An ns-3 Implementation of a Bursty Traffic Framework for Virtual Reality Sources

Mattia Lecci
Dept. of Information Engineering
University of Padova
Padova, Italy
leccimat@dei.unipd.it

Andrea Zanella
Dept. of Information Engineering
University of Padova
Padova, Italy
zanella@dei.unipd.it

Michele Zorzi
Dept. of Information Engineering
University of Padova
Padova, Italy
zorzi@dei.unipd.it

ABSTRACT

Next-generation wireless communication technologies will allow users to obtain unprecedented performance, paving the way to new and immersive applications. A prominent application requiring high data rates and low communication delay is Virtual Reality (VR), whose presence will become increasingly stronger in the years to come. To the best of our knowledge, we propose the first traffic model for VR applications based on traffic traces acquired from a commercial VR streaming software, allowing the community to further study and improve the technology to manage this type of traffic. This work implements ns-3 applications able to generate and process large bursts of packets, enabling the possibility of analyzing APP-level end-to-end metrics, making the source code, as well as the acquired VR traffic traces, publicly available and open-source.

CCS CONCEPTS

• **Networks** → **Network simulations**;

KEYWORDS

Virtual Reality, Video Traffic, Traffic Modeling, ns-3

ACM Reference Format:
Mattia Lecci, Andrea Zanella, and Michele Zorzi. 2021. An ns-3 Implementation of a Bursty Traffic Framework for Virtual Reality Sources. In *2021 Workshop on ns-3 (WNS3 2021), June 23–24, 2021, Virtual Event, USA.* ACM, New York, NY, USA, 8 pages. https://doi.org/10.1145/3460797.3460807

1 INTRODUCTION

The growing demand for high-performance telecommunication networks is driving both industry and academia to push the boundaries of the achievable performance.

The International Telecommunication Union (ITU) proposes requirements for International Mobile Telecommunication-2020 (IMT-2020) Enhanced Mobile BroadBand (eMBB) such as peak Downlink (DL) data rate of 20 Gbps with 4 ms user plane latency [8], for example by exploiting the large bandwidth available in the Millimeter Wave (mmW) spectrum. Similarly, Wireless Local Area Networks (WLANs) are also harvesting the potential of the mmW band with a family of standards known as Wireless Gigabit (WiGig), including IEEE 802.11ad and 802.11ay. While the former, first standardized in 2012 [7] and later revised in 2016 [6], is able to reach bit rates up to 8 Gbps, the latter is close to being officially standardized [14] and promises bit rates up to 100 Gbps.

These specifications for wireless systems enable a new generation of demanding applications such as high-definition wireless monitor, eXtended Reality (XR) headsets and other high-end wearables, data center inter-rack connectivity, wireless backhauling, and office docking, among others [15].

In particular, XR, an umbrella name including technologies such as Virtual Reality (VR) and Augmented Reality (AR), has been targeted as a key application with growing interest in the consumer market [5]. Compact and portable devices with limited battery and computing power should be enabled to wirelessly support this type of demanding applications, to provide a fully immersive and realistic user experience.

To reach the limits of human vision, monitors with a resolution of 5073×5707 per eye with 120 FPS refresh rate will be needed [5]. These specifications suggest that rendering might be offloaded to a separate server as head-mounted displays should be light, silent, and comfortable enough to be worn for long periods of time. This poses a significant strain on the wireless connection, requiring ∼167 Gbps of uncompressed video stream. Clearly, real-time 360 video compression techniques allow to largely reduce these throughput requirements down to the order of 100–1000 Mbps, at the cost of some processing delay.

While throughput requirements are already very demanding, low latencies are the key to the success or the failure of XR applications. In fact, many studies showed that users tend to experience what is called *motion* or *cyber sickness* when their actions do not correspond to rapid reactions in the virtual world, causing disorientation and dizziness [3–5, 9, 11]. *Motion-to-photon* latency is thus required to be at most 20 ms, translating into a network latency for video frames of 5–9 ms [5, 15].

In its simplest and most ideal form, raw XR traffic with a fixed frame rate F could be modeled by periodic traffic, with period $1/F$ and constant frame size S proportional to the display resolution. Real traffic, though, is first of all compressed with one of the many existing video codecs, and then properly optimized for real-time low-latency streaming, resulting in encoded video frames of variable size. Furthermore, the complexity of a given scene can also affect the time required to render it as well as the obtainable compression factor. Together, these factors make both the video frame

Permission to make digital or hard copies of all or part of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies bear this notice and the full citation on the first page. Copyrights for components of this work owned by others than ACM must be honored. Abstracting with credit is permitted. To copy otherwise, or republish, to post on servers or to redistribute to lists, requires prior specific permission and/or a fee. Request permissions from permissions@acm.org.
WNS3 2021, June 23–24, 2021, Virtual Event, USA
© 2021 Association for Computing Machinery.
ACM ISBN 978-1-4503-9034-7/21/06…$15.00
https://doi.org/10.1145/3460797.3460807

size and the period random, possibly correlated both in time and in the frame-size/period domains.

Given the interest of both industry and consumers in XR applications and the peculiarity of the generated traffic, networks and networking protocols might be optimized to better support this type of traffic, ensuring its strict Quality of Service (QoS) requirements.

To the best of our knowledge, no prior work on XR traffic modeling exists. Cloud gaming [1] was identified as a closely related problem, where a remote server renders and streams a video to a client with limited computational resources, which only feeds basic information to the rendering servers, such as keys pressed and mouse movements. The main difference with the problem under analysis is given by the more restrictive QoS constraints of XR applications, mainly due to the limits imposed by motion sickness. Furthermore, in cloud gaming, client and server are often in different WLANs, making it harder to obtain reliable measurements of packet generation times. In fact, due to the specific constraints and requirements of XR applications, we expect the rendering server to be in a local network rather than being remotely accessed via the Internet.

