For example, the following command issues 3 pings from the client to the server1.

```
mininet> client ping -c 3 172.64.3.10
PING 172.64.3.10 (172.64.3.10) 56(84) bytes of data.
64 bytes from 172.64.3.10: icmp_req=1 ttl=63 time=184 ms
64 bytes from 172.64.3.10: icmp_req=2 ttl=63 time=60.3 ms
64 bytes from 172.64.3.10: icmp_req=3 ttl=63 time=95.1 ms

--- 172.64.3.10 ping statistics ---
3 packets transmitted, 3 received, 0% packet loss, time 2003ms

rtt min/avg/max/mdev = 60.317/113.209/184.125/52.127 ms
```

You can also use traceroute to trace paths through the network, and wget to make HTTP requests:

```
mininet> client traceroute -n 172.64.3.10
traceroute to 172.64.3.10 (172.64.3.10), 30 hops max, 60 byte packets
 1   10.0.1.1 (10.0.1.1) 61.344ms 67.144ms 72.285ms
 2   172.64.3.10 (172.64.3.10) 143.898ms 139.766ms 79.988ms
mininet> client wget http://172.64.3.10
```

When you finish implementing your router code, it should make the above commands work correctly.

When you run the starter code:

```
> cd ~/cse123-p3/router
> make
> ./sr
```

You should see some output from"./sr" like the following:

```
Loading routing table from server, clear local routing table.
Loading routing table
---------------------------------------------
Destination     Gateway         Mask    Iface
192.168.2.2             192.168.2.2     255.255.255.255 eth1
172.64.3.10             172.64.3.10     255.255.255.255 eth2
10.0.1.100              10.0.1.100      255.255.255.255 eth3
---------------------------------------------
```

```
Client mininet connecting to Server localhost:8888
Requesting topology 0
successfully authenticated as mininet
Loading routing table from server, clear local routing table.
Loading routing table
---------------------------------------------
Destination     Gateway         Mask    Iface
192.168.2.2             192.168.2.2     255.255.255.255 eth1
172.64.3.10             172.64.3.10     255.255.255.255 eth2
10.0.1.100              10.0.1.100      255.255.255.255 eth3
---------------------------------------------
Router interfaces:
eth3    HWaddrc2:e3:78:bf:71:f9
        inet addr 10.0.1.1
eth2    HWaddr26:e4:1f:26:49:ab
        inet addr 172.64.3.1
eth1    HWaddr56:01:ec:fb:34:09
        inet addr 192.168.2.1
 <-- Ready to process packets -->
```

Moving back to the mininet terminal, since the skeleton code can't handle ping requests, you will get unexpected ping output:

```
mininet> client ping -c 3 172.64.3.10
PING 172.64.3.10 (172.64.3.10) 56(84) bytes of data.
From 10.0.1.100 icmp_seq=1 Destination Host Unreachable
From 10.0.1.100 icmp_seq=2 Destination Host Unreachable
From 10.0.1.100 icmp_seq=3 Destination Host Unreachable

--- 172.64.3.10 ping statistics ---
3 packets transmitted, 0 received, +3 errors, 100% packet loss, time
2000ms
```

Multi-router Topology

For the multi-router topology, you will have to do something slightly different to test. Running POX and Mininet are the exact same as PA3 and single router topology, just now in the cse123-p4 folder, not cse123-p3. Then, you need two extra terminals (4 total now, not 3) and run the following commands in the router folder:

./sr -v sw1 -r ../rtable1

./sr -v sw2 -r ../rtable2

This will start both the routers up. The multirouter setup contains multiple clients so instead of using "client ping" you will have to do something like "client1 ping"

*** IMPORTANT FOR M chip Mac Users: ***

There is a high chance that after following the above steps to start the multi-router topology, that the router terminals will not start correctly. In this case, again, just have POX and Mininet and the two router commands start. Then, WITHOUT shutting down mininet, use exit() in POX and restart POX. Then, start the routers again.

