**File name : VideoStream.py**

import struct

class VideoStream:

def \_\_init\_\_(self, filename):

self.filename = filename

try:

self.file = open(filename, 'rb')

print '-'\*60 + "\nVideo file : |" + filename + "| read\n" + '-'\*60

except:

print "read " + filename + " error"

raise IOError

self.frameNum = 0

def nextFrame(self):

"""Get next frame."""

data = self.file.read(5) # Get the framelength from the first 5 bytes

#data\_ints = struct.unpack('<' + 'B'\*len(data),data)

data = bytearray(data)

data\_int = (data[0] - 48) \* 10000 + (data[1] - 48) \* 1000 + (data[2] - 48) \* 100 + (data[3] - 48) \* 10 + (data[4] - 48)# = #int(data.encode('hex'),16)

final\_data\_int = data\_int

if data:

framelength = final\_data\_int#int(data)#final\_data\_int/8 # xx bytes

# Read the current frame

frame = self.file.read(framelength)

if len(frame) != framelength:

raise ValueError('incomplete frame data')

#print "frame length"

#print len(frame)

#if not (data.startswith(b'\xff\xd8') and data.endswith(b'\xff\xd9')):

# raise ValueError('invalid jpeg')

self.frameNum += 1

print '-'\*10 + "\nNext Frame (#" + str(self.frameNum) + ") length:" + str(framelength) + "\n" + '-'\*10

return frame

def frameNbr(self):

"""Get frame number."""

return self.frameNum

**Code for RTP packetization at server:**

**File name : RtpPacket.py**

import sys

from time import time

# from VideoStream import VideoStream

import VideoStream

HEADER\_SIZE = 12

class RtpPacket:

#header = bytearray(HEADER\_SIZE)

#HEADER\_SIZE = 12

def \_\_init\_\_(self):

self.header = bytearray(HEADER\_SIZE)

def encode(self, version, padding, extension, cc, seqnum, marker, pt, ssrc, payload):

"""Encode the RTP packet with header fields and payload."""

timestamp = int(time())

print "timestamp: " + str(timestamp)

self.header = bytearray(HEADER\_SIZE

#RTP-version filed(V), must set to 2

#padding(P),extension(X),number of contributing sources(CC) and marker(M) fields all set to zero in this lab

#Because we have no other contributing sources(field CC == 0),the CSRC-field does not exist

#Thus the length of the packet header is therefore 12 bytes

#Above all done in ServerWorker.py

#header[0] = version + padding + extension + cc + seqnum + marker + pt + ssrc

self.header[0] = version << 6

self.header[0] = self.header[0] | padding << 5

self.header[0] = self.header[0] | extension << 4

self.header[0] = self.header[0] | cc

self.header[1] = marker << 7

self.header[1] = self.header[1] | pt

self.header[2] = seqnum >> 8

self.header[3] = seqnum

self.header[4] = (timestamp >> 24) & 0xFF

self.header[5] = (timestamp >> 16) & 0xFF

self.header[6] = (timestamp >> 8) & 0xFF

self.header[7] = timestamp & 0xFF

self.header[8] = ssrc >> 24

self.header[9] = ssrc >> 16

self.header[10] = ssrc >> 8

self.header[11] = ssrc

# Get the payload from the argument

# self.payload = ...

self.payload = payload

def decode(self, byteStream):

"""Decode the RTP packet."""

#print byteStream[:HEADER\_SIZE]

self.header = bytearray(byteStream[:HEADER\_SIZE]) #temporary solved

self.payload = byteStream[HEADER\_SIZE:]

def version(self):

"""Return RTP version."""

return int(self.header[0] >> 6)

def seqNum(self):

"""Return sequence (frame) number."""

seqNum = self.header[2] << 8 | self.header[3] #header[2] shift left for 8 bits then does bit or with header[3]

return int(seqNum)

def timestamp(self):

"""Return timestamp."""

timestamp = self.header[4] << 24 | self.header[5] << 16 | self.header[6] << 8 | self.header[7]

return int(timestamp)

def payloadType(self):

"""Return payload type."""

pt = self.header[1] & 127

return int(pt)

def getPayload(self):

"""Return payload."""

return self.payload

def getPacket(self):

"""Return RTP packet."""

return self.header + self.payload