Using Machine Learning and In-band Network Telemetry for Service Metrics Estimation

Leandro C. de Almeida[1, 2], Rafael Pasquini[3], and Fábio L. Verdi[1]

[1]Computing Department, Federal University of São Carlos, Sorocaba, SP Brazil
[2]Academic Unit of Informatics, Federal Institute of Paraíba, João Pessoa, PB, Brazil
[3]Faculty of Computing, Federal University of Uberlândia, Uberlândia, MG, Brazil
leandro.almeida@ifpb.edu.br, rafael.pasquini@ufu.br, verdi@ufscar.br

***Abstract*—Data plane programmable devices used together with In-band Network Telemetry (INT) enable the collection of data regarding networks' operation at a level of granularity never achieved before. Based on the fact that Machine Learning (ML) has been widely adopted in networking, the scenario investigated in this paper opens up the opportunity to advance the state of the art by applying such vast amount of data to the management of networks and the services offered on top of it. This paper feeds ML algorithms with data piped directly from INT - essentially statistics associated to buffers at network devices' interfaces - with the objective of estimating services' metrics. The service running on our testbed is DASH (Dynamic Adaptive Streaming over HTTP) - the most used protocol for video streaming nowadays - which brings great challenges to our investigations since it is capable of automatically adapting the quality of the videos due to oscillations in networks' conditions. By using well established load patterns from the literature - sinusoid, flashcrowd and a mix of both at the same time - we emulate oscillations in the network, i.e., realistic dynamics at all buffers in the interfaces, which are captured by using INT capabilities. While estimating the quality of video being streamed towards our clients, we observed an NMAE (Normalized Mean Absolute Error) below 10% when Random Forest is used, which is better than current related works.**

***Index Terms*—In-band network telemetry, Machine learning, Service metrics estimation, Data plane programmability.**

I. Introduction

Thanks to recent advances in programmable hardware and the P4 language [1], network devices can report network status without intervention of the control plane [2]. In addition, the INT specification [3] defined a set of new fine-grained metrics that allows to extract network data at line rate.

This huge amount of network metrics allowed by INT, on one hand, brings to the network administrator new views of the network, such as packets' paths, buffers' occupancy and packets' waiting-time in queues. On the other hand, these data are very welcome for ML algorithms that are hungry for data. With this knowledge, ML algorithms can predict the status of services running in the network, giving the opportunity for new management actions.

In this context, this work intends to answer the following research questions: 1) Can INT metadata be used to feed ML algorithms? 2) Is it possible to obtain a better estimation of service metrics using INT and ML? If so, how accurate is this estimation? 3) What is the best ML method to work with INT?

Having these two powerful tools, INT and ML, we want to analyze if both, when used together, may help in service metrics estimation. Specifically in this work, we want to estimate the QoS of DASH service [4], considering that DASH is the current standard technology used by video companies such as Netflix® and Google® [5]. DASH offers a video service with an adaptive rate, in which the client has the possibility to consume the video in different configurations (resolution, bitrate and frames per second) according to the load of the network [6]. In this sense, estimating the metrics of an adaptive service like DASH is challenging given that network load conditions typically fluctuate.

We used the most classical ML algorithms to find the best estimator that fits with INT metadata: Decision Tree (DT), Random Forest (RF), K-nearest neighbors (KNN) and Neural Networks (NN). The network load was created through a variety of different traffic patterns using sinusoid, flashcrowd, and a mix of both at the same time. The purpose of these loads is to provide experience to the model to be trained, as they lead the system from low load to high load. We also created random microbursts so that the load is a mixture of different flows that vary in aspects such as size (long and short), type of applications (web services, management) and duration times. Results indicate that RF obtained the best figures having an NMAE below 10%, which is better than other related works [7]–[9].

Finally, our experiments revealed two important findings: 1) the queue depth is the feature (network metric) that influences most in the ML algorithm, and 2) the switch closest to the client (the host where the DASH client is running) is the one whose network metrics are more important for the ML algorithms. The latter result confirms the same finding of a previous work [7] whose evaluation was done using a combination of metrics from a computational cluster and OpenFlow networks.

In summary, the main contributions of this work are:

1) Propose and evaluate the joint usage of ML and INT for DASH service metrics estimation;
2) Analyse different ML algorithms and show which method obtains better results;
3) Reveal which network metric and which network equipment mostly influence ML algorithms.

