POLITECNICO DI TORINO
Repository ISTITUZIONALE

Where Did My Packet Go? Real-Time Prediction of Losses in Networks

Original
Where Did My Packet Go? Real-Time Prediction of Losses in Networks / Song, T., Markudova, D., Perna, G., Meo, M.. - ELETTRONICO. - (2023), pp. 3836-3841. (ICC 2023 - IEEE International Conference on Communications Rome, Italy 28 May 2023 - 01 June 2023) [10.1109/ICC45041.2023.10278583].

Availability:
This version is available at: 11583/2983314 since: 2023-10-25T12:52:55Z

Publisher:
IEEE

Published
DOI:10.1109/ICC45041.2023.10278583

Terms of use:

This article is made available under terms and conditions as specified in the corresponding bibliographic description in the repository

Publisher copyright
IEEE postprint/Author's Accepted Manuscript

©2023 IEEE. Personal use of this material is permitted. Permission from IEEE must be obtained for all other uses, in any current or future media, including reprinting/republishing this material for advertising or promotional purposes, creating new collecting works, for resale or lists, or reuse of any copyrighted component of this work in other works.

(Article begins on next page)

20 November 2026

Where did my packet go? Real-time prediction of losses in networks

Tailai Song, Dena Markudova, Gianluca Perna, Michela Meo
Politecnico di Torino
first.last@polito.it

***Abstract*—Real-time communication (RTC) platforms have undergone a consistent increase in popularity in recent years, and nowadays, they are fundamental for both work and leisure purposes. To ensure adequate Quality of Experience (QoE) for users of RTC services, we need proper traffic management policies, that, when critical network conditions are detected, react by operating either at the network configuration level or on the application to improve QoE. However, predicting critical network conditions, especially packet losses that are particularly harmful to QoE, is a very challenging task. In this paper, we propose a system for predicting packet losses that might occur in the near future (i.e., in a second) for RTP streaming traffic. We analyze several ML algorithms, from standard techniques to deep neural networks and anomaly detection algorithms, and we apply them to more than 66 hours of data from two popular RTC applications. The selection of the algorithm and its tuning turn out to be fundamental to achieving good performance. In one of the best settings, which are based on a Balanced Random Forest classifier, we obtain a recall of 0.82.**

I. Introduction

In recent years, there has been a consistent increase in the use of RTC applications, such as video-conferencing and cloud gaming, for both work and leisure purposes. The importance of RTC applications was especially evident during the COVID-19 pandemic in 2020, when the lockdown measures adopted to curb the outbreak forced millions of people to rely on these platforms all at once, leading to a global increase of RTC traffic by more than 200% [1]. With remote working becoming the norm for a lot of companies, the reliability of RTC applications has become critical. Supported by the ever-growing network infrastructure all over the world and higher available bandwidth, RTC services are evolving rapidly, with countless competing applications on the market today [2]. Each application adopts a different technical solution, but most of them rely on the Real-time Protocol (RTP) [3] over UDP. In web browsers, the applications use the open-source WebRTC framework over RTP.[1]

In this context, it is becoming increasingly important to maximize the Quality of Experience (QoE) of users of RTC applications. Improvements can either come from the application layer or the network layer. On the network side, QoE depends on many factors, such as the quality of the participants' connections, the network topology, and the network management. One of the main threats to good QoE is packet loss [4], [5].

In this paper, we propose a novel application based on Machine Learning (ML) for predicting losses in RTC traffic in real time. Predicting network conditions on in-network devices may significantly help network management. We pose the problem as time-series forecasting to be solved by supervised learning techniques. We base our prediction on carefully chosen features from network traffic statistics in the past few seconds, such as inter-arrival times and packet sizes. Our algorithm then predicts whether there will be packet loss in the next time bin. We build our study on 66 hours of network traffic from real calls captured at the client side using two RTC applications - *Cisco Webex* and *Jitsi Meet*[2]. To select the best ML algorithm for our purposes, we evaluate a large set of approaches, from standard techniques to deep neural networks and anomaly detection algorithms, measuring the classification recall as a performance metric. We further build our system considering constraints present in real networks - we are careful to train and test with data from different RTP streams and predict further in the future so we have time to implement network policies based on our predictions. We also analyze the occurrence of losses, considering the severity of loss events. The selection of the algorithm and its tuning turn out to be quite critical for good performance. In one of the best settings, using a Balanced Random Forest classifier, we obtain a recall of 0.82.

