Video Quality Experts Group

21-24 January 2014
Boulder, CO
English only
Original: English

JEG-Hybrid Contribution

Source: Intel Corporation

Title: Python Scripts for H.264 Bitstream Extraction from pcap Files

Introduction

Annexes I & II to this contribution contain two Python script files that were developed to extract the H.264 bitstreams from the zero-packet-error pcap files contained in the JEG264HMIX1 database on the IRRCyN website (http://www.irccyn.ec-nantes.fr/spip.php?article1033). They are provided in this contribution in case other researchers may find them useful, as an alternative to `H264AnnexBExtractor`[i] for these files, and as a basis for extending them to work with other pcap-embedded H.264 bitstreams. The H.264 bitstream files were run through the Modified JM decoder[ii], and produced XML files that matched the `HMIX2` files on the IRRCyN website.

Usage

The scripts consist of two files, `pcapto264script.py` and `make264frompcap.py`. They were developed for Python 3.32, Windows build, but can be modified to run in other environments. To use, copy both files to the directory containing the pcap files to be processed (for example, `C:\JEG264HMIX1\PCAP\`). In a command window, type:

```
>py pcapto264script.py N1 N2 N3 ... Nx
```

where `N1`, `N2`, etc. are numbers in the range 0-99, separated by spaces. The scripts will search the current directory and all subdirectories for pcap files whose base name contains the string `hrcNN`, where `NN` matches one of the numbers typed on the command line. For each such file, the H.264 bitstream will be extracted, and stored in an output file with the extension `.264`, in Annex B format. A new subdirectory is created with a name in the form `.\264files-<time><date>` and all the bitstream files are placed there (i.e., a unique subdirectory is created every time the script is run).

In addition, the scripts assume the pcap filenames are in the form `<text string 1>srcMx_hrcNy<text string 2>.pcap`. As the prefix and suffix text strings are often quite lengthy, for ease of use, the filenames of the output files do not contain them (in other words, the output filenames are in the form `srcMx_hrcNy.264`).

Discussion

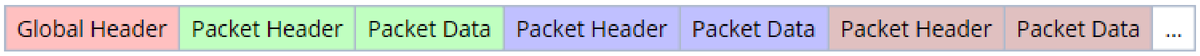
The JEG264HMIX1 database contains many MOS-scored H.264 bitstreams. Ten different SRC sequences are used, in combination with several types of HRC channels, both error-free and with packet errors introduced. Our group is interested in studying MOS estimation of compressed video sequences, so the bitstreams that were generated with the use of the error-free HRC's (i.e., HRC's 1 through 10) were of interest. After having difficulty getting `H264AnnexBExtractor` to reliably extract `.264` files from these pcap files without error, we created the attached scripts. These scripts

Contact: Barry O'Mahony
Intel Labs
Hillsboro, OR USA
Tel: +1 (503) 264-8579
Email: barry.omahony@intel.com

are nowhere near as capable as the `H264AnnexBExtractor` program, and will only extract bitstreams from the particular form of pcap file in question (i.e., zero packet errors, RTP packets, and other details described below). However, if desired, they could be enhanced in the future to work with other types of pcap files.

Pcap Format

Pcap files follow a simple format, consisting of a single global header, followed by a sequence of packets, each with its own header [iii]:



Global Header

The script files read the 24-byte byte global header and discard it. Little Endian order is assumed, although the script could be enhanced to read the 'magic number' in the global header, and determine the byte order to be used when reading the remainder of the file. Similarly, other information in the global header is not used, and Ethernet packet types are assumed.

Packet Header

Each 16-byte packet header contains information about packet timing and packet size. The script reads the packet size information and uses that to read the packet data field, and discards the timing information.

Packet Data

The scripts assume that the H.264 bitstream was encapsulated in RTP packets contained in Ethernet frames, with 14-byte Ethernet headers, 20-byte IPV4 headers, 8-byte UDP headers, and 40-byte RTP headers (i.e., 24-byte RTP header extension). These bytes are simply discarded.

