# Individual project report

RTP Traffic drop analyzer

Video Streaming using RTP/RTSP

Group-10
Venkata Adithya Boppana-010731201
Class id-08

## Contents

## 1. Introduction

In every video streaming scenario which uses UDP packets will definitely have a level of packet drop. Since RTP uses UDP packets for communication in our application we observed some packet drop at the client side. This packet drop is very variable and is changing due to many factors. So we needed a tool so that we can have the packet drop statistics in a particular scenario and then compare the packet drop in this scenario to packet drop in another scenario. We wanted this tool so that we can create an ideal situation such that there will be very less packet drop. As we couldn't find a tool which can show us the exact packet drop for RTP/RTSP video streaming with variable port selection I decided to implement a tool using 'dpkt' python library.

## 2. Usage

To use RTP traffic drop analyzer first we need the captures of network traffic at server side and client side to calculate the packet loss. These captures can be got by using Wireshark and capturing the traffic of the interface which is running Client, Server. Start Wireshark at the time of video streaming on Server, Client and select the interface using which Streaming service is performed then capture the traffic of that interface. After the capture we get two .pcap files which contains the network traffic of client side and server side.

Transfer the captured .pcap files to a system where test are performed. RTP traffic drop analyzer program is written in python so python interpreter has to be installed on the testing system. 'dpkt' python library also has to be installed on the system where tests are being conducted. The CLI program can be run with the following command

*python analyze.py arg1 arg2 arg3*

*arg1* = .pcap file from server side
*arg2*= .pcap file from client side
*arg3*= RTP port number

## 3. Working of analyzer

The network traffic captured on .pcap files is not human readable so we need a method to parse and extract data from such files. 'dpkt' python library is one such method, we can use the modules and functions in 'dpkt' library to extract data from .pcap files which contains the captured network traffic. So we import the 'dpkt' library. 'Sys' library is used to parse the command line arguments.

```
import dpkt
import sys
```

Let us go into the function which takes in the formatted .pcap file and returns the no of RTP packets transferred i.e. received and transmitted through a particular port. rtprtsp () function takes in formatted .pcap file. Then it separates the ipV4 packets from the bunch of other protocol packets in the line 'if eth.type==2048'. This specification is in correspondence to http://www.iana.org/assignments/ethernet-numbers. Even after filtering out many protocols packets are left aside like TCP, ICMP etc., which also use IPv4. To filter out data related to other protocols and only get UDP packets we use the command 'if ip.p==17'. This is in accordance with RFC5237. After performing this step we are left out with only UDP packets. But not all UDP packets in a network are used solely for RTP. So we have to remove all the UDP packets which do not correspond with the RTP protocol. RTP protocol can use any port and do not have limitations using any specified UDP ports, this means we have to specify the port every time we are initiating a client. Due to the dynamic nature of RTP protocol in using ports we take the port used by client through Command Line Arguments. Any UDP packets which used other port than the specified port in Command Line Arguments are filtered out. This leaves us with just UDP packets which are used by RTP protocol. The no of UDP packets used for RTP value is returned to function calling location.

```
def rtprtsp(pcap):
        rtp_packet=0
        rtsp_packet=0
        other=0
        total_packet=0

        for ts,buf in pcap:
                try:eth=dkpt.ethernet.Ethernet(buf)
                except:continue
        #counts total number of packets
                total_packet+=1
        #picks up IPv4 packets
                if eth.type==2048:
                        ip=eth.data
                if ip.p==17:            #picking only udp packets # In accordance to RFC5237
                        try:udp=ip.data
                        except:continue
                        if udp.sport == sys.argv[3] or udp.dport == sys.argv[3]: #picking only the rtp packets
                                rtp_packet+=1
                else:
                        other+=1
        return(rtp_packet)
```

We have seen how rtprtsp () function takes dpkt formatted .pcap file and returns no of RTP packets received/transmitted. Here we will look into formatting of original .pcap file and calculation of packet drop percentage. By using these line we open the .pcap file '*open(sy.argv[1], open(sys.argv[2])*)'. As the opened files are not in readable format we format such that dpkt modules can perform their operations on the .pcap file. After the formatting the formatted file is sent to rtprtsp () function which return the no of RTP packets. We are storing the no of RTP packets sent by the Server in server_rtp and RTP packets received in client_rtp variable. By using the following formula the drop rate of the RTP packets is calculated.

Drop rate= ((packets sent by server-packets received by client)/packets sent by server)*100

After calculating the Drop rate all the statistics calculated (# of Rtp packets sent by server, # of Rtp packets received by client, Drop rate) are printed out to the command line.

```
f=open(sys.argv[1])
pcap=dpkt.pcap.Reader(f)
server_rtp=rtprtsp(pcap)

f=open(sys.argv[2])
pcap=dpkt.pcap.Reader(f)
client_rtp=rtprtsp(pcap)

try:packetdrop=((server_rtp-client_rtp)/server_rtp)*100
except:packetdrop=0

print "rtp packets sent from server: "+str(server_rtp)
print "rtp packets received by the client: "+str(client_rtp)
print 'percentage of packet drop: '+str(packetdrop)
```

By using the above RTP traffic drop analyzer we checked the traffic drop for different scenarios and configured settings on AWS such that there is less packet drop.