The proposed application is envisioned to work as a software module in network devices (e.g., middleboxes, routers), as part of a comprehensive ML-based, RTC-aware traffic management system. The system enables application-level visibility at the network control plane and incorporates a feedback mechanism to inform the control plane of worsening network conditions. It would contain several classifiers, trained offline, that predict various network characteristics [6], [7], one of which is the expected packet loss in the near future. Then, a controller (e.g., an SDN orchestrator) can apply select network management techniques to alleviate the worsening conditions, like allocate more bandwidth to flows or send them on different paths.

To make our research reproducible, we publish the dataset, the underlying code, and the trained models mentioned in the paper[3].

This work has been supported by the SmartData@PoliTO center on Big Data and Data Science

[1] https://webrtc.org/

[2] https://www.webex.com/, https://meet.jit.si/

[3] https://smartdata.polito.it/real-time-prediction-of-packet-loss/

TABLE I: Summary of quantities for all packet flows

Description	Quantity[-]	Percentage[%]
Total packets	59393645	-
Total lost packets	216735	0.36
Total time bins	2137727	-
Time bins with loss	32472	1.52
Sparse loss	7374	0.34 (22.4)
Concentrated loss	25098	1.18 (77.6)

II. Related Work

Numerous works related to RTC applications address the topic of analysis and predictions of packet losses. Some works employ traditional mathematical models based on various network measurements. For example, the authors of [8] propose a model to predict packet loss and congestion for audio streams based on measurements of end-to-end delay variation. The authors of [9] focus on video loss prediction in wireless networks with the goal of video codec improvement. They use simulated data to devise a mathematical model, with parameters defined by linear regression. Other papers propose the use of machine learning for different predictions related to losses. The authors of [10] propose a lightweight ML tool to predict the number of retransmitted packets in bulky TCP flows. Unlike our real-time system, they propose an algorithm that works per-flow and predicts retransmissions a-posteriori. Authors of [11] develop hidden Markov chain-based models to predict the loss rate in the short-term future, by first estimating the distribution of the number of losses and then taking the expected value as a prediction of the loss rate. In contrast to our approach, they only use loss distributions to predict future losses, while we use various other network features, making the prediction more robust.

Whole systems to cope with packet losses also exist in the literature. Authors of [12] propose early packet loss feedback (EPLF), where the MAC layer of the local WiFi link sends a spoofed NACK to the RTP layer. In [13], instead, the authors propose an adaptive hybrid NACK/FEC method with temporal layers to handle packet loss in WebRTC.

Most works predict packet loss per-flow, while we are at a much finer granularity of per-500 ms time bin, which allows us to use our system in real-time. Numerous works use simulated data, while we build our study on real traffic from widely adopted RTC applications. Moreover, our system only requires the statistics of received packets at the receiver end, which makes it implementable in network equipment, not only on the application side. In addition, we also consider a wide range of ML algorithms of different types.

III. Problem statement

Our goal is to predict the presence of packet losses in the near future using statistical features of past packets. More specifically, we divide the traffic into 500 ms time bins and use the past 10 time bins, which correspond to 5 seconds of traffic, to predict if there is a loss in a future time bin. Formally, we can represent the loss as a function of traffic features in the past w time bins:

$$Y_{t+m\Delta t} = f(\overline{x}_{t-\Delta t}, \overline{x}_{t-2\Delta t}, .., \overline{x}_{t-w\Delta t})$$
$$\text{with } \overline{x} = (x_1, .., x_N) \text{ and } Y = \begin{cases} 1, \text{loss} \\ 0, \text{no loss} \end{cases}, \quad (1)$$

