import xml.etree.ElementTree as etree

import base64

from struct import unpack, pack

import sys

import io

import os

import time

import itertools

import xbmcaddon

import xbmc

import urllib2,urllib

import traceback

import urlparse

import posixpath

import re

import socket, struct

from flvlib import tags

from flvlib import helpers

from flvlib.astypes import MalformedFLV

import zlib

from StringIO import StringIO

import hmac

import hashlib

import base64

import bitstring

addon\_id = 'script.video.F4mProxy'

selfAddon = xbmcaddon.Addon(id=addon\_id)

\_\_addonname\_\_ = selfAddon.getAddonInfo('name')

\_\_icon\_\_ = selfAddon.getAddonInfo('icon')

downloadPath = xbmc.translatePath(selfAddon.getAddonInfo('profile'))#selfAddon["profile"])

#F4Mversion=''

defualtype=""

def getLastPTS(data,rpid,type="video"):

##print 'inpcr'

ret=None

currentpost=len(data)

##print 'currentpost',currentpost

found=False

packsize=188

spoint=0

while not found:

ff=data.rfind('\x47',0,currentpost-1)

##print 'ff',ff,data[ff-188]

if ff==-1:

#print 'No sync data'

found=True

elif data[ff-packsize]=='\x47' and data[ff-packsize-packsize]=='\x47':

spoint=ff

found=True

else:

currentpost=ff-1

##print 'spoint',spoint

if spoint<=0: return None

currentpost= spoint

found=False

while not found:

##print len(data)-currentpost

if len(data)-currentpost>=188:

##print 'currentpost',currentpost

bytes=data[currentpost:currentpost+188]

bits=bitstring.ConstBitStream(bytes=bytes)

sign=bits.read(8).uint

tei = bits.read(1).uint

pusi = bits.read(1).uint

transportpri = bits.read(1).uint

pid = bits.read(13).uint

##print pid

if pid==rpid or rpid==0:

##print pid

##print 1/0

try:

packet = bits.read((packsize-3)\*8)

scramblecontrol = packet.read(2).uint

adapt = packet.read(2).uint

concounter = packet.read(4).uint

except:

#print 'error'

return None##print 'errpor'#adapt=-1

decodedpts=None

av=""

##print 'adapt',adapt

if adapt == 3:

adaptation\_size = packet.read(8).uint

discontinuity = packet.read(1).uint

random = packet.read(1).uint

espriority = packet.read(1).uint

pcrpresent = packet.read(1).uint

opcrpresent = packet.read(1).uint

splicingpoint = packet.read(1).uint

transportprivate = packet.read(1).uint

adaptation\_ext = packet.read(1).uint

restofadapt = (adaptation\_size+3) - 1

if pcrpresent == 1:

pcr = packet.read(48)

restofadapt -= 6

if opcrpresent == 1:

opcr = packet.read(48)

restofadapt -= 6

packet.pos += (restofadapt-3) \* 8

if ((packet.len - packet.pos)/8) > 5:

pesync = packet.read(24)#.hex

if pesync == ('0x000001'):

pestype = packet.read(8).uint

if pestype > 223 and pestype < 240:

av = 'video'

if pestype < 223 and pestype > 191:

av = 'audio'

packet.pos += (3\*8)

ptspresent = packet.read(1).uint

dtspresent = packet.read(1).uint

if ptspresent:

packet.pos += (14)

pts = packet.read(40)

pts.pos = 4

firstpartpts = pts.read(3)

pts.pos += 1

secondpartpts = pts.read(15)

pts.pos += 1

thirdpartpts = pts.read(15)

#decodedpts = bitstring.ConstBitArray().join([firstpartpts.bin, secondpartpts.bin, thirdpartpts.bin]).uint

decodedpts =int(''.join([firstpartpts.bin, secondpartpts.bin, thirdpartpts.bin]),2)#

if dtspresent:

dts = packet.read(40)

dts.pos = 4

firstpartdts = dts.read(3)

dts.pos += 1

secondpartdts = dts.read(15)

dts.pos += 1

thirdpartdts = dts.read(15)

#decodeddts = bitstring.ConstBitArray().join([firstpartdts.bin, secondpartdts.bin, thirdpartdts.bin]).uint

decodeddts =int(''.join([firstpartdts.bin, secondpartdts.bin, thirdpartdts.bin]),2)#

elif adapt == 2:

