2ND WORKSHOP

"MACHINE LEARNING & NETWORKING" (MaLeNe) PROCEEDINGS

SEPTEMBER 4, 2023

CO-LOCATED WITH
THE 5TH INTERNATIONAL CONFERENCE ON NETWORKED SYSTEMS (NETSYS 2023)
POTSDAM, GERMANY

2nd Workshop on "Machine Learning & Networking"

The 2nd Workshop on "Machine Learning & Networking" was held in September 2023 at the Hasso-Plattner-Institut as part of the NetSys conference in Potsdam. This workshop also represented the 5th edition of the KuVS Fachgespräch "Machine Learning & Networking (MaLeNe)".

The workshop was organized by co-chairs Michael Seufert (University of Augsburg, Germany), Andreas Blenk (Siemens AG, Germany), and Olaf Landsiedel (Kiel University, Germany). The workshop attracted 16 full paper submissions. As it is a tradition at a Fachgespräch, MaLeNe aims to foster presentation of early results and discussions of these. Thus, all papers at MaLeNe are reviewed briefly and papers of sufficient quality and fit to the call for papers are accepted for presentation. In the case of the 2023 edition, all submissions were a good fit and of good quality and were accepted. From these 16 submissions and two invited keynotes, we assembled a full-day program. One paper was withdrawn on short notice due to illness of the presenter.

On the day of the workshop, the co-chairs welcomed 57 registered participants. The workshop started with a keynote by Nicola Michailow (Siemens AG, Germany) on “Embodied Agent-to-Agent Communications”. This talk was followed by a session on Machine Learning for Networking. The afternoon session began with a second keynote by Andreas Maeder from Nokia titled “Towards an AI-Native Air Interface in 6G” and was followed by a session on Machine Learning for Wireless Networking and TCP. After a coffee break, a third session on Machine Learning for Security and TSN concluded the workshop.

The workshop co-chairs closed the day with a short recap and thanked all speakers and participants, who joined in the fruitful discussions. Finally, we, the organizers, would like to thank Siemens AG, for supporting this instance of the MaLeNe Workshop series by sponsoring the Best Presentation Award**.** It was awarded to the following presentation: “In-Network Round-Trip Time Estimation for TCP Flows” by Daniel Stolpmann from Hamburg University of Technology.

We would like to thank all the authors and attendees for their contributions to the success of the workshop!

Michael Seufert, Andreas Blenk, and Olaf Landsiedel
MaLeNe 2023 Workshop Co-Chairs

Program

Welcome and Workshop Opening

Keynote 1
Embodied Agent-to-Agent Communications, Nicola Michailow (Siemens)

Session 1: Machine Learning for Networking
1. *Towards Benchmarking Power-Performance Characteristics of Federated Learning Clients*
   Pratik Agrawal, Philipp Wiesner and Odej Kao (Technische Universität Berlin)
2. *Demystifying User-based Active Learning for Network Monitoring Tasks*
   Katharina Dietz, Nikolas Wehner (University of Würzburg), Pedro Casas (AIT Austrian Institute of Technology, Vienna), Tobias Hoßfeld and Michael Seufert (University of Würzburg)
3. *Data Distribution Effects on Asynchronous Parameter Server Training in High Delay Differences Networks*
   Leonard Paeleke and Holger Karl (Hasso Plattner Institute)
4. *Replicable Machine Learning Workflow for Energy Forecasting*
   Stepan Gagin (University of Passau), Adrian Carrasco-Revilla (Inetum), Hermann de Meer (University of Passau) and Nuria Sánchez Almodóvar (Inetum)

Keynote 2
Towards an AI-Native Air Interface in 6G, Andreas Maeder (Nokia)

Session 2: Machine Learning for Wireless Networking and TCP
5. *Exploring AI-Based Adaptive Resource Management in 5G Networks*
   Ole Hendrik Sellhorn and Horst Hellbrück (University of Applied Sciences, Lübeck)
6. *Parameter Prioritization for Efficient Transmission of Neural Networks in Small Satellite Applications*
   Olga Kondrateva (Humboldt-Universität zu Berlin), Stefan Dietzel (Merantix Momentum GmbH), Ansgar Lößer and Björn Scheuermann (Technical University of Darmstadt)
7. *Bandwidth Prediction for Volatile Networks with Informer*
   Birkan Denizer and Olaf Landsiedel (Kiel University)
8. *Steps Toward a Supervised Machine Learning Scheduler for MPTCP*
   Reza Poorzare, Hadi Asghari and Oliver P. Waldhorst (Karlsruhe University of Applied Science)
9. *In-Network Round-Trip Time Estimation for TCP Flows*
   Daniel Stolpmann and Andreas Timm-Giel (Hamburg University of Technology)

Session 3: Machine Learning for Security and TSN

10. *Reducing Memory Footprints in Purity Estimations of Volumetric DDoS Traffic Aggregates*
    Hauke Heseding (Karlsruhe Institute of Technology), Timon Krack (KASTEL Security Research Labs) and Martina Zitterbart (Karlsruhe Institute of Technology)
11. *Impact of Adaptive Packet Sampling on ML-based DDoS Detection*
    Samuel Kopmann (KASTEL Security Research Labs) and Martina Zitterbart (Karlsruhe Institute of Technology)
12. *Dynamic Network Intrusion Detection System in Software-Defined Networking*
    Pegah Golchin, Jannis Weil, Ralf Kundel and Ralf Steinmetz (Technical University of Darmstadt)
13. *PicNIC: Image-based Diagnosis for Industrial Blackbox Networks*
    Marco Reisacher, Andreas Blenk and Hans-Peter Huth (Siemens AG)
14. *Towards Synthesizing Datasets for IEEE 802.1 Time-sensitive Networking*
    Doğanalp Ergenç, Nurefşan Sertbaş Bülbül (University of Hamburg),Lisa Maile, Anna Arestova (University of Erlangen-Nürnberg) and Mathias Fischer (University of Hamburg)
15. *Adapting to the Flow: Reinforcement Learning for Dynamic Priority Assignment in TSN*
    Nurefşan Sertbaş Bülbül and Mathias Fischer (University of Hamburg, Germany)

Closing Remarks and Best Presentation Award (sponsored by Siemens AG)

Towards Benchmarking Power-Performance Characteristics of Federated Learning Clients

Pratik Agrawal, Philipp Wiesner, and Odej Kao
Technische Universität Berlin
{pratikkumar.vijaykumar.agrawal, wiesner, odej.kao}@tu-berlin.de

***Abstract*—Federated Learning (FL) is a decentralized machine learning approach where local models are trained on distributed clients, allowing privacy-preserving collaboration by sharing model updates instead of raw data. However, the added communication overhead and increased training time caused by heterogenous data distributions results in higher energy consumption and carbon emissions for achieving similar model performance than traditional machine learning. At the same time, efficient usage of available energy is an important requirement for battery-constrained devices. Because of this, many different approaches on energy-efficient and carbon-efficient FL scheduling and client selection have been published in recent years.**

However, most of this research oversimplifies power-performance characteristics of clients by assuming that they always require the same amount of energy per processed sample throughout training. This overlooks real-world effects arising from operating devices under different power modes or the side effects of running other workloads in parallel. In this work, we take a first look on the impact of such factors and discuss how better power-performance estimates can improve energy-efficient and carbon-efficient FL scheduling.

***Index Terms*—Federated Learning, Energy Efficiency, Carbon Awareness, Battery-Powered Devices, Edge AI, IoT**

I. Introduction

While FL [1] mitigates privacy and data transfer overhead issues associated with centralized ML training, it has different set of challenges. Studies [2] have shown that FL consumes more energy and emits significantly more carbon to reach the same model performance as centrally trained ML models.

The usage of Internet of Things (IoT) and edge devices to train machine learning models in distributed and federated learning settings regularly have further worsened the energy and environment implications of AI/ML training. Moreover, regulators responsible for AI policy also explicitly put emphasis on sustainability and environmental aspects of AI[1]. Additionally, edge devices are battery powered and operate under strict energy budgets. To manage efficient usage of this limited battery power for both the non-FL baseloads and the FL training, it is necessary to profile power-performance characteristics of FL training under realistic scenarios.

To improve efficiency, researchers have proposed energy-efficient [3]–[5] and carbon-efficient [6], [7] approaches for FL scheduling and client selection. FL has also seen vast amount of work in benchmarking [8] in recent years. However, these benchmarks are often missing the critical energy consumption metrics of FL and are predominantly explored in server based simulated environments. Moreover, the benchmarks that include energy consumption metrics are missing granular level energy consumption data under the real-world impact factors such as non-FL baseloads and power modes of the devices. These approaches [6] rely on analytical energy models that are based on metrics such as CPU compute cycles or Floating Point Operations Per Second (FLOPS) to calculate energy required per batch. These models do not accurately take into account factors such as power modes and baseloads of the underlying FL device.

II. Problem Statement

Current FL studies over simplify the power consumption characteristics of FL clients. In comparison to simulation settings employed in FL studies, realistic on-device FL trainings exhibit complicated operational behavior patterns when it comes to energy consumption. For example, edge devices with and without hardware accelerators (GPUs or TPUs) have different energy requirements for per sample throughout. Furthermore, edge devices are usually executing workloads (i.e. baseloads) as such as inference or time-series data streaming which affects the energy per sample performance. These complex operational behavior patterns warrant a need for better energy management and energy-efficient FL. Client selection and scheduling are complex problems in FL, given the heterogeneous nature (hardware resources such as battery, compute, memory) of FL clients. Therefore, to enable better energy-efficient FL and carbon-efficient FL scheduling, metrics such as energy per sample, samples per second are necessary. Current FL studies assume the energy availability budgets and do not consider energy per sample performance under the influence of power modes or baseloads of the FL clients [3]–[7].

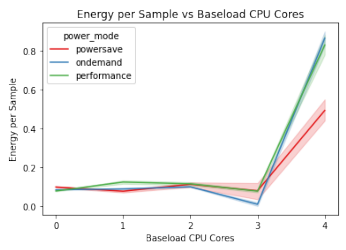
III. Preliminary Insights

To gain some preliminary insights into the variation of power-performance of FL clients, we evaluate an FL training under different baseloads and power modes of a Raspberry Pi.

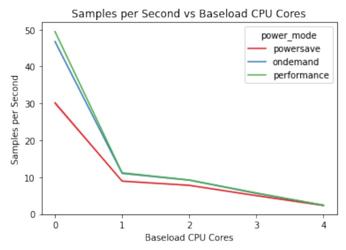
For Raspbeery Pi, we evaluated three power modes: Performance, Powersave and Ondemand. To simulate a baseload on an FL device, we utilized unix command *sysbench*[2]. It provides flexibility to simulate baseload in-terms of number

[1]https://digital-strategy.ec.europa.eu/en/library/ethics-guidelines-trustworthy-ai

[2]https://wiki.gentoo.org/wiki/Sysbench



(a) Energy Per Sample



(b) Samples Per Second

Fig. 1: Power-Performance Characteristics for RPI

of threads/CPU cores. For the FL training we utilized Flower framework [3]. We simulate a computer vision IoT training task using dataset CIFAR-10 [4] and the computer vision model SqueezeNet which is light-weight and deemed to be suitable for edge computer visions applications. We assign higher kernel priority to baseload process to ensure the FL training doesn't affect the CPU time of baseload in co-running scenario. For the energy consumption measurement, we utilized WLAN power socket switch[5]. We report mean energy per sample and samples per second values for different power-modes and CPU core baseloads.

$$EPS = \frac{P_{total} - P_{BL}}{N} \tag{1}$$

$$\begin{aligned} EPS &: \text{Energy per Sample} \\ P_{\text{total}} &: \text{Total power consumption (FL and Baseload)} \\ P_{\text{BL}} &: \text{Power consumption due to Baseload} \\ N &: \text{Number of Samples} \end{aligned}$$

Energy per sample values were calculated using Eq. 1.

Figure 1a illustrates the mean energy per sample and 95% confidence intervals for each powermode, based on 10 repeated measurements. We observe significant difference in energy per sample and samples per second values when there is no non-FL base load (0 baseload cores) compared to a scenario when non-FL baseload is executing and utilizing all CPU cores. We also observe that while samples per second (Figure 1b) doesn't vary significantly when non-FL baseload is co-running with FL, energy per sample values fluctuate for baseloads 3 and 4. For our experiments, ondemand mode with baseload cores 3 has an optimum energy usage when calculating same number of samples compared to other baseload and samples per second combinations.

IV. Conclusion and Future Work

Recent research studies have focused on energy-efficient and carbon-efficient FL scheduling and client selection. However, most of the research assumes simplistic energy consumption models for underlying FL clients. In this work, we showed that how energy per sample values under real-world scenarios such as different power modes and non-FL baseloads at CPU cores can vary and exhibit complex operational behavior patterns.

For future work, following open research questions and possibilities could be explored further,

- How do current FL systems communicate FL clients' energy related information? How to collect energy per sample, throughput per second and uncertainty related information at runtime?
- How can we predict the power-performance characteristics, what are the relevant metrics? With more data about real-world impact factors affecting energy footprint of edge devices, can we build predictive models for forecasting?
- How often do we need to measure before we can be certain? Can we report the uncertainty to be used in scheduling? FL trainings are usually executed multiple times due to data distribution drifts and hyperparameter search. This repetitive FL training execution could be leveraged to collect more data about power-performance behavior patterns of FL clients.
- What's the impact of hardware accelerated edge devices such as jetson nano on energy related metrics? What are the energy efficiency opportunities in FL and non-FL co-running scenarios?

References

[1] H. B. McMahan, E. Moore, D. Ramage, S. Hampson, and B. A. y Arcas, "Communication-efficient learning of deep networks from decentralized data," in *AISTATS*, 2016.

[2] X. Qiu, T. Parcollet, D. J. Beutel, T. Topal, A. Mathur, and N. D. Lane, "A first look into the carbon footprint of federated learning," *CoRR*, vol. abs/2010.06537, 2020.

[3] X. Zhou, J. Zhao, H. Han, and C. Guet, "Joint optimization of energy consumption and completion time in federated learning," in *ICDCS*, IEEE, 2022.

[4] C. W. Zaw, S. R. Pandey, K. Kim, and C. S. Hong, "Energy-aware resource management for federated learning in multi-access edge computing systems," *IEEE Access*, vol. 9, pp. 34938–34950, 2021.

[5] Y. G. Kim and C.-J. Wu, "Autofl: Enabling heterogeneity-aware energy efficient federated learning," in *IEEE/ACM International Symposium on Microarchitecture (MICRO)*, pp. 183–198, 2021.

[6] P. Wiesner, R. Khalili, D. Grinwald, P. Agrawal, L. Thamsen, and O. Kao, "Fedzero: Leveraging renewable excess energy in federated learning," *arXiv preprint arXiv:2305.15092*, 2023.

[7] B. Güler and A. Yener, "A framework for sustainable federated learning," in *International Symposium on Modeling and Optimization in Mobile, Ad hoc, and Wireless Networks (WiOpt)*, IEEE, 2021.

[8] B. Rupprecht, D. Hujo, and B. Vogel-Heuser, "Performance evaluation of ai algorithms on heterogeneous edge devices for manufacturing," in *International Conference on Automation Science and Engineering (CASE)*, pp. 2132–2139, IEEE, 2022.

[3]https://flower.dev
[4]https://www.cs.toronto.edu/ kriz/cifar.html
[5]https://www.delock.com/produkt/11826/merkmale.html

Demystifying User-based Active Learning for Network Monitoring Tasks

Katharina Dietz*, Nikolas Wehner*, Pedro Casas†, Tobias Hoßfeld*, Michael Seufert*
*University of Würzburg, Germany, †AIT Austrian Institute of Technology, Vienna
*{katharina.dietz, nikolas.wehner, tobias.hossfeld, michael.seufert}@uni-wuerzburg.de, †pedro.casas@ait.ac.at

I. Introduction and Problem Statement

In the past decade, Artificial Intelligence (AI), especially Machine Learning (ML), has enjoyed increasing popularity and has been widely applied in network monitoring and management. However, when it comes to sensitive topics such as network security, practical adoption has been poor, e. g., for anomaly and intrusion detection. Academic research on security-related topics lacks a holistic view [1], focusing on either human or technical aspects, while neglecting the interconnection of both [1]. Incorporating standard AI/ML approaches can further widen this gap and create barriers [2], as they take away the decision making from the experts/admins without giving any information about the confidence or severity of their decision, and provide no means to give feedback to the model, ultimately reducing the trust of the experts/admins. Thus, having an admin in the loop can be beneficial not only for the overall performance of the model by incorporating expert knowledge, but also increase its trustworthiness.

Originally, Active Learning (AL) aims to increase the performance of an ML model by manually labeling as few samples as possible via queries to an *oracle* [3], e. g., a (human) expert or a more powerful model, and add these few select samples to the (re)training data. A complementary technique to AL is self-training [4], which consists of a base classifier, which classifies the unlabeled data, but only adds the data to the (re)training on which it was the most confident on. While AL aims for the most informative data, self-training targets the most confident data to add to the model. Combining both techniques can yield great benefits [4], i. e., utilizing a base classifier to make automated decisions for obvious choices, while relaying inconfident decisions to the oracle, thus incorporating a human in the AI/ML loop.

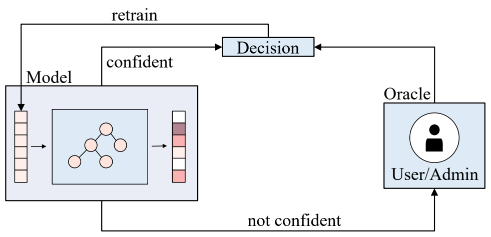
In the context of communication networks, these approaches help enabling the real-world deployment of AI/ML solutions by giving the users decisive power of critical administrative decisions, while enriching the monitored data with expertise. For example, when unseen devices, applications, or even zero-day attacks start appearing in the network (i. e., equating to potential labels the model was not trained on), the model may not be very confident in its decision and thus relay it to the expert admin for further inspection and relabeling. Ultimately, this also helps keeping the model up-to-date and adapt to network changes over time. In the following, we propose our general methodology, discuss specific use cases and related research fields, which can be incorporated in the future.

II. Background and Methodology

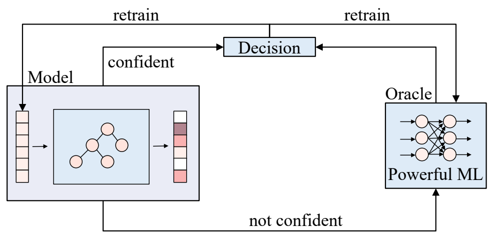
Active Learning. According to [3], AL can be divided into three different approaches, namely pool-based AL, stream-based selective sampling, and membership query synthesis. The pool-based approach ranks all datapoints regarding pre-defined criteria and relays the top ranked points to the oracle, while the latter two make an independent judgement for each sample [3]. Additionally, a membership query generates new samples on its own. The former two approaches make the most sense in context of our use case. Here, it is more of a question if we want to set a specific threshold (e. g., confidence of a decision) for every element to be relayed, which can potentially result in many requests/false alarms to the admin, or if we want to set a fixed number of queries beforehand and potentially miss out on important events. Thus, all approaches require a careful configuration of their parameters.

Querying Strategies. Regardless of the chosen AL approach, there is also a multitude of querying strategies, i. e., how to actually choose which elements to relay. In [3], the main querying strategies have been identified as diversity-based approaches, approaches based on the expected model change, and uncertainty-based approaches. The latter is a natural fit due to the self-training aspect of our use case, while the other two focus more on picking the most informative data-points, which do not necessarily need any admin supervision.