This work is organized as follows: in Section II the research problem is presented. The project decisions for the machine learning are described in Section III. In Section IV, the evaluation process is described, including a brief view about DASH video service and the workloads used. The results are presented in Section V and the related works is discussed in Section VI. Finally, the conclusions are in Section VII.

II. Problem statement

In this work, we are initially assuming a datacenter network. In this sense, the estimation can be useful in the context of network service providers (ISPs), which could adjust the infrastructure according to the estimates output by the ML algorithms. Furthermore, the hypothesis is that variations in service metrics (frames per second - FPS) have a positive correlation with INT metadata. The INT metadata used in this work are:

- *Ingress Global Timestamp*: the timestamp, in µs, of when the packet entered in the ingress (pipeline).
- *Egress Global Timestamp*: the timestamp, in µs, of when the packet started processing in the egress (pipeline).
- *Enq Timestamp*: the timestamp, in µs, of when the packet was initially placed in the queue for processing.
- *Enq Qdepth*: the queue depth when the packet was queued, in units of the number of packets.
- *Deq Timedelta*: the time, in µs, that the packet remained in the queue.
- *Deq Qdepth*: the queue depth when the package was dequeued, in units of the number of packets.

In our study, we analyzed how INT metrics (data set *X*) correlate with QoS metric (data set *Y*). In a nutshell, the fine-grained metrics of network *X* are INT metadata and the QoS *Y* metrics are data from the video components (FPS).

As in work [7], we consider a global clock, which can be read by all devices on the network so that all clocks are correctly synchronized. In addition, the evolution of metrics *X* and *Y* is treated as a **time series** $\{X_t\}$, $\{Y_t\}$ and $\{\ (X_t,\ Y_t)\ \}$, where each instant of time *t* represents the state of the network and the video service. The problem of estimating service metrics Y_t, in the time *t*, based on knowledge of fine-grained network telemetry metrics X_t, can be modeled with $M : X_t \rightarrow \hat{Y_t}$, where $\hat{Y_t}$ is a function of approximation of the function Y_t, for a given X_t. This is a regression problem that can be solved with supervised ML [10].

III. Project decisions for the machine learning algorithms

In this work, the ML algorithms are applied for regression problems to estimate service metrics from INT metadata. To evaluate the hypothesis defined in Section I, the following methods were used: Decision Tree (DT), Random Forest (RF), Neural Networks (NN) and K-nearest neighbors (KNN). To minimize overfitting problems in the data, Cross-validation (CV) was used to train and evaluate the models. In addition, a hyper-parameters' optimization was performed to find the parameters in each model that obtained the best results. To assess errors in predictive models, the Normalized Mean Absolute Error (NMAE) was calculated.

In the next sub-sections, we briefly explain the ML algorithms and the main concepts behind the CV, hyper-parameters optimization, and NMAE.

A. Machine learning models

DT is a method based on the divide-and-conquer strategy, seeking to solve a complex problem by decomposing it into smaller subproblems. In this case, the solutions of the subproblems are combined in a tree format, producing the solution of the original problem.

The objective of the DT algorithm is to minimize the residual sum of squares (RSS), described in Equation 1, where $\hat{y}_{R_j} = \sum_{i \in R_j} \frac{Y_i}{|R_j|}$.

$$\sum_{j=1}^{J} \sum_{i \in R_k} (y_i - \hat{y}_{R_j})^2 \tag{1}$$

In this case, the space *X* of *n* features is divided into *J* distinct, non-overlapping regions, R_1, R_2,...,R_j. For each observation that falls in a given region R_j, a prediction is made, which consists of the average of response values for the training observations in R_j. Furthermore, $\hat{y}_{R_j}$ is the average response for the $\mid R_j \mid$ observations of the j-th region.

RF is a model that extends DT using a multi-tree combination strategy. The final estimation is calculated from the average of the resulting estimates for each tree. Each tree is built using a fraction of the input X attributes at random, changing the format of each tree [7].

KNN is a ML model based on Euclidean distance (see Equation 2), which uses the intuition that objects related to the same concept are close to each other.

$$dist(x, y) = \sqrt{\sum_{i=1}^{n} (x_i - y_i)^2} \tag{2}$$

In Equation 2, the distance is calculated from the sum of differences between x and y, where x and y are vectors with the same dimension (n).

