Predicting Network Flow Characteristics using Deep Learning and Real-World Network Traffic

Christoph Hardegen*, Benedikt Pfülb*, Sebastian Rieger and Alexander Gepperth
Department of Applied Computer Science, Fulda University of Applied Sciences, Germany
Email: {christoph.hardegen,benedikt.pfuelb,sebastian.rieger,alexander.gepperth}@cs.hs-fulda.de

***Abstract*—We present a processing pipeline for flow-based traffic classification using a machine learning component leveraging Deep Neural Networks (DNNs). The system is trained to predict likely characteristics of real-world traffic flows from a campus network ahead of time, e.g., a flow's throughput or duration. Training and evaluation of DNN models are continuously performed on a flow data stream collected from a university data center. Instead of the common binary classification into "mice" and "elephant" (throughput) or "short-term" and "long-term" (duration) flows, predicted flow characteristics are quantized into three classes. Various communication contexts (subset of network traffic, e.g., only TCP) and flow feature groups (subset of flow features, e.g., only a flow's 5-tuple), which are supported through an enrichment strategy, are considered and investigated. An in-depth description of the data acquisition process, including preprocessing steps and anonymization used to protect sensitive information, is given. Additionally, we employ an accelerated variant of t-distributed Stochastic Neighbor Embedding (t-SNE) to visualize network traffic data. This enables the understanding of traffic characteristics and relations between communication flows at a glance. Furthermore, possible use-cases and a high-level architecture for flow-based routing scenarios utilizing the developed pipeline are proposed.**

***Index Terms*—Traffic Engineering, Network Management, Flow Prediction, Machine Learning, Deep Learning, NetFlow**

I. Introduction

TRAFFIC ENGINEERING has been an ongoing research field since the early days of computer networks [**?**], [**?**]. Recent advancements in machine learning (ML) have led to approaches that improve network and service management, e.g., using traffic forecasting [**?**], [**?**]. These include the prediction of traffic characteristics for efficient routing as well as load balancing, e.g., based on Equal-Cost Multipath (ECMP) [**?**]. A significant effort has been put in the classification of "mice" and "elephant" network flows to allow for an even link utilization [**?**]. The same applies to the differentiation between "short-term" and "long-term" flows [**?**]. As considered in [**?**] and [**?**], we specify a flow as metadata describing a stream of packets (≥ 1) that belongs to a coherent communication between a given source and destination system.

1) Problem Statement: In a classical routing scenario, the destination IP address only is used to select a link or path to forward a flow's packets. As a consequence, no "intelligent" routing is possible because the average traffic a flow will produce for a finite duration is not known a priori. Additionally, flows cause different traffic volumes and have varying durations. Hence, the following question arises: How precise can different flow characteristics be predicted by DNNs within the scope of various communication contexts and feature groups? The forecast results can be used for traffic engineering, e.g., predictive flow routing to tackle the disadvantages of state-of-the-art routing. The problem is significant for future communication networks because increasing traffic volumes have to be distributed across available network resources efficiently. Otherwise, the capacities are unevenly loaded and flows are negatively affected, e.g., regarding throughput or latency.

Because flows typically have to be forwarded over the same path during their lifetime, reactive load distribution across multiple paths is challenging. For example, this requirement is caused by stateful network components like firewalls and middleboxes [**?**], [**?**]. Thus, a proactive and forecasting-based solution that enhances traffic engineering, which is not limited to destination-based forwarding, is required. This could, for example, include Software-Defined Networking (SDN) [**?**] techniques that can be deployed to enable a fine-grained traffic steering based on the prediction of flow characteristics.

In order to improve traffic engineering, we apply DNNs to an application-oriented scenario, in which dynamic flow information of a real-world campus network is forecasted. Two scenarios clarify related challenges and motivate our work:

a) Example 1: Using only a single optimal path, e.g., the shortest one, can result in a high load or congestion. While alternative paths with higher costs but less load exist, individual flows are negatively influenced by the high load on the shortest path. This can lead to high latency, packet loss and low throughput (flow-level metrics) experienced by the applications, i.e., Quality of Experience. In addition, available links and paths, respectively, are unequally utilized, leading to a high Maximum Link Utilization (MLU) or Maximum Path Utilization (MPU), as well as a low and unbalanced average load (topology-level metrics) of the entire topology.