ML Confidence/Uncertainty. To actually evaluate the confidence or uncertainty of a decision, we need predicted probabilities for all possible classes instead of just the predicted label. Luckily, many traditional ML but also Deep Learning (DL) algorithms are capable of doing so, including Random Forests (or basically any bagging/boosting algorithms due to simple majority voting), Decision Trees, Neural Networks (via softmax layers), and many more. Given these class probabilities, we may now formulate our own definitions of confidence/uncertainty. For example, we may simply set a threshold of how big the probability for the most probable class is (e. g., relay all classified samples that have no class probabilities higher than 0.8), calculate the entropy of the class probabilities (e. g., maximum entropy means that all probabilities are distributed equally and thus there is no clear winner), or calculate the distance to a uniform distribution (e. g., low Kolmogorov-Smirnov or Wasserstein distances indicate uniformly distributed probabilities thus inconfidence). How to configure these parameters and which strategy to choose is a core question.



(a) Use case 1: Admin as oracle



(b) Use case 2: Powerful ML as oracle

Fig. 1: Potential use cases for AL in network monitoring.

III. Potential Use Cases

Admin as Oracle. Figure 1a illustrates the idea behind the previously described use case. Our approach to test the impact of AL in networking may look as follows. We may utilize any existing dataset concerning network monitoring, e. g., application classification, device fingerprinting, or intrusion detection, and train a suitable ML model with them. Then we just simply utilize the label probabilities to decide if we relay the decision to an admin. As access to experts is limited, we opt for a parameterized, simulative approach, i. e., we have virtual admins that exhibit different properties with respect to accuracy and time consumption, e. g., make correct decisions in short time or take a long time and have a high probability for erroneous decisions, or any value in-between. This way we can not only evaluate if the performance improvement is significant, but also if the time consumption is even worth it. In other words, we want to find a suitable threshold via parameter studies for the uncertainty/confidence, so that the model improves, while the admin is not overburdened. We can also analyse the impact of unseen traffic (e. g., new devices/apps or zero-day attacks), by excluding one class from the training data. We expect the ML models to be inconfident on unseen labels, thus relaying these datapoints to the admin.

In-network ML. The second use case is seen in Figure 1b. The core idea is, that in-network ML (e. g., via P4 switches) can extract features and classify at line rate, which is desirable for, e. g., real-time intrusion detection/prevention. Though, models are limited in their complexity as well as features to extract due to limited operations (in regard to their type as well as their amount), potentially resulting in lower accuracy. We can leverage the previous methodology by relaying all inconfident decisions from the switches to the oracle. In this case, this may be a powerful ML server with more complex models, aiming for high accuracy. In other words, for clear decisions we can provide low/no latencies, while for harder decisions we utilize some more time and resources, thus proposing a hybrid approach. Even though we now induce a small latency overhead for the sake of higher accuracy, this still costs a lot less time than introducing a human expert into the loop, thus combining the best of both worlds. As we now have two models we deploy in the network, we also need to update and retrain both of them, making this use case more complex from a technical point of view. Especially since P4 switches cannot be updated during runtime, this may be an interesting aspect for future research with regards to downtimes etc.

IV. Related Research and Future Work

Visualization. In [5] the authors also opt for a simulation-based approach for user-based active learning in the context of image classification. Instead of just simulating “good” or “bad” experts, they develop a visual approach via dimensionality reduction and base their metrics on these visuals. This may be another methodology to implement for our use case as well, as in the end, if we want to apply this in practise, the admin/user needs some help with their decision making.

User Studies. In conjunction with visualization, we can also conduct user studies instead of having virtual admins to make our analyses more impactful and to obtain more realistic values for our admin competence and time needed to complete the tasks, shown to be fruitful in security-related topics [6]. Though, as we potentially need network experts/admins to do so, this may be a challenging task. Alternatively, we can try to simplify the admin tasks [7], e. g., by giving non-experts a small handbook/guide and make the decision tasks less complex and more well-defined. However, regardless of the users participating in potential studies, we need to design an interface first, i. e., develop a fitting visualization.

Outlier Detection. Lastly, complementary to ML uncertainty/confidence, we can also correlate outlier detection to our research. In other words, we expect there to be some kind of relationship between both approaches, as both possibly indicate that a data sample like this has not been seen before. Thus, this may add another querying strategy for our use cases.

Acknowledgment

This work was partly funded by Deutsche Forschungsgemeinschaft (DFG) in project Usernet (SE 3163/3-1, project nr. 500105691). The authors alone are responsible for the content.

References

[1] M. Vielberth, F. Böhm, I. Fichtinger, and G. Pernul, “Security operations center: A systematic study and open challenges,” *IEEE Access*, vol. 8, pp. 227 756–227 779, 2020.

[2] D. Han, Z. Wang, W. Chen, Y. Zhong, S. Wang, H. Zhang, J. Yang, X. Shi, and X. Yin, “Deepaid: interpreting and improving deep learning-based anomaly detection in security applications,” in *Proceedings of the 2021 ACM SIGSAC Conference on Computer and Communications Security*, 2021, pp. 3197–3217.

[3] P. Ren, Y. Xiao, X. Chang, P.-Y. Huang, Z. Li, B. B. Gupta, X. Chen, and X. Wang, “A survey of deep active learning,” *ACM computing surveys (CSUR)*, vol. 54, no. 9, pp. 1–40, 2021.

[4] M. S. Hajmohammadi, R. Ibrahim, A. Selamat, and H. Fujita, “Combination of active learning and self-training for cross-lingual sentiment classification with density analysis of unlabelled samples,” *Information sciences*, vol. 317, pp. 67–77, 2015.

[5] C. Seifert and M. Granitzer, “User-based active learning,” in *2010 IEEE International Conference on Data Mining Workshops*. IEEE, 2010, pp. 418–425.

[6] A. Beaugnon, P. Chifflier, and F. Bach, “End-to-end active learning for computer security experts,” in *KDD Workshop on Interactive Data Exploration and Analytics (IDEA)*, 2018.

[7] H. Trittenbach, A. Englhardt, and K. Böhm, “Validating one-class active learning with user studies–a prototype and open challenges,” in *ECML PKDD Workshop*, 2019, p. 17.

Data distribution effects on asynchronous Parameter Server training in high delay differences networks

1st Leonard Paeleke
Hasso Plattner Institute (HPI)
University of Potsdam
Potsdam, Germany
leonard.paeleke@hpi.de

2nd Holger Karl
Hasso Plattner Institute (HPI)
University of Potsdam
Potsdam, Germany
holger.kar@hpi.del

***Abstract*—Model training with distributed Machine Learning (ML) methods, such as the well-known asynchronous Parameter Server (PS), is susceptible to network inhomogeneities, e.g. differences in link latency, computing resources, or data distribution. We evaluate how the combination of data distribution and high delay difference between workers affects asynchronous PS training. We show that a better ground-truth representation at the faster and more dominating worker increases model accuracy and reduces computational efforts by about five times.**

***Index Terms*—Distributed Machine Learning, Asynchronous Parameter Server, Networks for ML**

I. Introduction

Networked systems, such as IoT networks and future cellular communication networks, are envisioned to utilize Machine Learning (ML) for complex task. Training these ML models requires data that usually is collected on networked machines. However, data transmission to, e.g., a data center can heavily load the network and violate data privacy concerns. With *distributed* ML methods, e.g., using a Parameter Server (PS) [1], [2], the training can be distributed across multiple decentralized machines (*workers*). These workers maintain an instance of the model and process their local data to compute update gradients that describe necessary changes to the model parameters (bias, weights). Typically, the data set is distributed across all workers in non-overlapping portions. After processing a predefined number of data samples, the workers exchange their update gradients with a centralized machine (*parameter server*), which applies the update gradients of all workers to the model. The updated model is then returned to the contributing workers, which then continue generating update gradients. The parameter server also stores the model parameters.

Various PS implementations exist that primarily differ in how and how many workers contribute to a model update. In synchronous implementations, the parameter server aggregates the update gradients from all workers simultaneously before broadcasting, while the workers interrupt their processing until receiving the updated model. In contrast, in asynchronous PS implementations, the parameter server updates and returns the updated model immediately after receiving an update gradient, reducing the idle time of workers [3].

In practice, PS is exposed to varying link latencies or computing capacities. Such network inhomogeneities lead to long training times for synchronous PS and motivated asynchronous PS [3]. For asynchronous PS, network inhomogeneities change the order of model updates, which impacts the model development and makes the model development susceptible to data distribution [4]. Several methods have been described to mitigate these effects, e.g., back-up workers [5], limiting worker staleness [6], or changes to the typically used stochastic gradient descent (SGD) optimizer [7], [8]. Still, it is unclear how data should be distributed when the delays between worker and parameter server differ significantly. We help understand how the combination of inhomogeneous networks and data distribution affects model development for asynchronous PS.

II. Experiment design

In our experiment, an asynchronous PS in an inhomogeneous network trains an artificial neural network on the NSL-KDD data set [9] for different data distributions. The training infrastructure comprises three virtual machines (Ubuntu 22.04) on the same physical server with equal computing resources, setting up a network with two workers, each connected to a parameter server. We use Ray [10] to implement the asynchronous PS. To model an inhomogeneous network that causes changes in update order due to varying link latency or computation capacity, we impose delays of 1 ms and 100 ms, respectively, on the links from worker to parameter server using the `tc` package. This gives us fine-grained control over the transmission time from sending an update to receiving the updated model.

NSL-KDD is typically used for training models detecting intrusions in traffic data. Under supervised learning, a model learns to classify traffic into five classes: Denial of Service (DoS), User to Root (U2R), Remote to Local (R2L), Probing, and Normal. Here, we train the model on three quarters of the data set and evaluate the model's accuracy on the remainder. As the disproportionate impact of data distribution on model development in inhomogeneous settings is our key hypothesis, we distribute the data differently among the workers. For the experiments, the *fast* worker (with lower added link delay) accesses the DoS, R2L, and one-half of the Normal traffic data. The *slow* worker with a higher link delay accesses the other half of the Normal traffic data. These distributions stay fixed for all experiments; the distribution of U2R and Probing data varies across experiments.

The model to train is a fully connected artificial neural network with one hidden layer. It has 93 neurons in the input layer, 21 neurons in the hidden layer, and five neurons, one for every class as a one-hot-encoding, in the output layer. The hidden layer uses ReLU activation; the output layer uses logarithmic softmax activation.

III. Data distribution effects on model quality

In this section, we analyze the effects of data distribution on model development for asynchronous PS training in an inhomogeneous network. A uniform delay of 1 ms for the fast link and 100 ms for the slower link is used to form the inhomogeneous network. Throughout the experiment, we vary the proportions of U2R and Probing data accessed by each worker in increments of ten from zero to 100 percent. The data portions across both workers always add up to 100 percent. For each data distribution, we train twenty times with reshuffled data sets and with different model initializations. Model accuracy is the key metric; we show it over both number of training epochs as well as over data proportions.

Figure 1 shows that model accuracy increases the more the dominant (faster) worker knows about the ground-truth distribution. Hence, the reference independent identically distributed (IID) case achieves the highest accuracy, where data is equally distributed among the workers, such that each worker's data portion represents the entire ground-truth distribution. This is to be expected in such a vastly heterogeneous setup. However, in real-world applications with constrained computing capacity, (quickly) reaching specific model quality levels is of higher interest than the maximal accuracy. Given, e.g., the goal of 94% accuracy, even a little information about the classes at the slower worker reduces the necessary epochs to a third (cf. 0/100, 20/80 at 94% accuracy). The number of epochs can be reduced up to five times if the faster worker has more information about these classes.

IV. Summary and Outlook

We have shown that for the well-known asynchronous PS method in inhomogeneous networks, the distribution of the training data among the workers affects the model accuracy, in particular, its evolution over number of epochs. We found that the better the training data at the dominating worker represents the ground-truth data, the higher the achieved model accuracy. Furthermore, better ground-truth representation at the dominating worker can reduce by a factor of five the computational effort needed to achieve a certain accuracy level. We conclude that data transfers between workers before training can improve the training; however, we still need to take resource trade-offs for such transfers into account.

We hence plan to extend this study, short-term, to a quantitative characterization of the efforts for training and data load arising from different data sets and models. Further,

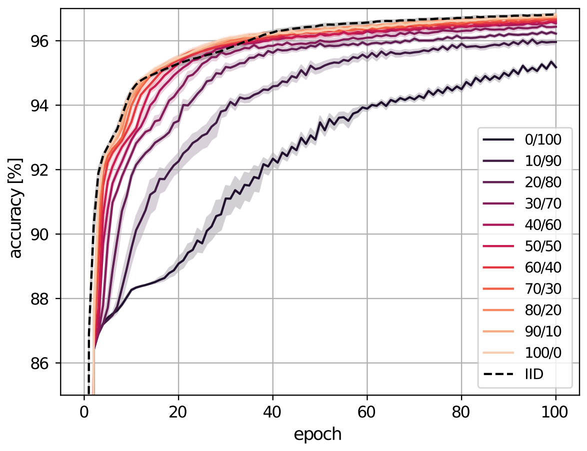


Fig. 1. Accuracy over epoch for different data distributions

we want to evaluate how susceptible different distributed ML architectures (e.g., All-Reduce or Federated Learning) are to data distribution among workers in inhomogeneous networks.

V. Acknowledgments

The authors acknowledge the financial support by the Federal Ministry of Education and Research of Germany in the project “Open6GHub” (grant number: 16KISK011).

References

[1] A. Smola and S. Narayanamurthy, “An architecture for parallel topic models,” *Proceedings VLDB Endowment*, vol. 3, no. 1-2, pp. 703–710, Sep. 2010.

[2] M. Li, D. G. Andersen, J. W. Park, A. J. Smola, A. Ahmed, V. Josifovski, J. Long, E. J. Shekita, and B.-Y. Su, “Scaling distributed machine learning with the parameter server,” 2014.

[3] J. Dean, G. S. Corrado, R. Monga, K. Chen, M. Devin, Q. V. Le, M. Z. Mao, M. Ranzato, A. Senior, P. Tucker, K. Yang, and A. Y. Ng, “Large scale distributed deep networks,” *Advances in neural information processing systems*, vol. 25, 2012.

[4] L. Paeleke and H. Karl, “Wip: How latency differences between workers affect parameter server training,” 2023, access: June 8. 2023. [Online]. Available: https://netaisys.github.io/wip_papers/netaisys23-final78.pdf

[5] J. Chen, X. Pan, R. Monga, S. Bengio, and R. Jozefowicz, “Revisiting distributed synchronous sgd,” *arXiv preprint arXiv:1604.00981*, 2016.

[6] Q. Ho, J. Cipar, H. Cui, S. Lee, J. K. Kim, P. B. Gibbons, G. A. Gibson, G. Ganger, and E. P. Xing, “More effective distributed ml via a stale synchronous parallel parameter server,” *Advances in neural information processing systems*, vol. 26, 2013.

[7] J. Jiang, B. Cui, C. Zhang, and L. Yu, “Heterogeneity-aware distributed parameter servers,” in *Proceedings of the 2017 ACM International Conference on Management of Data*, ser. SIGMOD ’17. New York, NY, USA: Association for Computing Machinery, May 2017, pp. 463–478.

[8] X. Lian, W. Zhang, C. Zhang, and J. Liu, “Asynchronous decentralized parallel stochastic gradient descent,” in *International Conference on Machine Learning*. PMLR, 2018, pp. 3043–3052.

[9] M. Tavallaee, E. Bagheri, W. Lu, and A. A. Ghorbani, “A detailed analysis of the kdd cup 99 data set,” in *2009 IEEE Symposium on Computational Intelligence for Security and Defense Applications*, 2009, pp. 1–6.

[10] P. Moritz, R. Nishihara, S. Wang, A. Tumanov, R. Liaw, E. Liang, M. Elibol, Z. Yang, W. Paul, M. I. Jordan, and I. Stoica, “Ray: A distributed framework for emerging AI applications,” Dec. 2017.

Replicable Machine Learning Workflow for Energy Forecasting

Stepan Gagin
Chair of Computer Networks and Computer Communication
University of Passau
Passau, Germany
stepan.gagin@uni-passau.de

Adrian Carrasco-Revilla
Data Scientis Senior (Innovation Department)
Inetum
Madrid, Spain
adrian.carrasco@inetum.com

Hermann de Meer
Chair of Computer Networks and Computer Communication
University of Passau
Passau, Germany
hermann.demeer@uni-passau.de

Nuria Sánchez Almodóvar
Head of Innovation Department
Inetum
Madrid, Spain
nuria.sanchez@inetum.com

***Abstract*—Energy forecasting is an essential functionality of energy management systems. In this paper, a machine learning workflow for energy forecasting is proposed. Integration of such forecasting service does not require preliminary knowledge in a machine learning domain. Moreover, the workflow is designed to be replicable. That means that it can be used for the forecasting of different measurements in the energy domain in multiple locations.**

***Index Terms*—machine learning, energy systems, replicability.**

I. Introduction

Energy forecasting is an essential functionality for the energy sector [1]. Many energy management systems (EMSs) use forecasting to control energy equipment and send recommendations to users and other applications to manage their energy consumption. However, a development and training of forecasting models as well as processing data from sensors require familiarity with machine learning (ML) concepts, while energy managers often lack experience in that domain.

The replicability of software systems used in research [2] is not always ensured in the ML domain. To mitigate that, in this paper the replicable ML workflow is proposed. A replicability of the workflow means, that it can be used without additional changes in different locations for time series forecasting in the energy sector. This workflow can be integrated into existing EMSs and does not require the involvement of a ML specialist. For an integration, only historic time series are necessary, however, exogenous variables can be accepted to improve the predictions.

II. Related Work

The concept of automated ML is becoming popular [3]. It aims to automate the training of multiple ML models and chooses the one that is performing best. ML can be encapsulated as a reusable software service [4]. However, the approach proposed in this paper focuses on the forecasting in energy sector, for instance on photovoltaic (PV) generation, building energy consumption in heating and electricity domains. That means that the models can be trained with specific types of measurements and interconnection between them in mind, which improves forecasting performance.

This paper has received funding by the European Union's Horizon 2020 research and innovation programme under grant agreement No. 957845: "Community-empowered Sustainable Multi-Vector Energy Islands - RENergetic".

III. Methodology

The replicable workflow introduced in this paper can be connected to the centralized data storage of an EMS as shown in Figure 1. The other EMS services can communicate with the storage too. There could be multiple copies of the workflow running in parallel in the system for different types of forecast data series, for instance, PV generation, electricity consumption of a building or heating consumption of a flat in a building. Moreover, the same workflow can be used for different buildings in multiple locations.

Replicability of a workflow means that it can be reused for forecasting of different time series without any internal changes. An interface between data storage and forecasting workflow consists of three elements:

- measurements - historic data from a sensor with a given timestamp for each measurement;
- weather data - historic data connected to the weather, such as temperature, relative humidity, wind speed and solar radiation;
- metadata - additional data about the type of sensor, physical quantity and units used for a measurement.

That allows for easy usage of this workflow by non-ML specialists because it does not require any adjustments of the parameters of underlying models.

The workflow can be implemented as a separate software service that communicates with other services using REST API. Inputs and outputs of the workflow can be represented as JSON objects with predefined values for attributes to model data and metadata.

Subsections III-A-III-E provide an overview of each step of the proposed workflow.