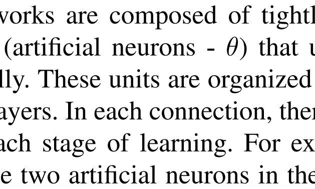


Fig. 1. Neural Network with one hidden layer.

NN is a ML method that mimics the biological structure of neurons. Networks are composed of tightly interconnected processing units (artificial neurons - θ) that use mathematical functions internally. These units are organized into one or more fully connected layers. In each connection, there is a weight (w) that adjusts to each stage of learning. For example, in Figure 1, we can observe two artificial neurons in the input layer: one artificial neuron in the hidden layer, and one artificial neuron in the output layer.

In this example, each neuron in the input layer has a value (x_i); These values are multiplied by the weights (w_i) and added together. The result (output) of the sum serves as input for the next layer of neurons, according to the Equation 3.

$$\sum_{i=1}^{n} x_i \times w_i \geq \theta \tag{3}$$

The algorithm of a NN involves an error correction rule, which uses the optimization of a quadratic error function between the outputs of the neural network and the expected values. In this case, artificial neurons are used to predict a value in regression problems.

B. Cross-validation

Overfitting is the problem in which the regressor have a good result in estimating the objective function $\hat{Y}$, since it fits almost perfectly to the input data X. This problem happens when the regressor has been trained with all the input data, or with a sizeable partition of the data.

To minimize this problem, the cross-validation strategy with k parts (k-fold cross-validation) was used. This strategy allows the division of data into k partitions, which are used for the training and validation of the regressor independently. In this work, the dataset was split into three parts: train, test and validation. The k-fold cross-validation was performed in the train and test parts. At the end, the validation part was used to assess the models.

C. Hyper-parameter optimization

In general, ML methods are sensitive to variations in their parameters. This implies that we could have different results for the same method in the same dataset [11]. In fact, there is a big space where the values of parameters can be chosen. Techniques like Grid Search, Random Grid Search and Genetic Algorithm are used to perform a search of the best parameters that fit with the dataset [11]. In this work, we used a Random Grid Search [12] considering that it is faster as it does not need to check all the possible combinations.

D. Performance evaluation metrics

Generally speaking, in ML (as in many other fields) there is no one-size-fits-all solution. The performance metrics can be used to measure the different aspects of the model, such as reliability, robustness, accuracy, and complexity [13]. For this reason, it is necessary to use performance evaluation metrics to extract conclusions from the results.

In regression problems, the evaluation of a ML algorithm is usually performed by analyzing the regressor generated by it in the estimation of new objects, not previously presented in its training. In this case, the evaluation metrics are based on the model's prediction errors, which represent how far the predicted value is from the actual value.

NMAE was used in this work to calculate the normalized mean of the absolute errors of the predictions made. Equation 4 describes the formula, where m is the size of the test set; y_i matches sample i from the test suite; $\hat{y}_i$ refers to the estimated value for the sample i; and $\bar{y}$ is the average of the responses of the test set samples. X is a self-contained metric, derived from MAE (Mean Absolute Error), that represents the percentage of error in relation to the value to be estimated.

$$NMAE(y, \hat{y}) = \frac{1}{\bar{y}}\left(\frac{1}{m}\sum_{i=0}^{m-1} |y_i - \hat{y}_i|\right) \tag{4}$$

IV. Evaluation

To make our evaluations, a virtualized-based environment was built running on a physical server model Dell EMC PowerEdge R720 with 2 Intel Xeon processors® E5-2630 v2 2.60GHz, 6 cores per socket (24 vCPUs), 48GB RAM, 2TB HDD and Ubuntu 18.04.5 LTS.

Virtualbox (6.1.8) was used as the hypervisor together with Vagrant (2.2.13) and Ansible (2.9.15) for infrastructure provisioning. All the artefacts are available for replication purposes in a public repository[1].

A. Components description

The topology described in Table I and shown in Figure 2 is composed by 10 virtual machines having all their connections provided by BMv2 switches, which are P4-capable virtual equipments.

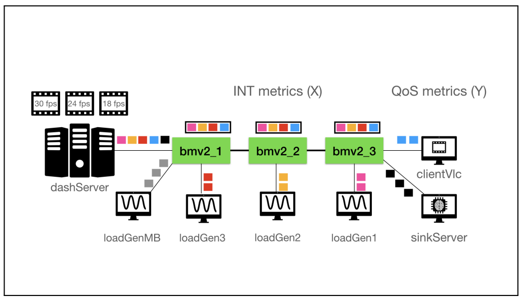


Fig. 2. Topology of experiment.