where t is time, Δt is the time bin size (500 ms), m is the number of time bins we consider as the "gap" between the time bins used as input and the time bin for which we want to predict, and $Y_{t+m\Delta t}$ is a binary variable that represents the presence of packet loss in the future time bin (at time $t+m\Delta t$). For the model development, we use m=0, but in the final results, we use m=1, predicting a loss 1 s later by skipping one time bin. $\overline{x}$ represents the input traffic statistics or features of the ML model (such as packet sizes and inter-arrival times), and N is the number of different types of statistics (features) considered. w is the window size - how many time bins we consider from the past. We fix w to 10. Then, $\overline{x}_{t-n\Delta t}$ is the vector of all types of traffic statistics calculated in the n^{th} time bin in the past. The supervised learning models learn the function $f(\cdot)$ that maps our input features $\overline{x} \in X$ to the binary classification target variable Y. An example of one feature vector (e.g., x_1 - standard deviation of inter-arrival time), with $w = 10$, $\Delta t = 500ms$, and $m = 0$, in function of different Y (0 - right or 1 - left) is shown in Figure 2.

IV. Dataset and loss characterization

In this section, we describe the dataset we used, from *.pcap* format to traffic statistics, showing the properties of the traffic before a loss occurs.

Data source. We collect data from 70 real multi-party calls, with audio and video, using two RTC applications: *Cisco Webex* and *Jitsi Meet*. The calls are made with 2–6 participants, who are either connected to WiFi or a cellular network. In total, we have 66.5 hours of traffic. We capture traffic at the client side, since this is where most losses in the network occur, and use only incoming streams for the prediction task. All the traffic is captured in *pcap* format.

RTP stream identification. We identify the RTP traffic with straightforward deep packet inspection by matching the protocol headers. We define an RTP flow as a tuple composed of $(IP_{src}, IP_{dst}, port_{src}, port_{dst}, SSRC, PT)$, where $SSRC$ is the Synchronization Source Identifier (a field in the RTP header that specifies an RTP stream) and PT is the Payload Type, indicating the video or audio codec that is being used.

Traffic aggregation and loss calculation. For each RTP stream, we aggregate the packets into time bins of duration Δt and calculate traffic statistics for each time bin. To do this, we use the open-source tool *Retina* [14]. *Retina* takes *pcap* files and the time bin size as inputs and outputs *.csv* files with 96 traffic statistics, such as the mean inter-arrival time, the standard deviation of packet size etc. We process all the *pcap* files with *Retina*. For each time bin, we also compute the loss by using the sequence number field from the RTP header. The sequence number is a 16-bit field, incremented by 1 for each

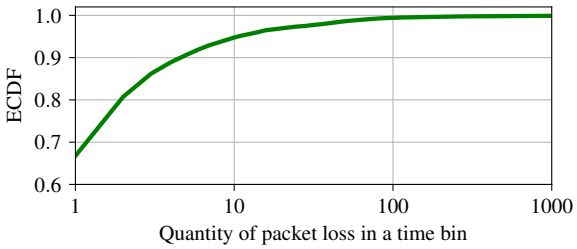


Fig. 1: ECDF of number of lost packets in *loss* time bins

new packet. Therefore, the number of losses can be computed as: $n_{\text{loss}} = \text{seq}_t - \text{seq}_{t-1} - 1$ in which n_{loss} is the number of packets lost between adjacent packets, seq_t is the sequence number of the packet at the current timestamp, and seq_{t-1} is the sequence number of the packet at the previous timestamp. We take care of edge cases like out-of-order delivery, packet duplicates, reaching the maximum RTP sequence number, or having a series of packet losses in two consecutive time bins.

A. Amount of packet loss

Table I describes the quantity of losses in our dataset. In general, out of 60 million observed packets, only 0.36% were lost. When we aggregate to time bins, out of 2 million time bins in total, 1.52% are time bins with at least one lost packet. In the following, we call these time bins *loss* time bins. Such a small percentage of *loss* time bins makes this a very imbalanced dataset, which poses one of the greatest challenges for prediction. Moreover, 77.6% of *loss* time bins are concentrated, meaning there are *loss* time bins in their 5-second proximity. As to the traffic from the different RTC applications, we have more Jitsi traffic (45.7 hours) than Webex traffic (20.81 hours). However, they both exhibit similar ratios of *no loss* and *loss* time bins, leaving no bias in the dataset.

