A Demand-aware Networked System Using Telemetry and ML with REACTNET

Seyed Milad Miri
TU Berlin
Berlin, Germany

Stefan Schmid
TU Berlin and Fraunhofer SIT
Berlin, Germany

Habib Mostafaei
Eindhoven University of Technology
Eindhoven, Netherlands

***Abstract*—Emerging network applications ranging from video streaming to virtual/augmented reality should provide stringent quality-of-service (QoS) guarantees in complex and dynamic environments with shared resources. A promising approach to meeting these requirements is to automate complex network operations and create self-adjusting networks. These networks should automatically gather contextual information, analyze how to efficiently ensure QoS requirements, and adapt accordingly. This paper presents REACTNET, a self-adjusting networked system designed to achieve this vision by leveraging emerging network programmability and machine learning techniques. Programmability empowers REACTNET by providing fine-grained telemetry information, while machine learning-based classification techniques enable the system to learn and adjust the network to changing conditions. Our preliminary implementation of REACTNET in P4 and Python demonstrates its effectiveness in video streaming applications.**

***Index Terms*—Self-adjusting networks, programmable data-plane, video streaming, machine learning**

I. INTRODUCTION

Communication networks have become a critical infrastructure of our digital society, imposing increasingly stringent requirements on their dependability and performance. These requirements, however, stand in stark contrast to today's manual and error-prone approach to managing and operating networks, as well as the increasing complexity and scale of networks. Indeed, many communication networks today need to serve a wide spectrum of applications with different performance requirements. These applications typically share network resources in complex ways and have demand patterns that may be hard to predict. For example, emerging applications such as online gaming, video streaming, or virtual/augmented reality may be latency-critical, while a distributed AI application may be bandwidth-hungry. In addition to the inherent complexity of meeting diverse requirements of the network applications, the efficient operation of such networks is further challenged by the limited visibility operators typically have into the current network traffic, as highlighted in prior research [1], [2]. The network operators need to develop several scripts to tailor the network to specific workloads, which are prone to bugs [3].

Automating the management and operation of communication networks is key to overcoming the complexities and dependability challenges of manual network operations [4]. A particularly appealing vision is a fully self-adjusting network: a network that automatically measures itself, gathers information about its context and environment, current demands, and loads, to then evaluate the most efficient and effective resource allocation to meet quality of service (QoS) requirements. Specifically, we are interested in self-adjusting networks that continuously measure, analyze, and adapt.

According to [5], the algorithms in self-adjusting networks must continuously be updated as the requirements and demands change frequently, and usually, they vary from one system to the other. The study also notes another challenge: the cost and risk calculation for each system if it encounters an unknown problem as humans do not manage it. So every aspect of the possible error should be considered, depending on which method we are using the self-driving

Programmable networks can play a vital role in the realization of self-adjusting systems, as they provide flexibility to collect fine-grained telemetry information about the network traffic flows and adapt the forwarding rules accordingly at line rate without delaying the packets of the flows [6]. To achieve the adaptation goal self-adjusting networks, steering traffic, and controlling connections among the endpoints are not enough since we should also consider the storage and processing capabilities of each compute element, particularly those responsible for adaptation and adjustment [7], [8]. Specifically, self-adjusting networks should dynamically observe their current state and automatically react to optimize for specific performance goals accordingly [1]. For example, some Internet customers cannot bear a connection with jitter and latency, which could negatively impact their systems or services. Furthermore, operators can leverage machine learning (ML) algorithms to automate parts of network management and simplify administration. By inspecting packets and flows through data plane programmability [9], they can perform further analysis and predict network behavior in various situations using ML methods. ML enables networked systems to adapt to different conditions and respond automatically based on trained data. Additionally, operators use ML algorithms to classify network traffic, balancing resource consumption in hardware devices with achieving reasonable classification accuracy [6].

This paper presents REACTNET [1], a self-adjusting networked system that aims to realize the vision of self-adjusting networks. REACTNET is enabled by the increased flexibility of communication networks today, particularly network pro-

[1]The preliminary version of this work has been published in [1].

grammability: e.g., programmable switches empower REACTNET through fine-grained telemetry information. REACTNET relies on machine learning-based classification techniques, allowing the system to learn and adjust the network to the new conditions. Hence, REACTNET can learn from the ongoing network conditions and adapt to the new state according to the desired QoS and Quality of Experience (QoE) needs. Our system measures the packet processing time of the desired flows and can set a threshold for them when the packet enters the network. This feature gives the system a powerful mechanism to tune the network application needs to the desired QoS or QoE requirements after applying the ML logic. In contrast, conventional capacity over-provisioning techniques lack such dynamicity in meeting the application needs.

We report on a prototype implementation of REACTNET in a programmable data plane, i.e., P4 [9], and Python, and also present a preliminary performance evaluation with case studies. Our prototype evaluation on a video streaming scenario shows that by adapting, REACTNET can indeed meet the QoE requirements of the application. We also test the accuracy of our ML classifiers on a trace of packets of IoT devices [10] and observe an accuracy of 99% for the classified packets.