#if adapt is 2 the packet is only an adaptation field

adaptation\_size = packet.read(8).uint

discontinuity = packet.read(1).uint

random = packet.read(1).uint

espriority = packet.read(1).uint

pcrpresent = packet.read(1).uint

opcrpresent = packet.read(1).uint

splicingpoint = packet.read(1).uint

transportprivate = packet.read(1).uint

adaptation\_ext = packet.read(1).uint

restofadapt = (adaptation\_size+3) - 1

if pcrpresent == 1:

pcr = packet.read(48)

restofadapt -= 6

if opcrpresent == 1:

opcr = packet.read(48)

restofadapt -= 6

elif adapt == 1:

pesync = packet.read(24)#.hex

##print 'pesync',pesync

if pesync == ('0x000001'):

pestype = packet.read(8).uint

if pestype > 223 and pestype < 240:

av = 'video'

if pestype < 223 and pestype > 191:

av = 'audio'

packet.pos += 24

ptspresent = packet.read(1).uint

dtspresent = packet.read(1).uint

##print 'ptspresent',ptspresent

if ptspresent:

packet.pos += (14)

pts = packet.read(40)

pts.pos = 4

firstpartpts = pts.read(3)

pts.pos += 1

secondpartpts = pts.read(15)

pts.pos += 1

thirdpartpts = pts.read(15)

#decodedpts = bitstring.ConstBitArray().join([firstpartpts.bin, secondpartpts.bin, thirdpartpts.bin]).uint

decodedpts =int(''.join([firstpartpts.bin, secondpartpts.bin, thirdpartpts.bin]),2)#

if dtspresent:

dts = packet.read(40)

dts.pos = 4

firstpartdts = dts.read(3)

dts.pos += 1

secondpartdts = dts.read(15)

dts.pos += 1

thirdpartdts = dts.read(15)

#decodeddts = bitstring.ConstBitArray().join([firstpartdts.bin, secondpartdts.bin, thirdpartdts.bin]).uint

decodeddts =int(''.join([firstpartdts.bin, secondpartdts.bin, thirdpartdts.bin]),2)#

if decodedpts and (type=="" or av==type) and len(av)>0:

##print 'currentpost',currentpost,decodedpts

return decodedpts

currentpost=currentpost-packsize

if currentpost<10:

#print 'came back to begin'

found=True

return ret

def getFirstPTSFrom(data,rpid, initpts,type="video" ):

##print 'xxxxxxxxxxxinpcr getFirstPTSFrom'

ret=None

currentpost=0#len(data)

##print 'currentpost',currentpost

found=False

packsize=188

spoint=0

##print 'inwhile'

while not found:

ff=data.find('\x47',currentpost)

if ff==-1:

#print 'No sync data'

found=True

elif data[ff+packsize]=='\x47' and data[ff+packsize+packsize]=='\x47':

spoint=ff

found=True

else:

currentpost=ff+1

##print 'spoint',spoint

if spoint>len(data)-packsize: return None

currentpost= spoint

found=False

while not found:

##print 'currentpost',currentpost

if len(data)-currentpost>=188:

bytes=data[currentpost:currentpost+188]

bits=bitstring.ConstBitStream(bytes=bytes)

sign=bits.read(8).uint

tei = bits.read(1).uint

pusi = bits.read(1).uint

transportpri = bits.read(1).uint

pid = bits.read(13).uint

##print pid

##print pid,rpid

##print 1/0

if rpid==pid or rpid==0:

##print 'here pid is same'

try:

packet = bits.read((packsize-3)\*8)

scramblecontrol = packet.read(2).uint

adapt = packet.read(2).uint

concounter = packet.read(4).uint

except:

#print 'error'

return None##print 'errpor'#adapt=-1

decodedpts=None

av=""

if adapt == 3:

adaptation\_size = packet.read(8).uint

discontinuity = packet.read(1).uint

random = packet.read(1).uint

espriority = packet.read(1).uint

pcrpresent = packet.read(1).uint

opcrpresent = packet.read(1).uint

splicingpoint = packet.read(1).uint

transportprivate = packet.read(1).uint

adaptation\_ext = packet.read(1).uint

restofadapt = (adaptation\_size+3) - 1

if pcrpresent == 1:

pcr = packet.read(48)

restofadapt -= 6

if opcrpresent == 1:

opcr = packet.read(48)

restofadapt -= 6

packet.pos += (restofadapt-3) \* 8

if ((packet.len - packet.pos)/8) > 5:

pesync = packet.read(24)#.hex

if pesync == ('0x000001'):

pestype = packet.read(8).uint

if pestype > 223 and pestype < 240:

av = 'video'

if pestype < 223 and pestype > 191:

av = 'audio'

packet.pos += (3\*8)

ptspresent = packet.read(1).uint

dtspresent = packet.read(1).uint

if ptspresent:

packet.pos += (14)

pts = packet.read(40)

pts.pos = 4

firstpartpts = pts.read(3)

pts.pos += 1

secondpartpts = pts.read(15)

pts.pos += 1

thirdpartpts = pts.read(15)

