Lab5 - Building a Router and Dive into TCP

200 points

Lab Environment:

- Part 1 and Part2 Section B of this lab have been tested on the VM image in the lab(s). We cannot provide support for other environments – some things might work and others not etc. If you have an M1/M2 Mac which does not work properly or are otherwise unable to run the VM, please use the lab computers to complete the assignment. Also, make sure your scripts are written Python 3.
- Part 2 Section A can be run either on your own computer outside of the VM or on the VM image. If you are having trouble running Wireshark, please see your TA.

Helpful Resources:

- Computer Networks: A Top Down Approach - Chapter 3 covers TCP
- **POX Wiki**
- Inside your VM, the /pox/pox/forwarding/l2_learning.py example file
- UDP:
  - Brief UDP Summary
  - RFC 768
- TCP Stevens Graph: helpful video!

Information on screenshots: For all questions that require a screenshot, make sure that a **date timestamp** is visible next to your results. Make sure to mark up your screenshots with a clearly visible red circle to get full credit. The graders have had trouble seeing the highlighting, so the red circle is required. Thank you!

Submission:

1. **cruzid-lab5.pdf** : PDF containing answers to all questions, submitted on Gradescope.
2. **cruzid-lab5_topo.py** : Your network topology for the router submitted on Canvas.
3. **cruzid-lab5_controller.py** : Your remote controller code for the router submitted on Canvas.

Introduction: In Lab4, we learned about Software Defined Networking (SDN) and the POX Controller to design a Firewall. In the first part of this Lab, we are going to use the POX Controller to build a router. This lab again assumes you will spend time reading the documentation and learning about Mininet, OpenFlow and the POX controller.

In the second half of the assignment, we explore the internal workings of the TCP protocol.

[100 pts] Part 1: Router

Your goal will be to allow or block traffic between the different devices based on the 4 Network Topology Rules given below. In this assignment, you **cannot** use flooding (i.e., you cannot use flooding: `of.OFPP_FLOOD`)**!** Instead, you will need to specify specific ports for all traffic and the forwarding must be done by subnet, not by enumerating all of the IP addresses**.** You might consider a method to determine if an IP Address is valid on a particular subnet. Notice that the rules are written according to the subnet.

You may implement your router however you choose—although as a suggestion, you may find it easiest to determine the correct destination port by using

- the source and destination IP addresses
- or the source port on the switch from which the packet originated.

These provided files will get you started—you will need to modify both:
`lab5_topo_skel.py` / `lab5_controller_skel.py`

We will use the same topology as Lab4, which is shown in Figure 1. **Note the departments are now organized into subnets**. You will need to do the following:

- Finish the topology file by adding all the hosts (be sure to manually specify the MAC address, IP address and subnet for each host), switches and links
- Implement the router based on Rule 1, Rule 2, Rule 3 and Rule 4.

Reminder: The same naming convention for the switches and hosts used in Lab4 must be followed here.