The remainder of this paper is organized as follows. We provide a preliminary discussion about the role of traffic classification in Section II. Section III presents the design of REACTNET. The proof-of-the-concept comes in Section IV. Simulation results are illustrated in Section V. In Section VI, we survey the related literature. Finally, Section VII concludes the paper.

II. NETWORK TRAFFIC CLASSIFICATION

Network traffic classification has become a crucial part of any networked system. Network operators seek solutions to classify network traffic to address various issues [11]. The network administrators need to monitor flows within their networks to take appropriate actions according to the underlying network traffic, ensuring meeting the application requirements [12]. Today, by analyzing flows and packets, we can identify distinct patterns, find correlations between features, and pinpoint failures within the network. Additionally, network traffic analysis offers benefits such as intrusion detection and achieving optimal QoS [13].

Initially, the concept of network traffic flows has been crucial for intrusion detection. In [14], a collection of data was used to analyze and classify ML methods. The algorithms used in ML should address problems that evolve and change over time, requiring continuous updates by technical experts [14]. The initial application of ML algorithms for traffic classification and intrusion detection in networks was in 1994. Generally, ML methods utilize features from a dataset as inputs to identify and illustrate patterns between features with different characteristics. After learning and recognizing these patterns, the output describes these patterns and structures [12].

Features are distinguishable elements of network traffic that can be identified in unknown IP traffic. These features include properties of IP packets or flows, such as protocol, flow duration, IP addresses, and source or destination ports. We use these features in ML algorithms for training on known datasets to analyze and classify data, detecting feature correlations among flows or packets. The algorithm then uses the trained data to classify other unknown data.

Supervised and unsupervised learning-based methods are the most applied ML techniques for traffic classification in the networked systems [15]. In supervised learning, the algorithm processes data based on a predefined set of labels and requires preprocessing of the dataset [16]. Conversely, in unsupervised learning, ML algorithms analyze the dataset and find patterns without prior modification or preprocessing, generally grouping features into clusters [17]. However, to classify the packets of different flows belonging to various applications, we use supervised learning-based methods in this paper [18].

III. REACTNET ARCHITECTURE

REACTNET has three main components: 1) Collecting valuable data from ongoing traffic, 2) learning the network status from collected data, and 3) adjusting the network based on the learned situation. We now explain these components and how REACTNET reaches the design goals.

A. Data Collection

Collecting data from the ongoing network traffic is a key operation of self-adjusting networks. The legacy approach to get insights from the network traffic is to use sampling. This technique efficiently collects sample packets from the ongoing traffic flow and provides partial information. Nevertheless, the sampling technique adds non-negligible costs to the network since it requires extra computing power to analyze the data. However, leveraging the programmable hardware [9] can greatly support efficient data collection. Programmable switches can provide insights about packets of all flows without degrading the performance of the network. Therefore, we build REACTNET on programmable networks for data collection.

We can collect insights from the packets by looking at the packet header, such as source and destination IP addresses, source and destination ports, and protocol number. However, the insights collected from the packets of different flows can be significantly improved using the In-band Network Telemetry (INT) provided by P4 [19]. Examples of such telemetry information are queue occupancy of network devices along the path from source to destination, packet inter-arrival time, and packet processing time.

The network operators of REACTNET can adjust data collection depending on the time of need using a customized flag. We use the mirroring feature of the programmable devices to collect data from the traffic flows. If the flag is set, the REACTNET mirrors the traffic to the designated egress port towards the collector. Otherwise, the traffic flows follow the forwarding rules for the routing decisions. The designed system can also mirror the traffic according to the desired interval using a timer if the information of the packets is

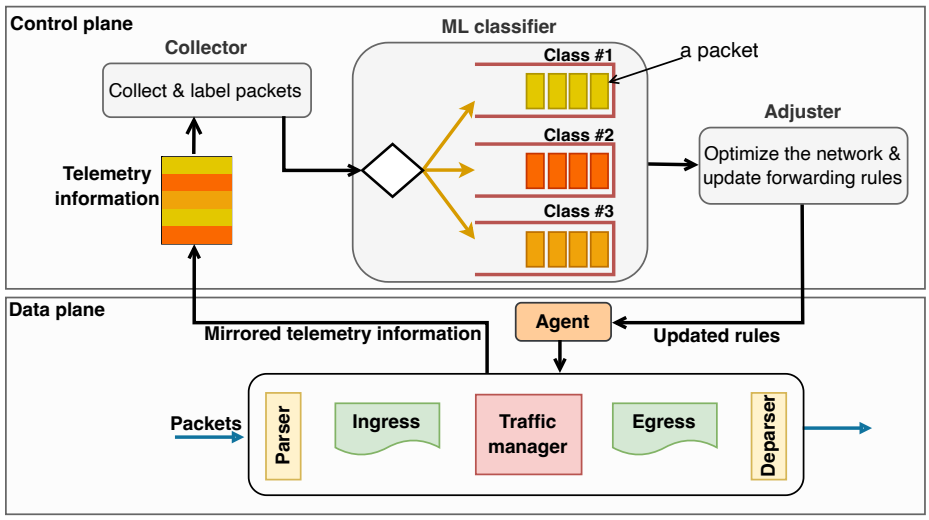


Figure 1. The architecture of REACTNET for self-adjustable networks, implemented on a P4 programmable switch. The data for training, i.e., packet headers and fine-grained telemetry information, of the system is collected via the data plane, while the classification of packets (to three classes in this figure), optimizing, and updating the network are performed via the control plane.

unnecessary. This feature can help the system save bandwidth and decrease the overhead of handling all the packets. We use P4 registers to store the flag value and the timer.

