Network Working Group
Internet-Draft
Intended status: Informational
Expires: January 18, 2018

X. Zhu
S. Mena
Cisco Systems
Z. Sarker
Ericsson AB

July 17, 2017

Modeling Video Traffic Sources for RMCAT Evaluations
draft-ietf-rmcat-video-traffic-model-03

Abstract

This document describes two reference video traffic source models for evaluating RMCAT candidate algorithms. The first model statistically characterizes the behavior of a live video encoder in response to changing requests on target video rate. The second model is trace-driven, and emulates the encoder output by scaling the pre-encoded video frame sizes from a widely used video test sequence. Both models are designed to strike a balance between simplicity, repeatability, and authenticity in modeling the interactions between a video traffic source and the congestion control module.

Table of Contents

1. Introduction

When evaluating candidate congestion control algorithms designed for real-time interactive media, it is important to account for the characteristics of traffic patterns generated from a live video encoder. Unlike synthetic traffic sources that can conform perfectly to the rate changing requests from the congestion control module, a live video encoder can be sluggish in reacting to such changes. Output rate of a live video encoder also typically deviates from the target rate due to uncertainties in the encoder rate control process. Consequently, end-to-end delay and loss performance of a real-time media flow can be further impacted by rate variations introduced by the live encoder.

On the other hand, evaluation results of a candidate RMCAT algorithm should mostly reflect performance of the congestion control module, and somewhat decouple from peculiarities of any specific video codec. It is also desirable that evaluation tests are repeatable, and be easily duplicated across different candidate algorithms.

One way to strike a balance between the above considerations is to evaluate RMCAT algorithms using a synthetic video traffic source model that captures key characteristics of the behavior of a live video encoder. To this end, this draft presents two reference models. The first is based on statistical modelling; the second is trace-driven. The draft also discusses the pros and cons of each approach, as well as how both approaches can be combined.

2. Terminology

The key words "MUST", "MUST NOT", "REQUIRED", "SHALL", "SHALL NOT", "SHOULD", "SHOULD NOT", "RECOMMENDED", "MAY", and "OPTIONAL" in this document are to be interpreted as described RFC2119 [RFC2119].

3. Desired Behavior of A Synthetic Video Traffic Model

A live video encoder employs encoder rate control to meet a target rate by varying its encoding parameters, such as quantization step size, frame rate, and picture resolution, based on its estimate of the video content (e.g., motion and scene complexity). In practice, however, several factors prevent the output video rate from perfectly conforming to the input target rate.

Due to uncertainties in the captured video scene, the output rate typically deviates from the specified target. In the presence of a significant change in target rate, it sometimes takes several frames before the encoder output rate converges to the new target. Finally, while most of the frames in a live session are encoded in predictive mode, the encoder can occasionally generate a large intra-coded frame (or a frame partially containing intra-coded blocks) in an attempt to recover from losses, to re-sync with the receiver, or during the transient period of responding to target rate or spatial resolution changes.

Hence, a synthetic video source should have the following capabilities:

- To change bitrate. This includes ability to change framerate and/or spatial resolution, or to skip frames when required.

- To fluctuate around the target bitrate specified by the congestion control module.

- To show a delay in convergence to the target bitrate.

- To generate intra-coded or repair frames on demand.

While there exist many different approaches in developing a synthetic video traffic model, it is desirable that the outcome follows a few common characteristics, as outlined below.

- Low computational complexity: The model should be computationally lightweight, otherwise it defeats the whole purpose of serving as a substitute for a live video encoder.

- Temporal pattern similarity: The individual traffic trace instances generated by the model should mimic the temporal pattern of those from a real video encoder.

- Statistical resemblance: The synthetic traffic should match the outcome of the real video encoder in terms of statistical characteristics, such as the mean, variance, peak, and autocorrelation coefficients of the bitrate. It is also important that the statistical resemblance should hold across different time scales, ranging from tens of milliseconds to sub-seconds.

- Wide range of coverage: The model should be easily configurable to cover a wide range of codec behaviors (e.g., with either fast or slow reaction time in live encoder rate control) and video content variations (e.g, ranging from high-motion to low-motion).

These distinct behavior features can be characterized via simple statistical models, or a trace-driven approach. We present an example of each in Section 5 and Section 6

4. Interactions Between Synthetic Video Traffic Source and Other Components at the Sender

Figure 1 depitcs the interactions of the synthetic video encoder with other components at the sender, such as the application, the congestion control module, the media packet transport module, etc. Both reference models, as described later in Section 5 and Section 6, follow the same set of interactions.

