Towards Adaptive ML Traffic Processing Systems

Johann Hugon
Univ Lyon, EnsL, UCBL, CNRS, LIP
Lyon, France

Gaetan Nodet
Univ Lyon, EnsL, UCBL, CNRS, LIP
Lyon, France

Anthony Busson
Univ Lyon, EnsL, UCBL, CNRS, LIP
Lyon, France

Francesco Bronzino
Univ Lyon, EnsL, UCBL, CNRS, LIP
Lyon, France

ABSTRACT

Machine learning techniques are a common solution used to solve a variety of network management tasks. Often, a network administrator chooses the model to deploy based on offline information, such as model performance and system load. Yet, network traffic is inherently dynamic making it hard to select an optimal model that can work throughout ever changing conditions. In this paper, we make the case that, instead of having to select the optimal candidate model based on offline information, systems should adapt based on the network conditions observed. We present a system design that takes as input a set of candidate models and their features, and adaptively selects the better configuration as a function of the network and system conditions.

CCS CONCEPTS

• **Networks** → **Network measurement**; **Network monitoring**.

KEYWORDS

Traffic analysis; Dynamic configuration; Machine Learning

ACM Reference Format:
Johann Hugon, Gaetan Nodet, Anthony Busson, and Francesco Bronzino. 2023. Towards Adaptive ML Traffic Processing Systems. In *Proceedings of the CoNEXT Student Workshop 2023 (CoNEXT-SW '23), December 8, 2023, Paris, France*. ACM, New York, NY, USA, 2 pages. https://doi.org/10.1145/3630202.3630227

1 INTRODUCTION

Nowadays Machine Learning (ML) techniques have become a common solution used to solve a variety of network management tasks. ML models simplify the identification of complex relationships between network traffic and important events occurring at different network layers. Thus, the networking community has developed ML models to help with traffic analysis tasks like QoE inference, traffic classification, intrusion detection, and more [2].

ML inference tasks are commonly identified by a three steps pipeline: the first step involves ingesting raw network traffic, where traffic undergoes multiple operations, including header parsing, flow tracking and reassembly, and more; the second step performs feature extraction, computing statistics and encoding information to prepare for the model; the third and final step inputs the processed features to the ML model (*e.g.*, random forest, neural network) for the final inference. Conventionally, the performance of these pipelines is evaluated at training time based on the quality of the input features and the trained model type. Unfortunately, different factors can impact the *actual* performance of the model. For example, the inability of a measurement system to cope with the amount of incoming traffic can terminate in packet losses that dramatically impacts the quality of the produced features and the final model performance.

To meet different tradeoffs across model performance and system constraints, different models are available for each task. For example, in the case of video quality inference, Bronzino *et al.* [3] presented a study of the impact that different feature sets have on the accuracy of video quality inference models as well as on the costs that collecting such features has on the measurement system. However, in most cases, models trained offline in a lab setting provide little to no representation of the system's ability to capture traffic nor the impact that packet loss can have on model performance. For these reasons, the users of these models have to decide a priori on which model might best fit their deployment, configuring their measurement system accordingly [3, 4]. Yet, network traffic is inherently dynamic and varies throughout the day due to continuous shifts in usage, making it hard, if not impossible, to select an optimal configuration that can work throughout these changes. Further, substituting such configurations requires to first observe that the system is not capable of processing the traffic, and then manually change the configuration causing further loss due to reboot times.

In this paper, we argue that the best advantage point to understand a system's ability to process traffic and generate the features for a target model is *the system itself*. Instead selecting the optimal candidate model based on offline information, systems should adapt based on up-to-date information of the traffic observed as well as on the system's ability to extract the requested features for a specific traffic load. Towards this, we present the high level design of a system that takes as input a set of candidate ML models, and the features they require as input, and adaptively selects the better fitting configuration as a function of the network and the system conditions.

2 USE CASE: VIDEO QUALITY INFERENCE

To demonstrate the impact that the offline choice of a model can have, we focus on the task of video quality inference from encrypted traffic. We partially recreate the experiment from Bronzino *et al.* [3] and train two candidate machine learning models to infer the resolution of video streaming applications over time. The two models use two different feature sets: the first model employs network-layer features, *e.g.*, packet counters and throughput, while the second model uses application-layer metrics gathered from observing packet patterns generated by video applications, *i.e.*, video segments sizes. We evaluate the performance of the two models in perfect conditions, *i.e.*,

CoNEXT-SW '23, December 8, 2023, Paris, France

ACM ISBN 979-8-4007-0452-9/23/12…$15.00
https://doi.org/10.1145/3630202.3630227

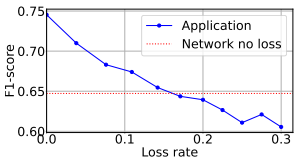

Figure 1: F1-score over Loss rate

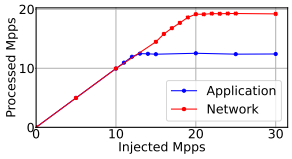

Figure 2: Processed over injected packet rate

with no packet loss, as well as the impact that packet loss can have on the best performing model. From Figure 1 we observe that at no loss the model that uses application layer features does indeed outperform the model that uses network layer features by more than 15%. However, when we introduce packet loss, the performance deteriorates rapidly, and the network-layer based model starts to outperform the other model at around 15% loss.