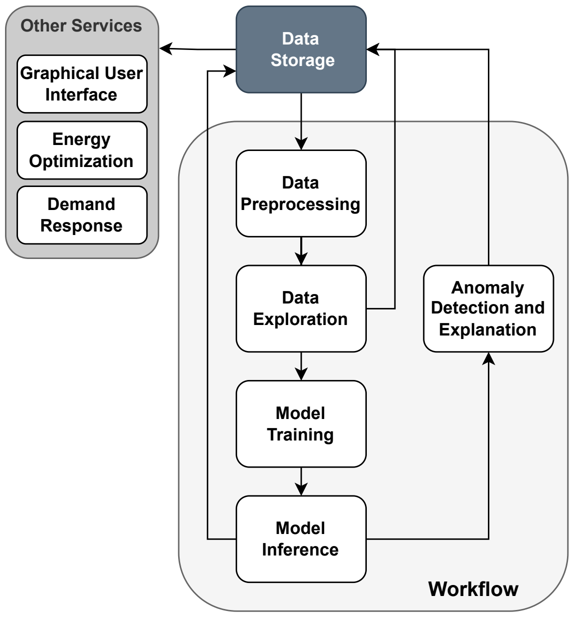


Fig. 1. Forecasting workflow

A. Data Prepossessing

Data collected from sensors and equipment can contain missing data points, thus they should be estimated, for instance, using the interpolation methods. Additionally, the data used in forecasting models should have a constant frequency of data points. However, data from sensors can arrive at the data storage at different time. Thus, data should be transformed and aggregated. Aggregation should be done considering the type of measurements. For instance, PV panels report about their generation power, although for energy optimization the data about energy generated are needed.

B. Data Exploration

In this step, statistics and other information about time series are collected. These can include a median, average and standard deviation of time series over a defined period. Additionally, the identification of patterns in time series is done. The identified shapelets [5] are used in the following steps by forecasting models. Anomaly detection is also performed at this step to discover and handle outliers in the data set before training. The output can be used by the forecasting models or by other services in the system, for instance, to display statistics in a graphical user interface.

C. Model Training

The workflow assumes that the model was already created. This step includes training of pre-defined models or their versions with different hyperparameters that can be used for the forecasting. It is important to note that training of the model on new data is performed, only if needed. It is decided based on the performance metrics of previously predicted data points. If performance is higher than set threshold, the re-training is not needed. If re-training is needed, several models are trained in parallel and the best-performing model is chosen. Such mechanism allows saving computational resources spent on the model training.

The models could be trained in competition (only one is saved) or in cooperation, meaning the final process combines all the results into a final model which is trained to take the benefits of each model and discard the weak spots.

D. Model Inference

In this step, the model resulting from the previous step makes predictions for the next time window, for instance, for 24 hours, 48 or an interval provided by the user.

E. Anomaly Detection and Explanation

Using the same processes introduced in Subsection III-B, the output of forecasting models can be analysed, for instance, allowing quick actions on results, such as notifications when high values are forecasted. Additionally, some models support the explainability of results - from feature importance they provide information on which elements of input data have the biggest influence on the output to seasonality disambiguation, which shows which of the time components impacts the forecast.

IV. Conclusion

In this paper, a ML workflow for energy forecasting is proposed. This workflow includes several steps that enable an integration of energy forecasting functionalities into EMSs without expertise in the ML domain. Replicability of the workflow means, that it can be used for energy forecasting in different locations. In an extended version of this paper, the models that can be used in such workflow are developed and evaluated on real-world data.

References

[1] T. Hong, P. Pinson, Y. Wang, R. Weron, D. Yang, and H. Zareipour, "Energy Forecasting: A Review and Outlook," vol. 7, pp. 376–388, 2020, publisher: Institute of Electrical and Electronics Engineers Inc.

[2] Association for Computing Machinery. [Online]. Available: https://www.acm.org/publications/policies/artifact-review-and-badging-currentappendix

[3] X. He, K. Zhao, and X. Chu, "AutoML: A survey of the state-of-the-art," *Knowledge-Based Systems*, vol. 212, p. 106622, Jan. 2021. [Online]. Available: https://linkinghub.elsevier.com/retrieve/pii/S0950705120307516

[4] M.-O. Pahl and M. Loipfinger, "Machine learning as a reusable microservice," in *NOMS 2018 - 2018 IEEE/IFIP Network Operations and Management Symposium*. Taipei: IEEE, Apr. 2018, pp. 1–7. [Online]. Available: https://ieeexplore.ieee.org/document/8406165/

[5] J. Grabocka, N. Schilling, M. Wistuba, and L. Schmidt-Thieme, "Learning time-series shapelets," ser. KDD '14. New York, NY, USA: Association for Computing Machinery, 2014, p. 392–401. [Online]. Available: https://doi.org/10.1145/2623330.2623613

Exploring AI-Based Adaptive Resource Management in 5G Networks

Ole Hendrik Sellhorn* and Horst Hellbrück*
*Department EI, Center of Excellence CoSA
Technische Hochschule Lübeck - University of Applied Sciences, Lübeck, Germany
ole.sellhorn@th-luebeck.de, horst.hellbrueck@th-luebeck.de

***Abstract*—Resource allocation is a key challenge for gNB scheduler in 5G networks, for distributing limited resources to end devices. The scheduler's goal is to optimize transmission quality and quality of service for applications like real-time communication, video streaming, IoT, and autonomous driving. Traditional scheduling algorithms struggle to cope with the dynamic and heterogeneous demands of 5G networks, which result from the increasing number of connected devices, network traffic and application requirements. We propose Artificial Intelligence techniques for gNB scheduler and enable intelligent and adaptive resource allocation. By utilizing AI, the scheduler learn from network data, adapt to changing conditions, and optimize resource utilization and transmission quality for different applications. Our approach involves implementing three AI techniques: Performance comparisons between AI-based and classical scheduling algorithms are conducted using metrics such as network throughput, latency, packet loss, resource usage, and user satisfaction. The proposed AI-based scheduling algorithms have the potential to improve network performance, reduce congestion, conserve resources, and enhance the user experience in 5G networks.**

***Index Terms*—5G, gNB, Artificial Intelligence, LSTM, Deep Reinforcement Learning, Online Learning, Machine Learning**

I. Introduction

We deployed a 5G standalone (SA) network, which consists of only 5G components [1], unlike the common 5G networks that rely on 4G components. The SA network enables us to conduct further research, such as demonstrating High Throughput and Ultra-Reliable Low-Latency Communications [2]. However, a major challenge in 5G networks is to optimize the complex configuration options and resource allocation. Previous approaches without AI [3], [4] have addressed this challenge, but they may not be flexible enough to cope with the dynamic and heterogeneous demands of 5G networks.

Various solutions with AI have already been developed. Some approaches support resource allocation with Deep Reinforcement Learning to optimize beamforming, power control and interference coordination [5]. Alternative solutions perform optimized and fast real-time resource slicing with deep-dueling neural networks [6]. Figure 1 illustrates an overview of AI in a 5G network.

II. AI Based Optimization of the gNB scheduler

In 5G networks, a gNB scheduler is usually controlled by rules and algorithms defined by standardization organizations

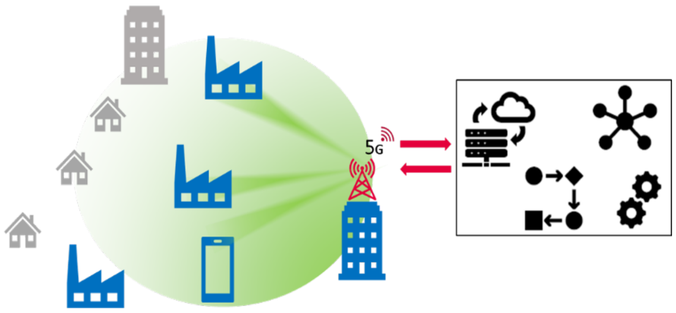


Fig. 1. AI implementation into 5g network

e.g. 3rd Generation Partnership Project (3GPP). These rules and algorithms define how resources are distributed to end devices based on quality of service (QoS) requirements, priorities, channel status, and network utilization [7].

However, the increasing complexity of 5G and the various applications limit the control with rule-based and classical algorithms. In summary, there is no standard configuration of the gNB scheduler that fits best for all environments. For this reason, artificial intelligence algorithms gain more importance in future. Our goal is the development of a scheduler that adopts to changes in the environment. Additionally, it evolves over time, i.e. enable continuous learning. The following three aspects represent optimization approaches for AI:

Resource Optimization: Deep Reinforcement Learning is an AI technique that combines reinforcement learning and deep neural networks [8]. Figure 2 shows the basic operation of a Deep Reinforcement Learning algorithm.

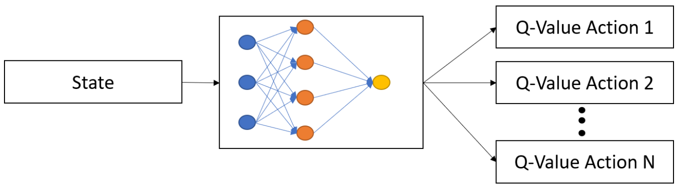


Fig. 2. Deep Reinforcement Learning

In the context of the gNB scheduler, Deep Reinforcement Learning optimizes the allocation of network resources, such as future slices. The scheduler learns by trial and error, combined with feedback on its decisions, and adjusts its resource allocation strategy accordingly.

Real-time Adaptation: Online Learning is a machine learning approach where the model learns and adapts in real-time as new data becomes available [9]. In the case of the gNB scheduler, online learning enables the scheduler to adapt and adjust its decisions based on changing network conditions. The scheduler continuously receives and processes real-time data on network parameters such as traffic load, signal quality, and interference levels. It optimizes resource allocation, user or service prioritization and network performance by updating its decision-making process to network conditions. For example, it dynamically allocates bandwidth based on traffic demands, prioritizes critical services during congestion, and adjusts transmission policies for improved network efficiency.

Network Faults: The second and third layer leverages Long Short-Term Memory models, a type of time series models, to detect or predict network faults, supporting the gNB Scheduler. Time series data represent a sequence of observations collected over time, such as network performance metrics. LSTM models are a type of recurrent neural network (RNN) that can capture dependencies and patterns in time series data [10]. By training an LSTM model on historical network data, the gNB Scheduler can detect anomalies or predict potential network failures. This allows proactive actions, such as traffic rerouting, resource reallocation or maintenance triggering, to prevent disruptions and maintain network stability and reliability.

III. Evaluation Concept

To evaluate the AI algorithms in 5G networks, a comprehensive experimental evaluation is conducted. The following methodology is used to evaluate the performance and effectiveness of the implemented AI algorithms:

- Data collection: Real data is collected from the 5G SA network, including network performance metrics, congestion levels, resource utilization, and user experience. This data forms the basis for evaluating the AI algorithms.
- Test Scenarios: Different scenarios are designed to evaluate the performance of the AI algorithms. The scenarios include different network conditions, traffic loads, and QoS requirements to simulate real-world operation.
- Performance metrics: Several metrics are compared for effectiveness of the AI algorithms. These metrics include network performance indicators such as throughput, latency, and packet loss, as well as congestion levels, resource utilization, and user satisfaction.
- Comparative analysis: The results of the AI-enabled gNB scheduler are compared to the performance of the classical algorithms. This comparative analysis provides insights into the improvements achieved by the AI algorithms in terms of network performance, congestion reduction, resource efficiency, and user experience.
- Statistical analysis: Statistical methods are applied to analyze the collected data and draw meaningful conclusions. The significance of the differences between the AI algorithms and the traditional algorithms will be evaluated using appropriate statistical tests.

Through rigorous experimental evaluation, this study aims to provide quantitative evaluation of the AI algorithms on the performance of the gNB scheduler in 5G networks. Analysis of the results based on various metrics provides valuable insights into the effectiveness and benefits of using AI techniques in resource management and transmission optimization.

IV. Conclusion and Future Work

We demonstrated how AI algorithms support a gNB scheduler and why this is increasingly important. We also presented the steps for implementing and evaluating such algorithms. Our future work involves implementing the AI algorithms in our 5G SA network's gNB scheduler and conducting an extensive evaluation. Moreover, we plan to deploy other AI algorithms for autonomous network management, such as learning, fault detection and recovery, self-optimization, efficiency, and scalability.

ACKNOWLEDGMENTS

This publication results from the research of the Center of Excellence CoSA at the University of Applied Sciences Lübeck. It is funded by the Federal State Schleswig-Holstein, Germany (Id 22021005, Project KI-5G, Project Management Agency: WTSH). Horst Hellbrück is an adjunct professor at the Institute of Telematics of the University of Lübeck

References

[1] F. John, J. Schuljak, L. B. Vosteen, B. Sievers, A. Hanemann, and H. Hellbrueck, "A reference deployment of a minimal open-source private industry and campus 5g standalone (sa) system," in *2022 IEEE 10th International Conference on Information, Communication and Networks (ICICN)*, 2022, pp. 1–9.

[2] F. John, B. Sievers, O. H. Sellhorn, and H. Hellbrueck, "Two industrial reference demonstrators for high throughput and low latency in 5g standalone network setups," in *Mobile Communication-Technologies and Applications; 27th ITG-Symposium*. VDE, 2023, pp. 1–6.

[3] W. S. Afifi, A. A. El-Moursy, M. Saad, S. M. Nassar, and H. M. El-Hennawy, "A novel scheduling technique for improving cell-edge performance in 4g/5g systems," *Ain Shams Engineering Journal*, vol. 12, no. 1, pp. 487–495, 2021. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S2090447920301878

[4] A. Mamane, M. Fattah, M. El Ghazi, M. Bekkali, B. Younes, and S. Mazer, "Scheduling algorithms for 5g networks and beyond: Classification and survey," *IEEE Access*, vol. 10, pp. 1–1, 01 2022.

[5] F. B. Mismar, B. L. Evans, and A. Alkhateeb, "Deep reinforcement learning for 5g networks: Joint beamforming, power control, and interference coordination," *IEEE Transactions on Communications*, vol. 68, no. 3, pp. 1581–1592, 2020.

[6] N. Van Huynh, D. Thai Hoang, D. N. Nguyen, and E. Dutkiewicz, "Optimal and fast real-time resource slicing with deep dueling neural networks," *IEEE Journal on Selected Areas in Communications*, vol. 37, no. 6, pp. 1455–1470, 2019.

[7] K. Pedersen, G. Pocovi, J. Steiner, and A. Maeder, "Agile 5g scheduler for improved e2e performance and flexibility for different network implementations," *IEEE Communications Magazine*, vol. 56, no. 3, pp. 210–217, 2018.

[8] Y. Li, "Deep reinforcement learning: An overview," 2018.

[9] S. C. H. Hoi, D. Sahoo, J. Lu, and P. Zhao, "Online learning: A comprehensive survey," 2018.

[10] R. C. Staudemeyer and E. R. Morris, "Understanding lstm – a tutorial into long short-term memory recurrent neural networks," 2019.

Parameter Prioritization for Efficient Transmission of Neural Networks in Small Satellite Applications

Olga Kondrateva
Technical University of Darmstadt
olga.kondrateva@kom.tu-darmstadt.de

Stefan Dietzel
Merantix Momentum GmbH
stefan.dietzel@merantix.com

Ansgar Lößer, Björn Scheuermann
Technical University of Darmstadt
{ansgar.loesser, scheuermann}@kom.tu-darmstadt.de

***Abstract*—Low-earth-orbit (LEO) satellites can be used for cost-effective Earth observation missions. Onboard processing using machine learning (ML) approaches is often proposed to reduce the amount of data transmitted back to Earth. However, the combination of LEO satellites and ML brings unique communication challenges, as requirements – and therefore ML models – often change throughout the lifetime of a satellite mission. In this paper, we propose a novel communication protocol that deals with model updates efficiently by providing incremental updates with low communication overhead.**

I. Introduction

Small, low-Earth-orbit (LEO) satellites allow us to deploy satellite missions more quickly and cost effectively. In particular, the CubeSat standard became popular due to the availability of off-the-shelf components [1]. Often, the amount of data acquired by the LEO satellites is too large to transmit everything to Earth [2] during their short, unreliable communication windows. To intelligently filter the most relevant information, machine learning have gained rising attention in the satellite community. But their deployment raises new communication issues in the upstream direction: how can machine learning models be updated efficiently?

We propose a novel communication protocol, which allows for efficient incremental model updates. Due to its large number of parameters, the updated model's transmission likely requires use of multiple communication windows. Yet, the new model must be used as soon as possible in order to benefit from the updated accuracy or adapt to new classification tasks quickly. Therefore, our communication protocol prioritizes the most important model weights in transmission. We use a space-efficient data structure to convey priority classes to the satellite with low communication overhead. Once its most important weights are received, they can be used to construct an approximation of the updated model. The approximation is then used immediately, and it is improved incrementally until all updated model weights are available.

Evaluation results show that our approach considerably outperforms the baseline and performs similar to an ideal update protocol while incurring significantly less overhead.

Next, we introduce our approach in Section II and evaluate it in Section III. Section IV concludes the paper.

The full version of this paper has previously been published at the IEEE MedComNet 2023 conference: https://www.medcomnet.org.
This work has been funded by the Federal Ministry of Education and Research of Germany in the project "Open6GHub" (16KISK014).
This work received funding from the German Research Foundation (DFG), CRC 1404: *FONDA: Foundations of Workflows for Large-Scale Scientific Data Analysis.*
This work has been co-funded by the LOEWE initiative (Hesse, Germany) within the emergenCITY center.

II. Efficient Incremental Weight Transmission

Recall that a machine learning model mainly consists of a description of its structure and a number of weights that describe how these neurons fire. We consider the model structure to be known in advance, as standard structures are often used for common tasks. Therefore, each model update can be considered as transmitting a list of new model weights.

We design our protocol such that the newly updated model can be approximated quickly using all weights received up to a certain point in time with the remainder of the weights all set to zero. Two questions arise in this context: (1) What is the *right* order of model weights. (2) How can we communicate the prioritized order of model weights efficiently?

A. Weight order representation

Our approach depends on prioritizing key parameters. To determine importance, we draw on ideas from model pruning, where less crucial neural network components are removed. Various criteria, like the magnitude criterion [3], L1 and L2 norms [4], and gradient magnitude [5] have been used to measure importance. In our work, we utilize the absolute magnitude criterion, ranking parameters by their absolute value, and deeming those with lower values less important.

Next, we design a compact representation of the model weights' order, using a lossy permutation compression approach. To this end, we adopt *sorting subsequences,* a straightforward yet shown to be optimal compression scheme [6]. Rather than encoding the exact priority order for *each* model weight, we subdivide the list of weights into k *groups* of decreasing priority. This approximate grouping reduces the size of additional communication overhead to $n \log(k)$ bits.

B. Transmission protocol

Figure 1 shows our approach for a simplified model with six weights. As we assume the model structure to be known, we can represent a model by a flattened array of weights W.

