RTP Streaming and NAT (Network Address Translation) Traversal (10% + 2% Bonus Points)

Deadline: May 1st , 2017

This assignment contains three parts: (1) Wireshark Labs, (2) UDP Streaming, and (3) NAT Traversal. Each part accounts for 4%, for a total of 12%.

Part 1: Wireshark Labs

To gain a solid understanding on network protocols, and to effectively troubleshoot, we have to look into the *bytes* transferred over the wire (or air medium). In addition, for fair comparisons when evaluating different algorithms and systems, we often have to collect *network traces* and replay the *same* traces in our testbeds or simulators multiple times. Being able to capture the *bytes* on the wire (or air medium) is therefore a very crucial skill for networking engineers like you.

Several tools are handy when it comes to collecting and analyzing packets. First, libpcap (http://www.tcpdump.org) is a packet capture library (hence, the name pcap), which allows you to access all the packets to/from the end host or other hosts (sniffering). For the latter case, libpcap (and the underlay system software and network drivers) puts the network cards in the *promiscuous* mode, so that the network cards no longer throw away packets to/from other hosts. Typically, we use libpcap via a command line tool, called tcpdump (http://www.tcpdump.org) , in order to save the packets into a trace file. Notice that, by default tcpdump only saves the prefix (parts of header fields) of the packets, but you can configure it to save more (say the payloads) through command line arguments. Like most Linux tools, *man page*s are your best friends.

However, libpcap/tcpdump only show you the binary without: (i) graphical user interface and (ii) rich protocol parser. Such limitation in turn makes reading through the captured (raw) packets very challenging. For one example, each TCP stream contains a bunch of IP packets. How to reassemble these IP packets back to a TCP stream is already very challenging. On top of that, depending on the application-layer protocol, we need different approach to interpret the TCP payload. This is why we need wireshark (https://www.wireshark.org), which is a frontend of libpcap/tcpdump. Wireshark is fairly user friendly, but because of its rich feature set, it may take you some time to get familiar with it. Nonetheless, being able to master wireshark is very important to networking engineers. For example, in the latter two parts of this project, in order to understand have the system works, I will ask you to use wireshark to collect and analyze the packets between the RTP streaming server and client, and between end hosts and NAT boxes.

For this part of the project, please check out the excellent tutorials from UMass at: http://www-net.cs.umass.edu/wireshark-labs/ . Please go through the following four topics:

1. Getting Started, v7.0 (PDF, Word)
2. TCP, v7.0 (PDF, Word)
3. NAT, v7.0 (PDF, Word)
4. 802.11, v7.0 (PDF, Word)

You are *highly* encouraged to finish all the questions in these four topics. For the project report, please turn in the answers of the following 8 problems: (i) 2 and 4 in Getting Started, (ii) 9 and 13 in TCP, (iii) 5 and 9 in NAT, and (iv) 6 and 7 in 802.11. Please write down you answers, ideally with screenshots or figures for illustrations, in the project report. Please submit the report in PDF format.

Part 2: UDP Streaming

In this part, you will implement a UDP streaming system using a mature streaming library called live555 (http://www.live555.com/liveMedia/), although that's not the only library you will use. You will need a server program that reads video files and sends video packets. You will write a client problem with GUI interface to receive the video packets, decode the video frames, and render the frames to users. To make your life easier, please use live555MediaServer as the streaming server. For the client side, have to integrate live555, FFmpeg, and Simple DirectMedia Layer (SDL: https://www.libsdl.org/). These three libraries are for: *streaming, decoding, and rendering (displaying),* respectively.

The recommended steps for completing this part are:

1. Compile Live555 and FFmpeg on your favorite Linux distribution.
2. Run the live555MediaServer program. You just need to put your video in the folder where live555MediaServer located.
3. Try and read the sample code named *testRTSPClient* in live555.
4. Integrate FFmpeg with the RTSP client. Use FFmpeg to decode the frame when the (live555) client gets a frame. You probably want to decode the frames into the YUV420 format.
5. Add SDL to the RTSP client. You need to create a GUI interface to play the decoded YUV420 videos in real-time.

There are some tutorial you may want to check out:

- http://www.live555.com/liveMedia/#config-unix
- https://trac.ffmpeg.org/wiki/CompilationGuide/Ubuntu
- http://wiki.libsdl.org/Installation

There are some addition libraries you may need, in order to compile your program on your Linux: libsdl1.2-dev, liblzma-dev, libfaac-dev, libfdk-aac-dev, libav-tools, zlib1g-dev, libswresample-dev. For Debian-like distirbutions, they can be installed using *apt-get*.

In this part, you need to submit the following files:

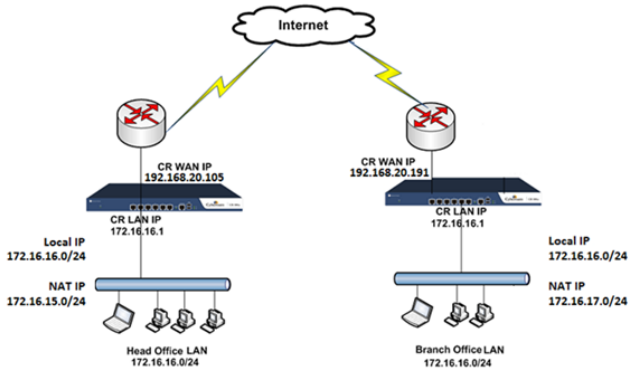
1. A report with how to compile your program, how you implement it, and the problem you faced.
2. Your source code (.cpp) and Makefile (or any file you used for compiling the binary).

Part 3: NAT Traversal

This part contains two challenges. First, find an NAT box, such as a WiFi access point, a Linux box with iptable, or a VM hypervisor, and run your streaming server on the public IP domain (WAN port), and your streaming client on the private IP domain (LAN port). Use wireshark to capture the packets while you stream a short video. Analyze the captured packet traces and discuss:

- What are the protocols used to setup and carry out the streaming session?
- How does the UDP streaming server sends the packet through the NAT box? Observe the IP header of the UDP packets before and after passing through the NAT box.

Please answer these two questions and elaborate your answers in your report.



Second, to make the problem even more exciting, let's find *two* NAT boxes, and put both the streaming server and client behind the NAT boxes (one behind each NAT box). The above figure illustrates a sample setup. Please try again to see if you UDP streaming server can still deliver the video to the client. Note that both sender and receiver now have private IP addresses. Please perform the following two tasks:

- Use wireshark to analyze what happened, and propose solution to solve the problem. Hint: you may want to study the NAT traversal approaches, such as connection reversal, NAT hole punching, and NAT configuration protocols.
- Enhance your streaming server and client to enable UDP video streaming over two NAT boxes.

In your report, please present your testbed in details. Please discuss all the NAT traversal approaches. Please justify your decision on which approach to implement. Please explain how you implement it, and report some experimental results.