CONTENTS

Introduction

With the penetration of the Internet technology and the vast boom in multimedia content available throughout the world, there have been many attempts to design a protocol for the transmission of live video streams across networks. This in itself is a vast subject of research, and this project aims to work on the lower level to see how a basic live stream media transfer takes places.

Live Stream refers to transmission of audio and video files such that they can be processed first without the entire file has been received. Since multimedia files are usually of very high size, it is imperative that streaming techniques be used for real-time presentation. This finds applications in video conferencing, remote diagnosis, and a stored-video streaming.

The choice of a protocol depends on the bandwidth of the network, the amount of accuracy required(i.e., can a frame be dropped?), proper synchronization, jitter and transmission error rate. Many more such factors decide the **Quality of Service(QoS)**. There are protocols to adjust the quality of service based on the need, this project aims to work without any QoS adjustments possible.

Features of Application

Format Independence

Progressive Streaming requires no specific protocols, just a format of the multimedia files that can be processed with partial information. Note here processing could simply mean displaying the content as well.

Our streaming mechanism is **not Progressive**. The format of the video file is independent of our streaming study. As a rule, we have converted all our video files to OpenCV Matrix Objects and transmitted them thereby completely removing any dependency on the formats of the media involved. However we may need codecs at both the transmitting and receiving end to encode and decode the file in the specific format.

Storage Overhead

We have attempted to use "True Streaming". Our aim is not to collect the packets and generate a corresponding file. It is to process them and forget about the packets. This prevents us from bothering with a lot of aspects such as the duration of video and reflects real world situations far more accurately.

Transmission of live video stream using UDP (User Datagram Protocol)

This project uses UDP for implementing a basic live video-transfer between peers, based on the request of the client(just like a LAN youtube). UDP is a connectionless datagram-oriented protocol present in the Transport Layer of the complete network model. It can used to send a datagram packet to any peer by simply using the **sendto() system call** from the Sockets Programming API. The peer's IP:Port are required to the same.

Methodology of Video Transmission:

This is a simple client-server model. Client requests a specific video stream and server sends it over UDP. The video is sent in real-time.

1) **The video from a file or webcam is read by the server program using Videocapture in OpenCV. The complete video file is located on the serve system.**
2) **Each frame is stored in a Mat object.**
3) **The data from the Mat object is placed in the buffer to be sent.**
4) **Each frame is sent from the buffer as a separate datagram across the network.**
5) **The client receives the datagram packet and converts it to a Mat object and plays it.**

NOTE : The sending of the frame requires the IP:Port of the peer. This is to be known before-hand.

Additional Play/Pause feature

This has been implemented using unix **signal handling,** which provides a toggle feature. Whenever the **client** (which is receiving the streamed the video) presses **Ctrl+C** i.e., passes a SIGINT signal, the play/pause is toggled. This is just like a play/pause button in youtube or other online streaming portals. The play/pause occurs at the server too, since the clients sends the same signal to the server using UDP.

Server	Frame Packet	Frame Packet	Frame Packet	Client

Why was UDP chosen?

1) Unlike TCP(Transmission Control Protocol), it is a very **fast** protocol, as it doesn't perform hand-shaking for establishing a connection(actually, it doesn't establish a connection, each time the peer IP:Port have to be mentioned in the sendto()).

2) For each segment of data, the TCP waits for acknowledgment and doesn't send other packets until the previous message was received by the peer. This feature of **reliability is not of much use in live stream of video, since it is not perceivable** if a few frames a lost between the peers. UDP provides the necessary speed by not providing the reliability. This **trade-off** is not significant and can be used for many applications.

3) Possibility of **Multicasting** and **Broadcasting** on UDP

The UDP protocol supports Multicasting(sending packets to a groups of users) and Broadcasting(sending packets to all users). The load incurred to the sender due to this is **not proportional** to the number of users, since the sender sends one datagram which is copied at the routers nearest to the destinations. This **avoids congestion** in the network path too.

Limitations of UDP

1) Size of maximum datagram that can be sent on UDP -

   1500 (MTU) − **8** byte UDP header − **20** byte IP header) = **1472** bytes

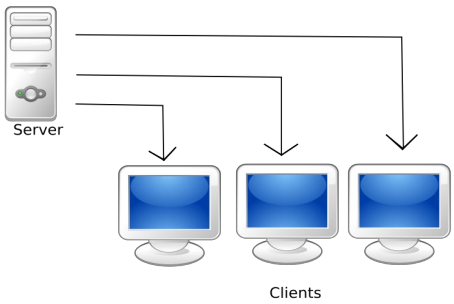
   There is a constraint that the video frames can't exceed a particular size. We could use IPv6 Jumbograms to tackle this issue.

2) Synchronization is difficult to achieve with both audio and video streams. For this RTP/RTCP are used nowadays. This are built on UDP and provide additional features. This project uses UDP for learning purposes, as to exactly how the video is transmitted.
3) The frames might not arrive the client in sequence, due to the absence of sequence number addition to the datagram packets, unlike TCP. But this again is not perceivable by humans and has rare chances to occur.

Broadcasting for video conferences

Broadcasting and Multicasting are unique features of the UDP protocol. Broadcasting refers to sending the packets to all addresses in the specified subnet. Multicasting is sending the packets to specific groups of addresses on the network. This is similar to a google group, wherein each system can request to join a group(each group is assigned a Class D IP). Each message for the group, is sent to each of the members.

These features require the permissions and multicasting settings enabled on the routers.



Author of Image : *Andrew D White*

RTP/RTCP

RTP provides a uniform means of transmitting data with consideration to real time constraints over IP (audio, video, etc.). The principal role of RTP is to implement the sequence numbers of IP packets(just as in TCP) to modify voice or video information even if the underlying network changes the order of the packets.

More generally, RTP makes it possible to:

- identify the type of information carried,
- add sequence numbers to the information carried, This provides sequencing not found in UDP.
- monitor the packets' arrival at the destination. This provides reliability not found in UDP.

RTP too may be used by multicast packets in order to route conversations to multiple recipients. This is just like that in UDP.

RTCP is usually used with RTP, to send control information to provide the synchronization information.

Further Avenues

Currently we have explored the transmission across devices with same Operating System, namely Ubuntu. However implementing the same functionality in different OS poses a challenge due to difference in the low level Network API's which are different from OS to OS.

Also given the ubiquitous nature of mobile devices and their growing usage, it is a natural possibility to consider if this application can be ported to a Smartphone or similar devices and if communicating with another mobile/ laptop and features for multicasting can be implemented using similar steps.

However the different nature of devices viz a mobile phone and a computer pose a syncing challenge.

For instance, the current implementation is in C whereas for Android App, it must be in Java using say Datagram class of Android SDK which imposes different limits for the maximum size of video file that can be transmitted.

Practically, the size limit may also vary from device to device especially in case of smartphones even if they run the same operating system.

Also we could consider the possibility of multicasting with all systems running different OS and studying the limits which will be compatible with all the different devices.

Tools and Platform Used

OpenCV 3.0.0 - For capture of video frames and transfer to UDP

Berkeley Sockets API - For creating client and server programs

The client and server programs were written specific to the Linux operating system. This can easily be extended to mobile devices and other systems.

References

https://help.ubuntu.com/community/OpenCV
http://beej.us/guide/bgnet/output/html/multipage/index.html
http://linux.die.net/man/