To analyze the amount of losses we find in the *loss* time bins, we refer to Figure 1. Figure 1 illustrates the empirical CDF (ECDF) of different quantities of packet loss for all *loss* time bins. Around 67% *loss* time bins only have 1 lost packet, and around 13% have 2 lost packets; in 95% of the cases, there are less than 10 lost packets in a time bin. To put it in perspective, a time bin of audio consists of 25 packets, while an average time bin of Standard Definition video holds around 100 packets. Having 67% of loss time bins with only one lost packet is another great challenge for the predictor.

B. Feature construction

In this section, we study the traffic statistics observed before a loss time bin to find out if they are correlated with losses, and we use this analysis to find meaningful features for our ML algorithms. To this end, we look at w=10 time bins in the 5 seconds preceding a loss time bin and compare them with normal conditions in which there are no losses. Figure 2 shows an example of the behavior of the standard deviation of packet inter-arrival time before a loss occurs (left) and in normal conditions (right). We observe an increasing trend as we approach a loss and stable behavior when there is no loss.

Finally, we chose 15 types of statistics with viable trends to identify *loss* time bins. We then check the correlation between any pair of features using the Pearson correlation coefficient: if the Pearson correlation coefficient is greater than 0.9, we remove one of the pair of features at random. This way, we decrease the number of features N to 11. Seven features among the 11 selected features are related to packet inter-arrival times; this is expected since the effect of network congestion that may lead to losses is first seen in the delays between adjacent packets. Three features are related to the packet size and one to the RTP timestamp. Note that, since we use traffic statistics from the past w=10 time bins as features, we end up with 110 features (inputs to the model).

C. Challenges

Our problem has numerous inherent and substantial properties that introduce challenges for the prediction of packet loss in a time bin. First, we observe different traffic trends before different loss time bins. Second, time bins with just one packet lost are very challenging to predict because the trend of statistics in the time window before such time bins is similar to the no-loss scenario. Third, the dataset is highly imbalanced due to the nature of losses. Moreover, the variation in traffic statistics in the time window before a loss time bin can be due to some other network conditions and not necessarily lead to a loss. Therefore, when evaluating the model performance, we need to take into account these difficulties.

V. Methodology

In this section, we present the different combinations of techniques we used to classify loss time bins. To solve the problem, we exploit six different ML approaches for classification and anomaly detection. For all models, we follow the same pipeline: train the model, fine-tune the hyperparameters, and evaluate performance. To validate the model, we use 70% of data for training and 30% for testing. To tackle the imbalanced dataset problem, we consider several sampling techniques and compare their performance: undersampling (reducing the quantity of the majority class), oversampling (increasing the quantity of the minority class) through the Synthetic Minority Oversampling Technique (SMOTE) [15] and a combination of both.

Since the dataset is highly imbalanced, accuracy and precision are highly biased by the number of samples from the majority class. The F1-score indicates the overall performance, but it is affected by precision. Consequently, we chose **recall** as our performance metric and optimization goal. Recall is the ratio between the true positive samples and all positive samples. Since it contains only the elements of one class, the results are not unbalanced, i.e., very close to 0 for the *loss* class or to 1 for the *no loss* class. Note that our classifier is conceived as a part of a network management system, which should react if it identifies a few *loss* time bins in a row. This means that if we misclassify a *no loss* time bin, the system would react to improve the conditions for a flow when it is not necessary, while if we misclassify a *loss* time bin, the system would not react when a reaction is necessary and keep the network status

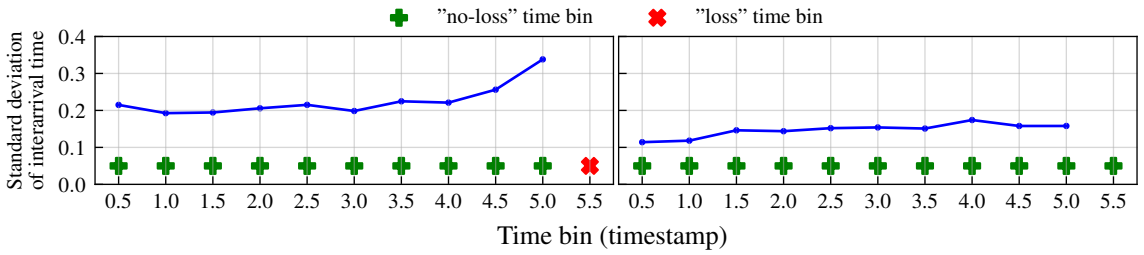


Fig. 2: The behaviour of standard deviation of packet inter-arrival time, before a loss (left) or when there is no loss (right)