Next, we perform a number of initialization steps on the ground station before transmission starts. We calculate a permutation P that prioritizes each model weight by its absolute value (Step 2). Then, we divide the array P into k groups of length m. First, the vector Q that maps each weight index to its priority group is transmitted to the satellite (Step 3), followed by all weights of the priority group 0. Within the priority group, the weight with the smallest index i is transmitted first, followed by the second-smallest, and so forth (Steps $4, \dots, n$). Therefore, the order within the group does not need to be communicated to the satellite but can be inferred from the index structure Q. When all weights of priority group 0 have been received, the process continues with priority group 1, and so forth until all weights have been

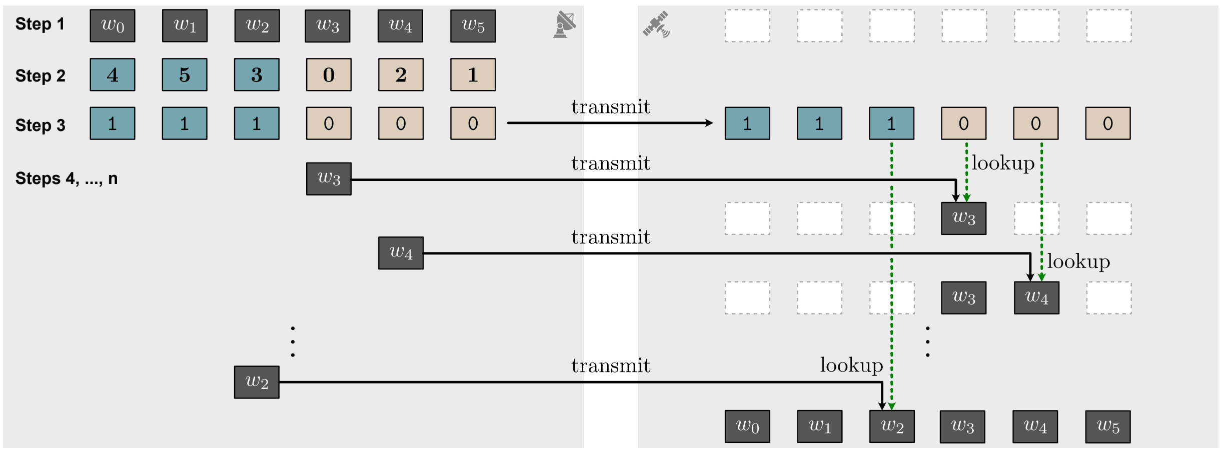


Fig. 1. Overview of the proposed communication scheme for an example model with six weights where $k = 2$ and $m = 3$.

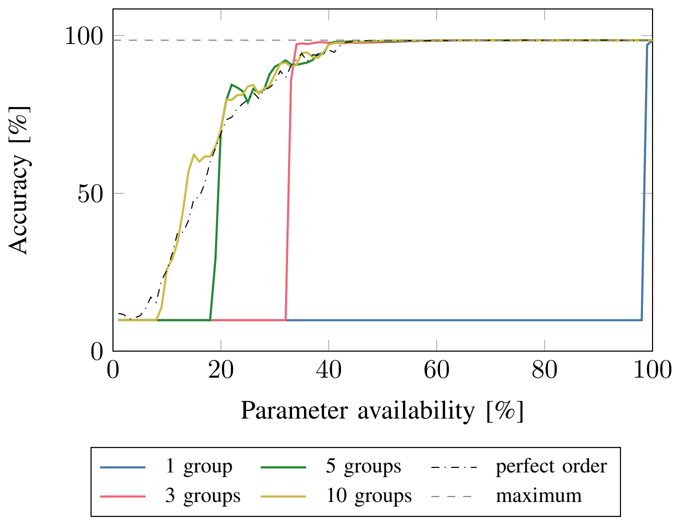


Fig. 2. Accuracy improvements depending on percentage of available model weights for MNIST trained on LeNet5.

received. In the example, the weights are transmitted in the order w_3, w_4, w_5 (group 0) followed by w_0, w_1, w_2 (group 1).

At the satellite, the updated model is used immediately by setting all unknown model weights to 0 as indicated by the dashed white boxes in Figure 1. Whenever more weights have been received, the model is incrementally updated on the satellite until the fully updated model is available.

III. Evaluation

To evaluate our approach, we use the LeNet5 [7] model trained on the MNIST [8] dataset. We compare our approach against a baseline and an optimal approach. The baseline is the naïve approach that puts all parameters into a single priority group, which achieves no prioritization. The optimal approach assumes that the weights are transmitted in the optimal order, i. e., according to their absolute value, highest first. In this case, the position of each weight in the model's structure needs to be communicated to reconstruct the model on the satellite. To isolate our mechanism's influence on accuracy, we evaluate it independent of communication effects, simply assuming that weights are available in a certain order.

Figure 2 shows our evaluation results. The x-axis shows the percentage of weights transmitted so far. All other weights are assumed to be 0. The y-axis shows the corresponding classification accuracy of the incrementally updated model. We compare different numbers of priority groups to assess how quickly they achieve good accuracies. It becomes clear that putting all parameters into one priority group does not allow for partial updates, since all parameters are required to achieve a meaningful accuracy level. It also can be seen that our approach allows to find a good tradeoff between the achieved accuracy and the amount of additional data that has to be transmitted. In addition, the results show that even the optimal parameter order gets outperformed, which indicates that the order of weights chosen (by absolute value) is not the only influence factor for model accuracy. In summary, the proposed approach considerably improves the accuracy when compared to the baseline and even the optimal approach.

IV. Conclusion

Performing efficient, incremental updates of machine learning models on satellites is a problem that has often been neglected. However, it is imperative to enable widespread use of LEO satellites despite changing classification requirements during satellite operation and short contact times with base stations. We have proposed a simple but effective mechanism to perform incremental model updates based on prioritizing weights into groups. Using this group-based ordering, we achieve significantly faster improvements in classification accuracy while keeping communication overhead to convey the prioritization order low.

References

[1] K. Woellert, P. Ehrenfreund, A. J. Ricco, and H. Hertzfeld, "Cubesats: Cost-effective science and technology platforms for emerging and developing nations," *Advances in Space Research*, vol. 47, no. 4, 2011.

[2] G. Furano, A. Tavoularis, and M. Rovatti, "Ai in space: applications examples and challenges," in *2020 IEEE International Symposium on Defect and Fault Tolerance in VLSI and Nanotechnology Systems (DFT)*, 2020.

[3] S. Han, J. Pool, J. Tran, and W. J. Dally, "Learning both weights and connections for efficient neural networks," *arXiv preprint arXiv:1506.02626*, 2015.

[4] H. Li, A. Kadav, I. Durdanovic, H. Samet, and H. P. Graf, "Pruning filters for efficient convnets," *arXiv:1608.08710*, 2017.

[5] N. Lee, T. Ajanthan, and P. H. S. Torr, "Snip: Single-shot network pruning based on connection sensitivity," *arXiv:1810.02340*, 2018.

[6] D. Wang, A. Mazumdar, and G. W. Wornell, "Compression in the space of permutations," *IEEE Trans. Inf. Theory*, vol. 61, no. 12, 2015.

[7] Y. Lecun, L. Bottou, Y. Bengio, and P. Haffner, "Gradient-based learning applied to document recognition," *Proc. IEEE*, vol. 86, no. 11, 1998.

[8] L. Deng, "The mnist database of handwritten digit images for machine learning research," *IEEE Signal Processing Magazine*, vol. 29, no. 6, 2012.

Bandwidth Prediction for Volatile Networks with Informer

Birkan Denizer
bde@informatik.uni-kiel.de
Kiel University
Kiel, Germany

Olaf Landsiedel
ol@informatik.uni-kiel.de
Kiel University
Kiel, Germany

Abstract

5G networks provide high throughput and low latency connections, crucial for remote monitoring and control of mission-critical operations. Managing buffer levels and accurate bandwidth estimations are essential for low-latency applications. However, wireless networks are susceptible to fluctuations in quality metrics due to mobility and interference, impacting link utilization. Sudden quality deterioration can lead to lower Quality-of-Experience (QoE). To address this, we propose a neural network-based bandwidth prediction system. Our system utilizes historical data for time-series forecasting using the Informer model. It achieves 10% lower errors on a publicly available LTE dataset and 51% lower errors on a publicly available 5G dataset. Future work includes multivariate predictions and the creation of a new 5G dataset.

Keywords: Bandwidth prediction, LTE, 5G, Informer

1 Introduction

5G networks enable high throughput and low latency connections. Remote monitoring and control of the vital infrastructure of mission-critical operations such as surgeries or production lines depend on 5G to provide a good Quality-of-Experience (QoE). To achieve low latency communication, such operations rely on accurately filling buffers to send data to minimize the latency of the end-to-end connection. Applications calculate estimations for the average bandwidth of a connection and generate packets. Low-latency applications keep buffer sizes to a minimum. Minimal buffer sizes result in higher dependence on accurate knowledge about future buffer states. However, applications experience low QoE when sudden quality deterioration happens. In this paper, we introduce a neural network-based approach for bandwidth prediction. It allows adjustment of application-specific parameters to increase QoE. Such a prediction system is not only vital for better link utilization but also for low latency.

Inherently, wireless networks are highly susceptible to fluctuations in quality metrics such as Signal-to-noise Ratio (SNR) or Received Signal Strength Indicator (RSSI). User mobility or sources of interference and blockage may cause variable bandwidth on the connection. This variance results in sub-optimal usage of the existing infrastructure as applications require time to adjust to newly attainable bandwidth.

Our performance metrics include data received from hardware and software sources such as SNR, RSSI, bandwidth, or round-trip time. Established approaches use historical data to generate the bandwidth predictions for a horizon. Time-series forecasting (TSF) is the process of analyzing and predicting future values based on historical data. Our approach builds on TSF as the main task to predict the bandwidth of a connection. In this domain, Long Short-term Memory (LSTM) and Transformer models gained popularity in extracting meaningful information [3] [5].

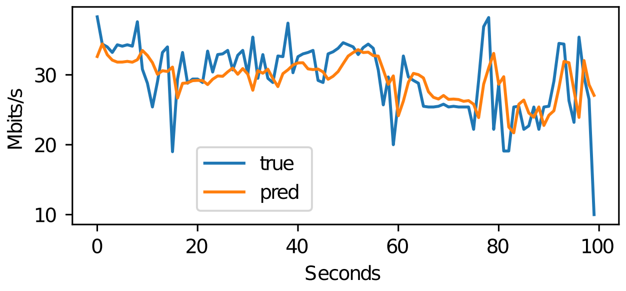


Figure 1. Bandwidth prediction on the LTE dataset

This paper makes the following contributions:

1. Design and development of a neural network-based approach that utilizes the Informer architecture to make bandwidth predictions based on historical metrics
2. Analysis of datasets and implementation of the MinMaxScaler for data normalization to a given range
3. Evaluation of our bandwidth prediction approach which shows up to 10% lower errors on the LTE dataset and up to 51% lower errors on the 5G dataset

2 Methodology

Transformer gained popularity since its introduction to Natural Language Processing (NLP) tasks [5]. One of the key challenges with Transformer-based approaches is their complexity in terms of time and memory during training.

Informer [6] is an efficient Transformer-based model for Long Sequence TSF tasks. It introduces the ProbSparse self-attention mechanism to reduce time and memory complexity during training, self-attention distilling to improve performance in the presence of extremely long input sequences, and a generative decoder to improve inference efficiency.

In this paper, we utilize the Informer architecture for bandwidth predictions. Namely, we use multivariate channel context information to predict several seconds of future bandwidth. Additionally, we implement the MinMaxScaler as our

Horizon	Metric	TPA-LSTM	Informer	Informer-M
1	RMSE	4.0038	4.0901	4.1281
1	MAE	2.9043	2.9786	3.0126
2	RMSE	4.6102	4.3954	4.4021
2	MAE	3.2362	3.1906	3.16
3	RMSE	5.0779	4.716	4.6033
3	MAE	3.5488	3.4356	3.2702

Table 1. Comparison on the LTE dataset: TPA-LSTM is better on horizon 1, Informer-M is better on Horizon 2&3

analysis of the datasets shows features with different ranges. The MinMaxScaler provides normalization of the data to a given range, often between zero and one. It scales each feature separately to the given range. Separate scaling prevents features with larger ranges from affecting the bias of scaling.

3 Datasets

We evaluate our approach on public LTE [3] and 5G [4] datasets. The LTE dataset contains several transportation scenarios like bus and subway lines. The LTE dataset contains bandwidth, LTE-neighbors, RSSI, Reference Signal Received Quality (RSRQ), change in ENodeB compared to the previous second, time advance to the next ENodeB, speed, and band. We choose "downloading" and "video streaming" use cases while driving from the 5G dataset. The 5G dataset includes metrics such as Reference Signal Received Power (RSRP, RSRQ), Channel Quality Indicator (CQI), SNR, RSSI, and download and upload bandwidths.

Traces include regularly sampled data. The 5G dataset has missing data points. Therefore, we use the forward-fill imputation method to remedy missing points.

4 Evaluation

In this work, we use the Mean Absolute Error (MAE) and the Root Mean Squared Error (RMSE) to evaluate prediction quality. We implement the MinMax Scaler and test its performance with the Informer (noted as Informer-M) on prediction horizons ranging from 1 to 24. We use bus line 15 and train line 7 of the LTE dataset, and "downloading while driving" as the mobility scenario in the 5G dataset.

Azarid et al. [1] compare Autoregressive Integrated Moving Average (ARIMA) and LSTM models and find that LSTM outperforms ARIMA for predictions. Mei et al. [2] show that the Temporal Pattern Attention LSTM (TPA-LSTM) model outperforms Recursive Least Squares (RLS), Random Forest (RF), and LSTM. We select TPA-LSTM as one of our baselines.

As Table 1 shows, TPA-LSTM performs better for horizon length 1 on the LTE dataset. Also in Table 1, Informer performs between 1-8 % lower on MAE, and 5 to 10 % lower on RMSE for horizon lengths 2 and 3.

On the 5G dataset, our solution with imputation (noted as Informer-I) achieves a 45 to 48% lower RMSE and 49 to 51% lower MAE compared to prior Informer-based solutions on different horizons, see Table 2.

Horizon	Metric	Informer + Lasso	Informer-I
1	RMSE	0.72	0.3801
1	MAE	0.35	0.1764
6	RMSE	1.19	0.6151
6	MAE	0.63	0.2994
24	RMSE	1.33	0.7318
24	MAE	0.73	0.3613

Table 2. Comparison on the 5G dataset: Informer-I performs overall better compared to prior Informer-based solution

5 Conclusion

This work introduces a neural network-based design that applies a state-of-the-art Transformer-based Informer model to predict bandwidth. Our bandwidth prediction system aids in helping users to have better QoE. In this work, we apply our solution to various datasets. Our results show up to 10% lower errors on the LTE dataset, and up to 51% lower errors on the 5G dataset compared to state-of-the-art approaches.

For future work, we plan to make multivariate predictions based on multivariate data. We also plan to generate a new regularly-sampled 5G dataset for new mobility scenarios.

6 Acknowledgements

This research has been partially funded by the German Ministry of Transport and Digital Infrastructure within the projects CAPTN Förde Areal (45DTWV007B) and CAPTN Förde Areal II (45DTWV08D).

References

[1] Amin Azari, Panagiotis Papapetrou, Stojan Denic, and Gunnar Peters. 2019. Cellular traffic prediction and classification: A comparative evaluation of LSTM and ARIMA. In *Discovery Science: 22nd International Conference, DS 2019, Split, Croatia, October 28–30, 2019, Proceedings 22.* Springer, 129–144.

[2] Lifan Mei, Jinrui Gou, Yujin Cai, Houwei Cao, and Yong Liu. 2022. Realtime mobile bandwidth and handoff predictions in 4G/5G networks. *Computer Networks* 204 (2022), 108736.

[3] Lifan Mei, Runchen Hu, Houwei Cao, Yong Liu, Zifa Han, Feng Li, and Jin Li. 2019. Realtime mobile bandwidth prediction using lstm neural network. In *Passive and Active Measurement: 20th International Conference, PAM 2019, Puerto Varas, Chile, March 27–29, 2019, Proceedings 20.* Springer, 34–47.

[4] Darijo Raca, Dylan Leahy, Cormac J Sreenan, and Jason J Quinlan. 2020. Beyond throughput, the next generation: a 5G dataset with channel and context metrics. In *Proceedings of the 11th ACM multimedia systems conference.* 303–308.

[5] Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N Gomez, Łukasz Kaiser, and Illia Polosukhin. 2017. Attention is all you need. *Advances in neural information processing systems* 30 (2017).

[6] Haoyi Zhou, Shanghang Zhang, Jieqi Peng, Shuai Zhang, Jianxin Li, Hui Xiong, and Wancai Zhang. 2021. Informer: Beyond efficient transformer for long sequence time-series forecasting. In *Proceedings of the AAAI conference on artificial intelligence*, Vol. 35. 11106–11115.

Steps Toward a Supervised Machine Learning Scheduler for MPTCP

Reza Poorzare, Hadi Asghari, and Oliver P. Waldhorst
Data-Centric Software Systems Research Group at the Institute of Applied Research,
Karlsruhe University of Applied Science, Karlsruhe, Germany
reza.poorzare@h-ka.de, asha1011@h-ka.de, and oliver.waldhorst@h-ka.de

Abstract— **The functionality of the MPTCP scheduler is a hurdle in the way of the protocol in achieving high performance. This drawback is even more severe in heterogeneous networks, where the differences in the characteristics of the paths impair the functionality of the scheduler drastically. In this paper, we introduce a dataset generated by an emulation environment, including diverse scenarios and traffic types, as an initial step toward having a supervised learning scheduler.**

Keywords—Transport protocols, Machine Learning, MPTCP, Proxy, OpenStack

I. Introduction

MPTCP (Multipath TCP), which is an extension of conventional TCP (Transmission Control Protocol), was developed to allow having more than one network path in the very same connection. This protocol can indeed provide some benefits relying on its scheduling mechanisms and simultaneous distribution of traffic into different paths. However, it cannot achieve full bandwidth aggregation, due to flaws such as out-of-order packets, frequent re-ordering processes, or HoL (Head of Line) blocking that waste time and energy. Most schedulers exploit single-criterion or multi-criteria approaches for traffic management. In the former, a parameter such as RTT (Round Trip Time) is used, while in the latter, more than one parameter is considered to select the path for the transmission. These reactive approaches lack the ability to properly distribute traffic to prevent the above-mentioned issues and are easily confused by random packet losses or other shortcomings that may occur in the network [1]. Therefore, to overcome these issues and make the best use of the available bandwidth aggregation, it is necessary to design schedulers based on machine learning techniques that can not only detect the current state of the subflows, i.e., paths, but also predict upcoming situations to enhance the functionality of MPTCP.

By considering these facts, the main questions in this ongoing work are: *(i)* How to create an emulation environment that can reflect various real-time networking circumstances? *(ii)* How to have a centralized node in the topology that has insight into the whole traffic to generalize the data set? *(iii)* Based on the generated data set, how can appropriate features be selected to be used in supervised learning techniques such as deep neural networks?

II. Related work

Several approaches have been proposed to deal with the inefficient functionality of the MPTCP scheduler. The initial steps approached existing problems, such as out-of-order delivery, reactively, leading to schedulers like BLEST (BLock ESTimation) [2] and ECF (Earliest Completion First) [3]. BLEST estimates the blocking time for different subflow selections and traffic distributions in a way that alleviates out-of-order and HoL blocking issues. With some similarities to BLEST, ECF attempts to find the path with the minimum transmission delay by exploiting parameters such as RTT and cwnd (congestion window).

There have been some other proposals, but most of them suffer from having static and non-intelligent methods. Thus, they are not able to fully utilize the available resources. A reinforcement learning scheduler called MPTCP-RL has been proposed recently in [4] to find the best optimal path and mitigate packet loss and network heterogeneity adverse impacts. This scheduler tries to create a table containing scheduling rules for subflow selection, and by relying on the rules, it could enhance the network's throughput. However, this approach can waste some time in the decision-making phase, since it should update itself frequently. Moreover, it cannot be generalized easily for different scenarios. As a result, there is a need for supervised learning techniques so that the training and decision-making parts can be separated. In this case, a machine learning engine that is frequently updated offline resides in the scheduling component. This mechanism can dramatically reduce the time spent on the decision-making process. However, to the best of our knowledge, there is no public dataset for this, so the first steps should be taken toward its creation.

III. MPTCP Proxy Deployment for Data Set Creation

The main problem in a supervised learning scheduler establishment is the lack of a stereotyped data set. A data set should be a reflection of diverse real-world scenarios, so it can generalize to most of the existing situations. As a result, we have divided the existing scenarios into three different categories, including short- medium- and long connections. For the first scenario, web page loading, for the second one, video streaming, and for the third one, test file download were the representatives. After identifying the problems and counterpart representatives for the scenarios, an approach for data gathering should be selected. As a result, we decided to use an intermediate node as a MPTCP proxy so that all traffic between clients and public servers goes through it and it can monitor all the traffic.