Figure 1 shows an example with three paths between routers R_1 and R_7. Path $P_1 = (R_1, R_2, R_7)$ with three router hops is the shortest one. $P_2 = (R_1, R_3, R_4, R_7)$ and $P_3 = (R_1, R_5, R_6, R_7)$ are two alternatives with four hops. In a classic routing scenario, P_1 will be preferred over P_2 and P_3. Each path has a capacity

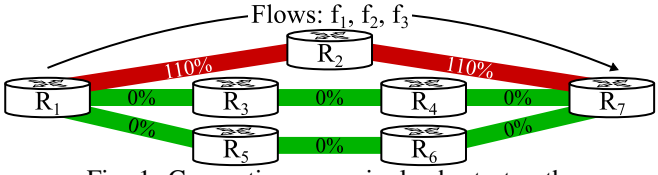


Fig. 1: Congestion on a single shortest path.

*Both authors contributed equally to this work.

of 100 %. Three load levels are distinguished: low (<10 %, green), medium (10 - 90 %, orange) and high (>90 %, red). There are three flows f_1, f_2 and f_3 with given bit rate levels $B(f_x)$ that need to be routed from R_1 to R_7: $B(f_1) = B(f_3) = 40\,\%$, $B(f_2) = 30\,\%$. Forwarding all over the optimal path leads to a theoretical path load of $L(P_1) = 110\,\%$ and results in a congestion on P_1 ($L(P_1) > 100\,\%$), while the full capacity is available on both alternatives with $L(P_2) = L(P_3) = 0\,\%$.

b) Example 2: Multiple paths with equal costs have to be shared equally to avoid unequal utilization. This is, for instance, challenging in ECMP-based load balancing techniques that use flow-based hashing or round robin. An uneven distribution of traffic loads across available paths can lead to a high MLU/MPU. Depending on the throughput of individual flows, an unequal distribution can cause a high load on one of the equal-cost paths. This results in, e.g., high latency, packet loss or a low throughput for flows and for the entire topology.

An example with two simple paths of equal costs (three hops) between the routers R_1 and R_4 is given in Figure 2. The paths $P_1 = (R_1, R_2, R_4)$ and $P_2 = (R_1, R_3, R_4)$ have a uniform capacity of 100 %. Four flows need to be forwarded from

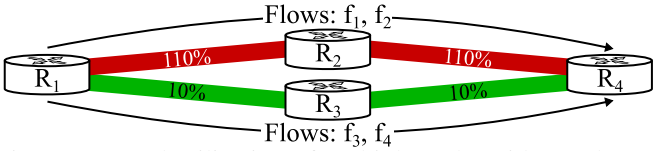


Fig. 2: Unequal utilization of multiple paths with equal cost.

R_1 to R_4 and have the following throughput requirements: $B(f_1) = B(f_2) = 55\,\%$ and $B(f_3) = B(f_4) = 5\,\%$. Routing f_1 and f_2 over P_1 and forwarding f_3 and f_4 via P_2 leads to an unequal utilization. While P_1 is under heavy load with theoretically $L(P_1) = 110\,\%$, whereby f_1 and f_2 may be exposed to the mentioned negative impacts, P_2 is less loaded with $L(P_2) = 10\,\%$.

2) Contributions: In this article, we propose a *flow data stream pipeline* to train and deploy a deep learning based prediction model for the forecasting of several network flow characteristics, i.e., bit rate, duration and number of bytes/packets. Compared to existing traffic classification mechanisms, e.g., distinguishing mice from elephant flows [**?**] or short-term from long-lasting flows [**?**], we introduce a flow feature estimation categorized into multiple traffic classes. This allows for a more fine-grained traffic engineering/steering of network flows.

Traffic characteristics are highly variable, e.g., with respect to the environment and size of the network [**?**]. For instance, the amount of traffic fluctuates over time, and spikes usually occur. Therefore, a key challenge in our work is to obtain realistic network traffic characteristics to be used as input for an ML model. Consequently, a systematic collection of network traffic metadata was conducted in our university campus network. We provide insights into the network topology, traffic data and the used experimental environment. Various communication levels and groups of flow characteristics are investigated to evaluate the prediction of individual flow metadata in different contexts. In order to address gradual changes of collected traffic characteristics, i.e., concept drift, we use a stream processing approach over consecutive time intervals to achieve continuous learning and adaption of the ML model. This setting requires an efficient implementation of the processing pipeline to keep up with the stream of flow data. Privacy is a further obstacle to process flow data and to obtain a realistic dataset. Flows contain IP addresses, which can be viewed as sensitive information. Accordingly, anonymized flow data is used at all processing stages in the proposed pipeline to ensure privacy protection.