The synthetic video encoder takes in raw video frames captured by the camera and then dynamically generates a sequence of encoded video frames with varying size and interval. These encoded frames are processed by other modules in order to transmit the video stream over the network. During the lifetime of a video transmission session, the synthetic video encoder will typically be required to adapt its

encoding bitrate, and sometimes the spatial resolution and frame rate.

In our model, the synthetic video encoder module has a group of incoming and outgoing interface calls that allow for interaction with other modules. The following are some of the possible incoming interface calls --- marked as (a) in Figure 1 --- that the synthetic video encoder may accept. The list is not exhaustive and can be complemented by other interface calls if deemed necessary.

- Target rate R_v: target rate request to the encoder, typically from the congestion control module and updated dynamically over time. Depending on the congestion control algorithm in use, the update requests can either be periodic (e.g., once per second), or on-demand (e.g., only when a drastic bandwidth change over the network is observed).

- Target frame rate FPS: the instantaneous frame rate measured in frames-per-second at a given time. This depends on the native camera capture frame rate as well as the target/preferred frame rate configured by the application or user.

- Frame resolution XY: the 2-dimensional vector indicating the preferred frame resolution in pixels. Several factors govern the resolution requested to the synthetic video encoder over time. Examples of such factors are the capturing resolution of the native camera; or the current target rate R_v, since very small resolutions do not make sense with very high bitrates, and vice-versa.

- Instant frame skipping: the request to skip the encoding of one or several captured video frames, for instance when a drastic decrease in available network bandwidth is detected.

- On-demand generation of intra (I) frame: the request to encode another I frame to avoid further error propagation at the receiver, if severe packet losses are observed. This request typically comes from the error control module.

An example of outgoing interface call --- marked as (b) in Figure 1 --- is the rate range, that is, the dynamic range of the video encoder's output rate for the current video contents: [R_min, R_max]. Here, R_min and R_max are meant to capture the dynamic rate range the

encoder is capable of outputting. This typically depends on the video content complexity and/or display type (e.g., higher R_max for video contents with higher motion complexity, or for displays of higher resolution). Therefore, these values will not change with R_v, but may change over time if the content is changing.

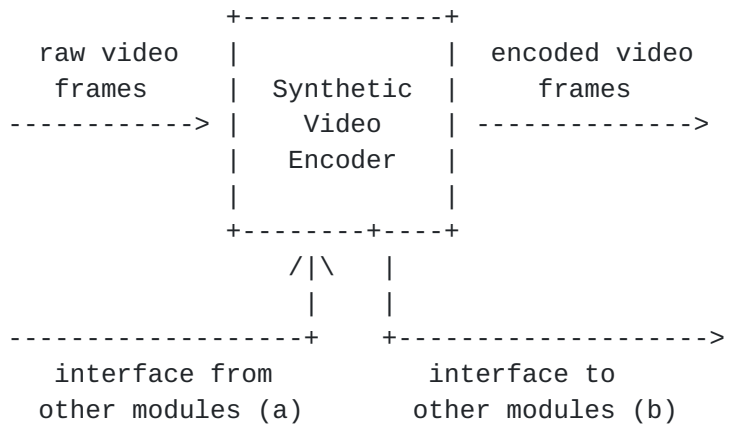

Figure 1: Interaction between synthetic video encoder and other modules at the sender

5. A Statistical Reference Model

In this section, we describe one simple statistical model of the live video encoder traffic source. Figure 2 summarizes the list of tunable parameters in this statistical model. A more comprehensive survey of popular methods for modelling video traffic source behavior can be found in [Tanwir2013].

Notation	Parameter Name	Example Value
R_v	Target rate request to encoder	1 Mbps
FPS	Target frame rate of encoder output	30 Hz
tau_v	Encoder reaction latency	0.2 s
K_d	Burst duration during transient	8 frames
K_B	Burst frame size during transient	13.5 KBytes*
t0	Reference frame interval 1/FPS	33 ms
B0	Reference frame size R_v/8/FPS	4.17 KBytes
SCALE_t	Scaling parameter of the zero-mean Laplacian distribution describing deviations in normalized frame interval (t-t0)/t0	0.15
SCALE_B	Scaling parameter of the zero-mean Laplacian distribution describing deviations in normalized frame size (B-B0)/B0	0.15
R_min	minimum rate supported by video encoder or content activity	150 Kbps
R_max	maximum rate supported by video encoder or content activity	1.5 Mbps

* Example value of K_B for a video stream encoded at 720p and 30 frames per second, using H.264/AVC encoder.