Most works in the literature focus on network performance and limitations of cloud gaming [16], and we could find only two main contributions addressing traffic analysis and modeling. The authors of [2] provide a simple traffic analysis for three different games played on *OnLive*, a cloud gaming application that was shut down in 2015. The analysis focuses on packet-level statistics, such as packet size, inter-packet time, and bit rate. They measured the performance of the streaming service under speed-limited networks, showing an evident frame rate variability. In [12], the authors tried to model the traffic generated by two games, also played on the *OnLive* platform. In particular, they recognized that video frames were split into multiple fragments, and re-aggregated them before studying their statistics. A number of DL and Uplink (UL) data flows were recognized, and characterized in terms of *application packet data unit size* and *packet inter arrival time*. Unfortunately, correlation among successive video frames was not modeled and the analysis referred to a single game played with an average data rate of about 5 Mbps.

The novelty of this paper can be summarized as follows:

(1) To the best of our knowledge, this is the first generative model for APP-level XR traffic based on over 90 minutes of acquired and processed VR traffic, with adaptable data rate and frame rate;
(2) we provide a flexible Network Simulator 3 (ns-3) module for simulating applications with bursty behavior able to characterize both fragment-level and burst-level performance;
(3) we provide an implementation of the proposed XR traffic model, as well as a trace-based model, on the bursty application framework;
(4) as a side contribution, several acquired VR traffic traces are made available, allowing researchers to (i) use real VR traffic in their simulations and (ii) further analyze and improve VR traffic models.

In the remainder of this work, we will describe the traffic acquisition and analysis in Section 2, propose a flexible XR traffic model in Section 3, discuss about the ns-3 implementation of the *bursty application* framework in Section 4, validate the model and show a possible use case in Section 5, and finally draw the conclusions of this work and propose future works in Section 6.

2 VR TRAFFIC: ACQUISITION AND ANALYSIS

The traffic model that will be described in Section 3 is based on a set of acquisitions of VR traffic. In the remainder of this section, the acquisition setup and the statistical analysis of the acquired VR traffic traces will be presented.

2.1 Acquisition Setup

The setup comprises a desktop PC (equipped with an i7 processor, 32 GB of RAM, and an nVidia GTX 2080Ti graphics card) acting as a rendering server, and transmitting the information to a smartphone acting as a passive VR headset. The two nodes are connected via USB tethering to avoid random interference from other surrounding devices and the less stable wireless channel.

To stream the VR traffic, the rendering server runs the application *RiftCat 2.0*, connected to the *VRidge* app running on the smartphone.[1] This setup allows the user to play VR games on the SteamVR platform for up to 10 minutes, enough to obtain multiple traffic traces that can be analyzed later. The application allows for a number of settings, most notably (i) the display resolution and scan format (kept fixed at the smartphone's native resolution, i.e., 1920×1080p), (ii) the frame rate, allowing the user to choose between 30 FPS and 60 FPS, and (iii) the target data rate (i.e., the data rate the application will try to consistently stream to the client) which can be set from 1 to 50 Mbps.

To simplify the analysis of the traffic stream in this first work, no VR games were played, acquiring only traces from the SteamVR waiting room with no audio. The waiting room is still rendered and streamed in real time, thus allowing the capture of an actual VR stream. Furthermore, in order for SteamVR to fully start and load the waiting room, traces were trimmed down to approximately 550 s each.

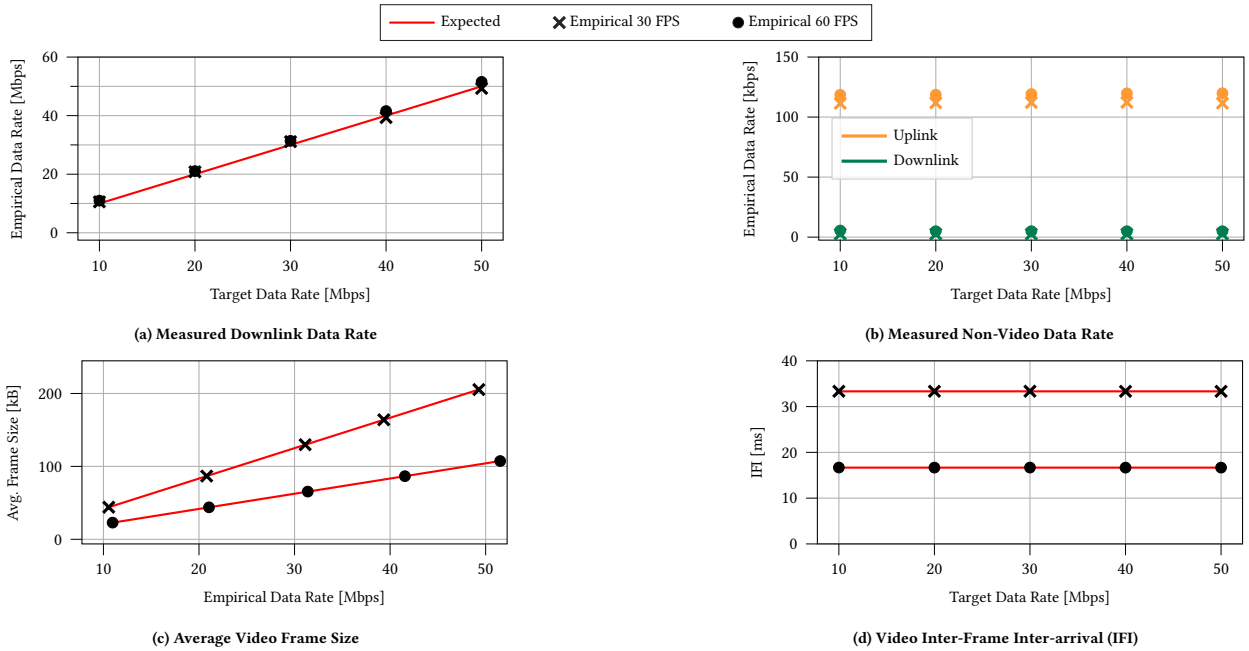
We noticed that the encoder used to stream the video from the rendering server to the phone was able to reduce the stream size when the scene was fairly static, thus preventing us from reaching data rates higher than 20 Mbps with the phone in a fixed position. Small movements from the phone were sufficient to obtain a data rate close to the target one.