Name	OS	vCPUs	Memory
dashServer	Ubuntu 16.04.7 LTS	12	4GB
clientVlc	Ubuntu 20.04.1 LTS	12	8GB
sinkServer	Ubuntu 20.04.2 LTS	4	8GB
loadGen1	Ubuntu 20.04.1 LTS	6	8GB
loadGen2	Ubuntu 20.04.1 LTS	6	8GB
loadGen3	Ubuntu 20.04.1 LTS	6	8GB
loadGenMicroBurst	Ubuntu 20.04.1 LTS	1	4GB
bmv2_1	Ubuntu 20.04.1 LTS	4	1GB
bmv2_2	Ubuntu 20.04.1 LTS	4	1GB
bmv2_3	Ubuntu 20.04.1 LTS	4	1GB

TABLE I
VMs details.

The dashServer is the component responsible for providing video streaming in the DASH standard for the client and the load generators. In the evaluation, two video streams were made available: the transmission of a soccer game for the

[1]https://github.com/leandrocalmeida/.

client access; and a playlist containing the ten most accessed videos on Youtube® for the load generators. Apache version 2 applications were installed as the web server; FFmpeg (2.8.17) was used for encoding the videos; and MP4box (0.5.2) for creating the MPEG-DASH manifest files.

The clientVlc is the component responsible for consuming the video streaming of the soccer game, in which the VLC video player (3.0.8) was executed, with modifications to collect service metrics.

The BMv2 switches were programmed to append INT metadata in all INT packets. In this work, we adopt an out-of-band approach, that is, specific INT probes are sent from the DASH server to the sink node. So, no data packets are changed to carry out INT metadata.

The sinkServer is the component responsible for collecting INT traffic and storing it in the format supported by the ML methods. Code written in Python was used to perform the collection and storage functionalities.

The load generators (loadGen{1/2/3}) are components responsible for consuming the streaming of the ten most accessed videos on Youtube®. They run two load patterns: a sinusoidal and flashcrowd.

The sinusoidal function is described in Equation 5, where: A represents an amplitude; F the frequency; and λ is a phase in radians. The loadGen{1/2/3} execute video clients obeying the sinusoid load function, increasing and decreasing over time.

$$f(y) = A\sin(F + \lambda) \tag{5}$$

The flashcrowd load describes a flash event, that is represented by a large spike or surge in traffic to a particular Web site [14]. The flashcrowd is divided into three phases: ramp-up, sustained and ramp-down.

Ramp-up is modeled by shock level (S), that is an order of magnitude increase in the average request (video clients) rate. Furthermore, it starts in t_0 and ends in t_1.

$$rampup = \frac{1}{\log_{10}(1 + S)} \tag{6}$$

Sustained represents the maximum traffic (clients) level at the time interval t_1 and t_2. It is also modeled by S.

$$sustained = \log_{10}(1 + S) \tag{7}$$

Ramp-down represents the end of the flash event, gradually decreasing the amount of traffic (video clients). In this phase, n is a constant that defines the speed of reduction. Ramp-down is modeled by n and S.

$$rampdown = n \times \log_{10}(1 + S) \tag{8}$$

The loadGenMicroBurst is a component that runs a microburst [15] generator used to create noise in the network load. The purpose of having microbursts is to mimic as much as possible the real traffic in a datacenter and make it difficult for the ML methods to characterize the load patterns.

B. Experiment description

The experiment lasted approximately 19 hours (8h for sinusoid, 6h for flashcrowd and 5h for mix load). Mix load means that both sinusoid and flashcrowd were used at the same time in the experiment. The dashServer hosts the video with different configurations (from high quality to low quality), as shown in Table II, so that the client can use each one (transition) depending on the traffic load in the network.

Type	**Resolution**	**FPS**	**GOP**[2]	**Kbps**	**Buffer**	**Codec**
vídeo	426x240	18	72	280	140	h264
vídeo	854x480	24	96	980	490	h264
vídeo	1280x720	30	120	2080	1040	h264
áudio	-	-	-	128	-	AAC
áudio	-	-	-	64	-	AAC

TABLE II
VIDEO PARAMETERS USED IN A DASHSERVER.

