Media Streaming using Socket Programming

Bommisetti Vara Sai Venkat
AM.EN.U4AIE20019
Amrita Vishwa Vidhyapeetham
Amritapuri

Devan Manoj
AM.EN.U4AIE20020
Amrita Vishwa Vidhyapeetham
Amritapuri

Dhanush Krishna R
AM.EN.U4AIE20021
Amrita Vishwa Vidhyapeetham
Amritapuri

amenu4aie20019@am.students.amrita.edu,amenu4aie20020@am.students.amrita.edu,amenu4aie20021@am.students.amrita.edu

***Abstract*—In this project we have developed a socket programming model for streaming video and audio. Our model supports two kinds of functionalities which include streaming a video and live webcam stream. Streaming a video with audio basically means the server system transfers a video to the client system and they can view the stream. In live streaming, a cache server exists, in which multiple clients will be able to access the live stream. The backbone of the entire data transfer system is socket programming as data transfer done here uses packets which are sent out by the server and received by the client using sockets.**

I. Introduction

Streaming refers to any media content live or recorded delivered to computers and mobile devices via the internet and played back in real time. Podcasts, webcasts, movies, TV shows and music videos are common forms of streaming content.Music, video and other types of media files are prearranged and transmitted in sequential packets of data so they can be streamed instantaneously. And unlike traditional downloads that are stored on your device, media files are automatically deleted once you play them.Live streaming is the broadcast of an event over the internet as it happens. Awards shows, sports, boxing matches, video games and special one-time events are the most popular types of live streaming with an ever-growing menu of topics.Social media platforms and others broadcast everything from celebrity events, promotions and life streaming to streaming between users. You can live stream on any compatible smartphone, tablet, TV, computer or gaming console with a relatively fast internet connection. Media streaming is one platform in which one can share video or audio using internet . It can either be live streaming or recorded which is being stored and streamed later .

II. Overview and Objective

Streaming media is video or audio content sent in compressed form over the Internet and played immediately, rather than being saved to the hard drive.. With streaming media, a user does not have to wait to download a file to play it. Because the media is sent in a continuous stream of data it can play as it arrives.

III. Socket Programming

Socket programming is a way of connecting two nodes on a network to communicate with each other. One socket(node) listens on a particular port at an IP, while the other socket reaches out to the other to form a connection. The server forms the listener socket while the client reaches out to the server. They are the real backbones behind web browsing. In simpler terms, there is a server and a client. Socket programming is started by importing the socket library and making a simple socket.

```
import socket
s = socket.socket(socket.AF\_INET,
    socket.SOCK\_STREAM)
```

Here we made a socket instance and passed it two parameters. The first parameter is AF_INET and the second one is SOCK_STREAM. AF_INET refers to the address-family ipv4. The SOCK_STREAM means connection-oriented TCP protocol. Now we can connect to a server using this socket.3

We can find ip by the below code snippet.

```
import socket
ip = socket.gethostbyname('www.google.com')
print ip
```

First of all, we made a socket. Then we resolved google's IP and lastly, we connected to google. Now we need to know how can we send some data through a socket. For sending data the socket library has a function which allows you to send data to a server to which the socket is connected and the server can also send data to the client using this function. Sockets and the socket API are used to send messages across a network. They provide a form of inter-process communication (IPC). The network can be a logical, local network to the computer, or one that's physically connected to an external network, with its own connections to other networks. The obvious example is the Internet, which you connect to via your ISP.

A. Background

Sockets have a long history. Their use originated with ARPANET in 1971 and later became an API in the Berkeley Software Distribution (BSD) operating system released in 1983 called Berkeley sockets.

When the Internet took off in the 1990s with the World Wide Web, so did network programming. Web servers and browsers weren't the only applications taking advantage of newly connected networks and using sockets. Client-server applications of all types and sizes came into widespread use.

Today, although the underlying protocols used by the socket API have evolved over the years, and new ones have developed, the low-level API has remained the same.

