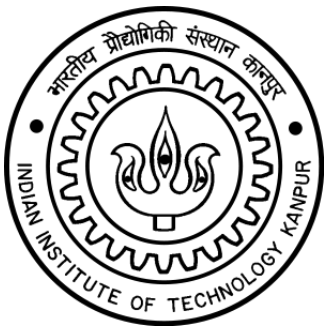


CS 425 - Computer Networks

"Multimedia Streaming using IPv6 "

Submitted By :

Lovish	Anirudh Kumar	Kapil Garg
Y9309	Y9088	Y9273

Guide : Dr Ratan.K.Ghosh

Date of Submission - 10th November 2011

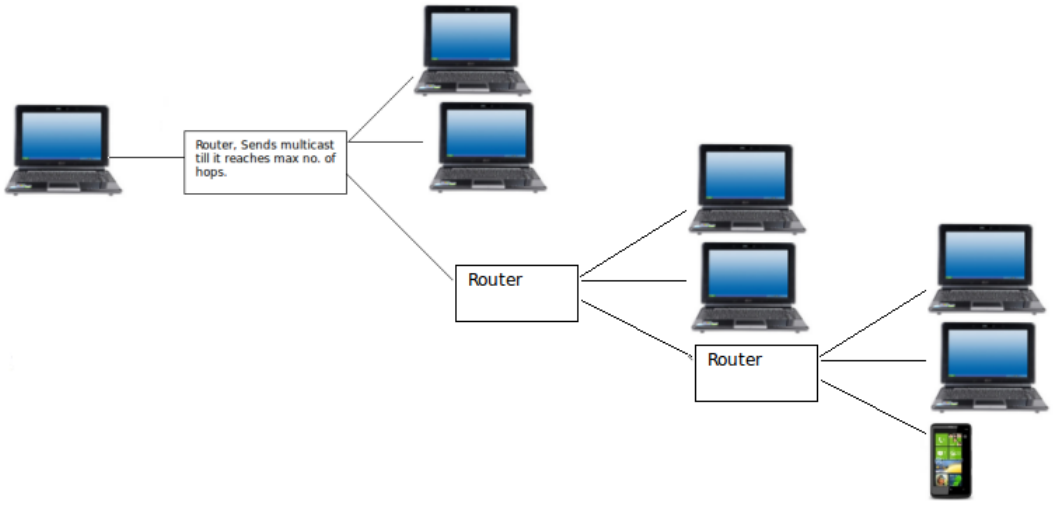
We are very thankful to Prof. RK Ghosh for giving us this opportunity and supporting us during the project.Also we are very thankful to CS425 course TA's for extending their help and guiding us whenever we needed throughout the project.

1 Problem Statement

Live streaming of videos is required to be sent at the same time to different computers while maintaining the same video quality. IPv6 has got native multicast support which will be helpful in transfer of this type of data.Design an application in which live streaming of videos can be sent to multiple clients using IPv6.

2 Indroduction

Live streaming of videos requires a continuous stream of data to be sent from a server to a client. If the number of clients is very large,say 1 million, the network resource and processing power consumed by the server in sending the data individually to each of the client will be huge. Further, a huge delay will be introduced in the streaming. Therefore, instead of sending of data individually to each client the server can multicast the data to a common address known as multicast address. The client which wants to receive the data can join the multicast group by sending a membership request to the group address. Once it is granted membership it can connect to this address and receive data. This greatly reduces the load of the server because each client that acquire a copy of the data from its own router which in turn acquires it from another router or directly from the server.



The figure above describes, point to multipoint multicasting as we have imple-

mented it, where a single computer acts as the server and sends a stream of multimedia data to the multicast address. This stream is replicated at all the branching points in a switched network untill it reaches the maximum number of hops specified by the TTL.

Our implementation uses IP Version 6 which provides inbuilt support for multicasting.

Multicasting in IPv6

Multicasting is a part of base specification of IPv6. It has also taken over IPv4 broadcast and there are no broadcast addresses in IPv6. Instead, multicasting in IPv6 can be used to emulate broadcasting as in IPv4 by sending the packets to link-local all nodes group defined at address ff02::1. Also, since IPv4 was limited to 32 bit addresses, it was difficult to get a globally routable multicast group. This led to difficulties in inter-domain multicasting. This issue has however been taken care of in IPv6 as due to the large address space of IPv6(128 bits), the first 64 bits are assigned by local Internet Registry for routing prefix and even after embedding the unicast address prefix into the IPv6 multicast address , we have 32 bits left. This means that IPv6 provides approximately 4.2 billion multicast group identifiers. This way each user in an IPv6 subnet has a set of globally routable SSM groups for multicasting.