This approach can bring some advantages, including: *(i)* Public servers do not need to support MPTCP as the connections between the clients and proxy will be MPTCP ones. *(ii)* As the whole traffic is passed through a centralized node, the creation of a data set by using it can be a reflection of the network. On the proxy server, we have used microsock5, which is a lightweight proxy that handles traffic redirection without heavy use of resources.

IV. Emulation Setup and Methodology

The laboratory experiment conducted in this study utilized the OpenStack platform as the virtualization infrastructure, including thirty Linux clients, as shown in Figure 1. Moreover, a Python automation script was employed for streamlined deployment. By using a Linux traffic shaper tool on individual subflows, characteristics such as packet loss probability, latency, or bandwidth, for 5G (Fifth Generation) and Wi-Fi 6 networks were emulated. The experiment encompassed various scenarios, including data download, video streaming, and webpage loading, to imitate real-world requirements and have different traffic types.

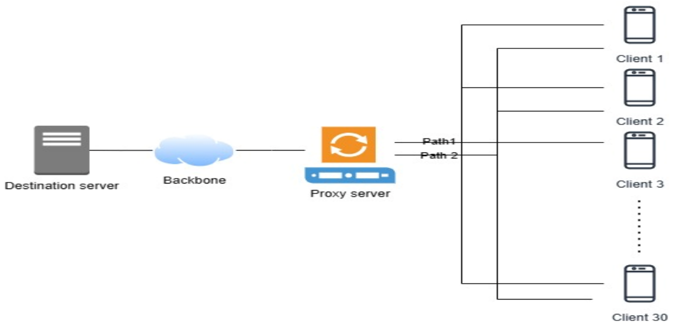


Figure 1 The emulation environment in OpenStack

Table I Selected parameters to create the data set

Parameter	Description
srtt_us	Round trip time for subflows
basertt	The minimum seen RTT of individual subflows
snd_cwnd	Sending congestion window
max_window	The maximal window ever seen from a subflow
bytes_acked	How many bytes were acked
bytes_sent	Total number of data bytes sent
prior_cwnd	The cwnd right before loss recovery
lost	Total data packets lost, including retransmissions

Once we have the emulation environment, we should choose the parameters carefully. This process should follow two main goals: 1. The parameters should be able to help the trained scheduler distinguish different states in the network 2. They should provide the feasibility of differentiating various circumstances in the network, such as shadowing or fading. With these goals in mind, the parameters in Table I were extracted from the tcp.h file. However, parameters can be added or deleted as needed. In the feature selection phase, some important criteria should be considered, including 1. When the network state is changed, it should be reflected. 2. Important states and conditions such as peak data rates, low latencies, recovery times, traffic load portion, and reasons for the packet loss should be distinguishable. In the next steps, a selection of variables can form the inputs of a supervised learning technique, such as a deep neural network, to train the engine. These variables should be affected by changes in the network to reflect the state of the network, and preferably reside between zero and one to avoid normalization. Some selective examples are given in Table II. For the final step and labeling of the outputs, random hashing is used to weigh the subflows and find out which weights appropriately reflect the states based on fair use of the bandwidths and user experience measurements such as packet loss and data rate.

Table II Selective inputs to feed the deep neural network

Variable	Goal
basertt/srtt_us	Traffic load detection
snd_cwnd /max_window	Determine the aggressiveness of the sending rate adjustment
bytes_acked /bytes_sent	Estimation of the BDP (Bandwidth-Delay Product)
prior_cwnd /max_window	Having a faster recovery
lost/bytes_sent	Distinguishing random losses

V. Conclusion and Future Work

The MPTCP scheduler has some drawbacks in selecting the best possible subflow because of its static and non-intelligent mechanism. As a result, in this work, we took the first steps toward having a supervised machine learning-based scheduler. We have established an emulation environment reflecting different network conditions, and then, by using a MPTCP proxy node, a stereotyped data set was created that can be used in supervised learning approaches. In future work, we will feed this data set to a deep neural network to conceive of an intelligent scheduler that can function properly in various circumstances.

Acknowledgment

This work was supported by the bwNET2020+ project which is funded by the Ministry of Science, Research and the Arts Baden-Württemberg (MWK). The authors alone are responsible for the content of this paper.

References

[1] R. Poorzare and O. P. Waldhorst, "Toward the Implementation of MPTCP Over mmWave 5G and Beyond: Analysis, Challenges, and Solutions," *IEEE Access*, vol. 11, pp. 19534-19566, Feb. 2023, DOI: 10.1109/ACCESS.2023.3248953.

[2] S. Ferlin, Ö. Alay, O. Mehani, and R. Boreli, "BLEST: Blocking estimation-based MPTCP scheduler for heterogeneous networks," *2016 IFIP Networking Conference (IFIP Networking) and Workshops*, 2016, pp. 431-439, DOI: 10.1109/IFIPNetworking.2016.7497206.

[3] Y.-S. Lim, E. M. Nahum, D. Towley, and R. J. Gibbens, "ECF: An MPTCP path scheduler to manage heterogeneous paths," *in Proc. 13th Int. Conf. Emerging Networking Experiments and Technologies (CoNEXT)*, Incheon, Republic of Korea, Dec. 2017, pp. 147-159. DOI: 10.1145/3143361.3143376.

[4] P. Dong *et al.*, "Multipath TCP Meets Reinforcement Learning: A Novel Energy-Efficient Scheduling Approach in Heterogeneous Wireless Networks," *IEEE Wireless Communications*, vol. 30, no. 2, pp. 138-146, April 2023, doi: 10.1109/MWC.013.2100658.

In-Network Round-Trip Time Estimation for TCP Flows

Daniel Stolpmann, Andreas Timm-Giel
Hamburg University of Technology, Institute of Communication Networks, Germany
{daniel.stolpmann, timm-giel}@tuhh.de

***Abstract*—Transmission Control Protocol (TCP)'s tendency to fill up buffers in the network results in long standing queues. Active Queue Management (AQM) tries to solve this issue but typically requires manual tuning as the optimal parameters depend on the Round-Trip Time (RTT) of the flow, which is unknown to the AQM. In this paper, we use network simulation and supervised learning to train a neural network to infer the RTT of a TCP flow from its queuing behavior. We transfer our model into a real-time network emulator and show that it is able to estimate the RTT with an error of only a few milliseconds.**

***Index Terms*—Active Queue Management, Machine Learning, Dataset Generation, Supervised Learning, Network Emulation**

I. Introduction

Transmission Control Protocol (TCP)'s tendency to fill up the buffer in front of the bottleneck link often results in a long standing queue also known as bufferbloat. Active Queue Management (AQM) tries to solve this issue by prematurely dropping or, if combined with Explicit Congestion Notification (ECN), marking packets to signal congestion to the Congestion Control (CC) algorithm at the sender. However, a common problem of AQM algorithms is the necessity to tune their parameters to the operating conditions. In [1], it was shown that the optimal buffer size for a TCP flow that results in full link utilization and minimal queuing delay depends on its base Round-Trip Time (RTT) (without queuing), which is unknown at the bottleneck link.

In this paper, we present a Machine Learning (ML) model that infers the base RTT of a TCP flow from its queuing behavior. Our approach is based on an efficient way to generate training data using a high-level network simulation, which is used to train a neural network using supervised learning. Finally, we transfer the model into a real-time network emulator to evaluate its performance on real traffic.

II. Related Work

In [2], the RTT of a TCP flow was estimated based on the timestamps in the TCP header using ML. In [3], the size of the buffer at the bottleneck link was adapted for a TCP flow depending on its RTT and the used CC scheme to achieve high throughput and low delay based on queue statistics using Deep Reinforcement Learning (DRL).

III. System Model

Our training data is generated by a network simulation implemented using Python and SimPy[1]. The simulation models a general communication network with a packet sender and a receiver that are connected via a full-duplex connection. The sender performs CC with a slow-start and a congestion avoidance phase as in TCP. In each direction, a configurable delay of half the base RTT is added to the packets. In the direction from the sender to the receiver, there is a packet queue. Packets are dequeued at a configurable time interval to simulate a link with a fixed capacity. Whenever the queue exceeds the buffer size or the queuing delay exceeds a controllable threshold, packets are marked as in ECN. The marking of a packet is signaled back from the receiver to the sender, which sees it as a congestion signal and reduces the Congestion Window (CWND) accordingly. The model is depicted in Figure 1.

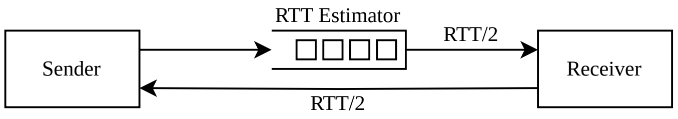


Fig. 1. Illustration of the system model

At the queue, the link capacity, the link utilization, the queuing delay and the queue length are measured every $10\,\mathrm{ms}$. To obtain the link capacity and the throughput, the number of transmission opportunities and transmitted packets are counted over the same interval. To get the queuing delay, it is calculated as the queue length divided by the link capacity [4] instead of timestamping each packet. As input for the estimator, a history of the values at the last 200 time steps is used. Additionally, the differentials $\Delta x_t = x_t - x_{t-1}$ are calculated for every value and also included in the input. To make the trained model robust against real-world imperfections, jitter is added to the measuring interval as well as to the packet pacing at the sender.

The estimator neural network is created and trained using Keras[2]. It has an input layer with 1600 neurons followed by two fully connected hidden layers with 256 neurons each using the Rectified Linear Unit (ReLU) activation function. Batch normalization is applied at the input, while layer normalization is performed after each hidden layer. Finally, the network condenses to a single neuron with linear activation in the output layer to estimate the base RTT.

[1]https://simpy.readthedocs.io (Accessed: 03.07.2023)

[2]https://keras.io (Accessed: 03.07.2023)

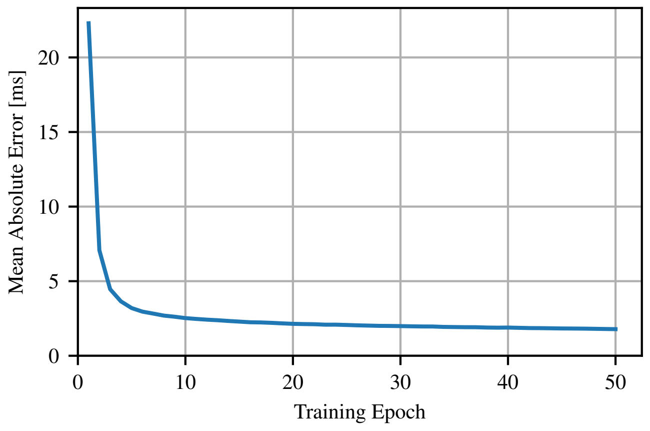


Fig. 2. Error in the base RTT estimation over the training epochs

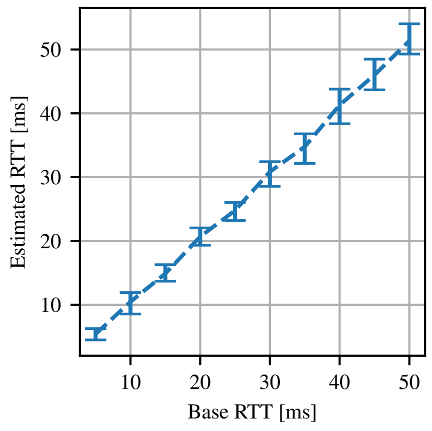


Fig. 3. Estimated RTT over the configured base RTT in the emulator

IV. Training

A. Dataset Generation

Multiple simulations are executed in parallel using the Ray framework[3] to create the dataset. Each training batch contains the data from one step in each simulation, resulting in a batch size of 128 with an equal number of simulations. The simulations are executed in episodes of 30 s. In the beginning, each simulation is executed for a random time of up to the length of the first episode where no samples are collected to desynchronize the simulations. At the start of each episode, a random base RTT between 1 and 60 ms and a queuing delay threshold between 0 and 100 ms is chosen. During the episode, the link capacity is resampled every 1 to 10 s to be between 100 and 2500 packets/s. Varying these three parameters creates a diverse training dataset for different base RTTs, link capacities and buffer sizes. The dataset contains 10 000 batches and thus a total of 1 280 000 training samples.

B. Supervised Learning

The neural network is trained supervised using randomly sampled training batches from the dataset. The queue statistics are used as the input and the base RTT as the label. The Mean Squared Error (MSE) is used as the loss function. The length of each training epoch corresponds to the size of the dataset and the network is trained for a total of 50 epochs using the Adam optimizer with a learning rate of 10^{-3}.

The estimation error over the training epochs is shown in Figure 2. It can be seen that the Mean Absolute Error (MAE) starts at around 23 ms, but goes down quickly and reaches a value of around 2 ms at the end of the training.

V. Evaluation

To evaluate the performance of the RTT estimator with a real TCP flow, it is implemented as a module in the FlowEmu network emulator [5]. The module acts as a packet queue and uses the TensorFlow C++ Application Programming Interface (API)[4] to load the trained neural network. The output of the neural network is filtered using an Exponentially Weighted Moving Average (EWMA) with a weight of 0.1 for the new sample. The emulated network corresponds to the one described in Section III, but iPerf[5] with TCP New Reno is used as the sender and the receiver. The buffer size is limited to 80 packets and the link capacity is set to 2000 packets/s.

The estimated RTT over the configured base RTT is shown in Figure 3. The plot shows the mean value over an experiment duration of 60 s, where the first 5 s are omitted as warm-up period. The bars mark the 5% and 95% percentiles, respectively. It can be seen that the estimated RTT is close to the configured base RTT with an error of only a few milliseconds.

VI. Conclusion

We trained a neural network on simulation data to estimate the base RTT of a TCP flow at the bottleneck link and showed its accuracy in a real-time network emulator. The estimated RTT can be used in future work to tune the parameters of classic AQM algorithms or as input for novel DRL approaches.

References

[1] G. Appenzeller, I. Keslassy, and N. McKeown, "Sizing Router Buffers," *ACM SIGCOMM Computer Communication Review*, vol. 34, no. 4, pp. 281–292, Oct. 2004.

[2] D. H. Hagos, P. E. Engelstad, A. Yazidi, and C. Griwodz, "A Deep Learning Approach to Dynamic Passive RTT Prediction Model for TCP," in *2019 IEEE 38th International Performance Computing and Communications Conference (IPCCC)*, London, England, UK, Oct. 2019.

[3] M. Bachl, J. Fabini, and T. Zseby, "LFQ: Online Learning of Per-flow Queuing Policies using Deep Reinforcement Learning," in *2020 IEEE 45th Conference on Local Computer Networks (LCN)*, Sydney, NSW, Australia, Nov. 2020, pp. 417–420.

[4] P. Goyal, A. Agarwal, R. Netravali, M. Alizadeh, and H. Balakrishnan, "ABC: A Simple Explicit Congestion Controller for Wireless Networks," in *17th USENIX Symposium on Networked Systems Design and Implementation (NSDI '20)*, Santa Clara, CA, USA, Feb. 2020, pp. 353–372.

[5] D. Stolpmann and A. Timm-Giel, "FlowEmu: An Open-Source Flow-Based Network Emulator," *Electronic Communications of the EASST*, vol. 80: Conference on Networked Systems 2021 (NetSys 2021), Sep. 2021.

[3]https://www.ray.io (Accessed: 03.07.2023)

[4]https://www.tensorflow.org/api_docs/cc (Accessed: 03.07.2023)

[5]https://iperf.fr (Accessed: 03.07.2023)

Reducing Memory Footprints in Purity Estimations of Volumetric DDoS Traffic Aggregates

Hauke Heseding[†], Timon Krack[*], Martina Zitterbart[†]
[†]*Institute of Telematics,* [†]*KASTEL Security Research Labs*
Karlsruhe Institute of Technology (KIT), Karlsruhe, Germany
hauke.heseding@kit.edu, timon.krack@student.kit.edu, zitterbart@kit.edu

***Abstract*—Distinguishing between attack and legitimate traffic in volumetric DDoS scenarios is challenging. Hierarchical Heavy Hitter algorithms can efficiently monitor high-volume traffic aggregates, but provide no insight into traffic composition. Monitoring complementary traffic features enables classification of traffic aggregates with machine learning, but increases the memory footprint of Hierarchical Heavy Hitter algorithms. Since the performance of these algorithms depends on the efficiency of memory usage, we evaluate feature importance to find a compact feature set for accurate distinction of legitimate and attack traffic.**

***Index Terms*—Distributed denial of service, hierarchical heavy hitters, machine learning, feature importance**

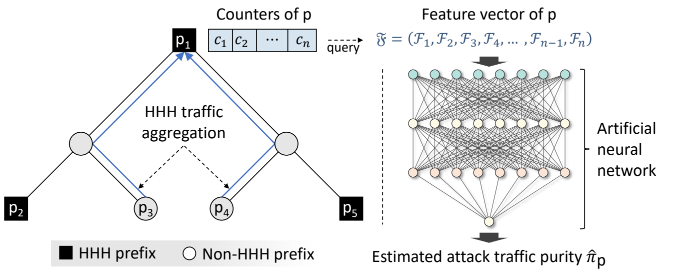

Counters of p
c_1 c_2 ⋯ c_n
query
Feature vector of p
$\mathfrak{F} = (\mathcal{F}_1, \mathcal{F}_2, \mathcal{F}_3, \mathcal{F}_4, \dots, \mathcal{F}_{n-1}, \mathcal{F}_n)$
p_1
HHH traffic aggregation
p_2
p_3
p_4
p_5
Artificial neural network
■ HHH prefix
○ Non-HHH prefix
Estimated attack traffic purity $\widehat{\pi}_p$

Fig. 1: ML-based regression to estimate attack traffic purity of aggregates.

I. Introduction

Hierarchical Heavy Hitter (HHH) [1] algorithms can process high-volume traffic in volumetric DDoS attack scenarios at line speed (e.g., [2], [3]). These algorithms monitor traffic volume distribution, but not traffic composition. To make HHH algorithms useful for attack traffic removal, additional information is required to distinguish between attack and legitimate traffic. By monitoring complementary features (besides traffic volume), machine learning (ML) can be used to estimate the purity of attack traffic in a traffic aggregate and to blacklist subnets that primarily send attack traffic. However, using complementary features increases the memory footprint of HHH algorithms, which impedes monitoring efficiency.

To reduce the number of required features, we evaluate the impact of individual features on attack traffic purity estimations. For this, we utilize permutation feature importance [4] to measure the increase in the absolute estimation error after shuffling the values of a single feature randomly. Our results based on authentic MAWI traffic and synthesized attack patterns indicate that only few features are required to achieve high accuracy in attack traffic purity estimations.

II. Attack Traffic Purity Estimation

HHH algorithms perform traffic volume aggregation to detect high-volume IP prefixes, i.e., prefixes whose traffic comprises a certain fraction ϕ of the total traffic (not including the volume of longer high-volume prefixes). Estimating the attack traffic purity of such high-volume prefixes enables blacklisting of highly malicious IP source subnets to protect network infrastructures from attack traffic.

[*]Research performed while student at KIT.

Specifically, we denote the *attack traffic purity* of a prefix p as the ratio $\pi_\mathsf{p} = A_\mathsf{p}/V_\mathsf{p}$ of attack traffic volume A_p to total traffic volume V_p. To obtain an estimate $\widehat{\pi}_\mathsf{p}$ of π_p, ML-based regression can be applied to complementary features as depicted in Fig. 1. To obtain these features, we modify an HHH algorithm to count how often packets in the source address range of a high-volume prefix exhibit certain characteristics. For example, the counter c_1 of prefix p_1 can count the number of times the TCP protocol occurred in its address range. By continually monitoring different traffic characteristics, the HHH algorithm provides insight into aggregate composition.