TABLE II: Summary of the results of all machine learning models

Category	Model	Sampling strategy	Recall (class 0)	Recall (class 1)	Macro average F1 score
Classification	Decision tree	Original	0.98	0.14	0.55
	Decision tree	Undersampling	0.68	0.67	0.41
	Decision tree	Oversampling	0.94	0.29	0.53
	Decision tree	Combination	0.84	0.48	0.47
	Deep neural network		**0.82**	**0.69**	0.47
	Long short-term memory neural network		**0.83**	**0.69**	0.47
	Random forest without class weight		0.96	0.47	0.60
	Balanced random forest with class weight	Original	0.85	0.64	0.51
	Balanced random forest without class weight		**0.81**	**0.80**	0.50
	XGBoost GBDT targeting overall performance		0.97	0.49	0.62
	***XGBoost* GBDT targeting average recall**		**0.85**	**0.74**	0.52
Anomaly detection	Isolation forest		0.99	0.01	0.50
	Autoencoder		0.96	0.05	0.50

quo. It is important to note that both of these decisions would not lead to worsening network conditions, since the system is not aggressive. However, it means that we are more interested in the performance of the *loss* class.

A. Machine learning models

We now present several key technical aspects for each of the chosen ML techniques.

The first family of ML algorithms we consider is tree-based algorithms, since they have been proven to work well with traffic statistics data [6], [7]. We try Decision Tree (DT), Random Forest (RF), Balanced Random Forest (BRF), and XGBoost (XGB). The idea of RF and XGB is to combine weak learners to create a stronger classifier. These algorithms implement intrinsic mechanisms to cope with the class imbalance [16]. We also implement two types of neural networks: the Deep Neural Network (DNN) with three hidden layers and the Long Short-term Memory (LSTM) neural network that specifically works for time-series data. Packet losses can also be considered anomalies with respect to normal traffic. We refer to two popular anomaly detection methods, Isolation Forest (IF) and AutoEncoder (AE). Both of them are unsupervised models, and in order to enable them to work in a supervised manner, we need to set thresholds to detect anomalies. For AE, we set the error reconstruction threshold to 0.13. For IF, we set the contamination parameter to 0.0139, which is the share of loss time bins in the training data.

VI. Experimental Results

In this section, we present the results of our experiments. We develop the models under basic assumptions, and then we introduce some optimizations to adapt them to a more realistic network scenario.

A. Results of model development

Table II summarizes the classification results of all algorithms and sampling strategies we consider. We denote the *no loss* class as Class 0 and the *loss* class as Class 1. To rank the models, we devise the following criteria: First, the recall for the *no loss* class has to be at least 0.80, meaning at most 20% of samples can be misclassified. Almost all models pass this criteria. We then choose the model with the highest recall for the *loss* class.

Generally, well-performing models are the two neural networks: DNN and LSTM, which exhibit recalls for the *no loss* class of 0.82 and 0.83, respectively, and a recall of 0.69 for the *loss* class. XGBoost (targeting macro-averaged recall) exhibits even better performance, with a recall of 0.85 for the *no loss* class and 0.74 for the *loss* class. The best-performing model according to our criteria is the Balanced Random Forest without class weight, with a recall of 0.81 for the *no loss* class and 0.80 for the *loss* class. It also shows the most balanced performance between the two classes. These four well-performing models are marked with bold in Table II. We notice that simple Decision Trees do not result in good performance, even when combined with the different sampling strategies. Anomaly detection models perform well for the *no loss* class but fail in classifying the *loss* class. Their performance is dependent on the thresholds mentioned in Section V. However, lowering these thresholds results in more misclassification for the *no loss* class and no significant improvement for the *loss* class.