REACTNET can also collect datasets from the network for different purposes since our data collection mechanism for adjusting the network is generic. We use *influxDB* to store the mirrored telemetry data in a database.

B. Learning from Data

The second pillar of a self-adjusting network is the capability to learn from the ongoing network conditions. It empowers the system to make decisions without human interaction based on experience.

One common way to learn from data is to use ML learning techniques. For example, we can use these techniques to classify packets of different flows collected previously. Some widely used classifiers in networking [20] are Decision Trees, Support-Vector Machine (SVM), Random Forest (RF), and K-Nearest Neighbor (KNN). These classifiers are supervised learning techniques that use labeled datasets to train the system and predict outcomes accurately. The classifiers of REACTNET need the label information from the network operators to classify the incoming packets accordingly. REACTNET labels the data before mirroring them. Our system can learn from the trained dataset and properly decide on upcoming conditions by checking the precise information collected from the packets.

C. Adjusting the Network

The self-adjustable network adapts itself to a new state after learning from the current state of the system. This adaption can be accomplished in several ways, such as updating the forwarding rules to balance the traffic of different links or adjusting the priority of the flows to state a few. For instance, consider a scenario where a network should adapt itself to forward the packets of business transactions without delaying them when competing for network resources. The system can learn from the collected data and the packet processing time of the packets if it needs to take a reaction to the ongoing flows. The network operators can specify the requirements of the applications to adjust the network.

The current implementation of REACTNET adjusts the network flows by assigning the desired priority for the packets of different flows. This happened by setting the proper queue ID for the packets of each specific flow. However, this is a design choice rather than a system limitation. REACTNET can also forward the traffic toward multiple links to balance the load of the network. This feature of our system needs to be tuned according to the application's needs. For example, we test the system for a video-streaming scenario to check the impact of self-adjusting on the video quality when the network has to carry non-responsive UDP traffic in §V.

REACTNET adjusts the network– via the proper API of the programmable switches– by updating the priority of the flows to handle the upcoming traffic based on the detected traffic pattern. The forthcoming traffic of the tuned flow follows the new pattern to improve the QoS or QoE. The network operator of REACTNET can also set the flag of mirroring the packets via the switch API whenever the dataset needs updating with the new telemetry information.

D. Bringing It All Together

We now explain the architecture of our system by putting all the mentioned properties together. REACTNET comprises two parts implemented in programmable networks' data and control planes. The data plane includes at least one P4 programmable switch, which can be the switch of the access

network, and provides all the means to our architecture. In contrast, the system control plane gets the applications' needs as the input and adapts the forwarding rules accordingly. We name the components of REACTNET as follows. *P4 switch*, *collector*, *ML classifier*, and *Adjuster*.

Fig. 1 shows the architecture of REACTNET. The *P4 switch* forwards the incoming packets to the designated egress ports according to the forwarding rules. If the corresponding flag to the mirror is set, the switch also appends the telemetry information into the packets. Then, it mirrors the packet with the desired header fields, including telemetry information to the collector. The *collector* receives the mirrored packets and puts them into the database by adding the proper label to each packet. We need data labeling to train our system using supervised machine-learning techniques. REACTNET gets the label information from the control plane and stores it in a proper data structure in the data plane. The *ML classifier* of REACTNET reads the data from the database and classifies the packets into different classes according to the network policy. Then, it sends the classified packets to *Adjuster* component that optimizes the rules and updates them accordingly. The P4 runtime agent updates the rules on the switch, and upcoming traffic flows will follow the updated rule.

Our system measures the packet processing time of the desired flows and can set a threshold for them when the packet enters the network. This feature empowers the system with a more sophisticated mechanism to tune the network application needs to the desired QoS or QoE requirements. While the conventional capacity over-provisioning techniques lack such dynamicity in meeting the application needs.

IV. PROOF-OF-CONCEPT

We implement REACTNET in P4 using the Behavioral Model v2 (BMv2) switch in Mininet and the ML part in Python.

Flow identification. REACTNET needs to identify the packets of different flows after classification by the ML techniques. We assign an ID for each flow and use P4 registers to track them. We also define a register called `prio_reg` to set the priority of the packets for different flows. Since the amount of available memory on the programmable device is limited, the network operators of REACTNET can prioritize the IDs based on the application need and service level agreements after adjusting the network. The switch forwards the packets based on their ID and corresponding priority value.

