

Screen Mirroring Video Trace Datasets from User-Generated Data: A Quick Reference Guide

Last Update: 27 November 2016

Marios Kastrinakis^a, Ghada Badawy^b, Mohammed N. Smadi^c and Polychronis Koutsakis^d

^a*School of ECE, Technical University of Crete, Greece*

^b*Computing Infrastructure Research Center, McMaster University, Canada*

^c*ECE Department, McMaster University, Canada*

^d*School of Engineering and Information Technology, Murdoch University, Australia*

Email: makastrinakis@electronics.tuc.gr, gbadawy@gmail.com, smadim2@mcmaster.ca, p.koutsakis@murdoch.edu.au

1. Introduction

For the purposes of our study entitled as “Video Frame Size Modeling for User-Generated Traffic in an Enterprise-Like Environment”, we worked with two different datasets, encoded with the H.264 video coding standard. Those datasets are based on real user-generated screen mirroring traffic data from a large shared cube space resembling an enterprise environment. Each participant in the data collection ran trace collection scripts on his/her PC, for about a month. One script polled the operating system every 33.3 msec to record the name of the main application that the participant was working on. Another script recorded the participant’s screen at 30 fps using encoding parameters that resembled a Miracast hardware encoder as much as possible (more details on this in section 3). The actual video was not recorded, but only the statistics of the encoded video were collected (i.e., I and P frame sizes). The scripts started automatically each work day at 8am and stopped at 7pm. When a participant locked his/her screen the scripts would report that the user is idle for that duration and video traces collection would stop till the user unlocks his/her machine. All of the users were using Windows 7 machines.

2. Recording Method

A recording framework was deployed on every host machine. It was running and logging in the background during the recording period. The FFmpeg program was used for video traffic recording. It logged the compressed H.264 video information (i.e., frames sizes, GOP structure, frames’ time of arrival etc.) of the host’s machine desktop. The frame resolution was the same as the PC’s screen resolution (i.e., it is not a constant) and the frame rate was 30 fps. It is worth mentioning that even though FFmpeg was running constantly, it was capable to log video traffic information only if the host’s machine GPU was active (i.e., the host machine was not in hibernation, sleep or monitor energy saving mode). As for active applications usage recording, a Windows PowerShell script was used for logging the name of the application in the foreground, followed by the current timestamp. Windows PowerShell was programmed to log the application’s name every 33.3 msec (in order to keep up with FFmpeg logs, where we had one frame every 33.3 msec). We should also note that Windows PowerShell is capable to report the application’s name only if the host machine is unlocked and the user is not logged off.

3. Datasets Encoding

The main difference between the two datasets lies in the different encoding of video traces. The first dataset (Dataset 1) has been encoded with the High 4:2:2 Profile of the H.264 standard, which is typical for professional applications. This profile can generate I, P and B frames. However, in our datasets the `-tune zero latency` command was used in FFmpeg to prohibit the encoder from producing B-Frames, in order to minimize latency. For this dataset, we have a GOP structure of 60 frames in length, where every GOP starts with an I-Frame and the rest 59 frames are of type P. The second dataset (Dataset 2) has been encoded with different encoding parameters. Those parameters try to resemble a Miracast hardware encoder as closely as possible. Since I-Frames size are much larger than P-Frames, Miracast encoders do not use I-Frames but use Periodic Intra Refresh instead. This enables each frame (in our case each I-Frame) to be capped to the same size by using a column of intra blocks that move across the video trace from one side to the other, thereby “refreshing” the image. In effect, instead of a big keyframe (in our case an I-Frame), the

keyframe is spread over many frames (in our case P-Frames). For this dataset, we do not have a GOP structure. We only have P-Frames with an exception of one I-Frame whenever the host computer starts or its user logs on.

4. Datasets Structure

The datasets are already organized and stored as Matlab files, in order to facilitate their easy and direct usage. Every dataset is partitioned into two separate files named as “part1.mat” and “part2.mat”. Those files can be loaded to Matlab workspace directly and each one contain the video traces information organized and stored in Matlab tables. As for Dataset 1, every distinct file contains a table with dimensions of 7892610x4 and for Dataset 2, a table with dimensions of 10107570x4 (for that reason, please ensure that your machine has enough memory available). Every table has four columns named as “Top_App”, “Frame_Type”, “Frame_Size” and “GoP_number” and every row corresponds to a recorded frame. The first column contains the application’s name that the user had opened on the foreground, the second column contains the type of the recorded frame, the third column contains the frame’s size in bytes and the fourth column the index number of the GOP that the frame belongs to. In addition, we would like to mention that we have sorted the frames according to applications name but we have preserved the time consistency of recording. Finally, the two tables from the different files of a dataset must be concatenated, in order to have in place the complete dataset.