We then implement the two models in a real system, Retina [4], and evaluate the ability of the system to extract the required features at varying traffic loads. To emulate realistic traffic, we deploy the system on a testbed with two servers connected by a 100gbps programmable switch. On the second server we deploy TRex [1] injecting an IMIX "SFR" profile, a combination of various traffic templates that we enhance with video traffic, at different rates. In Figure 2 we show the amount of packets per second the system is capable of processing with respect to the injected rate. We observe that, when processing application-layer features, the system starts dropping packets at around 12 Mpps, a 39% lower processing capacity compared to the network-layer features.

These two experiments show that, as traffic loads increase, it might be beneficial to lower-cost configurations and models as a preferred solution to maximize performance. More broadly, they support the argument that it is challenging to train a single model that works in every condition, suggesting the need for an adaptive system, as described in the next section.

3 DYNAMIC CONFIGURATION

In this section, we propose the high level design of a system that adaptively choses the best ML model to use at any point in time according to the trade-off between model accuracy, system capabilities, and network load. Starting from a pool of candidate models, and their underlying features to collect, the system comprises three key building blocks, as illustrated in Figure 3.

Model Selection. The model selection module has the goal of tracking the system performance and select the best candidate model given the observed environment. The module consists of two parts which create the capabilities for the system to self adapt this configuration without intervention or reboot. The profiler analyzes the performance of the system itself (CPU and memory usage, number of instructions required per feature, etc.). The monitor continuously aggregates information from the profiler as well as statistics from the Packet Capture module (*e.g.*, packets received, packets dropped, etc.). Using this information, the monitor decides whether it is necessary to update the features collected by the system in the likelihood of reaching a bottleneck.

Packet Capture. To support high network rates and fast packet processing, the system relies on state-of-the-art packet capture libraries process. Further, the system enables the implementation of

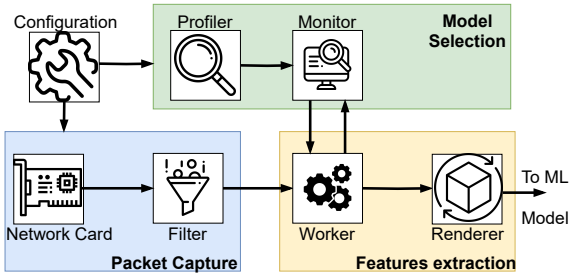

Figure 3: Scheme of the system

packet filters to drop early any unnecessary flow and focus on traffic of interest.

Feature Creation. The Feature Creation module is the final step of the packet processing pipeline and it is in charge of transforming raw packets in features to be used by ML models. The module is composed of a set of workers that compute the features selected by the monitor as fast as possible. After processing, the Renderer aggregates the collected features and compute the final representations to be used by the ML models (*e.g.*, averages, min/max, etc.).

In summary, the system consists of three fundamental building blocks, each strategically designed to optimize the tradeoff between model accuracy, system capabilities, and network context. Together, these modules enable our system to seamlessly adapt remaining attuned to the inherent dynamicity of network traffic.

4 CONCLUSION

In this paper, we have proposed the design of a system that adapts in real time to varying network conditions to select the best features and models for ML-based network management tasks. Such a system can be used to simplify the deployment of ML models in in concrete applications for network administrators and operators to more easily rely on machine learning solutions to drive their networks. Our first implementation shows encouraging early results. We are currently developing a prototype that implements the whole system from the feature extraction components, the ML models, to the monitor that selects features and ML models in real time.

This early work presents several interesting challenges to overcome and future work directions to explore. One key challenge involves expanding the array of feature sets to satisfy the requirements of various machine learning models. Furthermore, exploring more complex profiling metrics beyond the scope of packet loss alone promises to be an interesting avenue of research. With the inclusion of these metrics, our system will be more proficient in extracting the most suitable feature set for the current network context, thus ensuring the choice of the best model.

REFERENCES

[1] 2023. TRex, Realistic Traffic Generator. https://trex-tgn.cisco.com.
[2] Raouf Boutaba et al. 2018. A comprehensive survey on machine learning for networking: evolution, applications and research opportunities. *Journal of Internet Services and Applications* 9, 1 (2018), 1–99.
[3] Francesco Bronzino et al. 2022. Traffic refinery: Cost-aware data representation for machine learning on network traffic. In *ACM SIGMETRICS 2022.*
[4] Gerry Wan et al. 2022. Retina: Analyzing 100GbE Traffic on Commodity Hardware. In *ACM SIGCOMM 2022.*