Figure 2: List of tunable parameters in a statistical video traffic source model.

5.1. Time-damped response to target rate update

While the congestion control module can update its target rate request R_v at any time, our model dictates that the encoder will only react to such changes after tau_v seconds from a previous rate transition. In other words, when the encoder has reacted to a rate

change request at time t, it will simply ignore all subsequent rate change requests until time t+tau_v.

5.2. Temporary burst and oscillation during transient

The output rate R_o during the period [t, t+tau_v] is considered to be in transient. Based on observations from video encoder output data, we model the transient behavior of an encoder upon reacting to a new target rate request in the form of high variation in output frame sizes. It is assumed that the overall average output rate R_o during this period matches the target rate R_v. Consequently, the occasional burst of large frames are followed by smaller-than average encoded frames.

This temporary burst is characterized by two parameters:

- burst duration K_d: number of frames in the burst event; and
- burst frame size K_B: size of the initial burst frame which is typically significantly larger than average frame size at steady state.

It can be noted that these burst parameters can also be used to mimic the insertion of a large on-demand I frame in the presence of severe packet losses. The values of K_d and K_B typically depend on the type of video codec, spatial and temporal resolution of the encoded stream, as well as the video content activity level.

5.3. Output rate fluctuation at steady state

We model output rate R_o during steady state as randomly fluctuating around the target rate R_v. The output traffic can be characterized as the combination of two random processes denoting the frame interval t and output frame size B over time. These two random processes capture two sources of variations in the encoder output:

- Fluctuations in frame interval: the intervals between adjacent frames have been observed to fluctuate around the reference interval of t0 = 1/FPS. Deviations in normalized frame interval DELTA_t = (t-t0)/t0 can be modelled by a zero-mean Laplacian distribution with scaling parameter SCALE_t. The value of SCALE_t dictates the "width" of the Laplacian distribution and therefore the amount of fluctuations in actual frame intervals (t) with respect to the reference t0.
- Fluctuations in frame size: size of the output encoded frames also tend to fluctuate around the reference frame size B0=R_v/8/FPS. Likewise, deviations in the normalized frame size DELTA_B =

(B-B0)/B0 can be modelled by a zero-mean Laplacian distribution with scaling parameter SCALE_B. The value of SCALE_B dictates the "width" of this second Laplacian distribution and correspondingly the amount of fluctuations in output frame sizes (B) with respect to the reference target B0.

Both values of SCALE_t and SCALE_B can be obtained via parameter fitting from empirical data captured for a given video encoder. Example values are listed in Figure 2 based on empirical data presented in [IETF-Interim].

5.4. Rate range limit imposed by video content

The output rate R_o is further clipped within the dynamic range [R_min, R_max], which in reality are dictated by scene and motion complexity of the captured video content. In our model, these parameters are specified by the application.

6. A Trace-Driven Model

We now present the second approach to model a video traffic source. This approach is based on running an actual live video encoder on a set of chosen raw video sequences and using the encoder's output traces for constructing a synthetic live encoder. With this approach, the recorded video traces naturally exhibit temporal fluctuations around a given target rate request R_v from the congestion control module.

The following list summarizes the main steps of this approach:

1) Choose one or more representative raw video sequences.

2) Encode the sequence(s) using an actual live video encoder. Repeat the process for a number of bitrates. Keep only the sequence of frame sizes for each bitrate.

3) Construct a data structure that contains the output of the previous step. The data structure should allow for easy bitrate lookup.

4) Upon a target bitrate request R_v from the controller, look up the closest bitrates among those previously stored. Use the frame size sequences stored for those bitrates to approximate the frame sizes to output.

5) The output of the synthetic encoder contains "encoded" frames with zeros as contents but with realistic sizes.

Section 6.1 explains steps 1), 2), and 3), Section 6.2 elaborates on steps 4) and 5). Finally, Section 6.3 briefly discusses the possibility to extend the model for supporting variable frame rate and/or variable frame resolution.

6.1. Choosing the video sequence and generating the traces

The first step we need to perform is a careful choice of a set of video sequences that are representative of the use cases we want to model. Our use case here is video conferencing, so we must choose a low-motion sequence that resembles a "talking head", for instance a news broadcast or a video capture of an actual conference call.