Traffic traces were obtained using Wireshark, a popular open-source packet analyzer, running on the rendering server and sniffing the tethered USB connection. The traffic analysis was performed at 30 and 60 FPS for target data rates of {10, 20, 30, 40, 50} Mbps, for a total of over 90 minutes of analyzed VR traffic.

2.2 Traffic Analysis

By analyzing the sniffed packet traces, we discovered that *VRidge* uses UDP sockets over IPv4 and that the UL stream contains several types of packets, such as synchronization, video frame reception information, and frequent small head-tracking information packets. In DL, instead, we found synchronization, acknowledgment,

[1] riftcat.com/vridge



(a) Measured Downlink Data Rate

(b) Measured Non-Video Data Rate

(c) Average Video Frame Size

(d) Video Inter-Frame Inter-arrival (IFI)

Figure 1: Results from Acquired VR Traffic Traces

and video frame packets bursts. We also found out that the application stream is based on ENet, a simple and robust network communication layer on top of UDP.

Video traffic is, as expected, the main source of data transmission (Figure 1a). Video frames are easily categorized by their transmission pattern: a single frame is fragmented into multiple smaller 1278 B packets sent back-to-back. By reverse-engineering the bits composing the UDP payload, it was possible to identify 5 ranges of information in what appears to be a 31 B APP-layer header, specifically (i) the frame sequence number, (ii) the number of fragments composing the frame, (iii) the fragment sequence number, (iv) the total frame size, and (v) a checksum. Thanks to this, we were able to reliably gather information on video frames, allowing for robust data processing.

Figure 1b shows that UL traffic only accounts for 110 kbps for 30 FPS acquisitions and 120 kbps for 60 FPS acquisitions, while non-video DL traffic is only about 2.5 kbps for 30 FPS acquisition and 5 kbps for 60 FPS acquisitions, regardless of the target data rate, and for this reason, they were ignored in the proposed analysis.

It follows that, considering R the target data rate and F the application frame rate, the *average video frame size* is expected to be close to the ideal $S = R/F$, as shown in Figure 1c. Note that the x-axis reports the empirical data rate rather than the target data rate, i.e., the average data rate estimated from the acquired traces, which differ slightly as shown in Figure 1a.

Furthermore, Figure 1d shows that the average Inter-Frame Inter-arrival (IFI) perfectly matches the expected $1/F$.

Usually, when compressing a video, both intra-frame and inter-frame compression techniques are exploited. Specifically, Intra-coded frames (I-frames) use compression techniques similar to those of a simple static picture. Instead, Predictive-coded frames (P-frames) exploit the temporal correlation of successive frames to greatly reduce the compressed frame size. Similarly, Bipredictive-coded frames (B-frames) exploit the knowledge from subsequent frames as well, other than previous frames, to further improve the compression efficiency at the cost of non-real-time transmission. Video compression standards such as H.264 [13] define the pattern of compressed frame types between two consecutive I-frames, which is commonly referred to as Group of Pictures (GoP).

The traffic traces show that GoPs are not deterministic, and tend to be larger at lower target data rates, likely to take advantage of the higher compression generally provided by P-frames. Furthermore, lower target data rates may introduce a small delay to also encode the more efficient B-frames, thus improving the visual quality while decreasing the overall responsiveness. However, the strategy used by the application to map the specified target rate into a certain GoP format is proprietary and undisclosed, so that we could only observe some general trends.