This journal article is an extended version of the conference paper "Flow-based Throughput Prediction using Deep Learning and Real-World Network Traffic" from the Conference on Network and Service Management (CNSM) 2019 [**?**].

a) Main Contributions of the CNSM Paper:

- We perform multi-class training and prediction of a flow's bit rate using DNNs working in a streaming setup.
- We give insights into the process of collecting real-world flow data and apply t-SNE as a visualization technique.
- We provide a dataset (525 million flows) collected during one week and the code for our flow data stream pipeline.

b) Extended Contributions of this Article:

- We extend our investigations with regard to various flow communication contexts, e.g., only TCP, and feature groups, e.g., 5-tuple, while considering different flow characteristics as labels, e.g., duration.
- The class boundaries for each prediction experiment are estimated automatically by equal frequency binning.
- We reorganize the flow data stream pipeline and give more insights on its implementation[1] and the experimental setup.
- We validate our research from the CNSM paper as we collected a new dataset and obtained similar results. An optimized t-SNE implementation is used to analyse 10 000 instead of only 1 000 flow data samples at a glance.
- We investigate the influence of an exporter misbehaviour on the prediction results by comparing the results with those of using an alternative exporter.
- We discover accuracy instabilities observed when using the Adam optimizer, which do not exist when using vanilla Stochastic Gradient Descent for DNN models in the context of continuous learning on flow data.
- We present exemplary scenarios, in which the use of predicted flow characteristics results in an optimized flow routing. Additionally, a more detailed overview of a distributed and centralized approach are given.

c) Summarized Main Findings:

- The validity of the CNSM paper is reconfirmed by conducting additional streaming experiments.
- Forecasting different flow characteristics for streamed flow data is feasible with DNNs.
- The prediction accuracies depend on the considered communication context, feature group and the selected label.

II. Related Work

A survey of techniques for traffic classification using ML is given in [**?**], [**?**], [**?**] and [**?**]. An example for using ML to classify flows and their throughput in simulated data center networks is presented in [**?**]. In contrast to simulated traffic,

[1] https://gitlab.cs.hs-fulda.de/flow-data-ml

flow characteristics in real-world networks are often fluctuating and complex [?]. Other techniques for ML-based routing can be found in [?]. [?] presents an SDN-based adaptive traffic engineering framework. Both papers focus on only two traffic classes (mice and elephant). An approach that proposes the use of deep reinforcement learning on synthetic network traffic for routing can be found in [?]. [?] introduces the combination of learning from existing Dijkstra-based routing algorithms and imitating them with higher performance using a dynamic routing framework for SDN to optimize network throughput for simulated data. A solution with supervised deep learning for routing decisions based on real traffic demands is presented in [?]. The model uses aggregated known traffic demands as an input to optimize the overall path utilization. [?] analyzes real-world network flows captured from a university campus network. Thereby, the prevalence of small flows and the classification of features is discussed and the importance of data collection in real networks is emphasized.

The relevance of deep learning based traffic classification and prediction in SDN is explained in [?]. Traffic analysis and routing optimization with deep learning are explicitly named as major future research problems. This is also supported by the necessary shift from rule-based network traffic control to mechanisms using artificial intelligence (AI), e.g., due to steadily increasing traffic volumes [?]. [?] argues that a network AI can be used to predict future network traffic from past data to evolve network management and automation. Using a network AI, [?], [?] and [?] focus on intelligent traffic routing for aggregated traffic characteristics and improved network analytics. For verification, prediction models can be cross-checked, e.g., with existing evaluations of the interpretability of deep learning models used in the area of computer networks [?]. An option is generative replay, whereby generated characteristic traffic is combined with prior data to ensure the adaptability of the prediction model [?].

As proposed for knowledge-defined networking [?] or cognitive network management [?], our approach can be combined with SDN [?] and Virtual Network Functions [?].

Main differences between this article and related works are:

- multi-class instead of binary classification
- real-world compared to synthetic network data
- flow-level instead of aggregated path data
- stream compared to batch processing
- early prediction based on first packet features
- consideration of communication contexts and feature subsets
- discussion of use-cases for improved flow routing

III. Flow Data Stream Pipeline

The flow data stream pipeline (Figure 3) performs data collection and preparation as well as learning from flow data.

