Team 5
Catherine Smith
John Robertson
Zak Hussain

Abstract/Summary

It is our intention to build a small-scale, live streaming application. Rather than focus on a complete front-end framework, we directed our attention to the transport/network layer protocol for communicating stored data between a server and client. The client initializes communication with the server from a simple Gui built using the tkinter python library; at which point the server will stream content from a .Mjpeg file to the client host. In terms of protocols and techniques, we initialize a transport protocol using the real-time streaming protocol (RTSP). The video data is encoded within a real-time transport packet (RTP). To evaluate our network, we based performance on observations made about the quality of the streamed data (i.e., does the video contain jitter). Upon completion of building a basic server-to-client connection, we will have a better understanding of what evaluation means are best practice when using the RTP.

Reason for Exploring Live Stream Networks

Live streaming content has become a prominent means of disseminating information in the modern age. Platforms that originally focused on providing users with on-demand content have, over the last few years, added livestream functionality. This list includes platforms such as Instagram, YouTube, twitch, etc.

Given the integration of live stream functionality into major social media platforms, we, the members of team 5, see potential in taking time to better comprehend how these platforms are built from a networking perspective.

Project Activity (Roles)

The roles where be built in three stages. They are as follows:

<u>Stage 1 - Research</u>

During this stage, each team member was responsible for gathering information shared and discussed at the weekly meetings. The important focuses for doing research were to:

(1) Identify the high-level layout of how livestreamed networks are

Used by major platforms.

(2) Identify what protocols are generally used to stream live media

content—UDP, RTP, H.264.

(3) Identify which tools exist to construct our own,

controlled, live-stream application-based network (i.e. would we need additional hardware or software aside from what we have been using in CS5700?).

(4) Identify how to work around video properties that affect the size of files we will transport:

* Bit Rate (from 150 to 2,000 Kbps and higher).
* Resolution (from 480×270 to 1280×720 and higher).
* Sample Rate (22,050 / 44,100 / 48,100 Hz).
* Stereo Audio Bitrate (from 48 to 320 Kbps)
* Frame Rate (30 / 25 fps).

(5) Identify workarounds for factors that affect Streaming performance and decide how many simultaneous connections our server can support.

* decoding/recoding time on processor
* Processor time for downgrading quality.
* Video and Audio processing.
* Disk Space for saving.
* Server/user side bandwidth.

Stage 2 – Implementation

During this stage we divided our workload into 4 tasks:

Client-side code - client will receive request from web app layer to

view livestream content. The client will communicate to the server.

Server-side code - The server will transmit live stream content to the client.

Gui application layer - a simple interface for triggering the client-server interaction. This is where the user sees the live stream content. It also contains the event handlers that change the state of the client-server interaction.

Camera application - content streamed from raspberry pi camera module to the server. This was not implemented within the scope of our project. Instead we used a pre-recorded .Mjpeg file proveded by Kurose.

Stage 3 - Iterate

We hoped to get to this stage to achieve the following:

* build an emulator to simulate loss packets.

* apply concealment measures to allow content to show without jitter and delay.

From our research we have become aware of how we could better structure our network architecture to a more effective object-oriented approach. This would allow for us to easily expand on the features/functionality of our system, without crashing the entire codebase.

Outcomes

We did not develop a full-fledge, streaming service set by the end of the semester; rather, we were able to develop a strong intuition on how to effectively stream content across hosts. This includes understanding what factors cause congestion, dropped frames, and corruption across these networks- which now that we are aware, we can move towards concealing these factors within out service.

Evaluation (means of gauging progress)

The only evaluation we ran was that we could see the video streaming clearly and without delays on a local host server. Our next step would be to emulate a network of routers. This would allow us to take next steps in implementing concealment methods to allow our content to stream smoothly over an emulated network.

Resources (Not Cited Using Proper Academic Citations, Just Useful Links):

Kurose textbook Chapter 9 on multimedia transport.

Example from a company that show how they went about setting up a streaming app with pre-existing libraries on web, android, and ios (doesn't give much insight into the networking.

https://gbksoft.com/blog/how-to-develop-live-streaming-app-for-business/

Web

https://github.com/gbksoft/ampache

https://github.com/gbksoft/MediaStreamRecorder

Mobile

https://github.com/gbksoft/libstreaming – Android

https://github.com/gbksoft/Player – iOS

More technical video on streaming with python socket.

https://www.youtube.com/watch?v=bWwZF_zVf00

The main transport protocol for streaming: https://en.wikipedia.org/wiki/Real-time_Transport_Protocol

More geared towards the network architecture containing RTP :

https://www.youtube.com/watch?v=l71kGy69S8g (30 minute vid) https://www.youtube.com/watch?v=9dRY2_NmAuc (10 minute vid)

https://www.youtube.com/watch?v=z8yMSN9H2UE (RTSP and H.323)