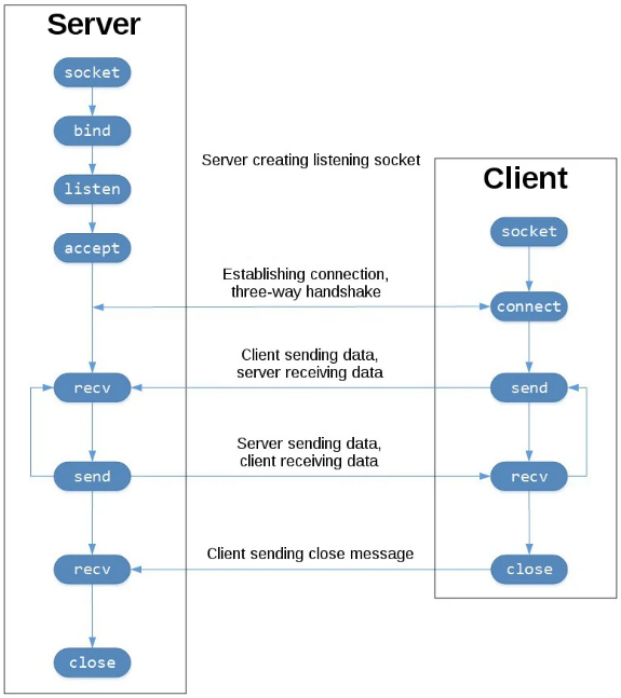
The most common type of socket applications are client-server applications, where one side acts as the server and waits for connections from clients. This is the type of application that you'll be creating in this tutorial. More specifically, you'll focus on the socket API for Internet sockets, sometimes called Berkeley or BSD sockets. There are also Unix domain sockets, which can only be used to communicate between processes on the same host.

B. Socket API Overview

Python's socket module provides an interface to the Berkeley sockets API. This is the module that you'll use in this tutorial.

The primary socket API functions and methods in this module are:

socket()
.bind()
.listen()
.accept()
.connect()
.connect$_ex()$
.$send()$
.$recv()$
.$close()$



TCP socket flow.

The left-hand column represents the server. On the right-hand side is the client.

Starting in the top left-hand column, note the API calls that the server makes to set up a "listening" socket:
socket()
.bind()
.listen()
.accept() A listening socket does just what its name suggests. It listens for connections from clients. When a client connects, the server calls .accept() to accept, or complete, the connection.

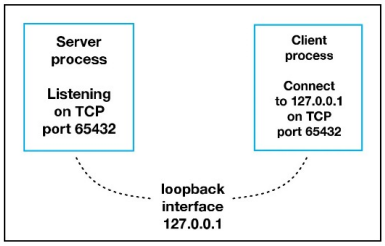
The client calls .connect() to establish a connection to the server and initiate the three-way handshake. The handshake step is important because it ensures that each side of the connection is reachable in the network, in other words that the client can reach the server and vice-versa. It may be that only one host, client, or server can reach the other.

In the middle is the round-trip section, where data is exchanged between the client and server using calls to .send() and .recv().

At the bottom, the client and server close their respective sockets.

Communication Breakdown

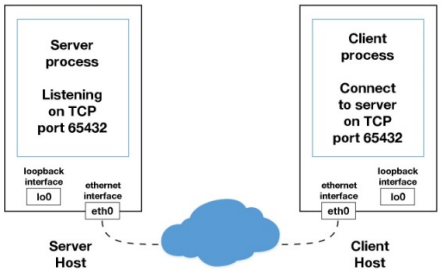
Now you'll take a closer look at how the client and server communicated with each other:



When using the loopback interface (IPv4 address `127.0.0.1` or IPv6 address `::1`), data never leaves the host or touches the external network. In the diagram above, the loopback interface is contained inside the host. This represents the internal nature of the loopback interface and shows that connections and data that transit it are local to the host. This is why you'll also hear the loopback interface and IP address `127.0.0.1` or `::1` referred to as

You can see this in action if you have an application server that uses its own private database. If it's not a database used by other servers, it's probably configured to listen for connections on the loopback interface only. If this is the case, other hosts on the network can't connect to it.

When you use an IP address other than `127.0.0.1` or `::1` in your applications, it's probably bound to an Ethernet interface that's connected to an external network. This is your gateway to other hosts outside of your "localhost" kingdom:



Be careful out there. It's a nasty, cruel world. Be sure to read the section Using Hostnames before venturing from the safe confines of "localhost." There's a security note that applies even if you're not using hostnames but are using IP addresses only.

C. Types of socket programming

1) Datagram Socket: Datagram Sockets: Datagram sockets allow processes to use the User Datagram Protocol (UDP). It is a two-way flow of communication or messages. It can receive messages in a different order from the sending way and also can receive duplicate messages. These sockets are preserved with their boundaries. The socket type of datagram socket is SOCK_DGRAM.

2) Stream Sockets: Stream socket allows processes to use the Transfer Control Protocol (TCP) for communication. A stream socket provides a sequenced, constant or reliable, and two-way (bidirectional) flow of data. After the establishment of connection, data can be read and written to these sockets in a byte stream. The socket type of stream socket is SOCK_STREAM.

TCP stands for Transmission Control Protocol a communications standard that enables application programs and computing devices to exchange messages over a network. It is designed to send packets across the internet and ensure the successful delivery of data and messages over networks.

3) Raw Sockets: Raw Socket provide user access to the Internet Control Message Protocol (ICMP). Raw sockets are not used for most applications. These sockets are the same as the datagram oriented, their characteristics are dependent on the interfaces. They provided support in developing new communication protocols or for access to more facilities of an existing protocol. Only the superusers can access the Raw Sockets. The socket type of Raw Socket is SOCK_RAW.