A. Data Collection

Data collection takes place in a real-world production network at Fulda University of Applied Sciences.

a) Network Architecture: The campus network covers faculty as well as teaching facilities (about 30 buildings) and hence connects several subnets from the data center, laboratories, researchers, administration (about 600) and also students (about 10 000; primarily WiFi). It is designed according to the core-distribution-access model, in which each building represents an element in the distribution layer, being connected to two core routers as shown in Figure 4. Two data centers are also connected to the core layer, providing central IT services. This architecture allows us to export traffic metadata from both

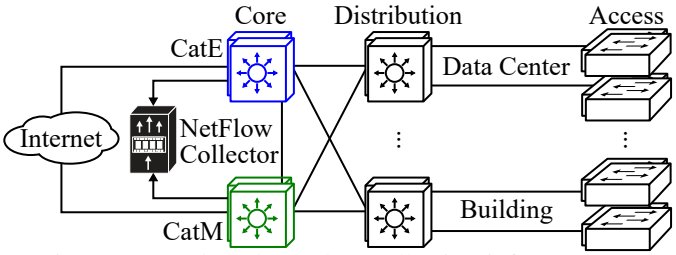


Fig. 4: Network and NetFlow collection infrastructure.

central core routers using the NetFlow protocol, catching WiFi, data center, internal and external traffic, except for packets that are routed within the distribution layer.

The Cisco Catalyst platform is used for the entire network. At the core layer, two Cisco Catalyst 6509-E switches in Virtual Switching System mode build the primary router in building *E* (CatE), while a Cisco Catalyst 4500 switch is available for redundancy in building *M* (CatM). All devices are capable of exporting Cisco Flexible NetFlow [?][?].

b) Flow Export: Flow information for all application protocols is continuously exported by two core routers. As both devices connect most of the campus infrastructure, traffic includes a variety of realistic patterns. This traffic may be affected by constraints (e.g., QoS policies) that are reflected in exported flow data. A configured flow record selects flows to export based on a set of matching criteria, i.e., the 5-tuple (source/destination IP address, source/destination port, protocol). Collect criteria are added to flows before they get exported to the collector. This metadata consists of timestamps for a flow's start and end time, the number of packets/bytes transmitted and a union of all observed TCP flags. A defined flow monitor references the flow record and the configured flow collector. Timeouts ensure that data is either exported at periodic intervals (longer lasting flows) or after inactivity. An active timeout of 600 s and an inactive one of 30 s are used.

Flow records are exported only for ingress traffic to avoid duplicated exports from one router. The collector handles ∅ ≈2 000 flow records per second. Only unicast flows are

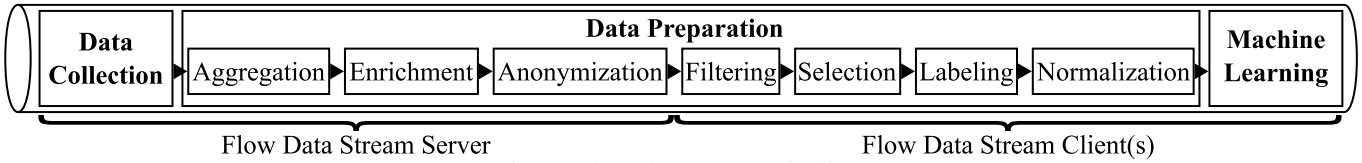


Fig. 3: Flow data stream pipeline.

considered. Since the proportion of multicast traffic is quite low in our network and it does not offer the same feature variety as unicast traffic, multicasts and broadcasts are filtered.

Depending on the switches, a flow can pass through both routers. As both devices export flows, the same data is potentially collected multiple times. Duplicated flows are filtered out and only unique records are further processed (Section III-B1).

In general, there is considerably more work-time network usage but a certain traffic level remains at night. In order to get an examplary overview on the traffic volume within the considered network as well as on the trend of the flow cache sizes and the number of flow records exported per second for a complete week, we refer to our CNSM paper [**?**].

c) Misbehavior of the NetFlow Exporter: While all generated and retraced flows [**?**] were correctly exported by the core switches in various tests, during data collection some flows were affected by a misbehavior of the NetFlow exporter. These flows had a faulty duration, i.e., the NetFlow inactive timeout was added to the flows' duration. For example, there were some flow records specifying only one exchanged packet but a duration of nearly the inactive timeout. The problem may be related to flow inactivity management. Though no significant switch load was monitored during the flow export and the issue was observed for different times and communication contexts. After detecting the issue and trying different configurations for the Cisco exporter, the NetFlow exporter *nprobe* [**?**] was connected to a packet mirror configured on CatE (Figure 5) to investigate the error in the exporter. Due to the potentially critical impact on the network, the packet mirror extracted only WiFi traffic based on VLAN tags.