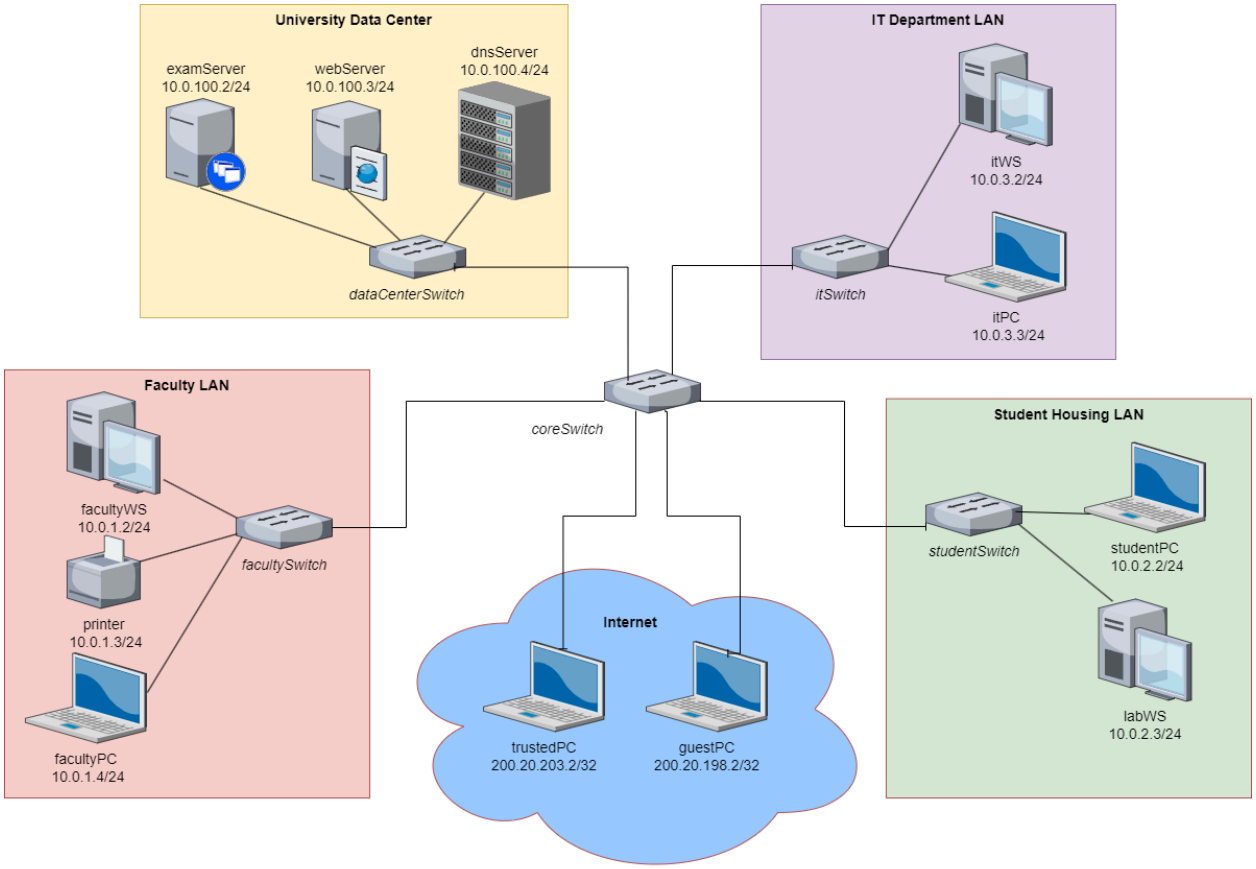

Figure 1

In your code use the naming convention below:

Device	Required Naming Convention					
	Core	**Faculty LAN**	**Student Housing LAN**	**IT Department LAN**	**University Data Center**	**Internet**
Workstations		‘facultyWS’	‘labWS’	‘itWS’		
Laptops		‘facultyPC’	‘studentPC’	‘itPC’		‘trustedPC’, ‘guestPC’
Servers					‘examServer’, ‘webServer’, ‘dnsServer’	
Other		‘printer’				
Switches	‘s0’	‘s1’	‘s2’	‘s3’	‘s4’	

Table 1 - Standard Names

Network Topology and Rules to be implemented:

- **Rule 1:** `ICMP` traffic is forwarded only between the Student Housing LAN, Faculty LAN and IT Department subnets or between devices that are on the same subnet.
- **Rule 2:** `TCP` traffic is forwarded only between the University Data Center, IT Department, Faculty LAN, Student Housing LAN**, trustedPC**, or between devices that are on the same subnet; however, **only the Faculty LAN may access the exam server**.
- **Rule 3:** `UDP` traffic is forwarded only between the University Data Center, IT Department, Faculty LAN, Student Housing LAN, or between devices that are on the same subnet
- **Rule 4:** All other traffic should be dropped.