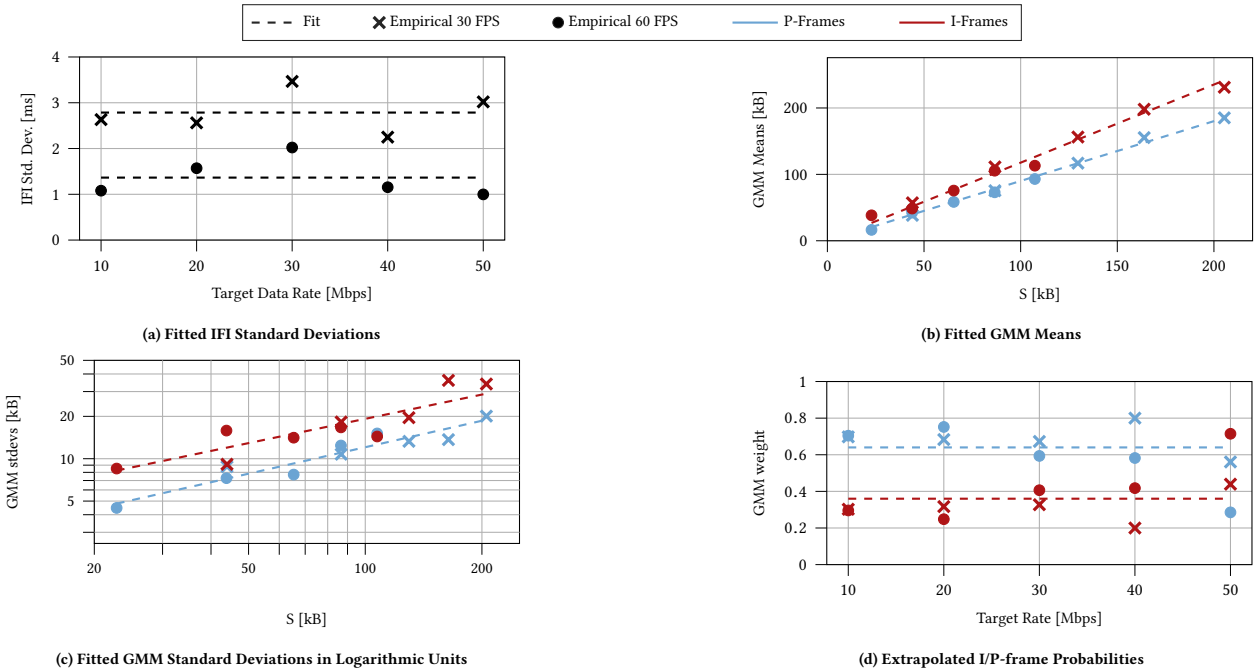
3 TRAFFIC MODEL

In this section, we will describe how we model frame sizes and inter-frame periodicity, leaving the model validation to Section 5.

3.1 Modeling Frame Periodicity

To fully characterize and thus generate a realistic frame period, more information is needed such as (i) the distribution of the frame period, (ii) the parameters of this distribution, (iii) the correlation between successive frame periods, and (iv) the correlation between the current frame size and the frame period.

To simplify the model, in this first analysis we assume the current frame size and the frame period to be independent, and we consider the stochastic process representing the frame period to be



(a) Fitted IFI Standard Deviations

(b) Fitted GMM Means

(c) Fitted GMM Standard Deviations in Logarithmic Units

(d) Extrapolated I/P-frame Probabilities

Figure 2: Fitted Models. For GMMs, the Red Lines Correspond to I-frame Statistics, the Blue Ones Correspond to P-frame Statistics

uncorrelated. We thus focus only on the distribution of the frame period.

Among all measurements taken at both 30 and 60 FPS, the logistic distribution was often the best-fitting one, among all of the tested distributions. For this reason, we choose to model the frame periods as independent and identically Logistic-distributed random variables $X \sim \text{Logistic}(\mu, s)$ with Probability Density Function (PDF)

$$p(x|\mu, s) = \frac{e^{-(x-\mu)/s}}{s(1 + e^{-(x-\mu)/s})^2}, \tag{1}$$

where μ is the *location* parameter, $s > 0$ is the *scale* parameter, and $\mathbb{E}[X] = \mu$, $\text{std}(X) = \frac{s\pi}{\sqrt{3}}$.

Given the great accordance between the expected frame period and the empirical one (shown in Figure 1d), we can easily set $\mathbb{E}[X] = \mu = \frac{1}{F}$. Instead, to model the standard deviation of the proposed Logistic random variables, we need to further process the acquired data, although for a powerful enough rendering server we expect the standard deviation to also be inversely proportional to the frame rate.

Figure 2a shows the average standard deviation of the acquired traces at both 30 and 60 FPS. We notice that the average standard deviation at 30 FPS is 2.7855 ms, while at 60 FPS it is equal to 1.3646 ms, with a perfect ratio of 2.04. This suggests that our rendering server is powerful enough to reliably handle streams at both frame rates since the standard deviation of the IFI nearly doubles as the frame rate halves, as expected.

While the data is not enough for a robust generalization, it does suggest that a higher frame rate F yields lower IFI standard deviation d, keeping a roughly constant ratio with the average IFI, with an inverse law: $d = \frac{c}{F}$. Assuming that this inverse law holds, we compute the average c obtained by the sets of acquisitions at 30 and 60 FPS, equal to $c = 0.0827$.

3.2 Modeling Frame Size

Following the discussion in Section 2.2, we propose to model the frame size distribution of the video frame with a Gaussian Mixture Model (GMM), i.e., $V(S) \sim GMM(\boldsymbol{\mu}(S), \boldsymbol{\sigma}^2(S))$, with PDF

$$V(S) = \chi_I(S)V_I(S) + (1 - \chi_I(S))V_P(S), \tag{2}$$

where $\chi_I(S)$ is the indicator function for I-frames which takes the value of 1 with probability $w_I(S)$ and 0 otherwise, and $V_f(S) \sim \mathcal{N}\left(\mu_f(S), \sigma_f^2(S)\right)$, $f \in \{I, P\}$. Clearly, the fitted normal variable with the lower mean will be associated to P-frames while the one with the higher mean will be associated to I-frames.

To generalize the model, the parameters of the GMM should be extended to arbitrary target data rates.

Starting from the GMMs of the acquired traffic traces, the mean frame sizes of I- and P-frames are generalized by fitting linear models, while their standard deviations are better fitted by a power law, both as a function of the expected average frame size S. Since a target data rate approaching zero would require video frames to also approach zero, we force the linear fit to have no intercept, i.e., $\mu_I(S) = s_I S$, $\mu_P(S) = s_P S$, $\sigma_I(S) = a_I S^{b_I}$, and $\sigma_P(S) = a_P S^{b_P}$ as depicted with dashed lines in Figures 2b and 2c.

By setting $\mathbb{E}[V(S)]$ equal to S we get

$$\begin{cases} \mathbb{E}[V(S)] = w_I(S)\mu_I(S) + w_P(S)\mu_P(S) = S \\ w_I + w_P = 1 \end{cases}, \tag{3}$$

from which $w_I(S) = w_I = \frac{1-s_P}{s_I-s_P}$ and $w_P(S) = w_P = \frac{s_I-1}{s_I-s_P}$, regardless of S. Setting $0 \leq w_I, w_P \leq 1$ results in a requirement for s_I and s_P, specifically, $s_P \leq 1 \leq s_I$.