RFC-3984 Encapsulation

The H.264 bitstream is assumed to be contained in the RTP packet payload data, formatted according to RFC-3984[iv]. Examination of the target pcap files with Wireshark[v] confirmed that the only packet types used were those that contained single NAL Units, where an RTP packet contained exactly one NAL Unit, and FU-A fragmentation packets, where a NAL Unit was spread out over an integral number of multiple RTP packets. The scripts assume these two alternatives are the only fragmentation methods used. Single NAL Unit packets are written into the H.264 Annex B-formatted output files after being prepended with the Annex B start code prefix. Fragmented NAL Units are reassembled from the multiple RTP packets, with the correct NAL Unit header computed, and then written to the output file with the start code prefix.

Conclusion

The scripts provided in this contribution are being made available at the request of some members of the JEG group. The author greatly appreciates the assistance they have provided in answering questions about using the files in the JEG264HMIX1 database, and in the use of the Virtualbox environment used by the group. While these scripts are very simple and limited in capability, other members of the group may find them of some value in their research.

Annex I. File `pcapto264script.py`

```
#! python

#
# Script for converting a group of pcap files to H.264 Annex B bitstream files
# Rev. 09-Dec.-2013 Barry O'Mahony Intel Corp.

# Copyright (C) 2013 Intel Corporation
#
# This program is free software; you can redistribute it and/or modify it under the terms
# of the GNU General Public License as published by the Free Software Foundation; either
# version 2 of the License, or (at your option) any later version.
#
# This program is distributed in the hope that it will be useful, but WITHOUT ANY WARRANTY;
# without even the implied warranty of MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.
# See the GNU General Public License for more details.
#
# You should have received a copy of the GNU General Public License along with this
# program; if not, write to the Free Software Foundation, Inc., 51 Franklin Street,
# Fifth Floor, Boston, MA  02110-1301, USA.
#

# Developed for Python 3.32, Windows build.
# Copy this script and the script make264frompcap.py to the directory
# containing the pcap files to be converted.  The script searches the directory
# and all its subdirectories for pcap files (with extension .pcap) matching
# the critereon defined by the command line parameters, and converts them to
# bitstream files with extension .264.  They are all placed in a subdirectory
# with a name in the form .\264files-<time><date>, i.e., a unique subdirectory
# is created every time the script is run.
# The script searches for pcap files whose base name contains the string
# 'hrcNN', where NN is a two digit numeral in the range 00-99.  The values of
# NN entered in the command line determines the pcap files that are converted.
#
# usage:  >py pcapto264script.py N1 N2 N3 ... NN
# where N1, N2, etc. are numbers in the range 0-99, separated by spaces, that
# determine the values of hrcNN that are converted.
#
# Also, pcap filenames are assumed to be in the form:
# <text string 1>srcMM_hrcNN<text string 2>.pcap.  The prefix and suffix text
# strings are often quite lengthy, so the filenames of the output .264 files do
# not contain them.
#

import sys
import os
import time
import subprocess
sys.stdout.write("Python %s\n\n" % (sys.version,))


#
# first let's make the output subdirectory.
# define a unique directory name every time it is run,
#

outdir = '.\\'+'264files-'+time.strftime("%H%M%S-%d-%m-%y")
os.mkdir(outdir)

#
# now let's make the .264 bitstream file for every selected pcap file.
# Walk through all the subdirectories, looking for files with the extension
# .pcap, containing one of the strings 'hrcNN', where NN is a two-digit
# representation of one of the numbers entered in the command line.
# The script make264frompcap.py is called to do the actual conversion
#

for subdir, dirs, files in os.walk(".\\", topdown=False) :
    for file in files:
        if (file.find('.pcap') != -1)  :
            for rawarg in sys.argv[1:] :
                if file.find('hrc' + ("%02d" % int(rawarg))) != -1 :
#                    print (os.path.join(subdir,file ))
                    shortfile = file[file.find('src'):]
                    outputfile = shortfile[: shortfile.find('hrc')+5] + '.264'
#                    print (os.path.join(subdir,shorterfile ))
                    subprocess.call(["make264frompcap.py", os.path.join(subdir,file),
                                    os.path.join(outdir,outputfile)], shell=True)
```