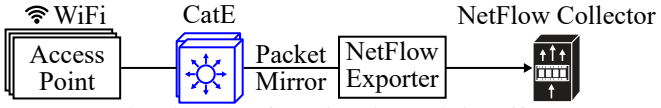


Fig. 5: Setup for mirroring WiFi traffic.

Flow data was collected by each exporter for one work day (8 h). The analysis of both datasets states slight differences regarding the data distribution of a flow's duration and throughput. As the distribution is comparable, the impact on our experiments is exemplarily investigated by predicting the bit rate. We trained two DNNs with identical parameters on each dataset and evaluated the results (Figure 6).

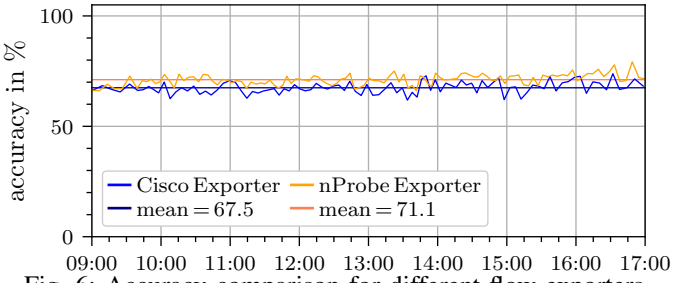


Fig. 6: Accuracy comparison for different flow exporters.

Since the accuracy trends for both experiments show only minor deviations, the influence of the exporter's misbehavior is negligible. Moreover, our focus remained on collecting such realistic traffic data from a real-world network.

Another issue concerns flow timestamps. The exporter reported a higher timestamp for the first compared to the last packet, resulting in negative durations. Out of ≈480 million flows collected during a week (2019-12-02 15:00 to 2019-12-09 15:00), ≈15 000 records distributed over a continuous time interval are affected. As the proportion of involved flows is small, the impact on subsequent steps is assumed to be low.

The described exporter issues show that the collection and preparation of real-world flow data is a major issue since it requires an understanding of the flow exporter's behavior. In general, a misbehavior of an exporter directly influences the quality of collected data but the experiments show that the effect on the prediction results is neglectable.

B. Data Preparation

After receiving a collection of about 100 000 flow records (*block*), data preparation is initiated. To prepare flow information for the training performed in the ML module, several operations (see below) are applied to each block of flow records. Most steps of the data preparation are executed by a central flow data streaming server that multiple flow data streaming clients can connect to. Several clients can implement various experiments, e.g., different learning techniques. The streaming server is responsible for data aggregation, enrichment and anonymization. Additionally, the collected and preprocessed data is stored as an offline dataset. The streaming clients filter and select flow data, before class labeling and normalization are performed (Figure 3). All data preparation operations are parallelized on both, server- and client-side. Therefore, each block is temporarily divided into data chunks that are processed independently of each other.

1) Data Aggregation: One *flow record* might not describe a complete communication because of exporter configurations or hardware limitations (i.e., timeouts or cache sizes). To obtain *flow entries* that represent an entire communication, flow records are aggregated block-wise. The applied method is based on a flow's 5-tuple, timestamps and flags. Flow properties like timestamps, duration, number of packets and bytes as well as bit rate are updated. Because of the timestamp resolution (50 ms steps), the bit rate for short-term flows (duration $<1\,ms$) cannot be calculated and is set to 0 (∅ ≈17 000 records per block). On average ≈20 000 records per block describe communications with only 1 packet (duration $= 0\,ms$). Records that cannot be aggregated are dropped (∅ ≈2 500 records per block). Duplicate flow records from exporters in both switches are filtered based on their first occurrence (∅ ≈4 200 records per block). Aggregation and filtering of a block reduces ≈100 000 records to ∅ ≈75 000 entries.

