NTNU
Norwegian University of
Science and Technology
Department of Information Security
and Communications Technology

TTM4100

Communication – Services and Networks

Lab 7: “RTP”

This lab is one of the 8 programming labs in this course. You must deliver and pass at least 6 of these labs to be qualified for the exam. Any questions about the exercises can be posted on the forum or you can come to the tuition. The exercises are designed to focus on learning and understanding – not debugging code. All lab-answers should be delivered on the blackboard for review.

Deadline of submission: 15.04.2018

Lab 7: RTP

By the end of this lab, you will have acquired a better understanding of how pulse-code modulation (PCM) audio is streamed through network with RTP protocol.

The real-time transport protocol (RTP) used to transfer real-time data is usually used in conjunction with the RTP control protocol (RTCP) to exchange control information, such as reports, for the data stream between the sender and the receiver. In addition, several media servers use the real-time streaming protocol (RTSP) for session establishment and stream controls, such as play, stop, pause, teardown. Then the stream is transferred using RTP and RTCP. To avoid complexity, this lab focuses on only RTP does not include RTCP and RTSP.

This lab consists of two programs: RTP server and RTP client. The real-time data to transfer is converted from a file, in the server program, with any supported audio file format to pulse-code modulation (PCM or in Windows, WAV) audio data. The server divides the audio data into chunks and packetizes each of them then send it over UDP to a client with specified IP address and port. Once the server starts to send out a stream, it cannot be paused, rewinded, or fast forwarded. We have to either wait until the whole stream is sent out, or terminate the program. Moreover, it does not wait for the client to receive packets, but just keeps sending out until finishing. When the client runs, it receives each packet from the server, extracts the header and the payload, then displays data in the header and put the payload data into play immediately.

Required Programs and Modules to Install

This programming lab requires that your computer have FFMpeg program and PyAudio Python module installed. FFMpeg is used for audio conversion that is used behind the scene by the AudioSegment class as you can see in the source code. The purpose is to convert an input audio file of any supported format to PCM/WAV. PyAudio Python module provides audio stream playback feature that is used by the client program when it receives packetized audio data from the server. **After the installation, you can test whether the program and the module work correctly by running the provided program "TestAudio.py" and it should play a segment of "TheGall.mp3" and you should hear the music. Then you are good to run your RTP server and client code smoothly (if there are no erros). Make sure that you have placed the directory "pydub" in the same directory/folder as the TestAudio program.**

For Mac OS X

It is convenient to have Homebrew installed on your OS X machine before installing the following programs because then you can get Homebrew to install those programs. If you still do not have Homebrew on your machine, follow the instruction on its website: https://brew.sh/

Installing FFMpeg

Mac OS X

1. Make sure you have installed Homebrew mentioned above and updated it already.
2. In a terminal, type `brew install ffmpeg --with-ffplay`. If that does not work, do the following steps:
    a. `brew install sdl --use-gcc`
    b. `brew uninstall ffmpeg`
    c. `brew install ffmpeg --use-gcc`
3. Check if ffmpeg is recognized by the system by typing "ffmpeg" without quotes and hitting the return (enter) key. If it shows a description how to use ffmpeg, then you are good to go.

Linux

1. Use package management of your Linux distribution to install "ffmpeg" or install from downloaded package, or build it from source code available at https://www.ffmpeg.org/download.html .
2. Check if ffmpeg is recognized by the system by typing "ffmpeg" without quotes and hitting the return (enter) key. If it shows a description how to use ffmpeg, then you are good to go.

Windows

1. Download a Windows build from FFMpeg's build page: https://ffmpeg.zeranoe.com/builds/ Note that you have to choose the same architecture as your machine's. A stable version (a version with format X.X.X) with static linking is recommended.

2. Extract the ZIP package and you will get a folder with the same name. Change the folder name to only "ffmpeg" without quotes.

3. Copy or move the entire folder to "Program Files".

4. Copy the full path to "bin" folder under "ffmpeg"; it should be "C:\Program Files\ffmpeg\bin" and add this path to the **Path** system variable of Windows.

    a. To copy a full path of a file or folder, see http://www.howto-connect.com/copy-path-of-a-file-or-folder-in-windows-10/ .

    b. To add a new path to **Path** system variable, you can following the instruction in the page https://www.computerhope.com/issues/ch000549.htm .

5. Check if the Windows recognize the programs under "C:\Program Files\ffmpeg\bin" by running the command prompt and typing "ffmpeg" without quotes and hitting the return (enter) key. If it shows a description how to use ffmpeg, then you are good to go. Note that you have to open a new command prompt window after finish the installation.

Installing PyAudio

Mac OS X

1. In a terminal, type the following lines to install portaudio first using Homebrew then install PyAudio module using Python's pip.