Every second, service quality metrics were collected in the clientVlc component. In parallel, packets with INT instructions were sent from the dashServer to the sinkServer in every µs interval. INT metadata is then appended by each switch (bmv2_{1,2,3}) in the path as shown in Figure 3. At each hop, the INT packet size increases **32 bytes** (metadata) from its original size (48 bytes). Considering that the solution presented here is initially applicable in datacenter, where the maximum number of hops is generally not greater than five [16], it is expected that the packets will not be fragmented.

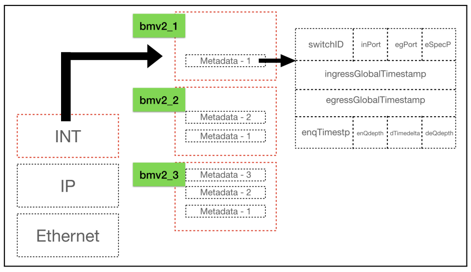


Fig. 3. Add INT metadata.

Upon arriving at the sinkServer, the metadata is extracted and stored in the proper format for the ML methods. In addition, the load generators (loadGen{1,2,3}) run the load patterns, sinusoid, flashcrowd[3] and mix, in independent executions. The parameters used are described in Table III.

Component	**Sinusoid**	**Flashcrowd**	**Mix**
loadGen1	A=4, F=15, λ=5	S=(8-15), n=(1-5)	A=2, F=7, λ=5, S=(3-8), n=(1-2)
loadGen2	A=4, F=15, λ=5	S=(8-15), n=(1-5)	A=2, F=7, λ=5, S=(3-8), n=(1-2)
loadGen3	A=4, F=15, λ=5	S=(8-15), n=(1-5)	A=2, F=7, λ=5, S=(3-8), n=(1-2)

TABLE III
PARAMETERS USED FOR LOAD GENERATORS.

The microburst generator sent 500 bytes-packets in a time interval between 0.01 - 1 second during all experiments.

[2]Group of Pictures.
[3]The value was chosen randomly from a range described in Table III.

After performing the experiments, the data (FPS and INT metrics) were integrated into a matrix M_{mxn}, where: m represents the number of samples (time series); and n represents the number of attributes used for the regression. Before submitting the data to the ML method, a pre-processing step was performed, in which the objective was to improve the quality and representation of the data. Pre-processing was carried out following the steps described below:

Incomplete/missing data handling: Incomplete/missing data (*NaN - Not a Number*) may lead to problems in the execution of the methods [17]. For this reason, through the function `removeNaN`, samples with this values were removed.

Removal of same-valued attributes: single-valued attributes do not have information that helps distinguish objects, so they are considered irrelevant [17]. For this reason, single-valued attributes have been removed with the `removeSameValuedAttr` function.

Attribute normalization: attributes that have very different scales can cause problems in machine learning methods [17]. For this reason, the *z-score* [18] attribute normalization process was performed with the `StandardScaler` function of the Scikit-learn package. In this case, the mean of each attribute was equal to zero and standard deviation equal to one.

After the pre-processing step, the data was divided into three partitions (train/test and evaluation) using the function `train_test_split` from python's Scikit-learn package. In this sense, 80% of data went to training/test and 20% to evaluation. Train/test partition (80% from original) was split again using the `KFold` cross-validation function, with k=5, creating 5 sub-partitions, 4 for train and 1 for test. Each partition was submitted to the ML models of the Scikit-learn package, performing a grid search through the `RandomizedSearchCV()` function, in order to find the regressor with the smallest error. For each method, 75 ML models were analyzed (25 for each load), totalizing 300 models evaluated. The best estimator of each ML method was used to evaluate the data evaluation partition (20% from original) for each load.

V. Results

First of all, we want to verify what is the most important feature for the ML algorithms. Figure 4 shows which features (INT metadata) contributed the most for the learning in all load conditions. In this case, on the x-axis, we have the Gini importance [19] representing the impurity level of the feature, and in the y-axis we have the name of the feature (metadata).

The lower the level of impurity the better classified is the feature. Therefore, we can observe that the metadata related to buffers (highlighted with a box) has the most influence on the learning model. Such a finding is of great importance since it gives some hints that not all the network metrics may need to be collected. It points to a direction where the ML algorithms may have the same accuracy with less data.