2) Data Enrichment: Each flow entry is enriched with additional metadata that is extracted from the local and global network context (e.g., private/public prefixes, VLANs, ASNs), depending on a flow's source and destination IP address. IP address management system exports as well as a lookup service [**?**] are used as data sources. Data enrichment aims at enhancing the prediction accuracy and enables the consideration of different communication contexts and levels.

3) Data Anonymization: To ensure privacy protection, an anonymization of IP and network addresses is performed. Address octets are substituted using individual substitution tables, which are permuted based on a cryptographically hashed

password (seed) defined by the data center. This ensures that the semantics of addresses are kept apart from their adjacency. On the one hand, the strategy preserves relationships between IP addresses and their subnets. On the other hand, relations between neighbored IP addresses or subnets are not retained.

4) Data Filtering: The filtering of data allows to exclude flows that match a set of prespecified conditions for available flow features. Feature filtering enables the extraction of sub-datasets from the flow data stream, e.g., for individual experiments. For example, in order to keep only TCP traffic, other flows are skipped based on the transport protocol. Thus, the data filtering stage allows to reduce the number of flows.

5) Data Selection: The selection of one or more flow features supports the consideration of various feature combinations. Again, feature selection enables the extraction of sub-datasets from the flow data stream. For example, to retain only a flow's 5-tuple, the IP addresses, ports and the transport protocol are selected. Hence, the data selection stage allows to decrease the number of features for all flows.

6) Data Labeling: Each flow is assigned a class using predefined boundaries for the selected flow property. Flow properties that merit being predicted by the model are referred to as class *labels*. The *number of bytes* and *packets* as well as a flow's *duration* and *bit rate* are supported. Labels are determined during data collection or updated in the aggregation stage and afterwards transformed into the one-hot format.

7) Data Normalization: To feed the model (e.g., DNNs) with suitable inputs, normalization and data format transformation are performed. The latter includes the replacement of dynamically chosen ports $\geq 2^{15}$ by 0 and the splitting of timestamps. The normalization supports three formats: *float*, *bit pattern*, *one-hot* (e.g., Table I). The float normalization (min-max normalization) maps a single value to a predefined interval, e.g., $[0.0, 1.0]$. Methods like bit pattern and one-hot avoid the representation of numerical proximity, which is important for, e.g., IP addresses. The latter method transforms each categorical value to a bit pattern with a single 1.

TABLE I: Normalization and transformation examples.

Feature	Raw Data	Data Type	Output Data
IP address	81.169.238.182	Float	$0.317, 0.662, 0.933, 0.713$
Protocol	6	Bit pattern	0, 0, 0, 0, 0, 1, 1, 0
Locality	Private | Public	One-hot	0, 1 | 1, 0

An overview of the features in the data stream that results from the data preparation stage is given in Table II. The size of the output vector is stated for each feature. Data transformation types marked in gray are used for the experiments. The origin (Src) of each feature can be identified and features for which there is both a source and a destination are marked with ⇄.

TABLE II: Features in the data stream.

Feature	Data Format			Src
	Float	Bit	One-hot	
month	1	4	12	Data Collection
day	1	5	31	
hour	1	5	24	
minute	1	6	60	
second	1	6	60	
protocol	1	8	✗	
IP address ⇄	4	32	✗	
port ⇄	1	16	✗	

Feature	Data Format			Src
	Float	Bit	One-hot	
network ⇄	4	32	✗	Data Enrichment
prefix len ⇄	1	5	✗	
ASN ⇄	1	16	✗	
longitude ⇄	1	✗	✗	
latitude ⇄	1	✗	✗	
country code ⇄	1	8	240	
VLAN ⇄	1	12	✗	
locality ⇄	1	1	2	

C. Machine Learning

This module implements the training and inference stages. Various online and offline (i.e., not operating on live data streams) experiments can be performed to evaluate the validity of different ML models, their hyper-parameters and data normalization or enrichment strategies. When doing so, either for model training or inference, the prepared flow data is always processed in a block-wise manner. A (chronologically ordered) block is split into a training and test set, both of which are individually shuffled to preserve the chronological order of flow data. The machine learning stage outputs the predictions of the trained ML model for individual flow communications.