    a. `brew install portaudio`

    b. `pip install pyaudio`

Linux

1. In a terminal, type the following line to install PyAudio using Python's pip.

    a. `brew install portaudio`

Windows

1. In a command prompt, type the following line to install PyAudio using Python's pip.

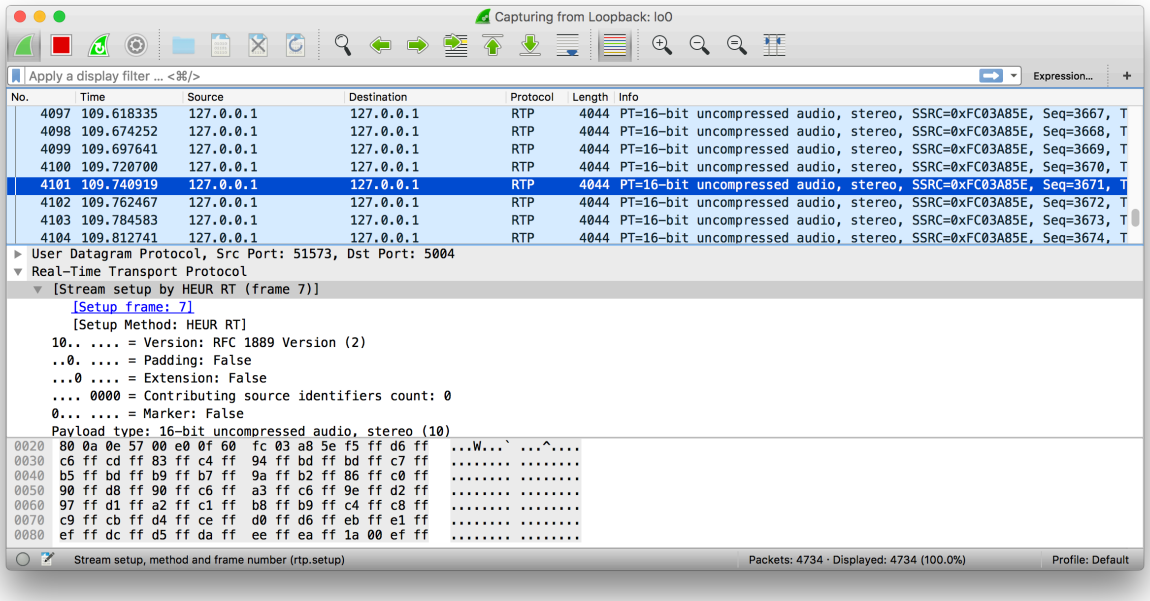
    a. `python -m pip install pyaudio`

You can consult the following webpage for more details https://people.csail.mit.edu/hubert/pyaudio/ .

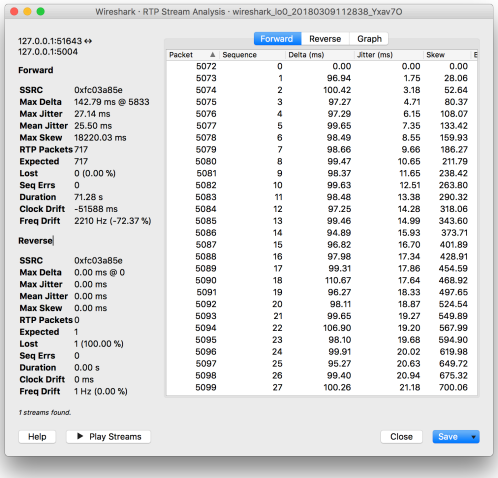
Tasks

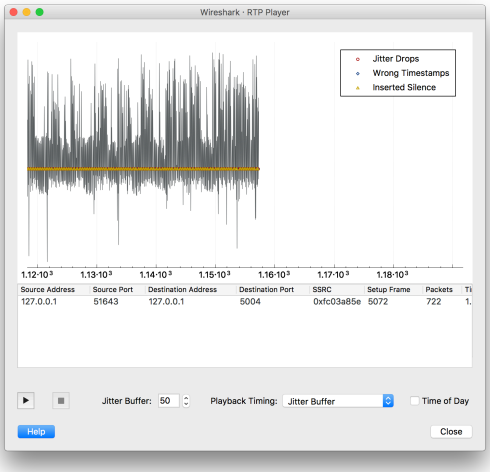
1. Make sure that you have placed the directory **pydub** in the same directory/folder of the client and server programs. The source code is complete, both server and client. You do not have to fill in any lines in the source code, but just run them. Test them with "TheGall.mp3". If it is successful, when you run both programs, either on the same or different machines, you will hear the music on the machine running the client program. If the music is not smooth, or too fast, or too slow, you can try adjusting N_FRAMES and/or SENDING_INTERVAL_MODIFIER in the server program. The description of these variables are embedded in the program as comments.

2. To run both client and server with all 4 different payload types provided in the program. Then capture packets between these programs using Wireshark. If your header is correct, Wireshark should show "RTP" in protocol column and correct RTP payload type in info column. Then capture this screen with any selected packet, also header information expanded as shown below. Therefore, you will have 4 captured screens, each of which is of a payload type.



3. To run both client and server with either PCMA or PCMU payload type and capture the packets using Wireshark. Then play a stream back on Wireshark via Telephony -> RTP -> RTP Streams ->*select a stream*-> Analyze -> Play Streams, then select RTP timestamp and click play.





Then capture 2 screens as shown above to demonstrate that you get this type done correctly. Note that Wireshark only supports PCMA and PCMU (both alternatively referred to as ITU-T G.711) in its RTP playback feature. You cannot playback any other RTP payload types in Wireshark.

Code

You can find the source code in the files "RTPServer.py" and "RTPClient.py". You do not have to fill in any part of the source code, but just run them.

What to Hand in

1. Four screen captures, each of which is for each selected payload type, of a fragment of the output from the client.

2. Four screen captures of Wireshark, each of which is for each selected payload type, correctly showing the protocol and payload type as described in 2. of the **Tasks** section above.

3. Two screen captures as described in 3. of the **Tasks** section above.

How to Run the Programs

Since these programs require command line arguments, you have to run it on a terminal or a command prompt. For instance, `python RTPClient.py -f TheGall.mp3 -c 10.33.22.123 -p 5007 -t l16_2`, providing that your script is saved as `RTPClient.py`.

Arguments are

- `-f` : audio file (only server program, required)
- `-c` : client IP address (both server and client, required)
- `-p` : client port number (both server and client, optional, default 5004)
- `-t` : payload type (both server and client, optional, default l16_2)