The length of the chosen video sequence is a tradeoff. If it is too long, it will be difficult to manage the data structures containing the traces. If it is too short, there will be an obvious periodic pattern in the output frame sizes, leading to biased results when evaluating congestion controller performance. In our experience, a sequence whose length is between 2 and 4 minutes is a fair tradeoff.

Once we have chosen the raw video sequence, denoted S, we use a live encoder, e.g. [H264] or [HEVC] to produce a set of encoded sequences. As discussed in Section 3, a live encoder's output bitrate can be tuned by varying three input parameters, namely, quantization step size, frame rate, and picture resolution. In order to simplify the choice of these parameters for a given target rate, we assume a fixed frame rate (e.g. 30 fps) and a fixed resolution (e.g., 720p). See section 6.3 for a discussion on how to relax these assumptions.

Following these simplifications, we run the chosen encoder by setting a constant target bitrate at the beginning, then letting the encoder vary the quantization step size internally while encoding the input video sequence. Besides, we assume that the first frame is encoded as an I-frame and the rest are P-frames. We further assume that the encoder algorithm does not use knowledge of frames in the future when encoding a given frame.

Given R_min and R_max, which are the minimum and maximum bitrates at which the synthetic codec is to operate (see Section 4), we divide the bitrate range between R_min and R_max in n_s + 1 bitrate steps of length l = (R_max - R_min) / n_s. We then use the following simple algorithm to encode the raw video sequence.

```
r = R_min
while r <= R_max do
    Traces[r] = encode_sequence(S, r, e)
    r = r + l
```

where function encode_sequence takes as parameters, respectively, a raw video sequence, a constant target rate, and an encoder algorithm; it returns a vector with the sizes of frames in the order they were encoded. The output vector is stored in a map structure called Traces, whose keys are bitrates and whose values are vectors of frame sizes.

The choice of a value for n_s is important, as it determines the number of vectors of frame sizes stored in map Traces. The minimum value one can choose for n_s is 1, and its maximum value depends on the amount of memory available for holding the map Traces. A reasonable value for n_s is one that makes the steps' length l = 200 kbps. We will further discuss step length l in the next section.

Finally, note that, as mentioned in previous sections, R_min and R_max may be modified after the initial sequences are encoded. Hence, the algorithm described in the next section also covers the cases when the current target bitrate is less than R_min, or greater than R_max.

6.2. Using the traces in the syntethic codec

The main idea behind the trace-driven synthetic codec is that it mimics a real live codec's rate adaptation when the congestion controller updates the target rate R_v dynamically. It does so by switching to a different frame size vector stored in the map Traces when needed.

6.2.1. Main algorithm

We maintain two variables r_current and t_current:

* r_current points to one of the keys of map Traces. Upon a change in the value of R_v, typically because the congestion controller detects that the network conditions have changed, r_current is updated to the greatest key in Traces that is less than or equal to the new value of R_v. For the moment, we assume the value of R_v to be clipped in the range [R_min, R_max].

```
r_current = r
such that
   ( r in keys(Traces)  and
     r <= R_v  and
     (not(exists) r' in keys(Traces) such that r < r' <= R_v) )
```

* t_current is an index to the frame size vector stored in Traces[r_current]. It is updated every time a new frame is due. We

assume all vectors stored in Traces to have the same size, denoted size_traces. The following equation governs the update of t_current:

```
if t_current < SkipFrames then
  t_current = t_current + 1
else
  t_current = ((t_current+1-SkipFrames) % (size_traces- SkipFrames))
              + SkipFrames
```

where operator % denotes modulo, and SkipFrames is a predefined constant that denotes the number of frames to be skipped at the beginning of frame size vectors after t_current has wrapped around. The point of constant SkipFrames is avoiding the effect of periodically sending a (big) I-frame followed by several smaller-than-normal P-frames. We typically set SkipFrames to 20, although it could be set to 0 if we are interested in studying the effect of sending I-frames periodically.

We initialize r_current to R_min, and t_current to 0.