In the following sections, when introducing the optimization techniques for a more realistic scenario, we build on two of the models: the Balanced RF classifier without class weight and the XGBoost that targets macro-averaged recall. XGBoost does not only exhibit good performance, it is also highly configurable.

B. Constraints in reality

Time constraint. To develop the model, we consider predictions in the next time bin. However, in a real scenario, our system would need time to perform a series of actions: compute the traffic statistics, run the model, predict the losses, and react accordingly. In particular, for a machine with an Intel i7-1165G7 CPU and 16 GB of RAM, in Python, the time consumption is 85.9 ms for feature construction and 5.55 ms (XGBoost) or 38.7 ms (BRF) for prediction. Therefore, we move the prediction target further into the future, setting the value of two parameters in (1). Namely, we set m to 1 and w to 9. This means that we leave a gap of Δt =500 ms between collecting the traffic and the prediction time bin, a sufficient amount of time to compute the statistics, run the model, and send the information to the network control plane. Even for a device with weaker computational capability, the total time consumption would be less than 500 ms and network devices operate on much faster programming languages. As a result of moving the prediction target further into the future, the models exhibit a slight performance drop of 1-2%.

RTP flow constraint. In the model development stage, data points are shuffled randomly to obtain the training and test sets. However, a more realistic approach would be to use data from different RTP flows in the training and test sets. This is because in a real network, one would always be predicting a new flow using a model pre-trained on other flows. To this end, we recreate the training and test sets, not allowing data from one flow to spill into both sets. Due to the higher variability of the results, depending on which flows fall in the training set and which in the test set, we get very different results. Thus, we perform 50 trials using different random shuffles of the flows. For each trial, we run Balanced RF and XGBoost and record the recall of both classes, as well as the macro-averaged recall. The results are shown in Figure 4(a). In this case, we observe huge performance drops. For Balanced RF the mean recall is 0.66 for the *no loss* class and 0.42 for the *loss* class. XGBoost gives a more balanced result, with a mean recall of 0.58 for both classes. On top of that, the predictions are much more variable for the *loss* class, since they are very dependent on the specific flows chosen in the test set.

C. Model optimization

To overcome the performance drop experienced when predicting further into the future and using different RTP flows in the training and test sets, we introduce three types of optimization: feature augmentation, feature selection, and a redefinition of loss time bins based on the quantity of lost packets.

Feature augmentation and selection. The first model optimization technique we employ is feature augmentation. Namely, a new binary feature is added to $\overline{x}$ from (1), that notes the presence of a loss in the last 10 time bins (5 seconds). This brings to a total of 120 features. The Recursive Feature Elimination (RFE) technique is then applied. RFE continuously examines the importance of features and recursively removes those that are less informative. Basically, we combine domain knowledge with standard feature selection techniques to finalize the features.

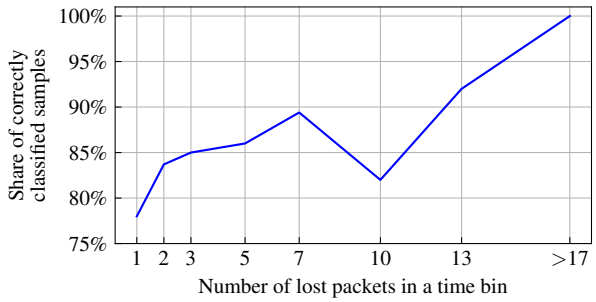
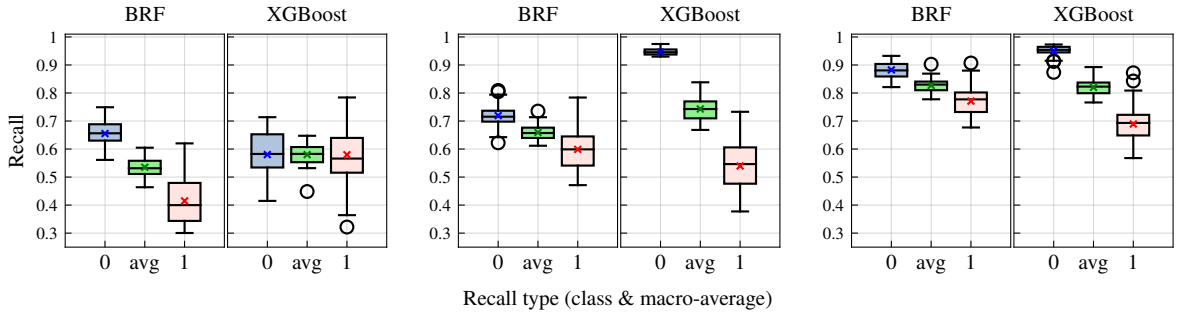