Questions and Rule Verification:

1. Annotate the topology in Figure 1 with port numbers associated with the end of each link and include a screenshot of the detailed image of Figure 1. Then, implement the topology in the skeleton file `lab5_topo_skel.py`.

2. Explain how packets are forwarded differently in this assignment compared to Lab4 (think about your previous `accept()` function.)

After you have finished writing your code:

3. **Verify Rule** 1 with *`pingall`*. Circle in red the lines of your output that indicate the inter-departmental communication is working as expected. Explain your answer. Include a **timestamped screenshot** and refer to it in your explanation.

4. **Verify Rule 2** with *`iperf`*. Are the individual results what you expect? Why? Include a **timestamped screenshot** and refer to it in your explanation after running *iperf* between these pairs of nodes:
    - examServer and itWS
    - itWS and facultyWS
    - facualtyWS, LabWS
    - itWS to itPC
    - trustedPC and itPC
    - trustedPC and guestPC
    - guestPC and webServer

    Important: If *`iperf`* takes longer than around 15 sec to run, this usually means the hosts can't reach each other; cancel with ctrl-c and include the result in your screenshot.

5. **Verify Rule 3** with the command (`iperfudp 10M <host1> <host2>`). Are the individual results what you expect? Why? Include a **timestamped screenshot** and refer to it in your explanation after running `iperfudp` between these pairs of nodes:
    - examServer and itWS
    - itWS and facultyWS
    - facualtyWS, LabWS
    - itWS and itPC
    - trustedPC and itPC
    - trustedPC and guestPC

    Important: A `UDP` test is considered successful if the `iperfudp` output displays two UDP bandwidth measurements between hosts. If one of the UDP bandwidth measurements is missing, it means that `UDP` traffic is not flowing. For example:
    Pass: `*** Results: ['10M', '10.1 Mbits/sec', '10.1 Mbits/sec']`
    Failed: `*** Results: ['10M', '', '10.0 Mbits/sec']`

[5 pts] Extra Credit:

Create a “discordServer” host on the Internet (with an public IP of your choice) that is only accessible from the Student LAN. Prove that students may access the server but the teaching staff cannot by including a **timestamped screenshot** of the following commands

- studentPC ping discordServer
- facultyPC ping discordServer
- iperfudp studentPC discordServer
- iperfudp facultyPC discordServer
- iperf studentPC discordServer
- iperf facultyPC discordServer

[100 pts] Part 2: TCP

In this section, we explore several aspects of the TCP protocol. First, we will upload a file to a webserver and observe the flow of packets over the TCP connection. Next, we will download a large file and introduce losses on the link to observe TCP's reliable data delivery service in action. You will be TCP experts by the end!

A. Upload a Great Story to a Web Server

In this section, you will observe TCP uploading a text file. You should be able to find packets being uploaded.

Environment: Use the **VIRTUAL MACHINE** for these problems.

Follow these instructions in the order they are given - the Wireshark capture is used for questions Q6 through Q11)

- Download the ASCII copy of Alice in Wonderland http://gaia.cs.umass.edu/wiresharklabs/alice.txt and store the file on your computer. (Read if you have time- it's a great story!)
- Load the page http://gaia.cs.umass.edu/wireshark-labs/TCP-wireshark-file1.html
- Start Wireshark and begin packet capture
- Upload the alice.txt file to the webpage. Once the file has been uploaded, a short congratulations message will be displayed in your browser window. End the Wireshark capture.

6. HTTP Uploads:

a. Review HTTP data and form upload methods. What methods can be used for HTTP uploads?

b. Observe the URL in your browser during the file upload. Does it change to reflect the upload process? Based on your observation of the URL once you upload the file, which HTTP method do you think is used? Determine the actual HTTP method used for this upload in Wireshark.

Circle the method in a **timestamped screenshot** of your Wireshark output.