Data collection. We implement the data collection part of our system from the network flows in P4 in the egress control flow since we have access to all telemetry information. Table I shows the telemetry information with their size in bits REACTNET extracts from every packet. We explain some of the features that need more clarification. *Flow interval time* specifies how long each packet spends between the ingress and egress ports. *enq-qdepth* shows the depth of the queue when the packet was enqueued. *deq-qdepth* specifies the depth of the queue when the packet was dequeued. *deq-timedelta* is the time that the packet was in the queue. We append the telemetry information to the packet and forward it to our data collector.

Table I
FEATURE USED TO BUILD OUR DATASET FOR THE ML-BASED CLASSIFICATION.

Feature	Size in bit
Ingress_port	9
Flow interval time	48
enq-qdepth	19
deq-qdepth	19
deq-timedelta	32
Protocol number	8
Source port	16
Destination port	16
IPv4 source address	32
IPv4 destination address	32

ML classification. We implement the ML classification component of REACTNET using the scikit-learn library in Python. The library has a reach set of implementations for different ML techniques. Our system applies different supervised learning techniques to classify the packets. The user can also specify the desired classification algorithm to apply to the collected data. We use the recommendation of [21] to provide the required label information for packet classification.

Self-tuning dataset. REACTNET by default collects telemetry information of each packet. However, the network operator can tune our system to collect the amount of the cloned packets using a predefined time interval. The time interval information is an input for the system provided by the control plane. This feature avoids the overloading of the collector by capturing many packets.

We define a dedicated timer for each flow using P4 registers to measure the packet processing time in the switch and implement it as follows. We store the current timestamp of the packet in the register and then subtract this value from the timestamp of the next packet. By comparing the subtracted result with an arbitrary threshold, the switch decides to either clone or forward the packet of that flow.

REACTNET exploits the cloning mechanism of programmable switches to send a copy of the telemetry information of each packet. The switch forwards the cloned packets via the egress port connected to the switch. However, having a direct link connection from the switch to the collector is unnecessary since we can modify the packet header to reach the collector according to its destination IP address.

We use Logstash [22] to filter the received packets by providing the key elements or features in the configuration file. REACTNET stores the filtered data sent from Logstash into the Influx database. We use the Influxdb plugin of Logstash to send the data to Influxdb.

Adjusting the network by changing the priority of the packets. We use a set of registers to set the priority for different packets. REACTNET updates the values of these registers via *simple_switch* API. We check the value of the corresponding register to set the proper priority for the packet.

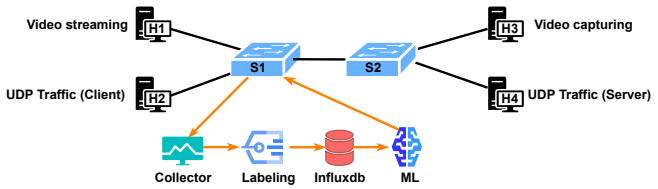


Figure 2. Our evaluation network topology for the video streaming application.

V. Performance Evaluation

This section reports the performance of ReactNet when applied to a video streaming scenario. We also test the performance of the ML classifiers on an IoT trace, including ≈ 600k packets.

Testbed network. We create a dumbbell topology with two P4 switches in Mininet connected with a 2Mbps link. We attach two hosts to switch *S1* to generate the traffic, namely, *H1* for video streaming and *H2* for UDP traffic. We generate UDP traffic using *Iperf*. The two hosts, i.e., *H3* and *H4*, connected to switch *S*2 receive traffic from the corresponding senders (see Fig. 2). We also attach another host, i.e., *collector*, to the switch *S1* to receive the cloned packets.

A. Video Streaming Scenario

This experiment aims to show the capability of ReactNet to adjust itself for a better QoE for the video streaming application. We stream the "Big Buck Bunny" video three times using FFmpeg [23]. We first stream the video without sending any background traffic. This experiment aims to obtain the QoE performance metrics as the ground-truth value for comparison. Then, we stream the video with background traffic without adjusting the network, i.e., without ReactNet. Finally, we stream the video with background traffic and apply ReactNet, i.e., with ReactNet. The UDP client H2 sends 2Mbps traffic to the corresponding receiver host, i.e., H4, after 10 seconds in both scenarios. The main reason for such a design choice is that it gives the system time to collect telemetry information from the current applications running in the network. Otherwise, we could train the system offline.

We use Logstash to collect and label the cloned packets from the streaming video and UDP traffic. We also tune the Logstash configuration to a higher timer precision to get all cloned packets. We apply higher priority to the packets of the video traffic in S1, while the priority of other flows remains at their default value.

Impact on the total frame rate. The frame rate of the original streaming video is 30 frames per second (FPS). The metric for the scenario without applying ReactNet is 26.11 FPS, while with ReactNet, it is 28.92 FPS.

Impact on the image quality metric. We report the Peak Signal-to-Noise- Ratio (PSNR) as the main image quality metric. PSNR indicates the ratio between the maximum possible value of a signal and the power of distorting noise that affects the quality of that image [24]. A higher PSNR value for the quality of images in video streaming applications is preferred. For the reference image f and the disported image g, with the size of $M \times N$, the PNSR in [25] is defined as follows.

$$\text{PSNR}(f,g) = 10\log_{10}(255^2/\text{MSE}(f,g)) \quad (1)$$

whereas:

$$\text{MSE}(f,g) = \frac{1}{MN}\sum_{i=1}^{M}\sum_{j=1}^{N}(f_{ij}-g_{ij})^2 \quad (2)$$

Fig. 3 shows that after adjusting the traffic rate of the video stream, the PSNR value of the streaming video significantly improves since the ReactNet sets the higher priority to those packets. The main reason for such an improvement in PSNR values relies on adjusting the priority of the streaming packets on the switch by using ReactNet.