The counters can be queried at regular intervals to obtain a feature vector $\mathfrak{F}$ with features $\mathcal{F}_1, \mathcal{F}_2, \dots$. Given a set of feature vectors and target values of π_p, an artificial neural network can be trained to estimate attack traffic purity. For this, we use a straightforward model architecture (implemented in TensorFlow). The model applies z-score normalization (fitted on training data) in the first layer followed by an alternating sequence of eight fully-connected layers (128 neurons and ReLU activation) and eight dropout layers that reduce the risk of over-fitting. A final layer outputs the attack traffic purity estimation $\widehat{\pi}_\mathsf{p}$ using a single neuron with a linear activation function. The model is trained on dynamic traffic patterns described in Sec. IV that model multiple volumetric DDoS attack vectors overlaid on authentic, legitimate traffic.

III. Feature Importance

To assess the impact of individual features on model performance, we use permutation feature importance on the features and vectors in Tab. I. We then shorten the vectors to reduce memory footprints and retrain the models for comparison.

TABLE I: Features and Feature Vectors

Feature	Meaning
$\mathcal{F}_{\text{PKT}}$	Aggregate packet count
$\mathcal{F}_{\text{VOL}}$	Aggregate traffic volume in bytes
$\mathcal{F}_{\text{TCP}}$, $\mathcal{F}_{\text{UDP}}$	#occurrences of TCP and UDP protocols
$\mathcal{F}_{\text{PORTS}\in[X,Y]}$	#occurrences of source ports in the range $[X, Y]$
$\mathcal{F}_{\text{SIZE}\in[X,Y]}$	#occurrences of frame sizes in the range $[X, Y]$
Feature vectors	**Included features**
$\mathfrak{F}^{\text{PROTO}}$	$\{\mathcal{F}_{\text{TCP}}, \mathcal{F}_{\text{UDP}}\}$
$\mathfrak{F}^{\text{PROTO+SIZE}}$	$\mathfrak{F}^{\text{PROTO}} \cup \left\{\mathcal{F}_{\text{SIZE}\in[0,199]}, \mathcal{F}_{\text{SIZE}\in[200,1999]}\right\}$
$\mathfrak{F}^{\text{PROTO+PORTS}}$	$\mathfrak{F}^{\text{PROTO}} \cup \left\{\mathcal{F}_{\text{PORTS}\in[0,2^{10}-1]}, \mathcal{F}_{\text{PORTS}\in[2^{10},2^{16}-1]}\right\}$
$\mathfrak{F}^{\text{ALL}}$	$\mathfrak{F}^{\text{PROTO}} \cup \{\mathcal{F}_{\text{PKT}}, \mathcal{F}_{\text{VOL}}\}$ $\cup \left\{\mathcal{F}_{\text{PORTS}\in[i\cdot 2^{10},(i+1)\cdot 2^{10}-1]} \mid i \in [0, 63]\right\}$ $\cup \left\{\mathcal{F}_{\text{SIZE}\in[i\cdot 200,(i+1)\cdot 200-1]} \mid i \in [0, 9]\right\}$

Initially, the vector $\mathfrak{F}^{\text{all}}$ uses all features and serves as a baseline. To find its important features, we perform two-steps:

Feature importance assessment. First, for a given feature vector, we calculate the cumulative distribution function (CDF) of the absolute errors $|\pi_{\text{p}} - \widehat{\pi}_{\text{p}}|$ of a trained model. We then calculate the area under the curve (AUC) of the CDF. A higher AUC implies better estimations. This can be used to assess permutation feature importance. For this, we shuffle each feature in a test dataset individually and perform attack traffic purity estimations with a trained model. A reduction in the AUC indicates the importance of a feature to the model.

Feature reduction. Second, we eliminate or combine features of $\mathfrak{F}^{\text{ALL}}$ that yield low AUC reductions (e.g., by combining short port ranges). This yields the vectors $\mathfrak{F}^{\text{PROTO+SIZE}}$, $\mathfrak{F}^{\text{PROTO+PORTS}}$, and $\mathfrak{F}^{\text{PROTO}}$ from Tab. I that focus on protocol, frame size, and/or port information. These vectors are significantly shorter to reduce the number of counters required by the HHH algorithm. By measuring the AUC after re-training on shorter vectors, we identify relevant complementary features.

IV. Evaluation

We train the model on a synthesized dataset with authentic, legitimate MAWI traffic [5] and synthesized attack patterns with randomized time-dynamic behavior:

- UDP-based DNS, NTP and OpenVPN amplification attacks with frame size distributions reported in [6].
- TCP-based flood attacks with random frame sizes and ports in the range $60 - 1492$ and $49152 - 65535$ (resp.).

The DNS and NTP attacks use high-volume sources, while OpenVPN and TCP attacks use wide-spread, low-volume sources to generate different aggregates with varying traffic volume. Randomizing sources and attack traffic characteristics renders the estimation of π_{p} challenging.

The feature importance of the most relevant features for each feature vector is shown in Fig. 2. Protocol information (particularly TCP) has the highest impact on model performance. Based in the feature importance of $\mathfrak{F}^{\text{ALL}}$ we determine shorter feature vectors. Notably, the short frame size and port ranges of $\mathfrak{F}^{\text{ALL}}$ have low individual impact. Therefore, we combine them into larger ranges to shorten vector length.

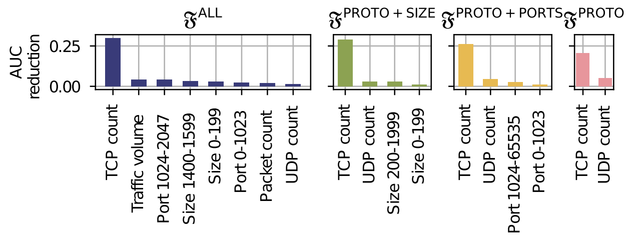


Fig. 2: Feature importance ranked by AUC reduction for all feature vectors.

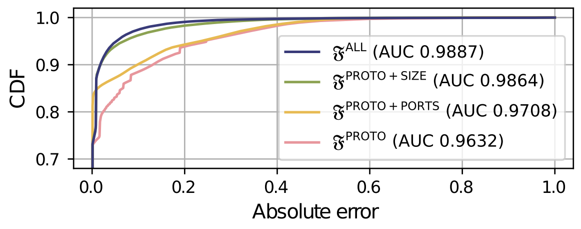


Fig. 3: Cumulative distribution function of the absolute errors in estimations of the attack traffic purity π_{p} when using different feature vectors.

Since there is no clear preference for port over frame size information, we use the complementary vectors $\mathfrak{F}^{\text{PROTO+PORTS}}$ and $\mathfrak{F}^{\text{PROTO+SIZE}}$ for comparison and additional size reduction.

Fig. 3 summarizes the CDF of absolute errors for different feature vectors. The vector $\mathfrak{F}^{\text{ALL}}$ achieves the highest estimation performance with an AUC of 0.9887. In comparison, protocol information alone results in a significant AUC reduction (0.9632 using $\mathfrak{F}^{\text{PROTO}}$). Including complementary port information in $\mathfrak{F}^{\text{PROTO+PORTS}}$ increases the AUC slightly to 0.9708. However, including frame size information instead (in $\mathfrak{F}^{\text{PROTO+SIZE}}$) retains an estimation performance close to the full feature vector $\mathfrak{F}^{\text{ALL}}$ (AUC = 0.9864). This provides a significant feature size reduction from 76 features ($\mathfrak{F}^{\text{ALL}}$) down to 4 features ($\mathfrak{F}^{\text{PROTO+SIZE}}$) with low impact on estimation performance. Through this, the memory footprint of an HHH algorithm used for blacklisting can be reduced by $94.7\,\%$.

Acknowledgment

This work was supported by funding of the Helmholtz Association (HGF) through the KASTEL Security Research Labs (KASTEL) (POF structure 46.23.01: Methods for Engineering Secure Systems).

References

[1] G. Cormode, F. Korn, S. Muthukrishnan, and D. Srivastava, "Finding hierarchical heavy hitters in data streams," in *Proceedings of the 29th International Conference on Very Large Data Bases*, 2003.

[2] D. A. Popescu, G. Antichi, and A. W. Moore, "Enabling fast hierarchical heavy hitter detection using programmable data planes," in *Proceedings of the Symposium on SDN Research*, 2017.

[3] R. Ben Basat, X. Chen, G. Einziger, and O. Rottenstreich, "Designing heavy-hitter detection algorithms for programmable switches," *IEEE/ACM Transactions on Networking*, 2020.

[4] L. Breiman, "Random forests," *Machine learning*, 2001.

[5] R. Fontugne, P. Borgnat, P. Abry, and K. Fukuda, "MAWILab: Combining Diverse Anomaly Detectors for Automated Anomaly Labeling and Performance Benchmarking," in *ACM CoNEXT '10*, 2010.

[6] D. Kopp, C. Dietzel, and O. Hohlfeld, "Ddos never dies? An IXP perspective on ddos amplification attacks," in *Passive and Active Measurement*, 2021.

Impact of Adaptive Packet Sampling on ML-based DDoS Detection

Samuel Kopmann* and Martina Zitterbart*†
*Institute of Telematics, Karlsruhe Institute of Technology, Karlsruhe, Germany
†KASTEL - Security Research Labs, Karlsruhe, Germany
{samuel.kopmann, martina.zitterbart}@kit.edu

Abstract—**Traffic monitoring can react to changing data rates by adapting the fraction of inspected packets (sampling rate). In this work, we investigate the resilience of a sampling rate agnostic machine-learning DDoS detector against a packet sampling rate adapting to changing data rates. We show with real-world data that an adapting packet sampling rate worsens the DDoS attack detection accuracy. To counter performance reduction, we employ upsampling and multi-rate training, showing that the resilience against a changing packet sampling rate improves.**

Index Terms—**DDoS detection, traffic monitoring, packet sampling, supervised machine learning**

I. Introduction

Supervised machine learning (ML) detectors rely on traffic monitoring in two ways. First, they are trained offline with traffic obtained from past monitoring. Second, they process traffic data obtained from current monitoring during deployment. ML detectors perform well if the characteristics of monitored data are similar during training and deployment.

Traffic monitoring can become a bottleneck at high data rates, e.g., 100^{+}Gbit/s. To prevent the monitoring from becoming the bottleneck during traffic bursts, as potentially caused by volumetric Distributed Denial of Service (DDoS) attacks, it can be throttled by limiting the fraction of inspected packets, i.e., packet sampling [1], [2].

However, packet sampling can reduce the stress on the monitoring infrastructure, but it skews observed traffic characteristics, as not all packets are inspected, and traffic information is lost. This leads to dissimilarities between training and deployment traffic data and causes a decrease in the performance of the ML-based detection.

One primary goal for attack detection considering monitoring resource efficiency is not to lose detection accuracy when packet sampling becomes necessary.

Contribution

We evaluate the impact of adaptive packet sampling rates on a DDoS detector, i.e., HollywooDDoS [4], trained in a supervisory manner. We show that HollywooDDoS, which is sampling rate agnostic, cannot preserve high-quality detection when monitoring is performed with sampling rates that have not been covered during the offline training process. We evaluate two countermeasures and show with real-world data that they enable the use of HollywooDDoS with monitoring applying adaptive packet sampling rates.

II. Background and Approach

HollywooDDoS is a DDoS detection approach representing arriving network traffic as two-dimensional images classified by a Convolutional Neural Network (CNN). Monitoring is performed in two dimensions, the source and the destination IP address space. A grid of source-to-destination IP subnet pairs is created and arriving packets are counted per subnet pair. All arriving packets account for image creation during monitoring.

When applying packet sampling, fewer packets are inspected and counter values per subnet pair are potentially smaller, breaking normalization during deployment and decreasing the detection accuracy. We counter this detection accuracy decrease with two methods, namely upsampling and multi-rate training.

Upsampling

We assume that the traffic distribution in the grid of subnet pairs is still correctly captured if enough packets arrive but at a lower traffic volume according to the sampling rate. Therefore, to compensate for the non-inspected traffic, every counter value in the grid of subnet pairs is multiplied with the inverse sampling rate. This provides an estimated reconstruction of the real traffic distribution without packet sampling, ensuring that the normalization does not break.

Upsampling is performed during deployment. Therefore, deployed ML models do not have to be retrained and can be further utilized.

Multi-rate Training

In contrast to upsampling, multi-rate training is not performed during deployment but changes the training process by creating multiple training data sets according to different sampling rates. Therefore, for every sampling rate potentially occurring during deployment, an individual data set is created. The ML model is trained on all data sets, leading to one model that generalizes well across all sampling rates.

III. Evaluation

Training data is composed of real-world attack traffic from CAIDA [3] and benign traffic from MAWI [5]. All data sets are balanced, i.e., they contain the same amount of benign samples as attack samples. Following best practices in ML, we split the dataset into training (70%) and test (30%) set, and present the

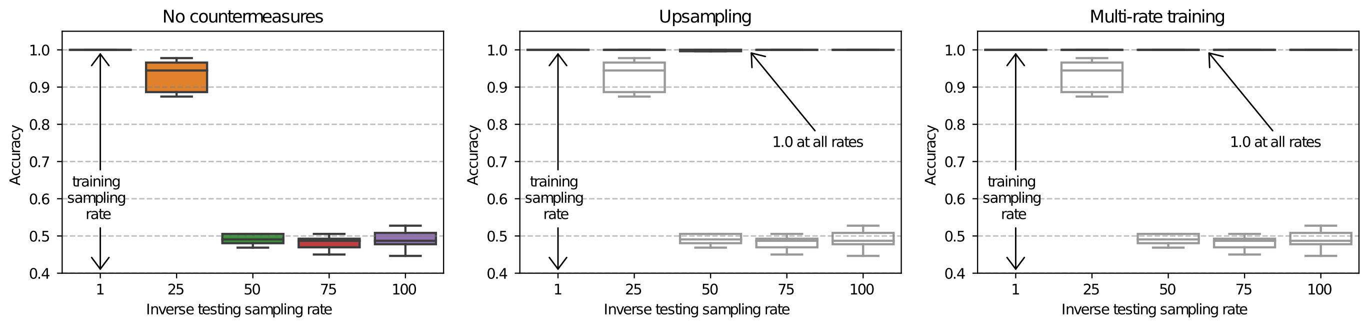


Fig. 1. The accuracy, with and without countermeasures, for different packet sampling rates resulting from a model with training sampling rate 1.0

results from testing. Each experiment has been conducted 20 times. Each result represents the median accuracy of all runs.

Fig. 1 provides accuracy results for an ML model trained with only one packet sampling rate (1.0) with countermeasures (two figures at the right) and without countermeasures (left figure). Tested sampling rates range from 1.0 to 100.

A. *Impact without Countermeasures*

From the left figure of Fig. 1, it is observable that the ML model performs well on the test data with the same sampling rate (1.0) as the training data, resulting in an accuracy of 100 percent. However, when the sampling rate decreases, the accuracy also decreases. For the sampling rate $\frac{1}{25}$ the accuracy drops to 94 percent and for sampling rates smaller than $\frac{1}{25}$ the accuracy drops to 50 percent, constituting a detection as good as guessing. Therefore, if HollywooDDoS would be deployed with adaptive sampling rates, only trained with data obtained from the sampling rate 1.0, detection results would not be reliable. To maintain high detection accuracy across all packet sampling rates, one model per sampling rate needs to be deployed. This is infeasible if sampling rates are not discrete.

B. *Upsampling*

The center figure of Fig.1 presents results derived from the same model as before, but the test data sets have been changed using upsampling by scaling the input with the inverse sampling rate before feeding them into the ML model. Previous results without countermeasures are carried over from the left figure to illustrate the improvement. It is observable that the model, only trained on data obtained from monitoring with a sampling rate 1.0, is now able to perfectly classify images obtained from monitoring with all tested sampling rates.

A significant advantage of upsampling is that one trained ML model can be used for the classification of images at multiple sampling rates. There is no need to change or swap the trained model during deployment because image scaling is performed as part of the monitoring.

C. *Multi-rate Traning*

The right figure of Fig.1 presents results applying the multi-rate training. Multi-rate training interferes the training process by training the ML model with data obtained from monitoring with all tested sampling rates. The goal is to train one model that generalizes well across all sampling rates, without the need for swapping models or rescaling images when using adaptive packet sampling.

Results show that HollywooDDoS trained with multiple sampling rates can perfectly classify images obtained from monitoring at all tested sampling rates, achieving an accuracy of 100 percent. Although the model training is more complex with multi-rate training than using upsampling, no adaptations to the monitoring are required during deployment in exchange.

IV. Conclusion

We outlined that adaptive packet sampling reduces the detection quality during deployment for the supervised ML-based DDoS detection approach HollywooDDoS. We evaluated two countermeasures, namely upsampling and multi-rate training. Upsampling rescales monitoring data according to the inverse packet sampling rate during deployment, while multi-rate training covers all packet sampling rates during the ML model training. We showed the effectiveness of both countermeasures with real-world data from CAIDA and MAWI achieving 100 percent accuracy across all sampling rates.

V. Acknowledgements

This work was funded by the German Federal Ministry of Education and Research (BMBF), RefNr. 16KIS1142K. This work was supported by funding of the Helmholtz Association (HGF) through the Kastel Security Research Labs (POF structure 46.23.01: Methods for Engineering Secure Systems).

References

[1] G. Androulidakis and S. Papavassiliou. Intelligent flow-based sampling for effective network anomaly detection. In *IEEE GLOBECOM 2007 - IEEE Global Telecommunications Conference*, pages 1948–1953, 2007.

[2] Baek-Young Choi, Jaesung Park, and Zhi-Li Zhang. Adaptive packet sampling for accurate and scalable flow measurement. In *IEEE Global Telecommunications Conference, 2004. GLOBECOM '04.*, volume 3, pages 1448–1452 Vol.3, 2004.

[3] Center for Applied Internet Data Analysis. The caida ucsd ddos attack. https://www.caida.org/catalog/datasets/ddos-20070804_dataset, 2007.

[4] Samuel Kopmann, Hauke Heseding, and Martina Zitterbart. Hollywood-dos: Detecting volumetric attacks in moving images of network traffic. In *2022 IEEE 47th Conference on Local Computer Networks (LCN)*, pages 90–97, 2022.

[5] MAWI. Backbone trace. http://mawi.wide.ad.jp/mawi/samplepoint-F/2019/201909011400.html, 2019.

Dynamic Network Intrusion Detection System in Software-Defined Networking

Pegah Golchin *, Jannis Weil *, Ralf Kundel *, Ralf Steinmetz *
*Multimedia Communications Lab (KOM), Technical University of Darmstadt, Germany
Contact: pegah.golchin@kom.tu-darmstadt.de

***Abstract*—Software-Defined Networking (SDN) enhances network management by separating control and data plane functionalities, but the centralized control plane increases the risk of cyber attacks. Therefore, detecting network intrusions, including unknown (zero-day) attacks, is crucial. Machine learning models may be a promising solution, but often lack adaptability due to their reliance on fixed datasets during training. This study investigates corresponding challenges and outlines the potential of employing online learning methods.**

***Index Terms*—Network Intrusion Detection, SDN, Machine Learning, Online Learning, Reinforcement Learning**

I. Introduction

Software-Defined Networking (SDN) is a network architecture that separates control and forwarding functions, enhancing the network management flexibility. The centralized control plane enables a comprehensive network view by establishing forwarding rules to the switch's flow table. However, the centralized control plane is vulnerable to network intrusions [1]. For instance, a Denial of Service (DoS) attack can exhaust flow table memory, preventing the switch from accepting new legitimate flows. Moreover, an attack can overload the controller with an excessive number of new packet flows, leading to disruptions and possible network outages.