A typical multicast address in IPv6 has the prefix ff00::/8. A multicast address in IPv6 can be broken into the following :
Bits 127-120 - prefix (1111 1111 always for multicasting).
Bits 119-116 - flags (only 3 of them defined right now and 1 is reserved for future use.)
Bits 115-112 - scope of multicasting
Bits 111-0 - Group identifier
The group identifier is allocated as per Allocation Guidelines specified in RFC3307. They are of three types:

- **Permanent IPv6 Multicast address**
  Allocated by IANA, they are assigned on Expert Review Basis in the range 0x00000001 to 0x3FFFFFFF. For such address the T and P flags must be set

to 0.

- **Permanent IPv6 Multicast group identifiers**
  Such identifiers are allocated to a group of numerous servers offering a common service such as NTP , NIS, DNS and many others.Permanent group IDs are allocated on an Expert Review basis, in the range 0x40000000 to 0x7FFFFFFF.
- **Dynamic IPv6 Multicast addresses**
  These can be allocated by an end-host or an allocation server. Their T flag must be set to 1.

3 Implementation

3.1 Technical Specifications

Language–Python 2.7.x.x

Modules – numpy,opencv,stringIO,pyaudio,pyffmpeg,pygame

3.2 Approach

To capture the frame from webcam we have used OpenCV module in python.It captures an image from the webcam and converts it to .jpeg format.Further this image is converted to string format through stringIO module.

Since only MTU of Ethernet is only 1500 B and an 640 x 480 image captured by webcam has average size of 30 kB(jpeg compression).We had to split the image in 40 parts and send each part seperately.

We have used UDP protocol instead of TCP. In live streaming the reliability of sent packets is not an issue, as long as most of the packets arrive correctly. Using TCP would slow down the streaming because of the large number of connections the server has to handle.

UDP protocol does not guarentee that the packets arrive in order so we had to attach redundant data with the packet.This includes:-

1. A unique Frame Number
2. Position of part within the Frame(0-39)

The client recieves a packet of data and extracts the video/audio data and the frame information from it. It uses this information to construct the frames of the video and displays them. It also has a buffer which stores the packets which arrive early. All the packets which arrive late are discarded. It uses a variable 'current_fno' which stores the frame number of currently playing frame to do this.

Also after a certain number of video packets have been transmitted, an audio packet has to be sent. The module pyaudio has been used to handle audio. Pyaudio records a small part of audio from the microphone at a time. The audio is split into 5 parts and an audio packet is constructed for each part by attaching the frame number and position information to it.For every 8 packets of video sent one such audio frame is sent. When all the parts of an audio frame have arrived on the client, it is played.

Sending audio frames between the video frames handles the problem of synchronisation of audio with the video.

When an audio or Video frame whose frame number is less than the current frame number being played arrives it is said to have arrived "out of order" and it increases a counter by one.When the counter reaches a specified limit the entire connection is reset and the client starts receiving afresh.

To display the video we have used pygame module.

3.3 Code Explained

3.3.1 server.py

- The **get_image** function takes a raw image from the camera and returns a image in standard python imaging library format.
- The **serverfromfile** function takes as its argument a valid movie file and gets video frames from it.The image received is in bgr format ,so we convert it to rgb format first.But to send it across the network we need to convert this into a

string.This is done with the help of StringIO module.Further we divide it into 40 parts and attach a frame number and sequence number with it to send it across the network.Note that while sending from a file no audio is sent across the network.

- **server** is the main function which plays role in live video streaming.It captures video frame from camera using opencv module and audio using pyaudio module.Now we divide each video frame in 40 parts and audio in 5 parts.To opimise and synchronise audio and video on the client side it sends an audio frame after 8 video frames.In each frame we add the following information :-
  - frame number
  - sequence number
  - 'aud' if it's a audio frame and 'vid' if it's a video frame

3.3.2 client.py

- Here **client** is the function which is used first initialises a pygame display window .Further it initialises a list of lists named video.It has a seperate variable to see it a packet arrives out of order.The data from the server is divided into 4 parts ie the type,frame number,sequence number and payload.Now it stores the audio and video data in seperate buffers,but if more than 10 packets of a any type arrive out of order then it flushes both the audio and video buffers.

3.4 Problems Faced

3.4.1 Resolved

1. The maximum transmission unit of ethernet was 1500 bytes.So we had to break each image.This initially led to synchronisation problems(as in UDP packets do not arrive in order) which we solved by buffering the images on client side.
2. If the server experiences some technical problems and need to shut down ,then for resuming the transmission we had to restart the client this was resolved by putting a limit on the pieces that arrive out of order.If more than 10 frames arrived out of order the client automatically resets the connection-variables.

3.4.2 Unresolved

1. We have added audio but there is certain amount of lag between the video and the audio.

3.5 Screenshots

4 Some Addon - Implementations

1. We have implemented streaming from a file on the server side along with the assigned task live streaming.
2. We have also tried to add audio along with video.

5 Future Implementations

- For a more challanging project we could have implemented p2p between the clients so that the broadcast speed would not be hindered.It also has a practical implementation as youtube and other websites such as `myp2p.eu` broadcast through this scheme.
- We could have implemented secure multicast ie the client should authenticate to view the multicast and also make makes sure that the data it is receiving is unmaniuplated.