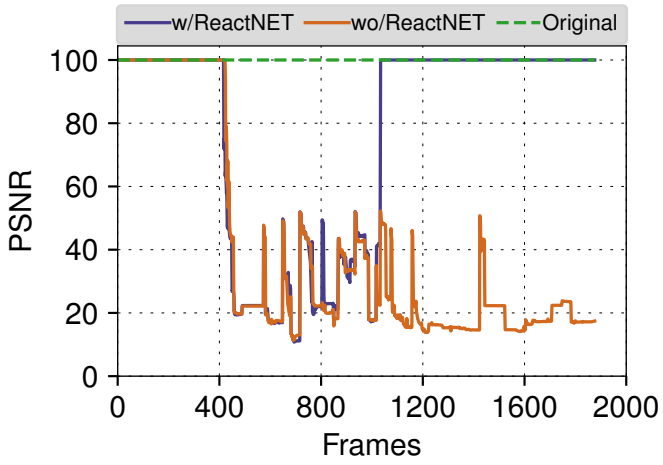


Figure 3. The effect of ReactNet on the PSNR metric of the streamed video.

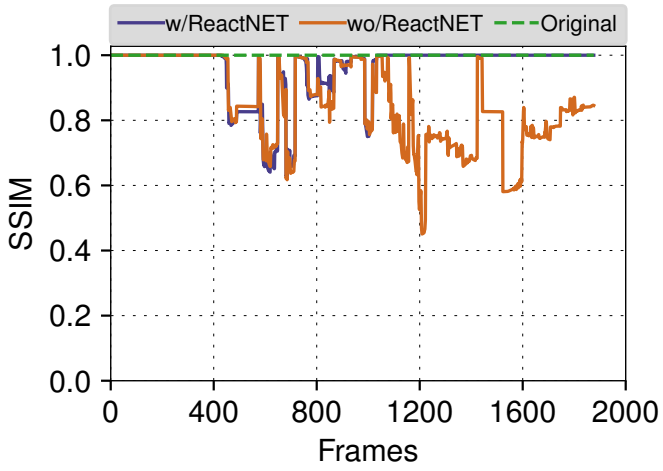


Figure 4. The effect of ReactNet on the SSIM metric of the streamed video.

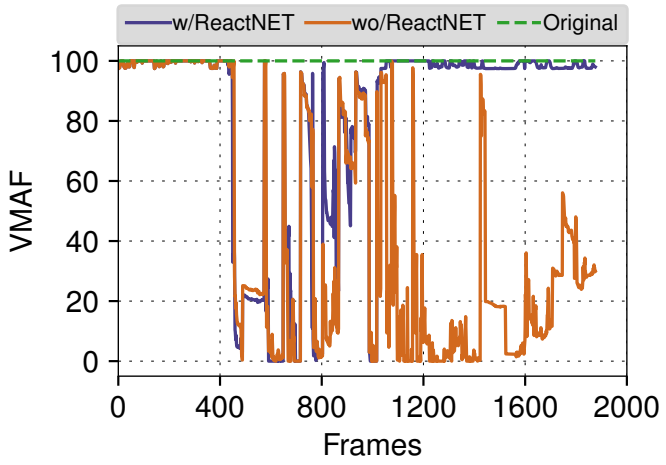


Figure 5. The impact of using REACTNET on the VMAF metric of the streamed video.

Fig. 4 shows the Structured Similarity Index Method (SSIM) metric of the streamed video. The SSIM is a full-reference quality metric that checks the similarity of the reference and the test images by combining three factors: loss of correlation, luminance distortion, and contrast distortion. The value of SSIM varies in the range of [0,1], and zero shows no correlation or similarity, and the value of 1 indicates both images are identical. The streamed video achieves better SSIM when applying REACTNET.

Impact on the video quality metric. Netflix introduced Video Multi-Method Assessment Fusion (VMAF) [26] as a video quality metric. VMAF assesses video quality after re-scaling and compression to detect degradation. The VMAF score ranges from 0 to 100, with higher values indicating better quality. As shown in Fig.5, after adjusting the traffic priority for video streaming, the VMAF score of the stream closely approximates the ground-truth value.

Figures 6(a) and 6(b) compare the identical frame of the "Big Buck Bunny" for the video outputs with background traffic for scenarios without and with REACTNET. The receiver loses many frames before applying the ML classifier with REACTNET resulting in suboptimal video quality for human observation.