Fig. 3: Share of correctly classified time bins with different numbers of lost packets

Redefining loss time bins. Since we are interested in detecting real network impairments, such as severe congestion, which would justify actuating operations on the network, it is more relevant to predict time bins with multiple lost packets than time bins in which an occasional loss occurs. Consequently, worse conditions lead to more distinct traffic patterns in our features. We thus set a threshold on the minimum number of lost packets in a time bin for it to be considered a *loss* time bin. To set the threshold, we analyze the prediction errors of the Balanced RF model trained during the model development phase. Figure 3 shows the share of correctly classified samples with respect to the number of lost packets in a time bin observed in our original test set. 78% of time bins with only one lost packet were correctly classified, while this number was 92% for time bins with 13 lost packets inside. We see that, as the number of lost packets inside a time bin grows, so does the percentage of correct classification. The curve caps at 17 packets, meaning that any time bin that contains more than 17 lost packets is always correctly classified. Note that the drop at 10 packets is due to the low number of data points for that particular case. Given the curve, we set the threshold to 3, meaning that if a time bin contains 3 or more lost packets, it is considered a loss time bin. For the following experiments, all the time bins that contain less than three lost packets are discarded, and they are also not used as *no loss* because they may impede the training.

Final results. Figure 4 shows the results when using the optimization techniques described above. Predictions refer to 1 s into the future, instead of the next time bin. The box plots refer to the distribution of recall for 50 random shuffles of the training and test sets. The split is still 70% for training and 30% for testing. Figure 4(a) shows the results with no optimization. As discussed in Section VI-B, in this configuration, the recall for both classes is unsatisfactory and the *loss* class exhibits high variance among different shuffles of the training and test sets. Figure 4(b), instead shows the performance after feature augmentation and selection. An overall increase in the recall for both algorithms can be observed. Balanced RF, as expected, gives a more balanced result among the *no loss* and *loss* classes,



(a) No optimization (b) Feature augmentation & selection (c) Feature augmentation & selection and redefinition of *loss* time bins

Fig. 4: Recall distributions under different model optimization

with a mean of 0.72 for the *no loss* class and 0.6 for the *loss* class. XGBoost shows very good performance for the *no loss* class, with a mean recall of 0.95, while it slightly worsens the *loss* class (mean recall 0.53).

Finally, Figure 4(c) shows the performance after implementing all the optimizations, feature augmentation, feature selection, and loss time bin redefinition. Here, a time bin is considered a *loss* time bin only if it contains 3 or more lost packets. Note that, by removing all time bins with fewer than three lost packets, the dataset shrinks a bit. Observe the dramatically improved performance, both for Balanced RF and XGBoost, especially for the *loss* class. Again, Balanced RF gives similar mean recalls for both classes: 0.89 for the *no loss* class and 0.78 for the *loss* class. XGBoost, instead, has plateaued for the *no loss* class with 0.95 mean recall and again exhibits a slightly worse recall for the *loss* class (0.69). With all three optimizations in place, we also observe a much lower variance in the different shuffles for both classes.

VII. Conclusion and Future Work

In this paper, we propose a ML-based system to predict lossy time segments in communication networks. Given the network statistics of the previous 5 seconds, we can predict whether there will be lost packets in the next second. We base our study on traffic data from two popular RTC applications carried through WiFi and mobile networks. As a methodology, we use time-series forecasting with ML classification algorithms. We find that predicting losses is a challenging task, but it can be solved by carefully choosing network traffic features and setting a threshold for the number of lost packets to distinguish occasional losses from losses due to undesired network conditions. With one of the most effective solutions, the Balanced Random Forest, we obtain a mean recall of 0.82.