TCP Basics

Continuing with the Alice in Wonderland upload, answer the following questions for the TCP segments you observe in Wireshark:

7. TCP Connection Setup:

a. Illustrate the Handshake:

Using a drawing program, construct a timing diagram to illustrate the TCP 3-way handshake process. Ensure your diagram includes:

- Two endpoints labeled with the IP addresses for the endpoints: Host A (your computer) and Host B (the server)
- The time the SYN packet is sent from Host A and Host B
- The time the SYN-ACK packet is received
- The time the final ACK packet is sent from Host A to Host B
- For each packet, include the relative and raw sequence number, relative and raw acknowledgement number, and the Reciever's Advertised Window size.

Clearly label all sequence and acknowledgment numbers.

What is the RTT between your client and the server? Label the RTT in your drawing and give its value.

Include **a timestamped screenshot** and circle all the information used in the diagram.

b. Purpose of Sequence Numbers and Receiver's Window Size:
   - Explain why the initial SYN packet must include a sequence number and its continued role throughout the handshake process.
   - What is the window size returned by the Server?
   - Explain the purpose of the window size number.

c. SYN-ACK Segment Details:
   - Why are the sequence numbers different in the SYN and SYN-ACK packets?
   - Explain how the acknowledgment number is calculated in the SYN-ACK packet.

d. ACK Segment Details:
   - Explain how the sequence number in the final ACK packet is determined.
   - What is the acknowledgment number in the final ACK packet referencing?

Ready to Send! – TCP File Upload

Now that the handshake is complete, we can look at the data segments.

8. Beginning the File Upload:

Hunting for the TCP Payload (a) and (b):

a. Find the first TCP segment that carries this HTTP message for the file upload in (Q6c). Convince yourself that the payload of this packet is the HTTP request message by analyzing this TCP packet. Find the bytes of the payload as we showed in lab.

What is the TCP **segment length** and **relative** sequence number of this TCP segment?

b. Next comes the data! Find the first TCP segment carrying part of the file transfer – it carries the very beginning portion of the actual Alice in Wonderland text. Convince yourself that the payload of this packet is the contents of the Alice in Wonderland file (find the bytes of the payload as we showed in lab).

   What is the TCP **segment length** and **relative** sequence number of this TCP segment?

   Screenshot: Demonstrate your findings in a **timestamped screensho**t that highlights the data that you see (visible in the payload bytes) and also the segment length and relative sequence number of the segment.

c. Look at the TCP **segment length** and **relative** sequence numbers for several subsequent segments (following the one in (b)). Can you discern a pattern? Explain how sequence numbers are calculated in subsequent segments and what they represent in the context of the TCP stream.

9. Graphing the RTT (Round Trip Time):
Wireshark has a nice feature that allows you to plot the RTT for each of the transmitted TCP segments. Click on a TCP segment that is being sent from the client to the gaia.cs.umass.edu server.
Navigate to: Statistics -> TCP Stream Graphs -> Round Trip Time to visualize the RTT data.
Note: The direction of the transfer is important! Switch direction until you display measurements of the transmitted segments.

a. How would the values shown in this graph be used as a part of TCP's retransmission timeout (RTO) calculation?

b. Discuss your observations in your RTT Graph. (Think about the discussion in class related to TCP's RTO calculation and the graph that was discussed. Chapter 3 is also a good reference.)

   Include a **timestamped screenshot** of this graph.

TCP Packet Loss and Congestion Control

10. Time-Sequence-Graph(Stevens): Select a TCP packet with len != 0 that has been transmitted by the client and use the Time-Sequence-Graph(Stevens) plotting tool to create a graph.

a. How many bytes are in the entire TCP segment you selected? (payload + header)

b. Explain what is displayed on the x and y-axis in the graph. What does each dot represent?

c. Put a red circle around any region where Slow Start begins.

d. Are there any retransmitted segments in your graph? If so, circle them in red on the graph. Make sure your graph shows the entire transfer and the lost segments, if any.

   Include a **timestamped screenshot** of this graph.