#decodedpts = bitstring.ConstBitArray().join([firstpartpts.bin, secondpartpts.bin, thirdpartpts.bin]).uint

decodedpts =int(''.join([firstpartpts.bin, secondpartpts.bin, thirdpartpts.bin]),2)#

if dtspresent:

dts = packet.read(40)

dts.pos = 4

firstpartdts = dts.read(3)

dts.pos += 1

secondpartdts = dts.read(15)

dts.pos += 1

thirdpartdts = dts.read(15)

#decodeddts = bitstring.ConstBitArray().join([firstpartdts.bin, secondpartdts.bin, thirdpartdts.bin]).uint

decodeddts =int(''.join([firstpartdts.bin, secondpartdts.bin, thirdpartdts.bin]),2)#

elif adapt == 2:

#if adapt is 2 the packet is only an adaptation field

adaptation\_size = packet.read(8).uint

discontinuity = packet.read(1).uint

random = packet.read(1).uint

espriority = packet.read(1).uint

pcrpresent = packet.read(1).uint

opcrpresent = packet.read(1).uint

splicingpoint = packet.read(1).uint

transportprivate = packet.read(1).uint

adaptation\_ext = packet.read(1).uint

restofadapt = (adaptation\_size+3) - 1

if pcrpresent == 1:

pcr = packet.read(48)

restofadapt -= 6

if opcrpresent == 1:

opcr = packet.read(48)

restofadapt -= 6

elif adapt == 1:

pesync = packet.read(24)#.hex

##print 'pesync',pesync

if pesync == ('0x000001'):

pestype = packet.read(8).uint

if pestype > 223 and pestype < 240:

av = 'video'

if pestype < 223 and pestype > 191:

av = 'audio'

packet.pos += 24

ptspresent = packet.read(1).uint

dtspresent = packet.read(1).uint

##print 'ptspresent',ptspresent

if ptspresent:

packet.pos += (14)

pts = packet.read(40)

pts.pos = 4

firstpartpts = pts.read(3)

pts.pos += 1

secondpartpts = pts.read(15)

pts.pos += 1

thirdpartpts = pts.read(15)

#decodedpts = bitstring.ConstBitArray().join([firstpartpts.bin, secondpartpts.bin, thirdpartpts.bin]).uint

decodedpts =int(''.join([firstpartpts.bin, secondpartpts.bin, thirdpartpts.bin]),2)#

if dtspresent:

dts = packet.read(40)

dts.pos = 4

firstpartdts = dts.read(3)

dts.pos += 1

secondpartdts = dts.read(15)

dts.pos += 1

thirdpartdts = dts.read(15)

#decodeddts = bitstring.ConstBitArray().join([firstpartdts.bin, secondpartdts.bin, thirdpartdts.bin]).uint

decodeddts =int(''.join([firstpartdts.bin, secondpartdts.bin, thirdpartdts.bin]),2)#

if decodedpts and (type=="" or av==type) and len(av)>0:

##print decodedpts

if decodedpts>initpts:

return decodedpts,currentpost

else:

found=True

currentpost=currentpost+188

if currentpost>=len(data):

##print 'came back to begin'

found=True

return ret

class TSDownloader():

outputfile =''

clientHeader=None

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

self.init\_done=False

def thisme(self):

return 'aaaa'

def openUrl(self,url, ischunkDownloading=False):

try:

post=None

openner = urllib2.build\_opener(urllib2.HTTPHandler, urllib2.HTTPSHandler)

if post:

req = urllib2.Request(url, post)

else:

req = urllib2.Request(url)

ua\_header=False

if self.clientHeader:

for n,v in self.clientHeader:

req.add\_header(n,v)

if n=='User-Agent':

ua\_header=True

if not ua\_header:

req.add\_header('User-Agent','VLC/2.2.2 LibVLC/2.2.17')

req.add\_header('Icy-MetaData','1')

#response = urllib2.urlopen(req)

if self.proxy:

req.set\_proxy(self.proxy, 'http')

response = openner.open(req)

return response

except:

#print 'Error in getUrl'

traceback.print\_exc()

return None

def getUrl(self,url, ischunkDownloading=False):

try:

post=None

openner = urllib2.build\_opener(urllib2.HTTPHandler, urllib2.HTTPSHandler)

if post:

req = urllib2.Request(url, post)

else:

req = urllib2.Request(url)

ua\_header=False

if self.clientHeader:

for n,v in self.clientHeader:

req.add\_header(n,v)

if n=='User-Agent':

ua\_header=True

if not ua\_header:

req.add\_header('User-Agent','Mozilla/5.0 (Windows NT 6.1; WOW64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/33.0.1750.154 Safari/537.36')

#response = urllib2.urlopen(req)

if self.proxy and ( (not ischunkDownloading) or self.use\_proxy\_for\_chunks ):

req.set\_proxy(self.proxy, 'http')

response = openner.open(req)

data=response.read()

return data

except:

#print 'Error in getUrl'

traceback.print\_exc()

return None

def init(self, out\_stream, url, proxy=None,g\_stopEvent=None, maxbitRate=0):

try:

self.init\_done=False

self.init\_url=url

self.clientHeader=None

self.status='init'

self.proxy = proxy

self.maxbitRate=maxbitRate

if self.proxy and len(self.proxy)==0:

self.proxy=None

self.out\_stream=out\_stream

if g\_stopEvent: g\_stopEvent.clear()

self.g\_stopEvent=g\_stopEvent

if '|' in url:

sp = url.split('|')

url = sp[0]

self.clientHeader = sp[1]

self.clientHeader= urlparse.parse\_qsl(self.clientHeader)

#print 'header recieved now url and headers are',url, self.clientHeader

self.status='init done'

self.url=url

return True #disable for time being

#return self.downloadInternal(testurl=True)

#os.remove(self.outputfile)

except:

traceback.print\_exc()

self.status='finished'

return False

def keep\_sending\_video(self,dest\_stream, segmentToStart=None, totalSegmentToSend=0):

try:

self.status='download Starting'

self.downloadInternal(dest\_stream=dest\_stream)

except:

traceback.print\_exc()

self.status='finished'

def downloadInternal(self,dest\_stream=None,testurl=False):

try:

url=self.url

fileout=dest\_stream

self.status='bootstrap done'

First=True

cont=True

lastbuf=None

lost=1

ignorefind=0

lastpts=None

fixpid=256

ignoredblock=None

sleeptime=0

firsttimeurl=False

while True:

if sleeptime>0:

xbmc.sleep(sleeptime)

sleeptime=0

starttime=time.time()

response=self.openUrl(url)

buf="start"

byteread=0

bytesent=0

firstBlock=True

wrotesomething=False

currentduration=0

limit=1024\*188

if testurl: limit=1024

lastdataread=limit

#print 'starting.............. new url',wrotesomething

try:

if self.g\_stopEvent and self.g\_stopEvent.isSet():

print 'event set'

return False

while (buf != None and len(buf) > 0 and lastdataread>0):

if self.g\_stopEvent and self.g\_stopEvent.isSet():

print 'event set'

return False

try:

buf = response.read(limit)##500 \* 1024)

lastdataread=len(buf)

byteread+=lastdataread

#print 'got data',len(buf)

if lastdataread==0: print 1/0

if testurl:

print 'test complete true'

response.close()

return True

except:

traceback.print\_exc(file=sys.stdout)

print 'testurl',testurl,lost

if testurl and lost>10:

print 'test complete false'

response.close()

return False

buf=None

lost+=1

if lost>10 or firsttimeurl:

fileout.close

return

break

firsttimeurl=False

writebuf=buf

if not First:

##print 'second ite',wrotesomething

if wrotesomething==False:

##print 'second ite wrote something false'#, len(lastbuf)

if lastpts:

#buffertofind=lastbuf#[lastbuf.rfind('G',len(lastbuf)-170):]

##print 'buffertofind',len(buffertofind),buffertofind.encode("hex")

#print 'pts to find',lastpts

lastforcurrent=getLastPTS(buf,fixpid,defualtype)

#print 'last pts in new data',lastforcurrent

if lastpts<lastforcurrent:#we have data

#print 'we have data', lastpts,lastforcurrent, (lastforcurrent-lastpts)/90000

try:

firstpts,pos= getFirstPTSFrom(buf,fixpid,lastpts,defualtype)#

except:

traceback.print\_exc(file=sys.stdout)

print 'getFirstPTSFrom error, using, last -1',# buf.encode("hex"), lastpts,

firstpts,pos= getFirstPTSFrom(buf,fixpid,lastpts-1,defualtype)#

#if ignoredblock and (lastpts-firstpts)<0:

# print 'ignored last block yet the new block loosing data'

# print lastpts,firstpts,lastpts-firstpts

# print ignoredblock.encode('hex')

# print buf.encode('hex')

#print 'last pst send',lastpts,

#print 'first pst new',firstpts

#if abs(lastpts-firstpts)>300000:

# print 'xxxxxxxxxxxxxxxxxx',buf.encode("hex")

#print 'last pst new',lastforcurrent

if firstpts>lastforcurrent:

print 'bad pts? ignore'#, buf.encode("hex")

#print 'auto pos',pos

if pos==None: pos=0

if pos>5000:

rawpos=buf.find(lastbuf[-5000:])

if rawpos>=0:

pos=rawpos+5000

#print 'overridin 1'

else:

#print 'rawpos',rawpos,lastbuf[-5000:].encode("hex")

#print 'buff',buf.encode("hex")

rawpos=(ignoredblock+buf).find((lastbuf)[-5000:])

if rawpos>len(ignoredblock):

pos=rawpos-len(ignoredblock)

#print 'overridin 2'

#else:

# print 'using next PTS', pos, firstpts

ignoredblock=None

#else: pos=0

#print firstpts,pos,(firstpts-lastpts)/90000

#fn=buf.find(buffertofind[:188])

#print 'BUFFER FOUND!!', (pos\*100)/len(buf)

if (pos\*100)/len(buf)>70:

sleeptime=0

buf= buf[pos:]

lastpts=lastforcurrent

#print 'now last pts',lastpts

wrotesomething=True

else:

#if lastforcurrent==None:

# print 'NONE ISSUE', buf.encode("hex")

print 'problembytes','diff',lastpts,lastforcurrent, lastpts, lastforcurrent

#buf.encode("hex")

ignoredblock=writebuf

ignorefind+=1#same or old data?

writebuf=None

#if lastpts-lastforcurrent>(90000\*10):

#lastdataread=0 # read again we are buffering

#response.close()

#xbmc.sleep(1000)

# print 'reconnect'

#if ignorefind>5:

# ignorefind=0

# #print 'not ignoring so write data'

#else:

# #print 'ignoring at the m'

# writebuf=None

#print 'Buffer NOT FOUND!!ignoring'

#else:

# writebuf=None

##print 'second ite wrote something false skipiing'

#else:

##print 'second ite wrote something so continue'

else:

#print 'found first packet', len(writebuf)

First=False

if not ('\x47' in writebuf[0:20]):

#fileout.write(buf)

#fileout.flush()

print 'file not TS', repr(writebuf[:100])

fileout.close()

return

starttime=time.time()

if writebuf and len(writebuf)>0:

wrotesomething=True

if len(buf)>5000 or lastbuf==None:

lastbuf=buf

else:

lastbuf+=buf

bytesent+=len(buf)

fileout.write(buf)

##print 'writing something..............'

fileout.flush()

lastpts1=getLastPTS(lastbuf,fixpid,defualtype)

if lastpts and lastpts1 and lastpts1-lastpts<0:

print 'too small?',lastpts , lastpts1,lastpts1-lastpts

#print lastbuf.encode("hex")

if not lastpts1==None: lastpts=lastpts1

try:

firsttime,pos=getFirstPTSFrom(lastbuf,fixpid,0,defualtype)#

#print lastpts,firsttime

currentduration += (lastpts-firsttime)/90000

##print 'currentduration',currentduration

#currentduration-=2

#f currentduration<=2:

# currentduration=0

#if currentduration>10: currentduration=2

##print 'sleeping for',currentduration

except: pass

try:

print 'finished',byteread

if byteread>0:

print 'Percent Used'+str(((bytesent\*100)/byteread))

response.close()

print 'response closed'

except:

print 'close error'

traceback.print\_exc(file=sys.stdout)

if wrotesomething==False :

if lost<10: continue

fileout.close()

#print time.asctime(), "Closing connection"

return

else:

lost=0

if lost<0: lost=0

#xbmc.sleep(len(buf)\*1000/1024/200)

#print 'finish writing',len(lastbuf)

##print lastbuf[-188:].encode("hex")

endtime=time.time()

timetaken=int((endtime-starttime))

#print 'video time',currentduration

#print 'processing time',timetaken

sleeptime=currentduration-timetaken-2

#print 'sleep time',sleeptime

#if sleeptime>0:

# xbmc.sleep(sleeptime\*1000)#len(buf)/1024/1024\*5000)

except socket.error, e:

print time.asctime(), "Client Closed the connection."

try:

response.close()

fileout.close()

except Exception, e:

return

return

except Exception, e:

traceback.print\_exc(file=sys.stdout)

response.close()

fileout.close()

return False

except:

traceback.print\_exc()

return