Still in Figure 4, it is clear that the node (bmv2_3), which is the closest to the client, has more influence on the learning than all the others. We conjecture that the intermediate nodes represent a good point of view of the state of the network,

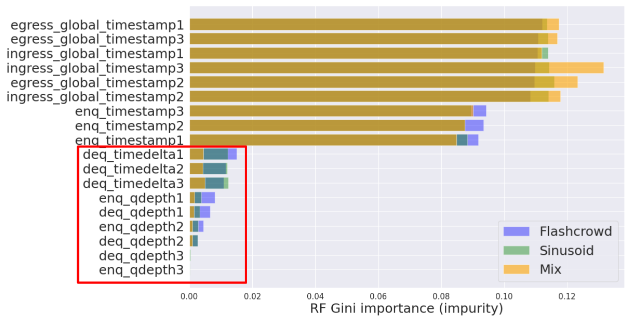


Fig. 4. Random Forest feature importance.

however, it is the closest node to the client that has the most accurate information for predicting video metrics. This result supports a similar finding in related work [7].

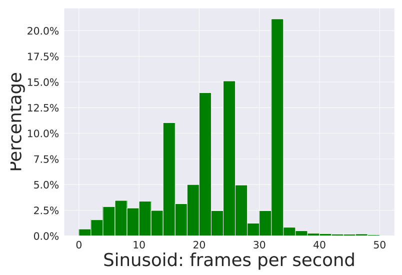
Second, we want to evaluate what is the best ML method in terms of NMAE, discussing the ability of ML models to predict QoS metrics from different load conditions. In this sense, the results in Table IV were obtained with the best estimator after the grid search and hyper-parameters optimization for each load pattern.

Load	DT	RF	KNN	NN
Sinusoid	33.74%	7.23%	25.81%	26.85%
Flashcrowd	30.28%	6.70%	20.54%	19.94%
Mix	25.71%	2.96%	29.82%	21.42%

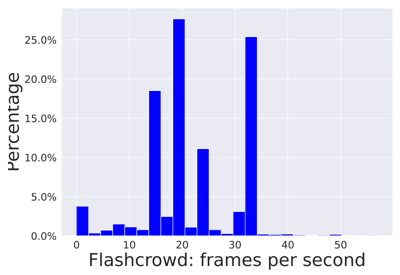
TABLE IV
NMAE FOR LOAD PATTERNS INDIVIDUALLY.

Table IV shows that RF obtained the lowest NMAE at evaluating individual datasets, that is, the model was trained, tested and evaluated in every individual load pattern. A NMAE of **7.23%** was obtained for the sinusoid load, **6.70%** for the flashcrowd load and **2.96%** for the mix load, having a mean NMAE of **5.63%**. Unexpectedly, the RF achieved its best performance in the mix load pattern, which is the worst scenario considering that both loads (sinusoid and flashcrowd) run at the same time. Although this pattern of traffic is challenging for the ML algorithms, it mimics the real traffic load in a datacenter. We believe that this behaviour was because the mix pattern generated a higher load, and made few transitions in the network when compared to the sinusoid and flashcrowd patterns. This can be inferred by observing histograms at Figure 5, in which the x-axis represents the number of the FPS played in the client and the y-axis the percentage of this FPS frequency in the experiment.

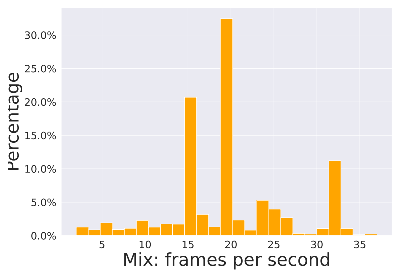
When looking at the sinusoid load (Figure 5(a)), we observe that the client played the video with a high resolution (around 30 FPS) in 21% of the time and made transitions to other minor resolutions (around 18 and 24 FPS). Figure 5(b) (flashcrowd load) shows that the client played the video in the lower resolution (around 18 FPS) in 27% of the time. Transitions to higher resolutions were done at other moments in time. It is important to say that about 4% of the time the client couldn't



(a) Sinusoid



(b) Flashcrowd



(c) Mix

Fig. 5. Load Patterns.

play the video. In the mix load (Figure 5(c)), the client played the video in high resolutions (around 30 FPS) in just 11% of the time. This indicates that the load in the network was higher for a longer time. This high load caused a lower number of transitions in the resolutions, when compared to the other two experiments, which is easier for ML algorithms' estimations, leading to a better NMAE.