When a new frame is due, we need to calculate its size. There are three cases:

a) R_min <= R_v < Rmax: In this case we use linear interpolation of the frame sizes appearing in Traces[r_current] and Traces[r_current + l]. The interpolation is done as follows:

```
size_lo = Traces[r_current][t_current]
size_hi = Traces[r_current + l][t_current]
distance_lo = ( R_v - r_current ) / l
framesize = size_hi * distance_lo + size_lo * (1 - distance_lo)
```

b) R_v < R_min: In this case, we scale the trace sequence with the lowest bitrate, in the following way:

```
factor = R_v / R_min
framesize = max(1, factor * Traces[R_min][t_current])
```

c) R_v >= R_max: We also use scaling for this case. We use the trace sequence with the greatest bitrate:

```
factor = R_v / R_max
framesize = factor * Traces[R_max][t_current]
```

In case b), we set the minimum to 1 byte, since the value of factor can be arbitrarily close to 0.

6.2.2. Notes to the main algorithm

* Reacting to changes in target bitrate. Similarly to the statistical model presented in Section 5, the trace-driven synthetic codec can have a time bound, tau_v, to reacting to target bitrate changes. If the codec has reacted to an update in R_v at time t, it will delay any further update to R_v to time t + tau_v. Note that, in any case, the value of tau_v cannot be chosen shorter than the time between frames, i.e. the inverse of the frame rate.

* I-frames on demand. The synthetic codec could be extended to simulate the sending of I-frames on demand, e.g., as a reaction to losses. To implement this extension, the codec's API is augmented with a new function to request a new I-frame. Upon calling such function, t_current is reset to 0.

* Variable length l of steps defined between R_min and R_max. In the main algorithm's description, the step length l is fixed. However, if the range [R_min, R_max] is very wide, it is also possible to define a set of steps with a non-constant length. The idea behind this modification is that the difference between 400 kbps and 600 kbps as bitrate is much more important than the difference between 4400 kbps and 4600 kbps. For example, one could define steps of length 200 Kbps under 1 Mbps, then steps of length 300 kbps between 1 Mbps and 2 Mbps; 400 kbps between 2 Mbps and 3 Mbps, and so on.

6.3. Varying frame rate and resolution

The trace-driven synthetic codec model explained in this section is relatively simple because we have fixed the frame rate and the frame resolution. The model could be extended to have variable frame rate, variable spatial resolution, or both.

When the encoded picture quality at a given bitrate is low, one can potentially decrease the frame rate (if the video sequence is currently in low motion) or the spatial resolution in order to improve quality-of-experince (QoE) in the overall encoded video. On the other hand, if target bitrate increases to a point where there is no longer a perceptible improvement in the picture quality of individual frames, then one might afford to increase the spatial resolution or the frame rate (useful if the video is currently in high motion).

Many techniques have been proposed to choose over time the best combination of encoder quatization step size, frame rate, and spatial resolution in order to maximize the quality of live video codecs [Ozer2011][Hu2010]. Future work may consider extending the trace-driven codec to accommodate variable frame rate and/or resolution.

From the perspective of congestion control, varying the spatial resolution typically requires a new intra-coded frame to be generated, thereby incurring a temporary burst in the output traffic pattern. The impact of frame rate change tends to be more subtle: reducing frame rate from high to low leads to sparsely spaced larger encoded packets instead of many densely spaced smaller packets. Such difference in traffic profiles may still affect the performance of congestion control, especially when outgoing packets are not paced at the transport module. We leave the investigation of varying frame rate to future work.

7. Combining The Two Models

It is worthwhile noting that the statistical and trace-driven models each has its own advantages and drawbacks. While both models are fairly simple to implement, it takes significantly greater effort to fit the parameters of a statistical model to actual encoder output data whereas it is straightforward for a trace-driven model to obtain encoded frame size data. On the other hand, once validated, the statistical model is more flexible in mimicking a wide range of encoder/content behaviors by simply varying the correponding parameters in the model. In this regard, a trace-driven model relies -- by definition -- on additional data collection efforts for accommodating new codecs or video contents.

In general, the trace-driven model is more realistic for mimicking ongoing, steady-state behavior of a video traffic source whereas the statistical model is more versatile for simulating transient events (e.g., when target rate changes from A to B with temporary bursts during the transition). It is also possible to combine both models into a hybrid approach, using traces during steady-state and statistical model during transients.