To make the model more robust, we first fit the GMM 50 times with random initial conditions and pick the best-fitting model, and then weigh the linear fit of the parameters proportionally to the goodness of the GMM fit.

To improve the reliability of our model, when fitting the GMM means and standard deviations we use the empirical data rate as the independent variable instead of the target data rate since the two differ slightly as noticed in Section 2.2. In fact, we aim at modeling the general behavior of the application while trying to generate the amount of traffic requested by the user as closely as possible.

Figure 2b shows that the means of the GMMs are fitted well by a linear model in the S domain, which is used to simultaneously process the data of the acquisitions obtained both at 30 and 60 FPS, yielding the fitter parameters $s_I = 1.1764$ and $s_P = 0.9008$. The fitted slopes result in $w_P = 0.64$, which fits well the 30 FPS acquisitions, with slightly worse performance at 60 FPS.

On the other hand, the GMMs standard deviations show a more noisy fit (Figure 2c), yielding parameters $a_I = 26.2065$, $b_I = 0.5730$, $a_P = 9.0399$, and $b_P = 0.6251$. This suggests an approximate square-root relationship between the average frame size S and the standard deviation of the GMMs. The logarithmic plot helps the visualization by transforming a power law into a simple linear relationship.

Frame sizes are independently drawn from the mixture model instead of simulating a GoP, since different GoPs were found for different target data rates, and always with a non-deterministic nature.

4 NS-3 IMPLEMENTATION

To properly model and test the performance of VR traffic over a simulated network, a flexible application framework has been implemented in ns-3 and made publicly available [10]. The framework is based on the ns-3.33 release and aims at providing a novel additional traffic model, easily customizable by the final user.

The proposed framework allows the user to send packet bursts fragmented into multiple packets by `BurstyApplication`, later re-aggregated at the receiver, if possible, by `BurstSink`. Since the generation of packet bursts is crucial to model a wide range of possibilities, a generic `BurstGenerator` interface has been defined. Users can implement arbitrary generators by extending this interface, and three examples have been provided and will be described in Section 4.2. Finally, each fragment comprises a novel `SeqTsSizeFragHeader`, which includes information on both the fragment and the current burst, allowing `BurstSink` to correctly re-aggregate or discard a burst, yielding information on received fragments, received bursts, and failed bursts.

More details on the implementation and the rationale behind these applications will be given in the following sections.

4.1 Bursty Application

Inspired by the acquired traffic traces described in Section 2.1, `BurstyApplication` periodically sends bursts of data divided into multiple smaller fragments of (at most) a given size. Since burst size and period statistics can be quite general, the generation of the burst statistics is delegated to objects extending the `BurstGenerator` interface, later described in Section 4.2. `BurstyHelper` is also implemented to simplify the generation and installation of `BurstyApplications` with given `BurstGenerators` to network nodes and examples are provided.

Each fragment carries a `SeqTsSizeFragHeader`, an extension of `SeqTsSizeHeader` which adds the information on the fragment sequence number and the total number of fragments composing the burst, on top of the (burst) sequence number and size as well as the transmission time-stamp. After setting a desired `FragmentSize` in bytes, the application will compute how many fragments will be generated to send the full burst to the target receiver, although the last two fragments may be smaller due to the size of the burst not being a multiple of the fragment size, and the presence of the extra header.

Traces notify the user when fragments and bursts are sent, while also keeping track of the number of bursts, fragments, and bytes sent, making it easier to quickly compute some simple high-level metrics directly from the main script of the simulation.

4.2 Burst Generator Interface

A generic bursty application can show extremely different behaviors. For example, an application could send a given amount of data periodically in a deterministic fashion, or the burst size or the period could be random with arbitrary statistics, successive bursts could be correlated (e.g., the concept of GoP for video-coding standards such as H.264 [13]), and even the burst size and the time before the next burst might be correlated.

To accommodate for the widest range of possibilities, a `BurstGenerator` interface has been defined. Classes extending this interface must define two pure virtual functions:

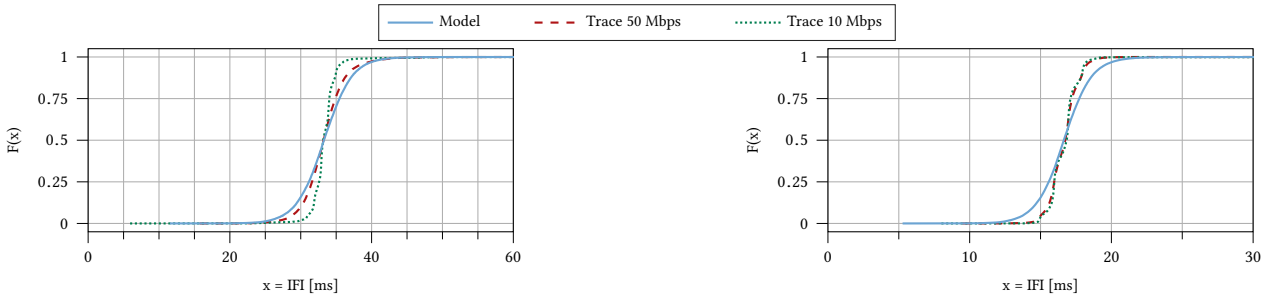
(1) `HasNextBurst`: to ensure that the burst generator is able to generate a new burst size and the time before the next burst (also called *next period* in the remainder of this paper);
(2) `GenerateBurst`: yielding the burst size of the current burst as well as the next period, if it exists.

Three classes extending this interface are proposed and briefly discussed in the remainder of this section, allowing users to generate very diverse statistics without the need to implement their own custom generator in most cases.