B. Accuracy of ML Classifiers of REACTNET

As the second application of our designed self-adjusting system, this section reports the accuracy of ML classifiers in classifying the packets of different flows. We use the traffic trace of IoT devices [10] and use Tcpreplay to inject the packets into the network from host H2 in Fig. 2. The trace contains $\approx$ 600k packets, and we replay them according to the capacity of the link between switch $S1$ and switch $S2$ in our topology. Table II shows the name of each class with the assigned number that we use in the classifiers.

We use the port numbers of different applications in the trace to label the mirrored data in our dataset. The main reason for such a choice is that IoT devices use a few port numbers compared with non-IoT devices. In addition, the devices made by the same manufacturer tend to use standard port numbers. This choice made our classification based on the IoT signaling pattern, which helped us easily classify the flows. We have five different classes, namely, energy, appliances, hubs, health-monitor, cameras, and others that indicate non-IoT devices. We used three models of ML methods, namely k-Nearest Neighbour (KNN), Decision Tree (DT) and Random Forest (RF) on the collected IoT data-trace to make a classification towards reaching optimal results. Table III shows that the decision tree and KNN-based classification have the most accurate results with 99% accuracy.

Name	Class
Energy	0
Appliances	1
Hubs	2
Health-Monitors	3
Cameras	4
Others	5

Table II
THE IOT CLASS NAMES AND THEIR ASSIGNED NUMBERS IN THE CLASSIFIERS

Table III
THE ACCURACY OF DIFFERENT ML CLASSIFICATION ALGORITHMS OF REACTNET ON IOT TRACE.

Model	**Accuracy**	**F1_score**	**MSE**	**Precision**
Decision Tree	0.99	0.1	0.06	0.75
K-Nearest Neighbors	0.99	0.99	0.001	0.87
Random Forest	0.98	0.99	0.11	0.57

We plot the Receiver Operating Characteristic (ROC) curve for DT, KNN, and RF packet classifiers in Figure 7. The ROC curve shows the True Positive Rate (TPR) against the False Positive Rate (FPR) for each class on the y and x-axis respectively for different threshold [27]. Fig. 7(a) shows the ROC curves for each class in II. For each class in the DT methods, the ROC curves lie above the diagonal line x=y. Notably, the minimum AUC for class zero (Energy) is 0.89, indicating that the model can effectively distinguish between positive and negative classes. The ROC for KNN methods for each class in Figure 7(b) show that the AUC for classes 0 (Energy) and 3 (Health-Monitors) is 0.82. Lastly, the ROC for RF in Figure 7(c) demonstrates superior performance compared to the DT and KNN methods. The highest accuracy is observed for classes 4 (Cameras) and 5 (Others), with an AUC of 0.97.

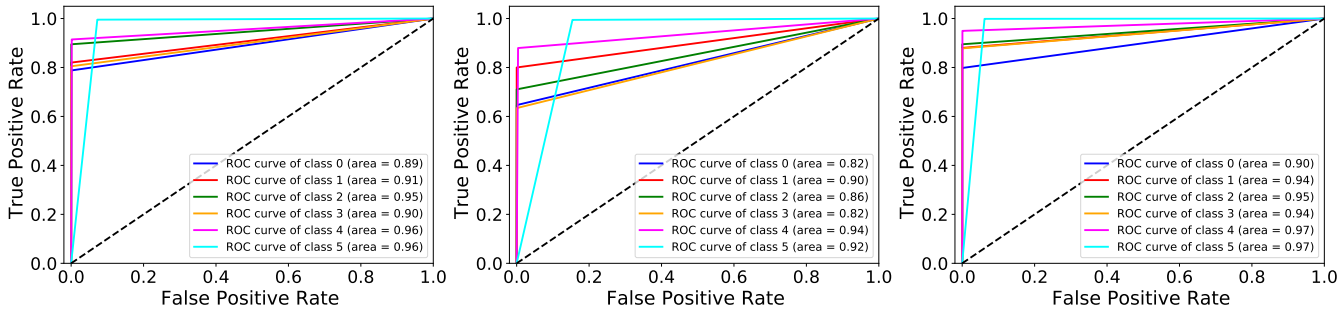
VI. RELATED WORK

The vision of self-adjusting and "self-driving" networks has recently received much attention [3], [28], [6], [29], [30]. It is enabled by the increasing programmability and flexibility of networks [4] as well as the success of AI in various domains. Indeed, softwarization facilitates a more automated

(a) Video output with background traffic

(b) Video output after using REACTNET

Figure 6. Comparison of the identical scene of the two video outputs and the impact of REACTNET on video quality.



(a) DT

(b) KNN

(c) RF

Figure 7. ROC curve results for different classifiers for five classes in the dataset.

administration, operation, and management of the networked systems, and monitoring [31], [32], [29].

We review some research and studies in the self-driven network area. ReNet [33] is a self-adjustable approach to optimizing route lengths in demand-aware networks (DANs). ReNet uses splay trees [34], a self-adjusting Binary Search Tree (BST), to adapt the network based on optimizing topology to facilitate routing issues. The system in [35] proposes a self-driving management system based on intents to reduce the complexity of network management.

AI-based approaches such as NetBOA [31] also generally allow measuring and estimating critical system information such as CPU performance or network latency. A deep reinforcement learning-based approach to coordinate microservices in self-driving networks is proposed in [36] to manage them based on traffic patterns.