As future work, we plan to strengthen our system by including data from additional lossy channels, e.g., other mobile scenarios. We will work on the length of the time bin to offer the service at different granularities or even on a per-packet basis. This work is only a first step towards a network management system designed to gather fine-grained information from RTC traffic at the network layer and enable policies to improve QoE.

References

[1] A. Feldmann, O. Gasser, F. Lichtblau, E. Pujol, I. Poese, C. Dietzel, D. Wagner, M. Wichtlhuber, J. Tapiador, N. Vallina-Rodriguez, O. Hohlfeld, and G. Smaragdakis, "The Lockdown Effect: Implications of the COVID-19 Pandemic on Internet Traffic," in *Proceedings of the ACM Internet Measurement Conference*, IMC '20, (New York, NY, USA), p. 1–18, Association for Computing Machinery, 2020.

[2] A. Nistico, D. Markudova, M. Trevisan, M. Meo, and G. Carofiglio, "A comparative study of rtc applications," in *2020 IEEE International Symposium on Multimedia (ISM)*, pp. 1–8, IEEE, 2020.

[3] R. Frederick, S. L. Casner, V. Jacobson, and H. Schulzrinne, "RTP: A Transport Protocol for Real-Time Applications." RFC 1889, Jan. 1996.

[4] D. Vucic and L. Skorin-Kapov, "The impact of packet loss and google congestion control on qoe for webrtc-based mobile multiparty audiovisual telemeetings," in *International Conference on Multimedia Modeling*, pp. 459–470, Springer, 2019.

[5] G. Carofiglio, G. Grassi, E. Loparco, L. Muscariello, M. Papalini, and J. Samain, "Characterizing the relationship between application qoe and network qos for real-time services," in *Proceedings of the ACM SIGCOMM 2021 Workshop on Network-Application Integration*, pp. 20–25, 2021.

[6] G. Perna, D. Markudova, M. Trevisan, P. Garza, M. Meo, M. M. Munafò, and G. Carofiglio, "Real-time classification of real-time communications," *IEEE Transactions on Network and Service Management*, 2022.

[7] D. Markudova, M. Trevisan, P. Garza, M. Meo, M. M. Munafo, and G. Carofiglio, "What's my App?: ML-based classification of RTC applications," *ACM SIGMETRICS Performance Evaluation Review*, vol. 48, no. 4, pp. 41–44, 2021.

[8] L. Roychoudhuri and E. S. Al-Shaer, "Real-time analysis of delay variation for packet loss prediction," 2004.

[9] J. V. C. CarmonaID, E. M. C. de Matos, B. S. L. Castro, F. J. B. Barros, M. C. de Alcaˆntara Neto, and E. G. Pelaes, "Video loss prediction model in wireless networks," 2019.

[10] A. Giannakou, D. Dwivedi, and S. Peisert, "A machine learning approach for packet loss prediction in science flows," *Future Generation Computer Systems*, vol. 102, pp. 190–197, 2020.

[11] F. S. Filho and E. de Souza e Silva, "A method for predicting packet losses with applications to continuous media streaming," 2006.

[12] L. Ma, W. Chen, D. Veer, G. Sternberg, W. Liu, and Y. Reznik, "Early packet loss feedback for webrtc-based mobile video telephony over wi-fi," 2015.

[13] S. Holmer, M. Shemer, and M. Paniconi, "Handling packet loss in webrtc," 2013.

[14] G. Perna, D. Markudova, M. Trevisan, P. Garza, M. Meo, and M. M. Munafò, "Retina: An open-source tool for flexible analysis of rtc traffic," *Computer Networks*, vol. 202, p. 108637, 2022.

[15] N. V. Chawla, K. W. Bowyer, L. O.Hall, and W. P. Kegelmeyer, "Smote: synthetic minority over-sampling technique," 2002.

[16] C. Chen, A. Liaw, and L. Breiman, "Using random forest to learn imbalanced data," 2004.