Simple Burst Generator. Inspired from `OnOffApplication`, `SimpleBurstGenerator` defines the current burst size and the next period as generic `RandomVariableStreams`. Users are thus able to model arbitrary burst size and next period distributions, by: using the distributions already implemented in ns-3; implementing more distributions; or simply defining arbitrary Cumulative Distribution Functions (CDFs) for `EmpiricalRandomVariables`.

Limitations for this generator lie in the correlation of the generated random variables: burst size and next period are independently drawn as are successive bursts.

VR Burst Generator. `VrBurstGenerator` is a direct implementation of the model proposed in Section 3, where bursts model video frames.



(a) 30 FPS

(b) 60 FPS

Figure 3: Comparison of Empirical CDFs of Inter-Frame Inter-arrivals

Similar to the *RiftCat* software described in Section 2.1, this generator makes it possible to choose a target data rate and a frame rate.

While traces were taken at specific frame rates and target data rates, the proposed model attempts to generalize them, although without any knowledge on the quality of the generalization beyond the boundaries imposed by the streaming software.

To generate the frame size and the next period, `LogisticRandomVariable` and `MixtureRandomVariable` have been implemented in ns-3.

A validation of the proposed model based on this burst generator will be discussed in Section 5.

Trace File Burst Generator. Finally, users might want to reproduce in ns-3 a traffic trace obtained by a real application, generated by a separate traffic generator, or even manually written by a user (e.g., for static debugging/testing purposes). For these reasons, `TraceFileBurstGenerator` was introduced, taking advantage of `CsvReader` to parse a csv-like file declaring a (burst size, next period) pair for each row. Once traces are imported, the generator will sequentially yield every burst, returning `false` as output to `TraceFileBurstGenerator::HasNextBurst` after the last row of the trace file is yielded, thus stopping the `BurstyApplication`.

A `StartTime` can be set as an attribute, allowing the user to control which part of the file trace will be used in the simulation. This can be especially useful when the total simulation duration is shorter than the traffic trace, making it possible to decouple users by setting different start times.

Several VR traffic traces using different frame rates and target data rates are available [10] in the described format for a total of over 90 minutes of processed acquisitions, comprising some relevant metadata as part of the commented header. Interested readers can thus simulate real VR video traffic in their ns-3 simulations, or expand the analysis performed in Sections 2.2 and 3.

4.3 Burst Sink

An adaptation of the existing `PacketSink`, called `BurstSink`, is proposed for the developed bursty framework. This new application expects to receive packets from users equipped with `BurstyApplications` and tries to re-aggregate fragments into packets.

While the current version of `PacketSink` is able to assemble byte streams with `SeqTsSizeHeader`, there are two reasons why `BurstSink` was created, specifically (i) to stress the dependence of this framework on UDP rather than TCP sockets, as the acquisitions suggested, thus expecting individual fragments sent unreliably rather than a reliable byte stream, and (ii) to trace the reception at both the fragment and the burst level.

The application implements a simple best-effort aggregation algorithm, assuming that (i) the burst transmission duration is much shorter than the *next period*, and (ii) all fragments are needed to re-aggregate a burst. Specifically, fragments of a given burst are collected, even if unordered, and, if all fragments are received, the burst is successfully received. If, instead, fragments of subsequent bursts are received before all fragments of the previous one, then the previous burst is discarded. Information on the current fragment and burst can easily be recovered from `SeqTsSizeFragHeader`, allowing the application to verify whether a burst has been fully received or not. If needed and suggested by real-world applications, future works might also introduce the concept of APP-level Forward Error Correction (FEC).

Traces notify the user when fragments are received and when bursts are successfully received or discarded, together with all the related relevant information. Furthermore, similarly to the `BurstyApplication`, also the `BurstSink` application keeps track of the number of bursts, fragments, and bytes received.

5 MODEL VALIDATION AND POSSIBLE USE CASES

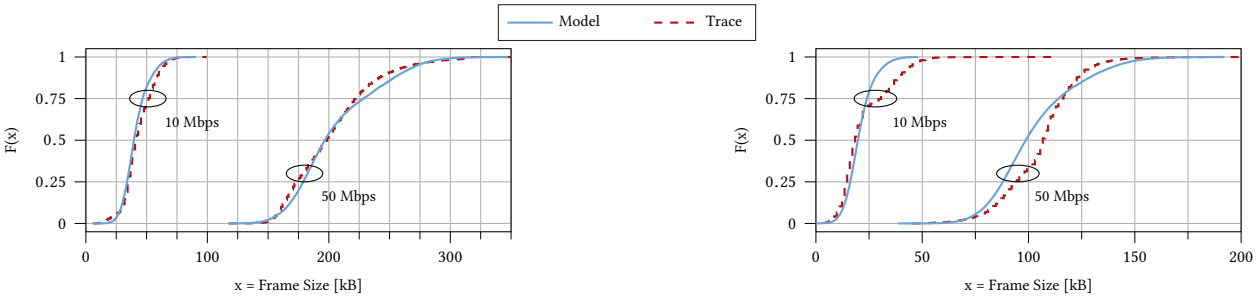
This section will present a comparison between the acquired VR traces and the proposed model, as well as an example use case.

For both the comparison and the example, we show the results of full-stack simulations highlighting the importance of accurately modeling a traffic source by using (i) the proposed model and (ii) the acquired traffic traces. For full-stack simulations we consider a simple Wi-Fi network based on IEEE 802.11ac, sending data over a single stream and using MCS 9 over a 160 MHz channel.

5.1 Model Validation

A comparison between the modeled distributions and the acquired traffic traces is shown in Figures 3 and 4.