To address these issues, increasing the number of SDN controllers can distribute the load to enhance resilience and scalability [2]. However, local network intrusions can still impact the entire network, emphasizing the need for prompt detection and mitigation. Therefore, an intrusion detection system (IDS) is required for an effective network management. By learning flow patterns, Machine learning (ML) has shown promising results in the area of network flow clustering and classification. A ML-based IDS can be implemented using various techniques, including supervised [3] and unsupervised models [4]. Supervised ML-based IDS relies on labeled datasets for effective training, but its ability to detect new attacks is limited to those with similar distribution to the training dataset. Unsupervised models require no labeled data and explore data to identify patterns, leading to better detection of zero-day attacks. However, both supervised and unsupervised approaches struggle with adaptability to dynamic network architectures and concept drift over time.

Concept drift refers to changes in network architecture that result in modifications to the data distribution. Network modifications, such as adding or removing nodes, can impact routing and alter the statistical features of flows over time [5].

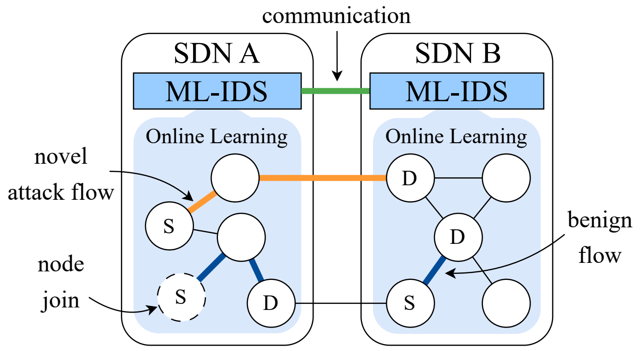


Fig. 1: A cooperative ML-based intrusion detection system (ML-IDS) refines its model in an online fashion and communicates with other instances to classify flows on a global scale.

This shift in distribution can lead to decreased performance in flow classification using fixed models. Network congestion is another example that can impact the distribution of benign flows. A static ML model may misclassify congested flows as attack flows, highlighting the problem of generalization.

To address these challenges, mechanisms are needed for model updates and adaptation to network architecture changes [6], concept drifts, and dynamic variations in flow distributions. Decentralization and cooperation among multiple SDN controllers are also crucial for detecting global trends and resolving attacks across subnetworks. The subsequent section discusses potential solutions and explores challenges associated with each approach.

II. Case Study

Online learning, also known as incremental learning, can be a potential solution. It enables regular updates of a ML model as new data is obtained, allowing an IDS to continuously refine its model to adapt to concept drifts and maintain the detection performance [7]. However, online learning introduces the problem of catastrophic forgetting, where an ML-based IDS may unlearn how to handle previously seen attacks. To mitigate this, the training data should be carefully shaped to adequately represent each attack type. Consequently, the well-crafted and split dataset can be played back as a sequence to continuously refine the IDS model over time. The individual datasets have to cover distribution shift, emergence of novel attacks and reemergence of previous attacks to evaluate the applicability of the approach. As all flows are taken from available datasets, the ground truth labels for the classification problem are known. These can be compared with the predictions of the IDS model to evaluate the efficacy of the online learning mechanisms. Moreover, cooperation between individual SDN controllers in a ML-based IDS can be achieved with a distributed training paradigm like federated learning [8] to propagate long-term knowledge, or with a direct communication link between the models for reactive coordination [9]. If a neural network is used to model the IDS, hidden state information could be exchanged between IDS models to share their view on the network and improve local decision making. Communication approaches from the area of multi-agent reinforcement learning [10] could be adapted for semi-supervised online learning. The overall idea is illustrated in Fig. 1, showing multiple benign flows from source nodes (S) to destination nodes (D) within the same subnetwork and an attack flow that spans from SDN A to SDN B.

Determining the ML-based IDS refining time is another crucial metric which should be considered [11]. A naive solution entails updating the ML-based IDS deployed in the control plane whenever the data plane encounters a new unmatched flow. However, this approach can overwhelm the control plane during instances of DoS attacks or network congestion. To address this challenge, a potential solution is to update the online model using multiple trigger mechanisms.

Therefore, to detect concept drifts, two updating phases are employed. The first phase investigates a subset of flow features, denoted as $F_c = [f_i, f_j, .., f_K]$, which exhibit common behaviors seen in network intrusions (e.g., packet interarrival time, packet size). Analyzing the patterns within these features enables the determination of shifts in the underlying data distribution. When concept drift occurs, the switch initiates a process where flows are forwarded to the controller for a forwarding duration of t_f. During this time, the ML model is updated to adapt to the new data patterns. Furthermore, to address potential shifts in the data distribution caused by network architecture changes, a second trigger mechanism for updating is proposed. This approach involves selecting a random updating time interval, denoted as T_u. After this interval elapses, network flows are forwarded to the control plane for a duration of t_f to update the ML model. Selecting T_u randomly helps to prevent attackers from predicting the precise timing of the ML model updates. The proposed aggregated trigger mechanism ensures that the IDS continuously adapts and remains effective in detecting potential intrusions.

As an alternative to using existing data sets, one could simulate the underlying network and the emergence of new benign and attack flows. The interaction of the IDS model with the network could then be modelled and approached with reinforcement learning. Generative models [12] could be viable to simulate the emergence of new flows in the environment, but constraining their output to realistic traffic data will presumably require the involvement of domain experts and manually created rules. Creating new flows by manipulating existing traffic patterns would also be feasible and require less manual intervention. While this setting would be more realistic than the first approach, the expected effort for design and implementation outweighs its benefits.

III. Conclusion

This work explores the insufficient adaptability of existing ML-based IDS and proposes solutions for their effective use. The main challenges for leveraging online learning include catastrophic forgetting and determining model refinement time. To tackle catastrophic forgetting, we emphasize the need for well-shaped training data that represents benign and attack flows. For refinement time determination, two trigger mechanisms are proposed to detect different concept drifts. If the effectiveness of this approach can be verified in a centralized setup, a potential next step would be the extension to a decentralized setting with cooperative SDN controllers.

Acknowledgment

This work is funded by the Federal Ministry of Education and Research of Germany (BMBF) trough Software Campus Grant 01IS17050 (AC3Net, and ML-based NIDS), the CELTIC-NEXT Flagship Project AI-NET-PROTECT and in parts by the German Research Foundation (DFG) within the Collaborative Research Center (CRC) 1053 MAKI.

References

[1] P. Golchin, C. Zhou, P. Agnihotri, M. Hajizadeh, R. Kundel, and R. Steinmetz, "Cml-ids: Enhancing intrusion detection in sdn through collaborative machine learning."

[2] Y. Zhang, L. Cui, W. Wang, and Y. Zhang, "A survey on software defined networking with multiple controllers," *Journal of Network and Computer Applications*, vol. 103, pp. 101–118, 2018.

[3] P. Golchin, R. Kundel, T. Steuer, R. Hark, and R. Steinmetz, "Improving ddos attack detection leveraging a multi-aspect ensemble feature selection," in *NOMS 2022-2022 IEEE/IFIP Network Operations and Management Symposium*. IEEE, 2022, pp. 1–5.

[4] M. Hajizadeh, S. Barua, and P. Golchin, "Fsa-ids: A flow-based self-active intrusion detection system," in *IEEE/IFIP Network Operations and Management Symposium*. IEEE, 2023, pp. 1–9.

[5] D. C. Mulimani, S. G. Totad, and P. R. Patil, "Concept drift adaptation in intrusion detection systems using ensemble learning," *International Journal of Natural Computing Research (IJNCR)*, vol. 10, no. 4, pp. 1–22, 2021.

[6] M. Ferriol-Galmés, J. Paillisse, J. Suárez-Varela, K. Rusek, S. Xiao, X. Shi, X. Cheng, P. Barlet-Ros, and A. Cabellos-Aparicio, "Routenet-fermi: Network modeling with graph neural networks," *IEEE/ACM Transactions on Networking*, 2023.

[7] G. I. Parisi, R. Kemker, J. L. Part, C. Kanan, and S. Wermter, "Continual lifelong learning with neural networks: A review," *Neural Networks*, vol. 113, pp. 54–71, 2019.

[8] C. Zhang, Y. Xie, H. Bai, B. Yu, W. Li, and Y. Gao, "A survey on federated learning," *Knowledge-Based Systems*, vol. 216, p. 106775, 2021.

[9] S. Sukhbaatar, A. Szlam, and R. Fergus, "Learning multiagent communication with backpropagation," in *Advances in Neural Information Processing Systems*, 2016, pp. 2244–2252.

[10] C. Zhu, M. Dastani, and S. Wang, "A survey of multi-agent reinforcement learning with communication," arXiv preprint arXiv:2203.08975 [cs.LG], 2022.

[11] L. Xie, S. Zou, Y. Xie, and V. V. Veeravalli, "Sequential (quickest) change detection: Classical results and new directions," *IEEE Journal on Selected Areas in Information Theory*, vol. 2, no. 2, pp. 494–514, 2021.

[12] S. Bourou, A. El Saer, T.-H. Velivassaki, A. Voulkidis, and T. Zahariadis, "A review of tabular data synthesis using gans on an ids dataset," *Information*, vol. 12, no. 09, p. 375, 2021.

PicNIC: Image-based Diagnosis for Industrial Blackbox Networks

Marco Reisacher, Andreas Blenk, Hans-Peter Huth
Siemens AG, Munich, Germany

***Abstract*—This research paper presents two tools, SieMonX and PicNIC, that can help monitor and diagnose faults in black box industrial networks. SieMonX is an agent-based network monitoring and data generation tool, while PicNIC is an image-based end-to-end network diagnosis tool that uses a convolutional neural network to diagnose inaccessible networks through simple delay measurements. These tools provide valuable insight into network infrastructure and can aid operators in making informed decisions. A look into both tools provides insight into the functionalities and the implementation.**

***Index Terms*—Machine learning Network Diagnosis, Industrial Networks, Network Monitoring**

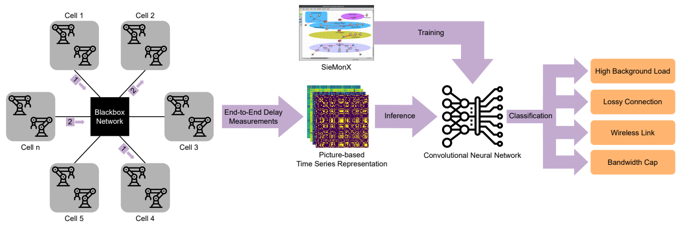


Fig. 1. Training and Inference Pipeline using SieMonX in an Industrial Blackbox Network.

I. Introduction

The Context: In recent years, the convergence of IT and OT networks in industrial environments has become increasingly prevalent. These complex networks involve multiple stakeholders coexisting, making them difficult to monitor and diagnose. Railway networks are a prime example of externally operated networks, which require operators to have insight into foreign networks to diagnose end-to-end connections. However, equipment in these networks is often connected through foreign networks, and there needs to be access to internal network equipment or monitoring data, making it challenging to pinpoint faults.

The Problem: The challenge of monitoring and diagnosing faults in complex industrial networks has been a research topic for many years. Operators often struggle to determine whether a fault is due to broken equipment or an underlying network. This lack of insight can lead to costly downtime and decreased productivity. Dynamic Time Warping (DTW) [1] and deep learning techniques, such as Convolutional Neural Networks (CNNs) [2]–[5]. Moreover, Recurrent Neural Networks (RNNs) [6], [7] are commonly used in time series classification for detecting anomalies, predicting network traffic, and identifying network events. Although these techniques are widely researched, their use case in black box network diagnosis, especially in industrial environments, is a hardly considered topic. This paper presents two tools to address these challenges: SieMonX and PicNIC.

Our Contribution: The contribution of this paper is to provide a comprehensive overview of SieMonX and PicNIC and their potential to address the challenges of monitoring and diagnosing faults in complex industrial networks, as seen in Fig. 1. SieMonX is an agent-based network monitoring and data generation tool for developing data-driven machine learning. It provides a framework for custom monitoring tools or load generators and stores and documents data consistently, making it easy to preprocess. This tool generates valuable data for network analysis and provides a robust foundation for machine learning models to detect faults in the network. PicNIC is an image-based end-to-end network diagnosis tool that uses simple delay-based measurements. This tool visually represents time series data as images, allowing for more straightforward interpretation of network behavior. It uses a Convolutional Neural Network to classify images and detect behavior in the black box network, such as high background load, lossy transmits, and bandwidth restrictions.

II. SieMonX

Motivation: To generate data for training of models or for statistical analysis, often complex tasks or measurement campaigns in testbeds have to be performed. A typical testbed will have several nodes and a communication network in between. Moreover, campaigns must be repeated using different setups to cover a broad range of scenarios. In a non-trivial environment, this can be a time-consuming and error-prone task. Correct and complete documentation of test runs is a must and again adds to the needed efforts. The Siemens Monitoring tool for Experimentations (SieMonX) automizes test campaigns quickly, allowing many test runs with different conditions and storing data consistently. It can even add tags to the data, so manual tagging is not needed in this case. A second problem in those training environments is often that custom functions are needed, which external tools cannot quickly provide. For example, SieMonX includes a simple UDP-based end-to-end monitoring tool that can - on demand - send a short packet immediately followed by a long packet so the effects of a network can be studied under the assumption that both short and long packets receive the same effects. Thus, SieMonX is also a framework for custom monitoring tools or load generators. In this respect, it offers timer handling, a storage service, and more.

State-of-the-Art: Typically, scripts are devolved and deployed in the testbed. This, however requires some development effort and is error prune. A better approach is to have agents on each node and one controller. This can be achieved by several automation tools (e.g., Ansible) or by monitoring tools (e.g., Nagios). However, these tools are built for other purposes, so it is still needed to develop scripts or plugins, and limitations may occur. Moreover, these tools do not address automatic tagging or data consolidation, which requires additional scripting. SieMonX, as a specialized tool, is easier to use and only needs one straightforward high-level script to perform a campaign. There is no need to change something on the nodes remotely.

Architecture: SieMonX has two components: a controller and agents. The controller is the user interface and sends commands to the agents, performing the needed actions. The agents need a simple configuration, the controller's IP address, and a directory for storing data. Upon start, the agent then registers at the controller. A researcher then can issue commands using the console of the controller or can start a simple script to initiate several commands in the testbed at once. This script can add supplementary info to the data from the test run, e.g., configuration data. Each measurement campaign stores its data in an individual directory, together with automatically produced metadata. Thus, preprocessing can quickly identify the correct data set and the conditions under which the data was produced.

III. PicNIC

This section introduces PicNIC, a picture-based approach to diagnosing black box industrial networks using machine learning. Fig. 1 shows the complete pipeline from training using SieMonX to inference in an industrial network.

Data Sources: Fig. 1 shows the two primary data sources for PicNIC. SieMonX mainly provides training data for the neural network, while the data from the real factory network is used for inferencing and diagnosis. It is also possible to use labeled factory data to train the model further to increase classification accuracy.

Data Preprocessing: To use a convolutional neural network for time series classification, the data is preprocessed and converted into a visual representation. These representations must be able to represent information in the time domain, not to lose information about the incoming signal. Gramian Angular Fields (GAFs), Markov Transition Fields (MTFs) [8], and Recurrence Plots (RPs) [9] represent this information accordingly and can keep temporal correlations of the incoming data.

The preprocessing step splits incoming time series data into chunks of n items, as each transformed image has a dimension of nxn pixels. Each chunk contains 100 delay measurements to keep images small to decrease the training and inference time and the model size. Small chunk sizes also decrease the delay between recording the chunk's first measurement and the image's classification. The preprocessor converts each chunk into a GAF, MTF, and RP as input for the CNN.

As the CNN needs exactly one image for classification, the last preprocessing step stitches the three separate images into one final image. The final image has n pixels in width and height with three channels, similar to an RGB picture. In contrast to an RGB image, each channel represents either the GAF, MTF, or RP, not the color value.

Training & Inferencing: Training and inference commence similarly, with the picture format as an intermediate between the data and the neural network. An InfluxDB buffers the streamed data from SieMonX or the black box network. From here, the preprocessing pipeline takes the images, transforms them, and feeds the neural network for training or diagnosis.

Use Case & Results: An emulated industrial network inside SieMonX with multiple agents delivers training data for no error and high background load scenarios between two agents to train PicNIC. The trained PicNIC model then classifies measurements between two different agents connected at different endpoints in the emulated network. PicNIC manages to classify high background load in the black box network with an accuracy of above 90%.

IV. Conclusion & Future Work

PicNIC and SieMonX show the potential of black box network diagnosis in industrial networks. We believe that our work opens exciting possibilities to diagnose black box networks for future emerging network layouts. Future work could include a reinforcement learning approach to let PicNIC auto-adapt to changing network behaviors and layouts, without the need for manual retraining.

References

[1] H. Wu, Y.-P. Zhao, and H.-J. Tan, "A novel neural network based on dynamic time warping and Kalman filter for real-time monitoring of supersonic inlet flow patterns," *Engineering Applications of Artificial Intelligence*, vol. 102, p. 104258, 2021.

[2] A. S. Khatouni, F. Soro, and D. Giordano, "A machine learning application for latency prediction in operational 4G networks," *2019 IFIP/IEEE Symposium on Integrated Network and Service Management, IM 2019*, pp. 71–74, 2019, iSBN: 9783903176157.

[3] D. Alekseeva, N. Stepanov, A. Veprev, A. Sharapova, E. S. Lohan, and A. Ometov, "Comparison of Machine Learning Techniques Applied to Traffic Prediction of Real Wireless Network," *IEEE Access*, vol. 9, no. Ml, pp. 159 495–159 514, 2021, publisher: IEEE.

[4] N. Desai and S. Punnekkat, "Enhancing Fault Detection in Time Sensitive Networks using Machine Learning," *2020 International Conference on COMmunication Systems and NETworkS, COMSNETS 2020*, pp. 714–719, 2020, publisher: IEEE ISBN: 9781728131870.

[5] A. H. Ahmed, S. Hicks, M. A. Riegler, and A. Elmokashfi, "Predicting High Delays in Mobile Broadband Networks," *IEEE Access*, vol. 9, pp. 168 999–169 013, 2021, publisher: IEEE.

[6] J.-R. Jiang, J.-B. Kao, and Y.-L. Li, "Semi-Supervised Time Series Anomaly Detection Based on Statistics and Deep Learning," *Applied Sciences*, vol. 11, no. 15, p. 6698, Jan. 2021, number: 15 Publisher: Multidisciplinary Digital Publishing Institute.

[7] T. Kubota and W. Yamamoto, "Anomaly Detection from Online Monitoring of System Operations Using Recurrent Neural Network," *Procedia Manufacturing*, vol. 30, pp. 83–89, Jan. 2019.

[8] Z. Wang and T. Oates, "Encoding Time Series as Images for Visual Inspection and Classification Using Tiled Convolutional Neural Networks," Jan. 2015.

[9] J.-P. Eckmann, S. O. Kamphorst, and D. Ruelle, "Recurrence Plots of Dynamical Systems," *EPL*, vol. 4, no. 9, p. 973, Nov. 1987.