Finally, our last evaluation tests the ability of a ML model, trained in one load pattern, to predict QoS metrics from other load patterns, as described below and shown in Table V.

- Sinusoid → Flashcrowd: ML model was trained with the sinusoid load and evaluated with the flashcrowd load.
- Sinusoid → Mix: ML model was trained with the sinusoid load and evaluated with the mix load.
- Flashcrowd → Sinusoid: ML model was trained with the flashcrowd load and evaluated with the sinusoid load.
- Flashcrowd → Mix: ML model was trained with the flashcrowd load and evaluated with the mix load.
- Mix → Sinusoid: ML model was trained with the mix load and evaluated with the sinusoid load.
- Mix → Flashcrowd: ML model was trained with the mix load and evaluated with the flashcrowd load.

Load	DT	RF	KNN	NN
Sinusoid → Flashcrowd	30.89%	33.22%	33.88%	35.69%
Sinusoid → Mix	28.75%	30.98%	41.00%	36.33%
Flashcrowd → Sinusoid	33.72%	36.15%	36.38%	42.69%
Flashcrowd → Mix	30.04%	30.34%	40.64%	43.57%
Mix → Sinusoid	35.11%	38.15%	37.42%	40.26%
Mix → Flashcrowd	31.11%	34.30%	34.03%	37.77%

TABLE V
NMAE FOR CROSS EVALUATIONS.

The general conclusion is that the models perform poorly when evaluating against other load patterns. It is conjectured that this happens because the load patterns explored different regions of the space of possibilities (system states), as seen in the histograms of Figure 5.

VI. Related works

The work presented in [7] used a combination of metrics from a computational cluster (CPU, memory and disk usage) together with metrics from an OpenFlow network (number of bytes/packets transmitted and received). An NMAE error of 9.17% was obtained with a sinusoid load. That work used a static service (Video on Demand) which does not adapt to the network load and then is easier to predict. Also, the number of metrics was higher (48) compared to our work (18).

In [8], the authors propose a method to predict QoE using ICMP probing in MPEG-DASH video service. For the QoE inference, a RMSE (Root Mean Squared Error) of 0.98 was achieved. The main limitation is that the prediction was carried out without any load on the network. Also, the topology used was simplistic, having only 1 node connecting the client to the Dash server. In this way, ICMP queries did not suffer great variations over successive buffers.

The work presented in [9] used TLS transactions to predict QoE in a video service, obtaining an accuracy of 72%. This work limits the scope only to video traffic with TLS, that is, video services that do not use encryption would not be anticipated. In addition, they used 38 features to predict QoE, while this work only used 18.

In general, this work is positioned in the context of predicting the quality of service metrics. The main difference of this work in relation to others is that it uses fine-grained network metrics, thanks to the programmable data plane together with the INT specification and ML algorithms.

VII. Conclusions

This work presents a step forward towards the estimation of DASH video QoS metrics using INT and ML. The results indicate a NMAE error below 10% in the individual datasets when the RF model is used. Results also indicate that the metadata extracted from buffers has more impact on the learning model. Also, the node closest to the client is the biggest contributor to the video metrics estimation. All datasets used to evaluate the ML models, as well as the setup infrastructure, are made available from the IaC (Infrastructure as a Code) perspective for replication and comparison purposes.

Despite the advances presented in this work, further studies still need to be carried out. Important questions that are still open include whether there is a "best" place in the network to collect INT metadata while keeping the same accuracy and the verification of having a reduced number of INT metadata (e.g. buffers-related metrics) is enough to maintain a low NMAE.

References

[1] P. Bosshart, D. Daly, G. Gibb, M. Izzard, N. McKeown, J. Rexford, C. Schlesinger, D. Talayco, A. Vahdat, G. Varghese, and D. Walker, "P4: Programming protocol-independent packet processors," *SIGCOMM Comput. Commun. Rev.*, vol. 44, no. 3, p. 87–95, Jul. 2014. [Online]. Available: https://doi.org/10.1145/2656877.2656890

[2] S. Arslan and N. McKeown, "Switches know the exact amount of congestion," in *Proceedings of the 2019 Workshop on Buffer Sizing*, ser. BS '19. New York, NY, USA: Association for Computing Machinery, 2019. [Online]. Available: https://doi.org/10.1145/3375235.3375245

[3] P4, "In-band network telemetry (int) dataplane specification," P4 Consortium, Tech. Rep., 2021.