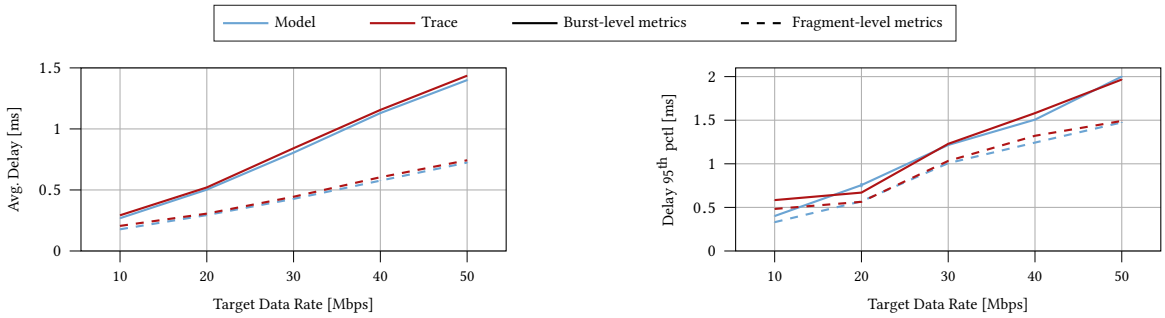
In particular, as expected from Figure 2a, the IFI standard deviation is a loose fit, and thus CDFs shown in Figure 3 show a fairly large deviation between the model and two examples of acquired



(a) 30 FPS

(b) 60 FPS

Figure 4: Comparison of Empirical CDFs of Frame Sizes



(a) Average Delay

(b) Delay 95th Percentile

Figure 5: APP-layer Metrics Plotted Against an Increasing Target Data Rate with 60 FPS Sources

traces, namely the highest and lowest target data rates acquired for both available frame rates. This is to be expected given the large variance of the acquired data, although the objective of our model is to generalize the behavior of a real application for both data rates and frame rates, making it extremely hard to represent well the acquired data.

Figure 4 shows the CDFs for frame sizes at different frame rate and target data rate. The 30 FPS data is overall fitted well by the proposed model, and two examples at different target data rates are shown. On the other hand, the 60 FPS data shows a different behavior with respect to the model. A first explanation is that the 60 FPS traces always have a higher empirical data rate than the target one, so the red dashed lines are generally to the right of the blue lines. Furthermore, doubling the frame rate at a constant data rate halves the average video frame size, making it necessary to use more sophisticated encoding techniques to still be able to obtain a good enough video quality by biasing the relative size and frequency of I- and P-frames. Still, our model is able to capture the overall distributions, making it possible to generate novel traffic data with parameters that were never acquired.

End-to-end results in Figure 5 show good accordance between models and empirical data, except for the 95th percentile of the delay. In fact, the proposed model shown in blue shows good accordance with the traffic traces, suggesting that it is able to emulate sufficiently well the traffic statistics to obtain similar full-stack results.

It is also important to notice the difference between fragment-wise and burst-wise statistics. For applications such as VR, where the whole burst needs to be received to acquire the desired information, it is crucial to measure burst-level performance to get a deeper understanding of the system performance. In general, fragment-level metrics will be more optimistic than those at the burst level, which may lead to incorrect conclusions.

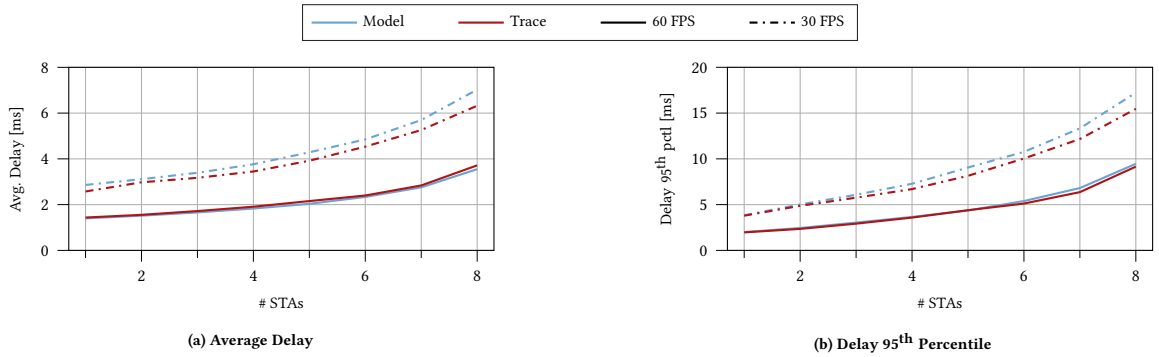
5.2 Examples of Use Cases

To exemplify the uses of the proposed model, we discuss a possible scenario of interest where we test how well an IEEE 802.11ac network can support intense VR traffic, for example in a VR arena.

In Figure 6, we show the simulation results for a scenario with multiple users running VR applications with a target rate of 50 Mbps in a Wi-Fi network. We compare the acquired trace file, where Stations (STAs) import and generate disjoint parts of the trace file, with the proposed model, both at 30 and 60 FPS.

Notice that the fixed target data rate of R =50 Mbps, average video frame size S and frame rate F will have the same ratio resulting in double the frame size for 30 FPS streams with respect to 60 FPS. For a network with fixed channel capacity, this translates to a delay which will also double assuming that processing, queuing, and other delays independent of the burst size are negligible, hence explaining the different delay performance of the two frame rates in Figure 6.

From Figure 6a, it is possible to see that the average burst delay is below the maximum tolerable delay of 5-9 ms specified in [5] for up



(a) Average Delay

(b) Delay 95th Percentile

Figure 6: APP-layer Metrics Plotted Against an Increasing Number of STAs. Each STA Uses a VR Application with a Target Data Rate of 50 Mbps

to 8 users at both 30 and 60 FPS, although the delays of the 30 FPS streams are higher than those of the 60 FPS streams, as expected.