This may also help those on Windows and Intel Macs too having problems.

Protocols to Understand

For help with understanding the different fields in the headers of the different protocols below, feel free to use the following: http://www.networksorcery.com/enp/default.htm (Click on **IP Protocol Suite** on the left hand side)

Ethernet

You are given a raw Ethernet frame and have to send raw Ethernet frames. You should understand source and destination MAC addresses and the idea that we forward a packet one hop by changing the destination MAC address of the forwarded packet to the MAC address of the next hop's incoming interface.

Internet Protocol

Before operating on an IP packet, you should verify its checksum and make sure it meets the minimum length of an IP packet. You should understand how to find the longest prefix match of a destination IP address in the routing table described in the "Getting Started" section. If you determine that a datagram should be forwarded, you should correctly decrement the TTL field of the header and recompute the checksum over the changed header before forwarding it to the next hop.

Internet Control Message Protocol

ICMP is a simple protocol that can send control information to a host. In this assignment, your router will use ICMP to send messages back to a sending host. You will need to properly generate the following ICMP messages (including the ICMP header checksum) in response to the sending host under the following conditions:

- Echo reply (type 0) Sent in response to an echo request (ping) to one of the router's interfaces. (This is only for echo requests to any of the router's IPs. An echo request sent elsewhere should be forwarded to the next hop address as usual.)
- Echo request (type 8) Received as a ping message to either the interfaces of the router or forwarded to any other devices in the network. NOTE: This packet is not generated by our simple router
- **Destination net unreachable (type 3, code 0) ** Sent if there is a non-existent route to the destination IP (no matching entry in routing table when forwarding an IP packet).**

- **Destination host unreachable (type 3, code 1) ** Sent if five ARP requests were sent to the next-hop IP without a response.**
- **Port unreachable (type 3, code 3) ** Sent if an IP packet containing a UDP or TCP payload is sent to one of the router's interfaces. This is needed for traceroute to work.**
- **Time exceeded (type 11, code 0) ** Sent if an IP packet is discarded during processing because the TTL field is 0. This is also needed for traceroute to work. The source address of an ICMP message can be the source address of any of the incoming interfaces, as specified in RFC 792. As it can be seen from above, the only incoming ICMP message destined towards the router's IPs that you have to explicitly process are ICMP echo requests. You may want to create additional structs for ICMP messages for convenience, but make sure to use the packed attribute so that the compiler doesn't try to align the fields in the struct to word boundaries:**

5.4. Address Resolution Protocol

ARP is needed to determine the next-hop MAC address that corresponds to the next-hop IP address stored in the routing table. Without the ability to generate an ARP request and process ARP replies, your router would not be able to fill out the destination MAC address field of the raw Ethernet frame you are sending over the outgoing interface. Analogously, without the ability to process ARP requests and generate ARP replies, no other router could send your router Ethernet frames. Therefore, your router must generate and process ARP requests and replies.

To lessen the number of ARP requests sent out, you are required to cache ARP replies. Cache entries should time out after 15 seconds to minimize staleness. The provided ARP cache class already times the entries out for you. When forwarding a packet to a next-hop IP address, the router should first check the ARP cache for the corresponding MAC address before sending an ARP request. In the case of a cache miss, an ARP request should be sent to a target IP address about once every second until a reply comes in. If the ARP request is sent five times with no reply, an ICMP destination host unreachable is sent back to the source IP as stated above. The provided ARP request queue will help you manage the request queue.