Annex II. File `make264frompcap.py`

```
#! python

#
# rev. 09-Dec.-2013 Barry O'Mahony Intel Corp.

# Copyright (C) 2013 Intel Corporation
#
# This program is free software; you can redistribute it and/or modify it under the terms
# of the GNU General Public License as published by the Free Software Foundation; either
# version 2 of the License, or (at your option) any later version.
#
# This program is distributed in the hope that it will be useful, but WITHOUT ANY WARRANTY;
# without even the implied warranty of MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.
# See the GNU General Public License for more details.
#
# You should have received a copy of the GNU General Public License along with this
# program; if not, write to the Free Software Foundation, Inc., 51 Franklin Street,
# Fifth Floor, Boston, MA  02110-1301, USA.
#

# script to make Annex B H.264 bitstream files from pcap files.
# Developed for Python 3.32, Windows build.
#
# this version assumes the file starts with a 24-byte global header;
# the RTP packets start after that.
# Each packet starts with an 16-byte pcap header which specifies capture time
# and length. It is assumed that H.264 was encapsulated in RTP packets with
# 14 byte ethernet headers, # 20 byte IPV4 headers, 8-byte UDP headers,
# and 40-byte RTP headers (24-byte header extension), with no transmission
# errors (no missing packets), and only RFC-3984 Single NAL Unit and
# FU-A fragmentation packets present.
#
# All start codes prefixes are 32-bit.
#
# Usage: py make264frompcap.py <input file> <output file>
#
#

import sys
# sys.stdout.write("Python %s\n\n" % (sys.version,))

rtpfile = open(sys.argv[1], 'rb')
# print('File: ' + sys.argv[1])

junkbytes=rtpfile.read(24) # toss the pcap global header
# print (junkbytes)

#
# OK we've opened the file and stripped off the header; now let's
# process each individual packet
#
startprefix = b'\x00\x00\x00\x01'
outname = sys.argv[2]
outfile = open(outname, 'wb')
i=0
while 1 :
    pcapjunk = rtpfile.read(8) # don't do anything with timestamp stuff
    sizebytes = rtpfile.read(4)
    if sizebytes :
        packsize = int.from_bytes(sizebytes, byteorder='little') - (42+40) # RTP payload size
        rtpfile.read(4+42+40) # throw away rest of PCAP header and Eth/IP/UDP/RTP header bytes
        packetcontents = rtpfile.read(packsize) # get the RTP payload

        if packsize : # some packets just have the marker bit set. with no payload
#
# We need to handle FU-A's specially.  Let's see if we have one
#
            if packetcontents[0]& 31 != 28 : # see if just a single NAL packet
                outfile.write(startprefix)
                outfile.write(packetcontents)
            elif packetcontents[1] & 128 :                 # see if first packet in FU-A group
                outfile.write(startprefix)
#
# compute NAL unit header if so
#
                realheader=packetcontents[0] & 224 | packetcontents[1] & 31
                outfile.write(realheader.to_bytes(1, byteorder='big'))
                outfile.write(packetcontents[2:])
            else :
                outfile.write(packetcontents[2:]) # if not 1st fragment, just append payload
        i=i+1
    else:
        break
# print("Number of packets: ", i)
print("Output file: " + outname)
rtpfile.close()
outfile.close()
```

References

[i] http://vqegstl.ugent.be/?q=node/31
[ii] http://vqegstl.ugent.be/?q=node/14
[iii] Libpcap File Format, http://wiki.wireshark.org/Development/LibpcapFileFormat
[iv] http://tools.ietf.org/html/rfc3984
[v] http://www.wireshark.org/