Instead, if a good overall quality of experience should be granted, the same bound for the 95th percentile of the delay would only allow up to 5 users in the system for 30 FPS streams, or 7 users for 60 FPS streams, as shown in Figure 6b, thus greatly reducing the arena capacity for increased reliability and overall user experience.

6 CONCLUSIONS

In this paper, we presented a simple VR traffic model based on over 90 minutes of acquired traffic traces. While being simple, ignoring second-order statistics, and being based on an ideal setting, this model marks a starting point for network analysis and optimization tailored for this novel and peculiar type of traffic, introducing a more realistic traffic model into ns-3.

The proposed ns-3 framework for bursty applications is publicly available and open source [10], together with the implementation of the proposed traffic model and the actual traffic traces experimentally obtained. We also attempted to generalize the model to arbitrary target data rates and frame rates, allowing users to experiment with arbitrary application-level settings that suit their specific research.

The model has been built upon a framework to simulate bursty applications in ns-3, where burst size and period can be customized with little additional code, and traces for burst-level metrics collections allow the user to better analyze a complex application QoS.

Future works will focus on improving the quality and generality of this approach. For example, second-order statistics will be taken into account, trying to better characterize the statistics of GoPs. More acquisitions will be taken, possibly longer, with different streaming and video encoding settings, on several VR applications. Finally, more complex settings will be considered, e.g., adding head movements, in order to analyze possible correlations between them and the generated traffic.

ACKNOWLEDGMENTS

Mattia Lecci's activities were supported by *Fondazione CaRiPaRo* under the grant "Dottorati di Ricerca 2018." This work was partially supported by NIST under Award No. 60NANB19D122.

REFERENCES

[1] Wei Cai, Ryan Shea, Chun-Ying Huang, Kuan-Ta Chen, Jiangchuan Liu, Victor C. M. Leung, and Cheng-Hsin Hsu. 2016. A Survey on Cloud Gaming: Future of Computer Games. *IEEE Access* 4 (2016), 7605–7620.

[2] Mark Claypool, David Finkel, Alexander Grant, and Michael Solano. 2014. On the Performance of OnLive Thin Client Games. *Multimedia Syst.* 20, 5 (Oct. 2014), 471–484.

[3] Eric L. Groen and Jelte E. Bos. 2008. Simulator Sickness Depends on Frequency of the Simulator Motion Mismatch: An Observation. *Presence: Teleoperators and Virtual Environments* 17, 6 (Dec. 2008), 584–593.

[4] Lawrence J. Hettinger and Gary E. Riccio. 1992. Visually Induced Motion Sickness in Virtual Environments. *Presence: Teleoperators and Virtual Environments* 1, 3 (Jan. 1992), 306–310.

[5] Huawei. 2016. *Empowering Consumer-Focused Immersive VR and AR Experiences with Mobile Broadband.* White Paper. Huawei. https://www.huawei.com/en/industry-insights/outlook/mobile-broadband/insights-reports/vr-and-ar

[6] IEEE P802.11. 2016. IEEE Standard for Information Technology – Telecommunications and Information Exchange Between Systems Local and Metropolitan Area Networks – Specific Requirements - Part 11: Wireless LAN Medium Access Control (MAC) and Physical Layer (PHY) Specifications. (Dec. 2016), 3534 pages.

[7] IEEE P802.11 - Task Group ad. 2012. 802.11ad-2012 - IEEE Standard for Information Technology – Telecommunications and Information Exchange Between Systems – Local and Metropolitan Area Networks – Specific Requirements-Part 11: Wireless LAN Medium Access Control (MAC) and Physical Layer (PHY) Specifications Amendment 3: Enhancements for Very High Throughput in the 60 GHz Band. (Dec. 2012). Superseded.

[8] ITU. 2017. Minimum Requirements Related to Technical Performance for IMT-2020 Radio Interface(s). Report ITU-R M.2410-0. (Nov. 2017).

[9] Hak Gu Kim, Wissam J. Baddar, Heoun-taek Lim, Hyunwook Jeong, and Yong Man Ro. 2017. Measurement of Exceptional Motion in VR Video Contents for VR Sickness Assessment Using Deep Convolutional Autoencoder. In *ACM Symposium on Virtual Reality Software and Technology (VRST)*. Gothenburg, Sweden.

[10] Mattia Lecci. 2021. Implementation of the VR Application Model. (2021). https://github.com/signetlabdei/ns-3-vr-app

[11] Sebastian von Mammen, Andreas Knote, and Sarah Edenhofer. 2016. Cyber Sick but Still Having Fun. In *ACM Conference on Virtual Reality Software and Technology (VRST)*. Munich, Germany.

[12] Marc Manzano, Manuel Urueña, Mirko Sužnjević, Eusebi Calle, Jose Alberto Hernández, and Maja Matijasevic. 2014. Dissecting the Protocol and Network Traffic of the OnLive Cloud Gaming Platform. *Multimedia Syst.* 20, 5 (Oct. 2014), 451–470.

[13] Iain E. Richardson. 2010. *The H.264 Advanced Video Compression Standard* (2nd ed.). Wiley Publishing.

[14] Task Group ay. 2015. Status of Project IEEE 802.11ay. (2015). http://www.ieee802.org/11/Reports/tgay_update.htm

[15] Task Group ay. 2017. TGay Usage Model. (Nov. 2017). https://mentor.ieee.org/802.11/dcn/15/11-15-0625-07-00ay-ieee-802-11-tgay-usage-scenarios.pptx doc.: 802.11-15/0625r7.

[16] Zheng Xue, Di Wu, Jian He, Xiaojun Hei, and Yong Liu. 2015. Playing High-End Video Games in the Cloud: A Measurement Study. *IEEE Transactions on Circuits and Systems for Video Technology* 25, 12 (Dec. 2015), 2013–2025.