[4] ISO, "Dynamic adaptive streaming over http (dash)-part 1: Media presentation description and segment formats," *ISO/IEC*, pp. 23 009–1, 2014.

[5] S. Lederer, "Why youtube & netflix use mpeg-dash in html5," https://bitmovin.com/status-mpeg-dash-today-youtube-netflix-use-html5-beyond/, 2015, accessed: 2020-03-24.

[6] ISO, "Dynamic adaptive streaming over http (dash)-part 1: Media presentation description and segment formats," *ISO/IEC*, pp. 23 009–1, 2014.

[7] R. Stadler, R. Pasquini, and V. Fodor, "Learning from network device statistics," *J. Netw. Syst. Manag.*, vol. 25, no. 4, pp. 672–698, 2017. [Online]. Available: https://doi.org/10.1007/s10922-017-9426-z

[8] G. Miranda, D. F. Macedo, and J. M. Marquez-Barja, "A qoe inference method for dash video using icmp probing," in *2020 16th International Conference on Network and Service Management (CNSM)*, 2020, pp. 1–5.

[9] T. Mangla, E. Halepovic, E. Zegura, and M. Ammar, "Drop the packets: Using coarse-grained data to detect video performance issues," in *Proceedings of the 16th International Conference on Emerging Networking EXperiments and Technologies*, ser. CoNEXT '20. New York, NY, USA: Association for Computing Machinery, 2020, p. 71–77. [Online]. Available: https://doi.org/10.1145/3386367.3431294

[10] G. James, D. Witten, T. Hastie, and R. Tibshirani, *An Introduction to Statistical Learning*, ser. Springer Texts in Statistics. Springer, New York, NY, 2013.

[11] P. Liashchynskyi and P. Liashchynskyi, "Grid search, random search, genetic algorithm: A big comparison for nas," 2019.

[12] J. Bergstra and Y. Bengio, "Random search for hyper-parameter optimization," *Journal of Machine Learning Research*, vol. 13, no. 10, pp. 281–305, 2012. [Online]. Available: http://jmlr.org/papers/v13/bergstra12a.html

[13] R. Boutaba, M. Salahuddin, N. Limam, S. Ayoubi, N. Shahriar, F. Estrada-Solano, and O. Caicedo Rendon, "A comprehensive survey on machine learning for networking: Evolution, applications and research opportunities," *Journal of Internet Services and Applications*, vol. 9, 05 2018.

[14] I. Ari, B. Hong, E. Miller, S. Brandt, and D. Long, "Managing flash crowds on the internet," in *MASCOTS 2003*, 11 2003, pp. 246– 249.

[15] R. Joshi, T. Qu, M. C. Chan, B. Leong, and B. T. Loo, "Burstradar: Practical real-time microburst monitoring for datacenter networks," in *Proceedings of the 9th Asia-Pacific Workshop on Systems*, ser. APSys '18. New York, NY, USA: Association for Computing Machinery, 2018. [Online]. Available: https://doi.org/10.1145/3265723.3265731

[16] Y. Li, R. Miao, H. H. Liu, Y. Zhuang, F. Feng, L. Tang, Z. Cao, M. Zhang, F. Kelly, M. Alizadeh, and M. Yu, "Hpcc: High precision congestion control," in *Proceedings of the ACM Special Interest Group on Data Communication*, ser. SIGCOMM '19. New York, NY, USA: Association for Computing Machinery, 2019, p. 44–58. [Online]. Available: https://doi.org/10.1145/3341302.3342085

[17] K. Faceli, A. C. Lorena, J. Gama, and A. C. P. d. L. F. d. Carvalho, *Artificial Intelligence: A Machine Learning Approach*. LTC, 2011.

[18] S. Kotsiantis, D. Kanellopoulos, and P. Pintelas, "Data preprocessing for supervised learning," *International Journal of Computer Science*, vol. 1, pp. 111–117, 01 2006.

[19] S. Nembrini, I. R. König, and M. N. Wright, "The revival of the Gini importance?" *Bioinformatics*, vol. 34, no. 21, pp. 3711–3718, 05 2018. [Online]. Available: https://doi.org/10.1093/bioinformatics/bty373