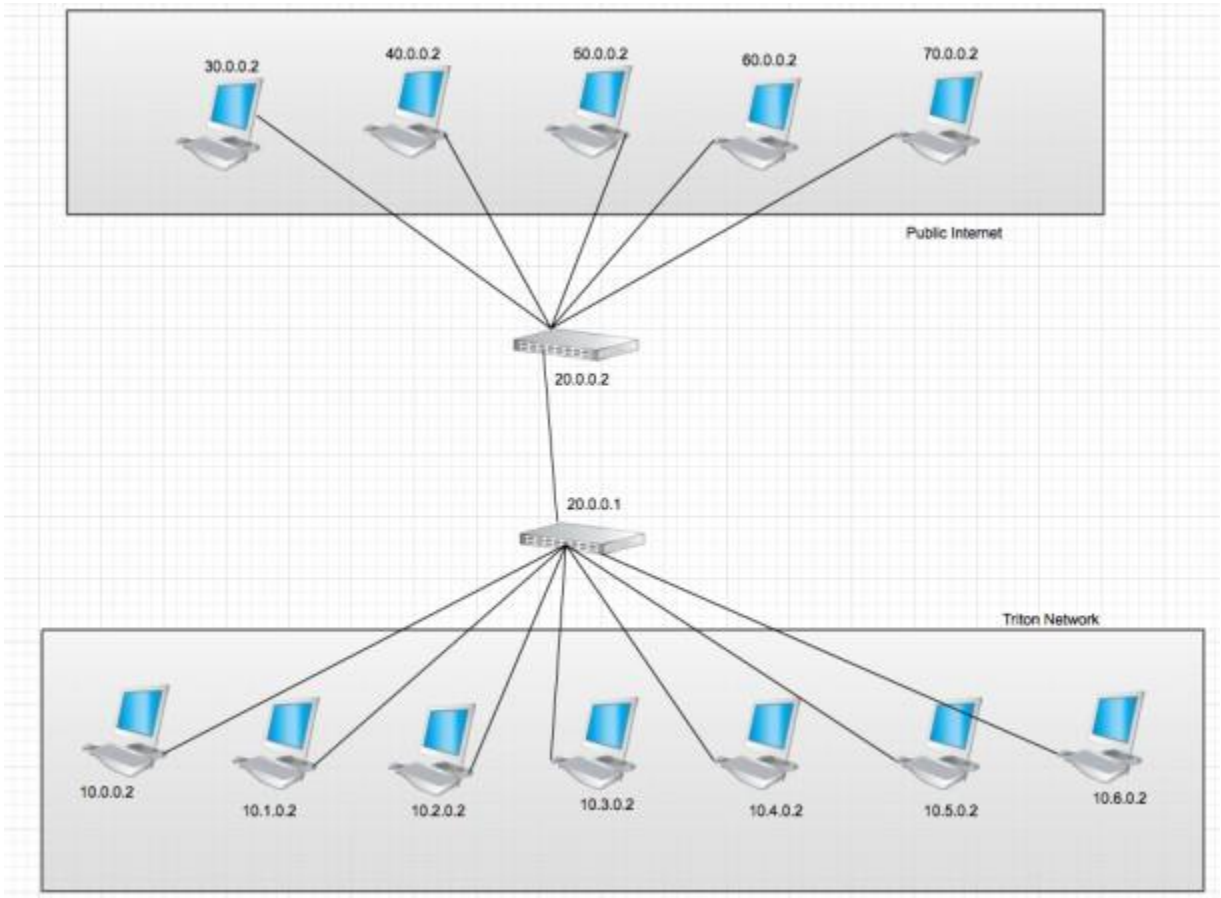
When handling ARP requests, you should only send an ARP reply if the target IP address is one of your router's IP addresses. In the case of an ARP reply, you should only cache the entry if the target IP address is one of your router's IP addresses. Note that ARP requests are sent to the broadcast MAC address (ff-ff-ff-ff-ff-ff). ARP replies are sent directly to the requester's MAC address.