4) Sequenced Packet Sockets: Sequenced Packet Sockets are similar to the stream socket, with the exception that record boundaries are preserved in-stream sockets. The given interface in this section is of Network System (NS) that has an abstraction of Sockets and is ordered in all the applications. The Sequenced Packet Sockets enable the user to multiply the sequence packet protocol or some IDP (Internet Datagram Protocol) which heads on the packet or a packet group by writing in the header of the prototype along with the data that has been sent. The socket type of Sequenced Packet Socket is SOCK_SEQPACKET.

IV. Cache-server

A dedicated serve acting as a storage of web content commonly for local area networks.Provides faster content closer to the local vicinity.Fetches content from remote source for future requests.Provides computation offloading from the remote server.Handles client requests, and only ask remote source if it doesn't have the requested content

V. Methodology

We included two kinds of functionality which include streaming a video and live video stream through web camera.We have 3 parts in our code client,server and cache server.In the Server code we have given two parts webcam and video streaming in the webcam live stream we create a server socket and define to a port streaming the live video as packets to cache server and through cache server multiple clients can access the live stream and in the video stream part we stream a video in this we will divide video and audio and transfer it separately we will create a temp.wav file in the directory and stream them simultaneously using sockets

VI. Results and Conclusion

We have implemented a media streaming model using socket programming which allows to stream live video using webcam and stream video with audio through socket path.

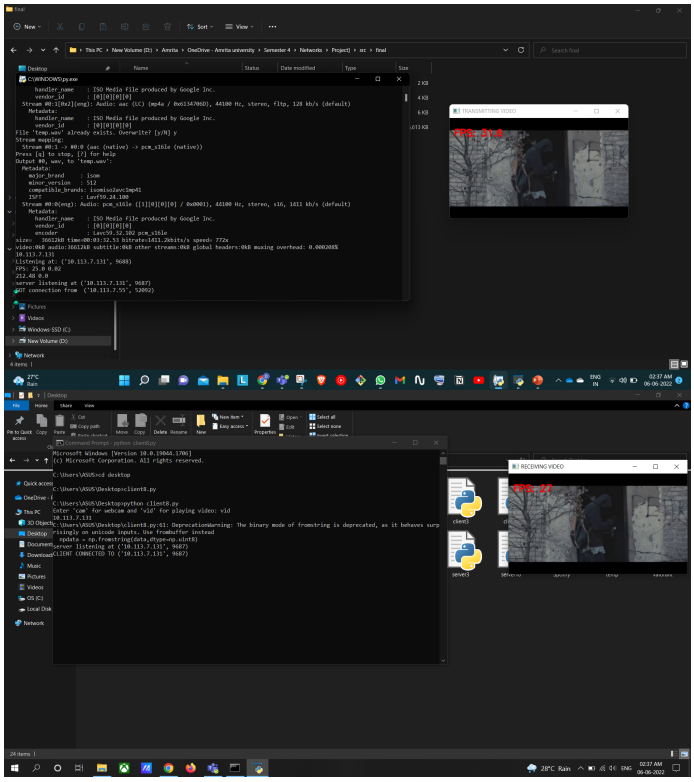


Fig. 1. Example screenshots of the server and client functions in action, both in different systems.

References

[1] S. C. Yadav and S. K. Singh, An introduction to client server Computing, New Age International (P) Ltd, 2009.

[2] C. S. Guynes and J. Windsor, "Revisiting client/server computing", Journal of Business Economics Research, vol. 9, no. 1, pp. 17-22, 2011.

[3] P. Uppu and S. Kadimpati, QoE of video streaming over LTE network, 2013.

[4] R. Gill, T. Farah and L. Trajkovic, "Comparison of WiMAX and ADSL performance when streaming audio and video content", 2011.

[5] L. Bellido, C. M. Lentisco, M. Aguayo and E. Pastor, "Supporting handover between LTE video broadcasting and unicast streaming", presented at the Next Generation Mobile Applications Services and Technologies 2015 9th International Conference on, pp. 329-334, 2015.

[6] A. G. Sheshjavani, B. Akbari and H. R. Ghaeini, "CMPVoD: A cluster mesh-based architecture for VoD streaming over hybrid CDN-P2P networks", presented at the Telecommunications (IST) 2016 8th International Symposium on, pp. 783-788, 2016.

[7] T. Kawakami, Y. Ishi, S. Matsumoto, T. Yoshihisa and Y. Teranishi, "An implementation of a video effect process allocation scheme for Internet live broadcasting", presented at the Consumer Electronics 2016 IEEE 5th Global Conference on, pp. 1-2, 2016.