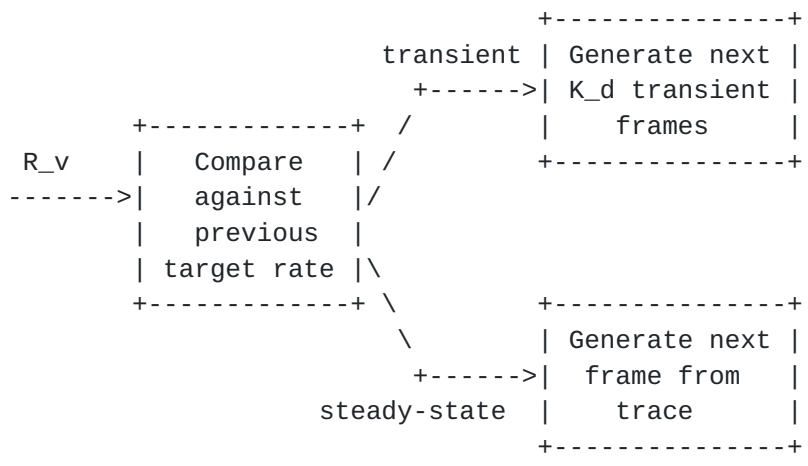

Figure 3: Hybrid approach for modeling video traffic

As shown in Figure 3, the video traffic model operates in transient state if the requested target rate R_v is substantially higher than the previous target, or else it operates in steady state. During transient state, a total of K_d frames are generated by the statistical model, resulting in 1 big burst frame with size K_B followed by K_d-1 smaller frames. When operating at steady-state, the video traffic model simply generates a frame according to the trace-driven model given the target rate, while modulating the frame interval according to the distribution specified by the statistical model. One example criterion for determining whether the traffic model should operate in transient state is whether the rate increase exceeds 10% of previous target rate.

8. Implementation Status

The statistical model has been implemented as a traffic generator module within the [ns-2] network simulation platform.

More recently, both the statistical and trace-driven models have been implemented as a stand-alone traffic source module. This can be easily integrated into network simulation platforms such as [ns-2] and [ns-3], as well as testbeds using a real network. The stand-alone traffic source module is available as an open source implementation at [Syncodecs].

9. IANA Considerations

There are no IANA impacts in this memo.

10. References

10.1. Normative References

[H264] ITU-T Recommendation H.264, "Advanced video coding for generic audiovisual services", 2003, <http://www.itu.int/rec/T-REC-H.264-201304-I>.

[HEVC] ITU-T Recommendation H.265, "High efficiency video coding", 2015.

[RFC2119] Bradner, S., "Key words for use in RFCs to Indicate Requirement Levels", BCP 14, RFC 2119, DOI 10.17487/RFC2119, March 1997, <http://www.rfc-editor.org/info/rfc2119>.

10.2. Informative References

[Hu2010] Hu, H., Ma, Z., and Y. Wang, "Optimization of Spatial, Temporal and Amplitude Resolution for Rate-Constrained Video Coding and Scalable Video Adaptation", in Proc. 19th IEEE International Conference on Image Processing, (ICIP'12), September 2012.

[IETF-Interim] Zhu, X., Mena, S., and Z. Sarker, "Update on RMCAT Video Traffic Model: Trace Analysis and Model Update", April 2017, <https://www.ietf.org/proceedings/interim-2017-rmcat-01/slides/slides-interim-2017-rmcat-01-sessa-update-on-video-traffic-model-draft-00.pdf>.

[ns-2] "The Network Simulator - ns-2", <http://www.isi.edu/nsnam/ns/>.

[ns-3] "The Network Simulator - ns-3", <https://www.nsnam.org/>.

[Ozer2011] Ozer, J., "Video Compression for Flash, Apple Devices and HTML5", ISBN 13:978-0976259503, 2011.

[Syncodecs] Mena, S., D'Aronco, S., and X. Zhu, "Syncodecs: Synthetic codecs for evaluation of RMCAT work", <https://github.com/cisco/syncodecs>.

[Tanwir2013]
Tanwir, S. and H. Perros, "A Survey of VBR Video Traffic Models", IEEE Communications Surveys and Tutorials, vol. 15, no. 5, pp. 1778-1802., October 2013.

Authors' Addresses

Xiaoqing Zhu
Cisco Systems
12515 Research Blvd., Building 4
Austin, TX 78759
USA

Email: xiaoqzhu@cisco.com

Sergio Mena de la Cruz
Cisco Systems
EPFL, Quartier de l'Innovation, Batiment E
Ecublens, Vaud 1015
Switzerland

Email: semena@cisco.com

Zaheduzzaman Sarker
Ericsson AB
Luleae, SE 977 53
Sweden

Phone: +46 10 717 37 43
Email: zaheduzzaman.sarker@ericsson.com