New Topology

You should copy your router code to the cse123-p3 folder first. You should then also copy your router code to the cse123-p4 folder. This is the folder that contains the setup for multi-router topology. To test multi-topology, while in the cse123-p4 folder, run the run_pox.sh script, then the run_mininet.sh script.
From there, now run your sr executable but now with some parameters. First run "./sr -v sw1 -r ../rtable1" and then run "./sr -v sw2 -r ../rtable2". You can then run commands in the mininet window as usual. The topology can be seen below. You can also see more about the topology in the IP_CONFIG_TOPOA file as well as the rtable1 and rtable2 files.

You do not need to be able to use wget for this topology as there is nothing setup for wget in this topology.

The clients are the 7 hosts on the bottom, and the servers are the 5 hosts on the top. You can refer to the clients as client1, client2, client3, etc.



Code Overview

Basic Functions

Your router receives a raw Ethernet frame and sends raw Ethernet frames when sending a reply to the sending host or forwarding the frame to the next hop. The basic functions to handle these functions are:

```
void sr_handlepacket(struct sr_instance* sr, uint8_t * packet,
unsigned int len, char* interface)
```

This method, located in *sr_router.c*, is called by the router each time a packet is received. The "packet" argument points to the packet buffer which contains the full packet including the ethernet header. The name of the receiving interface is passed into the method as well.

```
int sr_send_packet(struct sr_instance* sr, uint8_t* buf, unsigned int
len, const char* iface)
```

This method, located in *sr_vns_comm.c*, will send an arbitrary packet of length, len, to the network out of the interface specified by iface.

You *should not* free the buffer given to you in sr_handlepacket (this is why the buffer is labeled as being "lent" to you in the comments). You are responsible for doing correct memory management on the buffers that sr_send_packet borrows from you (that is, sr_send_packet will not call free on the buffers that you pass it).

```
void sr_arpcache_sweepreqs(struct sr_instance *sr)
```

The assignment requires you to send an ARP request about once a second until a reply comes back or we have sent five requests. This function is defined in *sr_arpcache.c* and called every second, and you should add code that iterates through the ARP request queue and re-sends any outstanding ARP requests that haven't been sent in the past second. If an ARP request has been sent 5 times with no response, a destination host unreachable should go back to all the sender of packets that were waiting on a reply to this ARP request.

Data Structures

The Router (sr_router.h):

The full context of the router is housed in the struct sr_instance (sr_router.h). sr_instance contains information about topology the router is routing for as well as the routing table and the list of interfaces.

Interfaces (sr_if.c/h):

After connecting, the server will send the client the hardware information for that host. The stub code uses this to create a linked list of interfaces in the router instance at member if_list. Utility methods for handling the interface list can be found at sr_if.c/h.

The Routing Table *(note: this is technically a forwarding table)* (sr_rt.c/h):

The routing table in the stub code is read on from a file (default filename "rtable", can be set with command line option -r) and stored in a linked list of routing entries in the current routing instance (member routing_table).

The ARP Cache and ARP Request Queue (sr_arpcache.c/h):

You will need to add ARP requests and packets waiting on responses to those ARP requests to the ARP request queue. **When an ARP response arrives, you will have to remove the ARP request from the queue and place it onto the ARP cache, forwarding any packets that were waiting on that ARP request**. **Pseudocode for these operations is provided in sr_arpcache.h. The base code already creates a thread that times out ARP cache entries 15 seconds after they are added for you.** You must fill out the sr_arpcache_sweepreqs function in sr_arpcache.c that gets called every second to iterate through the ARP request queue and re-send ARP requests if necessary. Pseudocode for this is provided in sr_arpcache.h.