The usage of ML has also been considered in the context of data planes [37], [38], [1]. For example, SwitchML offloads the distributed parallel training of part of machine learning systems to the network to reduce the amount of exchanged information and speed up their processing requirements using programmable networks [38]. REACTNET currently relies on an external entity to run the ML classification task. We can offload the classification task of REACTNET to the programmable switches. However, a careful architecture design needs to be considered due to the memory limitations of programmable switches.

VII. CONCLUSION

This paper introduced REACTNET, a self-adjustable network that can adapt to the application requirements given by the network operators. Our system is built on two key enabler technologies: programmable networks and machine learning. Leveraging programmable networks enables the system to get telemetry information from all ongoing packets without delay. This provides more accurate data to our machine learning-based classification algorithms. Our evaluations showed that the system could tune the network to meet the QoE requirements for video streaming applications. Also, the machine learning techniques are highly accurate in classifying the packets of different applications. We plan to extend REACTNET by adding more sophisticated strategies to optimize the network and leverage available resources.

ACKNOWLEDGMENT

This work was partially funded by the German Ministry for Education and Research as BIFOLD - Berlin Institute for the Foundations of Learning and Data (ref. 01IS18025A and ref. 01IS18037A), as well as by German Research Foundation (DFG) project ReNO (SPP 2378), 2023-2027.

References

[1] H. Mostafaei, S. M. Miri, and S. Schmid, "Reactnet: Self-adjusting architecture for networked systems," in *Proceedings of the 17th International Conference on Emerging Networking EXperiments and Technologies*, ser. CoNEXT '21, 2021, p. 473–474.

[2] 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," in *13th USENIX Symposium on Operating Systems Design and Implementation (OSDI 18)*, Oct. 2018, pp. 561–577.

[3] N. Feamster and J. Rexford, "Why (and how) networks should run themselves," *CoRR*, 2017. [Online]. Available: http://arxiv.org/abs/1710.11583

[4] N. Foster, N. McKeown, J. Rexford, G. Parulkar, L. Peterson, and O. Sunay, "Using deep programmability to put network owners in control," *SIGCOMM Comput. Commun. Rev.*, vol. 50, no. 4, p. 82–88, oct 2020.

[5] P. Kalmbach, J. Zerwas, P. Babarczi, A. Blenk, W. Kellerer, and S. Schmid, "Empowering self-driving networks," in *Proceedings of the afternoon workshop on self-driving networks*, 2018, pp. 8–14.

[6] W. Kellerer, P. Kalmbach, A. Blenk, A. Basta, M. Reisslein, and S. Schmid, "Adaptable and data-driven softwarized networks: Review, opportunities, and challenges," *Proceedings of the IEEE*, vol. 107, no. 4, pp. 711–731, 2019.

[7] A. Laghrissi and T. Taleb, "A survey on the placement of virtual resources and virtual network functions," *IEEE Communications Surveys & Tutorials*, vol. 21, no. 2, pp. 1409–1434, 2018.

[8] H. Hantouti, N. Benamar, T. Taleb, and A. Laghrissi, "Traffic steering for service function chaining," *IEEE Communications Surveys & Tutorials*, vol. 21, no. 1, pp. 487–507, 2018.

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

[10] A. Sivanathan, H. H. Gharakheili, F. Loi, A. Radford, C. Wijenayake, A. Vishwanath, and V. Sivaraman, "Classifying iot devices in smart environments using network traffic characteristics," *IEEE Transactions on Mobile Computing*, vol. 18, no. 8, pp. 1745–1759, 2018.

[11] M. Shafiq, X. Yu, A. A. Laghari, L. Yao, N. K. Karn, and F. Abdessamia, "Network traffic classification techniques and comparative analysis using machine learning algorithms," in *2016 2nd IEEE International Conference on Computer and Communications (ICCC)*. IEEE, 2016, pp. 2451–2455.

[12] T. T. Nguyen and G. Armitage, "A survey of techniques for internet traffic classification using machine learning," *IEEE communications surveys & tutorials*, vol. 10, no. 4, pp. 56–76, 2008.

[13] F. Pacheco, E. Exposito, M. Gineste, C. Baudoin, and J. Aguilar, "Towards the deployment of machine learning solutions in network traffic classification: A systematic survey," *IEEE Communications Surveys & Tutorials*, vol. 21, no. 2, pp. 1988–2014, 2018.

[14] J. Frank, "Artificial intelligence and intrusion detection: Current and future directions," in *Proceedings of the 17th national computer security conference*, vol. 10. Baltimore, MD, 1994, pp. 1–12.

[15] F. Pacheco, E. Exposito, M. Gineste, C. Baudoin, and J. Aguilar, "Towards the deployment of machine learning solutions in network traffic classification: A systematic survey," *IEEE Communications Surveys & Tutorials*, vol. 21, no. 2, pp. 1988–2014, 2019.