Towards Synthesizing Datasets for IEEE 802.1 Time-sensitive Networking

Doğanalp Ergenç*, Nurefşan Sertbaş Bülbül*, Lisa Maile†, Anna Arestova†, Mathias Fischer *
University of Hamburg* University of Erlangen-Nürnberg †
Email: *name.surname@uni-hamburg.de †name.surname@fau.de

***Abstract*—IEEE 802.1 Time-sensitive Networking (TSN) protocols have recently been proposed to replace legacy networking technologies across different mission-critical systems (MCSs). Design, configuration, and maintenance of TSN within MCSs require advanced methods to tackle the highly complex and interconnected nature of those systems. Accordingly, artificial intelligence (AI) and machine learning (ML) models are the most prominent enablers to develop such methods. However, they usually require a significant amount of data for model training, which is not easily accessible. This short paper aims to recapitulate the need for TSN datasets to flourish research on AI/ML-based techniques for TSN systems. Moreover, it analyzes the main requirements and alternative designs to build a TSN platform to synthesize realistic datasets.**

***Index Terms*—IEEE 802.1 TSN, machine learning, dataset**

I. Introduction

Modern mission-critical systems (MCSs) such as avionics and automobiles have evolved from static and close-loop networks of embedded devices to highly interconnected networks of different services. IEEE 802.1 Time-sensitive Networking (TSN) protocols have recently been proposed to satisfy the varying quality of service (QoS) and reliability requirements of such services over standard Ethernet equipment [1]. Replacing multiple domain-specific networking technologies in MCSs, TSN reduces their design and maintenance cost and offers additional configuration flexibility.

This flexibility enables us to (re)configure data streams for dynamically changing data traffic requirements due to the addition or removal of new devices, mobility of existing components, or any anomalies in case of potential failures and security incidents [2]. It requires capturing the complex system behavior to detect such changes and develop advanced reconfiguration strategies that can adapt MCSs in real-time to ensure end-to-end deterministic communication requirements [3].

Artificial intelligence (AI) and machine learning (ML) have been recently employed to model the complex nature of MCSs to enhance their safety, reliability, and efficiency [4]. In TSN-enabled MCSs, they can *autonomously classify different types of TSN traffic* such as video streaming, event- or time-triggered traffic. This classification capability aids in network management and QoS optimization to develop effective self-configuration mechanisms [3]. Moreover, AI/ML enables *prediction of future traffic behavior*, which is crucial for capacity planning, resource allocation, network optimization, and predictive maintenance. This leads to better resource efficiency and QoS in highly dynamic environments by rearranging resources based on estimated traffic patterns [5]. Besides, combined with an effective TSN monitoring tool [6], AI/ML models can accurately *detect anomalies in time-sensitive data traffic*, which could indicate network intrusions, security breaches, or performance issues.

However, AI/ML models usually require training with significant amounts of data that should accurately reflect network topology and communication. Since IEEE 802.1 TSN protocols have yet to be broadly deployed, it is challenging to find actual data. Besides, the lack of transparency in MCSs due to their safety and security obligations prevents even (potentially) existing data from being publicly available. Therefore, we need reliable sources and platforms to obtain extensive and realistic TSN datasets. Accordingly, in this short paper, our first goal was to recapitulate the need for TSN datasets to flourish research on AI/ML-based TSN design, configuration, and resilience methods. In the remainder, we explore the main requirements of an open and reusable platform to synthesize public TSN datasets (Section II). Then, we shortly review alternative designs to build such a platform (Section III).

II. Platform Requirements for Dataset Synthesis

The two primary reasons for the absence of public TSN datasets are the lack of widely-deployed TSN systems and the opaque nature of MCSs. Designing an accessible TSN platform, e.g., a TSN-based prototype, simulator, etc., should be the first step to synthesizing the required datasets, which should represent the overall behavior of the respective system in a reliable and verifiable manner. Accordingly, we outline the requirements of such a platform as follows.

- **Representation of a realistic MCS:** A TSN-based platform (and its respective dataset) should accurately reflect the design and operational principles of actual MCSs such as aircraft, automobiles, or industrial systems. This includes modeling a realistic network topology and different service classes with distinct QoS and reliability requirements.
- **Support for various TSN protocols:** This platform should support a broad set of TSN protocols to generate extensive datasets reflecting various MCS scenarios. For instance, P802.1Qav CBS or P802.1Qbv TAS could be alternatively used for scheduling mixed-criticality streams. P802.1CB FRER is required for redundant communication, and P802.1Qcc SRP could also be a prerequisite for the network-wide configuration of these protocols.

- **Verification of system behavior:** The design and configuration of the TSN platform should be verified to ensure a reliable dataset reflecting the desired network behavior. For instance, verifying the TAS scheduler guarantees that all streams in the synthesized dataset are realistically forwarded within their latency boundaries [7]. This requires an extensive analysis of the resulting dataset.
- **Visibility of individual components:** The platform should allow extracting local traffic from selected components alongside an extensive dataset with system-wide network communication. This enables the analysis of the behavior in target components more isolatedly.
- **Scalability:** A realistic topology size and connectivity of MCSs should also be considered for the platform design since they shape the interdependencies between TSN components. This mainly affects both amount and depth of a TSN dataset extracted from the respective platform.

III. Platform Design

We consider three potential designs for a TSN platform to synthesize datasets: a) a hardware-based prototype, b) a hybrid emulation, and c) a simulation platform. This section briefly analyzes them regarding the requirements mentioned earlier. Table I also summarizes this discussion.

TABLE I
Comparison of different platform designs.

	Hardware	**Hybrid**	**Simulation**
Representation	✓	✓	∼
Protocol Support	∼	∼	✓
Verification	✓	✓	✓
Visibility	∼	✓	✓
Scalability	✗	∼	✓

a) Hardware-based Prototype: A platform of actual off-the-shelf equipment, e.g., TSN bridges or more generic TSN-supporting Linux devices, represents an MCS the most realistically [8]. While generic equipment provides extensive visibility, TSN bridges require monitoring capabilities such as mirror ports, which may only exist in some commercial TSN bridges. Proportional to their visibility, the behavior of hardware-based components can be verified by manually accessing these components or processing the dataset derived from the platform. Unfortunately, such a platform potentially has a *partial* protocol support (marked as "∼" in Table I) since existing TSN equipment implements a limited set of protocols as it takes a significant engineering effort. Besides, it can only have a limited scalability that may not reflect the typical size and complexity of real MCSs [9].

b) Hybrid Emulation: When hardware capabilities and platform scalability are limited, it is possible to emulate certain parts of a platform by integrating software-based solutions, e.g., an emulator, into a basis hardware-based prototype. Mininet, for instance, could represent a network of generic Linux-based devices with TSN scheduling capabilities [10]. While this is partially scalable and provides more visibility and easier verification, integrating hardware- and software-based solutions is technically challenging and requires the synchronization of different environments.

c) Simulation: Simulation platforms have already been used in several TSN studies [3], [5], and it is the most flexible alternative regarding configurability, scalability, and visibility. They offer a wide range of TSN protocols with a convenient level of abstraction [11]. Besides, integrating (even run-time) verification mechanisms into the simulations is more straightforward. However, they can only approximate the actual dynamics of MCSs and potentially deviate from real-world network behavior.

IV. Conclusion

AI/ML models can significantly foster the development of advanced design, configuration, and maintenance techniques for emerging IEEE 802.1 TSN protocols. While these models typically require a significant amount of data for training, there does not exist any public TSN dataset due to the lack of broadly-deployed TSN systems. In this paper, we first recapitulate the need for an extensive TSN dataset to enable AI/ML research providing a standardized and reproducible basis for experimentation and validation. However, this first requires a realistic TSN platform that synthesizes and validates TSN datasets. Therefore, we next outline five major requirements and analyze alternative designs of such a platform. We aim to use this preliminary study as a road map for our ongoing effort to create extensive, realistic, and public TSN datasets.

References

[1] "Time-Sensitive Networking (TSN) Task Group." Available at https://1.ieee802.org/tsn/.

[2] L. Maile, K.-S. J. Hielscher, and R. German, "Delay-Guaranteeing Admission Control for Time-Sensitive Networking Using the Credit-Based Shaper," *IEEE Open Journal of the Comm. Soc.*, vol. 3, 2022.

[3] N. Sertbaş Bülbül, D. Ergenç, and M. Fischer, "SDN-based Self-Configuration for Time-Sensitive IoT Networks," *International Conference on Local Computer Networks (LCN)*, 2021.

[4] P. Laplante, D. Milojicic, S. Serebryakov, and D. Bennett, "Artificial Intelligence and Critical Systems: From Hype to Reality," *Computer*, vol. 53, no. 11, pp. 45–52, 2020.

[5] N. Sertbaş Bülbül, D. Ergenç, and M. Fischer, "Towards SDN-based Dynamic Path Reconfiguration for Time-sensitive Networking," *IEEE/IFIP Network Operations and Management Symposium (NOMS)*, 2022.

[6] D. Ergenç, R. Schenderlein, and M. Fischer, "TSNZeek: An Open-Source IDS for IEEE 802.1 TSN," in *IFIP Networking, International Workshop on Time-Sensitive and Deterministic Networking (TENSOR)*, 2023.

[7] A. Arestova, K.-S. J. Hielscher, and R. German, "Design of a hybrid genetic algorithm for time-sensitive networking," in *20th International GI/ITG Conference on Measurement, Modelling and Evaluation of Computing Systems*, 2020.

[8] M. Bosk, F. Rezabek, K. Holzinger, A. G. Marino, A. A. Kane, F. Fons, J. Ott, and G. Carle, "Methodology and infrastructure for tsn-based reproducible network experiments," *IEEE Access*, vol. 10, 2022.

[9] D. Ergenç, C. Brülhart, and M. Fischer, "Towards Developing Resilient and Service-oriented Mission-critical Systems," *9th IEEE International Conference on Network Softwarization (NetSoft)*, 2023.

[10] M. Ulbricht, S. Senk, H. K. Nazari, H.-H. Liu, M. Reisslein, G. T. Nguyen, and F. H. P. Fitzek, "TSN-FlexTest: Flexible TSN Measurement Testbed (Extended Version)," 2022. Preprint on arXiv:2211.10413.

[11] T. Steinbach, H. D. Kenfack, F. Korf, and T. C. Schmidt, "An Extension of the OMNeT++ INET Framework for Simulating Real-Time Ethernet with High Accuracy," in *4th International ICST Conference on Sim. Tools and Techniques*, 2011.

Adapting to the Flow: Reinforcement Learning for Dynamic Priority Assignment in TSN

Nurefşan Sertbaş Bülbül and Mathias Fischer
University of Hamburg, Germany
Email:{sertbas, mfischer}@informatik.uni-hamburg.de

***Abstract*—Real-time systems employ prioritization schemes to accommodate different traffic classes with specific quality of service (QoS) requirements. However, in some scenarios where numerous high-priority packages are transmitted, lower-priority packages may fail to meet their deadlines, leading to a significant decline in scheduling performance. Sending high-priority flows excessively early does not provide any additional benefits beyond meeting the deadline. Instead, it is more effective to utilize this buffer time for lower-priority traffic and ensure on-time transmission of high-priority traffic. We propose an adaptive dynamic priority assignment scheme that utilizes reinforcement learning (RL) to address this issue. This enables adaptation to changing network conditions and continual improvement in performance over time. Additionally, we present and discuss two potential configuration candidates that can be utilized within the proposed scheme.**

***Index Terms*—dynamic priority, reinforcement learning, time-sensitive networks, deadline**

I. Introduction

The Internet of Things (IoT) and time-sensitive networking (TSN), i.e., real-time communication protocols, are enablers for future mission-critical systems and respective applications that require bounded latency as well as seamless and fault-tolerant communication. In this context, the delivery of traffic within a specific timeframe, referred to as the deadline, is significant, with potentially severe consequences when packets are late. In certain critical applications like robotics, any failure to meet these deadlines can have catastrophic consequences. The respective traffic has a hard deadline. However, numerous applications operate with soft deadlines, e.g., best-effort traffic, meaning they can tolerate occasional misses without experiencing a significant decline in performance or output quality. For instance, in video streaming, a few late packets do not have a significant impact on the overall video quality, e.g., the codec contains enough redundancy to recover. In such cases, the system can be more flexible in meeting the deadlines, focusing on providing the best possible output while still striving to adhere to the specified deadline.

The IEEE task group has proposed the TSN standards for providing such a QoS for each traffic class within the same network by describing different handling mechanisms for varying traffic requirements. It uses eight priorities to distinguish traffic classes, representing which traffic is more important with hard real-time constraints. The TSN mechanisms handle traffic on switches based on these priority values.

Although the prioritization mechanism enables the coexistence of different traffic classes, when a large number of high-priority packages are sent, it can lead to lower-priority packages failing to meet their deadlines. Despite having a lower priority, these packages still have their deadlines. This becomes particularly unfortunate when high-priority packages reach their destination much earlier than they are supposed to arrive. From the application perspective, it does not make a huge difference to receive a packet shortly before its deadline or well in advance of it. Thus, instead of sending high-priority traffic early on, it might be better for all flows in the network to send high and low-priority traffic on time. To address that, we propose dynamic priority handling that reassigns packet priorities so that lower-priority packets can make up time and do not arrive late. This can increase the utilization of network resources and decrease the number of late packets.

The authors of [1] have proposed an elastic queuing structure to avoid frame drops due to queue size limitations and with the help of the underlying hardware. However, this is an additional dependency and also increases the hardware requirements. Meng et al. [2] have proposed to use the Fuzzy Analytic Hierarchy Process (FAHP) to compute the priority of packets considering energy consumption, running time, and deadline. Then, a heapsort-based dynamic sorting algorithm selects the optimal scheduling subset from the task set using the new priorities. Even though results promise to reduce the deadline miss rate, it is not directly applicable to TSN with design criteria like energy consumption.

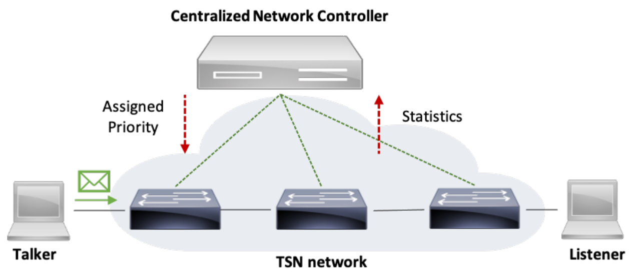
Unlike traditional approaches, RL offers several advantages for dynamic real-time networks as it can adapt and learn from experience in changing environments. In dynamic real-time networks, RL can continuously update its policies and make near-optimal decisions based on the current network state. This autonomous optimization capability allows RL to adapt to changing network conditions and improves network performance over time without human intervention. Moreover, the capabilities of RL to learn from interactions and effectively represent complex relationships position it as a highly suitable approach for addressing challenges encountered in dynamic network environments. Thus, RL has also been used in TSN with different goals, such as finding a routing path [3] or configuring per-hop latency guarantees [4]. Several studies leverage RL for dynamic priority assignment problems in the context of real-time environments [5]. However, the main objective of these studies is to determine a schedulable priority assignment that can accommodate a greater number of flows in the network. Instead, we aim to use resources more efficiently while accommodating, at least, the same number of flows but providing better QoS satisfaction regarding deadlines. This problem becomes more obvious in the case of imbalanced traffic classes, where certain classes may become overloaded while there is an available capacity for other traffic classes.

Accordingly, this paper outlines our roadmap for utilizing reinforcement learning for dynamic priority assignments in time-sensitive networks. Our main goal is to reduce the number of missed flow deadlines resulting from inefficient resource utilization. To achieve this objective, we propose two configuration schemes based on TSN standards: centralized and distributed. We examine these schemes and discuss their limitations in meeting near real-time requirements and providing strict QoS guarantees, considering the constraints imposed by the time-sensitive environment.

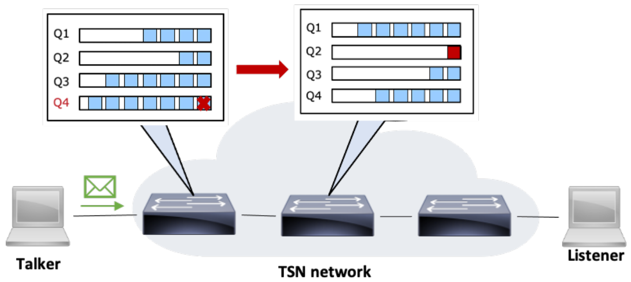
II. Dynamic Priority Assignment with Reinforcement Learning

To leverage the advantages of reinforcement learning, we present two potential configuration schemes in Figure 1 that can be applied to time-sensitive networks:

a) ***Centralized Scheme:*** In a centralized scheme, as illustrated in Figure 1a, it is assumed that the centralized network controller (CNC) has a global network view and collects statistics such as queue waiting time and queue utilization. The RL agent is deployed on top of the CNC so that it receives real-time network data and can utilize RL algorithms to learn and determine global network policies. Since it perceives the network as a whole, this scheme enables coordinated decision-making and optimization across multiple network elements. Thus, CNC can generate *best* policies for the dynamic priority assignment based on a centrally deployed RL agent.



(a) Centralized scheme



(b) Distibuted scheme

Fig. 1: RL-based dynamic priority assignment scheme.

Since there is no policy or apriori information about the network initially, CNC can monitor the current assignments and train the RL agent. In other words, CNC can get hop-by-hop statistics and merge them to compute a final reward value for the action, e.g., the current priority of the packet. Here, considering the QoS requirements of the packets, the reward can get negative values as well, e.,g., a penalty value. During this training (exploration) time, RL can assign random priorities to packets or leave them as they are. After the pre-training time, CNC can benefit from the developed RL policy to reassign a new priority to the packets dynamically on the runtime.

b) ***Distributed Scheme:*** In a distributed scheme, as illustrated in Figure 1b, RL agents are directly deployed at network nodes, e.g., at TSN switches. These agents locally monitor the network, collect real-time data, and learn optimal decision-making policies based on the observed conditions. Each switch aims to compensate for a potential latency in the previous hop by dynamically re-assigning packet priorities determined by RL. Thus, each switch has its own RL agent to develop a policy. It may also be possible to benefit from the *transfer reinforcement learning* concept. Switches may collaboratively help each other to develop the best assignment policy.

However, without a centralized controller, the problem gets harder. Now, the switch has to assess, based on limited knowledge, whether the packet will miss its deadline and needs to get reprioritized. For that, it may need to know the topology or path the packet will be routed. Also, the packet must be marked as a late packet to be handled differently at the next hop switch. Thus, there is a need for a communication protocol and a local or distributed algorithm between switches to address these points.

III. Conclusion

In conclusion, the proposed adaptive dynamic priority assignment scheme leveraging reinforcement learning presents a promising solution for future networks. It can dynamically reconfigure the priorities of existing packets and offers a practical approach to decreasing missed flow deadlines. By adaptively adjusting priorities based on real-time conditions, it can effectively manage resource allocation and meet stringent QoS requirements.

References

[1] A. G. Mariño, F. Fons, A. Gharba, L. Ming, and J. M. M. Arostegui, "Elastic queueing engine for time sensitive networking," in *IEEE 93rd Vehicular Technology Conference*, 2021, pp. 1–7.

[2] S. Meng, Q. Zhu, and F. Xia, "Improvement of the dynamic priority scheduling algorithm based on a heapsort," *IEEE Access*, vol. 7, pp. 68 503–68 510, 2019.

[3] N. S. Bülbül and M. Fischer, "Reinforcement Learning assisted Routing for Time Sensitive Networks," in *IEEE GLOBECOM Global Communications Conference*, 2022, pp. 3863–3868.

[4] A. Grigorjew, M. Seufert, N. Wehner, J. Hofmann, and T. Hoßfeld, "ML-Assisted Latency Assignments in Time-Sensitive Networking," in *IFIP/IEEE International Symposium on Integrated Network Management*, 2021, pp. 116–124.

[5] H. Lee, J. Lee, I. Yeom, and H. Woo, "Panda: Reinforcement learning-based priority assignment for multi-processor real-time scheduling," *IEEE Access*, pp. 185 570–185 583, 2020.