Protocol Headers (sr_protocol.h)

Within the router framework you will be dealing directly with raw Ethernet packets. The stub code itself provides some data structures in sr_protocols.h which you may use to manipulate headers easily. There are a number of resources which describe the protocol headers in detail. Network Sorcery's RFC Sourcebook provides a condensed reference to the packet formats you'll be dealing with:

- Ethernet
- IP
- ICMP
- ARP (For the actual specifications, there are also the RFCs for ARP (RFC826), IP (RFC791), and ICMP (RFC792))

Debugging Functions

We have provided you with some basic debugging functions in sr_utils.h, sr_utils.c. Feel free to use them to print out network header information from your packets. Below are some functions you may find useful:

- `print_hdrs(uint8_t *buf, uint32_t length)` - Prints out all possible headers starting from the Ethernet header in the packet
- `print_addr_ip_int(uint32_t ip)` - Prints out a formatted IP address from a uint32_t. Make sure you are passing the IP address in the correct byte ordering.

Required Functionality

- The router must successfully route **packets** between the client and the application servers.
- The router must correctly handle ARP requests and replies.
- **The router must correctly handle traceroutes through it (where it is not the end host) and to it (where it is the end host).**
- The router must respond correctly to ICMP echo requests.
- **The router must handle TCP/UDP packets sent to one of its interfaces. In this case the router should respond with an ICMP port unreachable.**
- **The router must maintain an ARP cache whose entries are invalidated after a timeout period (timeouts should be on the order of 15 seconds).**
- **The router must queue all packets waiting for outstanding ARP replies. If a host does not respond to 5 ARP requests(separated by 1 second), the queued packet is dropped and an ICMP host unreachable message is sent back to the source of the queued packet.**
- The router must not needlessly drop packets (for example when waiting for an ARP reply)
- **The router must enforce guarantees on timeouts--that is, if an ARP request is not responded to within a fixed period of time (5 ARP requests separated by 1 second each), the ICMP host unreachable message is generated even if no more packets arrive at the router. (Note: You can guarantee this by implementing the sr_arpcache_sweepreqs function in sr_arpcache.c correctly.)**
- **You must be able to handle a multi-router topology as well now.**

Additional examples

These examples show some expected traceroute and HTTP download output using the sr_solution binary.

```
mininet> client traceroute 192.168.2.2
traceroute to 192.168.2.2 (192.168.2.2), 30 hops max, 60 byte packets
 1  10.0.1.1 (10.0.1.1)  149.135 ms  147.469 ms  147.474 ms
 2  192.168.2.2 (192.168.2.2)  217.474 ms *  217.488 ms
mininet> client traceroute 172.64.3.10
traceroute to 172.64.3.10 (172.64.3.10), 30 hops max, 60 byte packets
 1  10.0.1.1 (10.0.1.1)  176.872 ms  176.724 ms  176.689 ms
 2  * * *
 3  * 172.64.3.10 (172.64.3.10)  176.748 ms  216.648 ms
```

The extra *s, and the destination IP address appearing on line 3 instead of line 2, are acceptable.

```
mininet> client wget -O- http://192.168.2.2
--2020-05-26 13:50:38--  http://192.168.2.2/
Connecting to 192.168.2.2:80... connected.
HTTP request sent, awaiting response... 200 OK
Length: 161 [text/html]
Saving to: ‘STDOUT’

 0% [                                       ] 0         --.-K/s           <html>
<head><title> This is Server1 </title></head>
<body>
Congratulations! <br/>
Your router successfully route your packets to server1. <br/>
</body>
</html>
100%[======================================>] 161       --.-K/s   in 0s

2020-05-26 13:50:38 (17.6 MB/s) - written to stdout [161/161]

mininet> client wget -O- http://172.64.3.10
--2020-05-26 13:50:56--  http://172.64.3.10/
Connecting to 172.64.3.10:80... connected.
HTTP request sent, awaiting response... 200 OK
Length: 161 [text/html]
Saving to: ‘STDOUT’

 0% [                                       ] 0         --.-K/s           <html>
<head><title> This is Server2 </title></head>
<body>
Congratulations! <br/>
Your router successfully route your packets to server2. <br/>
</body>
</html>
100%[======================================>] 161       --.-K/s   in 0s

2020-05-26 13:50:56 (1.08 MB/s) - written to stdout [161/161]
```