11. Average Transmission Rate for File Upload

    Based on the Time-Sequence-Graph(stevens), what is the average transmission rate (over the duration of the file upload)? Explain how you calculated this value (your process) and support your calculation by highlighting in your **timestamped screenshot**.

B. Downloading a File from a Server (with packet loss)

Environment: Use the **VIRTUAL MACHINE** for these problems.

In this section, we will examine how loss affects a TCP connection. Loss will be simulated with the `tc qdisc` command to apply a loss rate to an incoming interface. We will observe TCP's response to packet loss.

No introduced losses:

12. In this problem we will not intentionally introduce any loss. Start Wireshark and use `wget` to download this 10MB file.

a. Is packet loss possible during this download?

b. Select a TCP data segment received by your client and create TimeSequenceGraph(Stevens). Include a **timestamped screenshot** of the graph and highlight important areas of the graph such as transfer begin/end, retransmitted packets etc. Write a short summary of your observations.

c. Average throughput: Calculate the average throughput (consider the entire duration until the file transfer completes). Show your calculations and support with a **timestamped Wireshark capture with two separate highlights:** 1) the first packet and 2) the last packet.

Introducing Loss:

Now we will simulate loss by using the command ***`tc qdisc`*** on an interface. When the command is first used, you must use ***add dev*** for the interface being changed. After adding the interface, use ***change dev*** to set the loss rate.

The following sequence of commands can be used to simulate 100% loss on the `eth0` interface:

- `sudo tc qdisc add dev eth0 root netem loss 0%`
- **Change loss to 100%**
  `sudo tc qdisc change dev eth0 root netem loss 100%`
- **Change loss back to 0%**
  `sudo tc qdisc change dev eth0 root netem loss 0%`

Really dismal – 100% loss event:

Read through this paragraph before starting:

- Start Wireshark
- Open 2 terminals and have commands typed and ready before you begin:
  - Terminal 1: Download the 10MB.zip file using wget
  - Terminal 2: While the download is in progress, change loss to 100%. After a second, change loss back to 0%.

13. Answer the following questions by examining the Wireshark trace.
    Find a TCP data segment received by your client and create a TimeSequenceGraph(Stevens) graph with this packet selected.
    a. At what time in your graph does packet loss begin? At what time does it end?

    b. Search for an area of the graph where there is a retransmission. What Sequence Number is retransmitted?

       Timestamped screenshot: Circle in red the region where 100% loss begins and ends in the graph. Identify the retransmitted packet (label it) in your screenshot.

    c. Calculate the average throughput (consider the entire duration until the file transfer completes). Show your calculations and support with the **timestamped Wireshark capture** by circling in red the first and last packet used for your calculations.

Not so bad? – 20% loss event:

14. Restart Wireshark and retry the experiment, this time with 20% packet loss. Introduce 20% loss and then quickly return to loss 0%. Find a TCP data segment received by your client and create a TimeSequenceGraph(Stevens) graph. Note: **you can use the Zoom function** to zoom in on an area of the graph to understand what is happening.

Attach a **timestamped screenshot** that includes markup for (a) and (c). It is ok if you want to use two screenshots.

a. At what time does the region which is experiencing 20% loss begin and end? In your screenshot, put a red box around this region.
b. Examine in detail (zoom when necessary) the areas in the graph at the beginning of the data transfer and after the period of loss. What do you notice about these regions in the graph with respect to the TCP congestion control algorithms?

c. How do you expect to determine where there has been a retransmission? Search for a retransmission in the graph. What Sequence Number is retransmitted? Circle and label a retransmitted packet in your screenshot.

d. Calculate the average throughput (consider the entire duration until the file transfer completes). Show your calculations and support with your **timestamped Wireshark capture** by **circling in red** the first and last packet used for your calculations.

Comparison of Download Performance:

15. Make a table comparing the loss rates in the Q12, 13 and 14. Columns should show your throughput calculation and the total time it took to download the 10MB file.

Discuss your results – Do your results make sense to you? What does the difference in throughput tell you about the operation of TCP during periods of congestion or lossy links?