[16] Y. Lavinia, R. Durairajan, R. Rejaie, and W. Willinger, "Challenges in using ml for networking research: How to label if you must," in *Proceedings of the Workshop on Network Meets AI & ML*. New York, NY, USA: Association for Computing Machinery, 2020, p. 21–27. [Online]. Available: https://doi.org/10.1145/3405671.3405812

[17] M. W. Berry, A. Mohamed, and B. W. Yap, *Supervised and unsupervised learning for data science*. Springer, 2019.

[18] A. S. Jacobs, R. Beltiukov, W. Willinger, R. A. Ferreira, A. Gupta, and L. Z. Granville, "Ai/ml for network security: The emperor has no clothes," in *Proceedings of the 2022 ACM SIGSAC Conference on Computer and Communications Security*, ser. CCS '22. New York, NY, USA: Association for Computing Machinery, 2022, p. 1537–1551. [Online]. Available: https://doi.org/10.1145/3548606.3560609

[19] The P4.org Applications Working Group, "In-band network telemetry (INT) dataplane specification v2.1," https://github.com/p4lang/p4-applications/tree/master/docs, 2020.

[20] D. McGaughey, T. Semeniuk, R. Smith, and S. Knight, "A systematic approach of feature selection for encrypted network traffic classification," in *2018 Annual IEEE International Systems Conference (SysCon)*, 2018, pp. 1–8.

[21] Z. Xiong and N. Zilberman, "Do switches dream of machine learning? toward in-network classification," ser. HotNets '19, 2019, p. 25–33.

[22] "Logstash: Collect, Parse, Transform Logs," https://www.elastic.co/logstash/, 2021.

[23] "A complete, cross-platform solution to record, convert and stream audio and video," https://ffmpeg.org/, accessed: 2022-04-20.

[24] A. Horé and D. Ziou, "Image quality metrics: Psnr vs. ssim," in *2010 20th International Conference on Pattern Recognition*, 2010, pp. 2366–2369.

[25] A. Hore and D. Ziou, "Image quality metrics: Psnr vs. ssim," in *2010 20th international conference on pattern recognition*. IEEE, 2010, pp. 2366–2369.

[26] "Vmaf - video multi-method assessment fusion," https://github.com/Netflix/vmaf, accessed: 2022-04-20.

[27] Z. H. Hoo, J. Candlish, and D. Teare, "What is an roc curve?" pp. 357–359, 2017.

[28] C. Avin and S. Schmid, "Toward demand-aware networking: A theory for self-adjusting networks," *SIGCOMM Comput. Commun. Rev.*, vol. 48, no. 5, p. 31–40, 2019.

[29] H. Mostafaei and S. Afridi, "SDN-enabled Resource Provisioning Framework for Geo-Distributed Streaming Analytics," *ACM Trans. Internet Technol.*, vol. 23, no. 1, feb 2023. [Online]. Available: https://doi.org/10.1145/3571158

[30] J. Hao, P. Subedi, L. Ramaswamy, and I. K. Kim, "Reaching for the sky: Maximizing deep learning inference throughput on edge devices with ai multi-tenancy," *ACM Trans. Internet Technol.*, vol. 23, no. 1, Feb. 2023. [Online]. Available: https://doi.org/10.1145/3546192

[31] J. Zerwas, P. Kalmbach, L. Henkel, G. Rétvári, W. Kellerer, A. Blenk, and S. Schmid, "Netboa: self-driving network benchmarking," in *Proceedings of the 2019 Workshop on Network Meets AI & ML*, 2019, pp. 8–14.

[32] C. Kim, A. Sivaraman, N. Katta, A. Bas, A. Dixit, and L. J. Wobker, "In-band network telemetry via programmable dataplanes," in *ACM SIGCOMM*, vol. 15, 2015.

[33] C. Avin and S. Schmid, "Renets: Toward statically optimal self-adjusting networks," *arXiv preprint arXiv:1904.03263*, 2019.

[34] D. D. Sleator and R. E. Tarjan, "Self-adjusting binary search trees," *Journal of the ACM (JACM)*, vol. 32, no. 3, pp. 652–686, 1985.

[35] K. Dzeparoska, N. Beigi-Mohammadi, A. Tizghadam, and A. Leon-Garcia, "Towards a self-driving management system for the automated realization of intents," *IEEE Access*, vol. 9, pp. 159 882–159 907, 2021.

[36] S. Schneider, A. Manzoor, H. Qarawlus, R. Schellenberg, H. Karl, R. Khalili, and A. Hecker, "Self-driving network and service coordination using deep reinforcement learning," in *2020 16th International Conference on Network and Service Management (CNSM)*, 2020, pp. 1–9.

[37] A. Shukla, K. N. Hudemann, A. Hecker, and S. Schmid, "Runtime verification of p4 switches with reinforcement learning," in *Proceedings of the 2019 Workshop on Network Meets AI & ML*, ser. NetAI'19, 2019, p. 1–7.

[38] A. Sapio, M. Canini, C.-Y. Ho, J. Nelson, P. Kalnis, C. Kim, A. Krishnamurthy, M. Moshref, D. Ports, and P. Richtarik, "Scaling distributed machine learning with in-network aggregation," in *NSDI 21*, 2021, pp. 785–808.