D. Data Flow of Network Flow Data

The data flow diagram in Figure 7 depicts the processing of flow data on *Flow Data Stream Server* and *Clients*. Exported flows from multiple exporters are collected by the *NetFlow Collector* until an entire block has been received. Each block of 100 000 *Flow Records* is processed by the *Flow Processor* that splits a block into multiple chunks containing a subset of all flows and passes each chunk to one of n *Flow Processors* performing aggregation, enrichment and anonymization. The number of processing units n depends on the available CPU cores. Splitted blocks of flow records have to be processed in parallel to ensure that processing ends before a new block is available. Preprocessed chunks of *Flow Entries* are recombined to a block, which is compressed and transferred to connected Flow Data Stream Clients by the *Connection Handler*. Raw and preprocessed data can be exported to files, either for offline analysis or reproducible experiments.

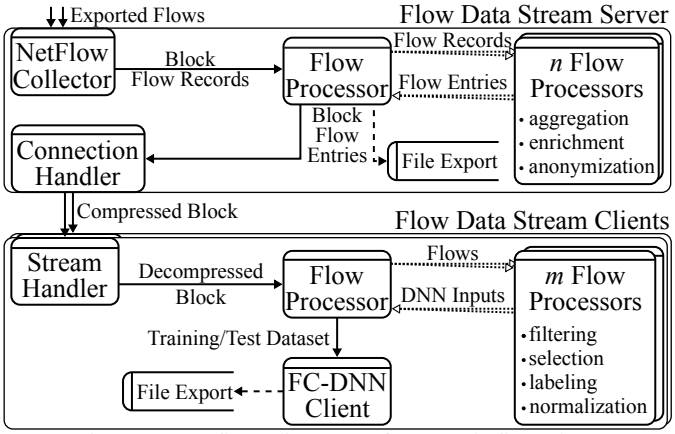


Fig. 7: Data flow diagram of network flow data.

On the *Flow Data Stream Client* side, the *Stream Handler* is responsible for decompressing a received block and passing the flow data to the *Flow Processor*. Again, the *Flows* are split into chunks and processed by m *Flow Processors*, while the number m depends on the available client hardware. Each Flow Processor applies a feature filter and selector, as well as a normalization and labeling function. Finally, the resulting data chunks are reconstructed into a block of suitable *DNN Inputs*, and then split into chronologically ordered *Training* and *Test Datasets*, which are individually shuffled and passed to the depicted *FC-DNN Client* (fully-connected DNN). The datasets and evaluation results can be exported to files.

This architecture supports a simultaneous evaluation of multiple machine learning models on a data stream. As data collection and most parts of data preparation are performed on the server-side, the models can be implemented on the client. Additionally, our streaming solution can replay a stored dataset, either with raw or preprocessed flow data.

The flow data stream pipeline supports the processing of exported IPv4 flows. In order to add support for IPv6 flows, only IP address operations in the affected pipeline stages need to be modified (128 instead of 32 bit). The same statement applies to ASN numbers (4 instead of 2 bytes) or the value range of other supported features in the flow data stream.

All pipeline stages have varying but constant runtime complexities that depend on the block size. In order to guarantee real-time capability, a block has to be processed faster than the subsequent one is collected. The latter is influenced by existing network conditions, i.e., number of flows per second, and available computational resources.

IV. Traffic Analysis Experiments

Unless stated otherwise, we selected aggregated flow entries from block 1340 (2019-12-03 at about 14:00) out of a stored reference dataset[2] for traffic analysis. The dataset contains about 6 800 blocks with overall ≈480 million flow entries collected for a continuous week (2019-12-02 15:00 to 2019-12-09 15:00). The data distribution was investigated based on chosen flow labels. Structural patterns within the flow data are visualized, while only the first 10 000 flow entries of the selected block are used. Variations were present for previous or following blocks, but the trend remained unchanged when considering day and night time separately. Both steps help to provide a better understanding of the data.

In order to verify the operation of each stage in the flow data stream pipeline, intended network traffic was generated and retraced in exported flow data. For example, we performed various downloads and compared the results with obtained flow information, e.g., a flow's duration and throughput. This is described in more detail in our previous work [**?**].

A. Label-based Data Distribution

Figure 8 outlines the data distribution within the complete selected block for each flow label. While respecting the observed value range, the histograms summarize data in 25 bins. Most flow communications are active for a relatively short time (Figure 8a) and/or transmit very few data while being active (Figure 8b). The same findings apply to the observed number of transferred bytes and packets (Figures 8c and 8d), which are both relatively small.

Flow data is unevenly distributed, which is clarified by the median values (Figure 8). As the number of classes and the related definition of boundaries used to label each flow depends on the observed data, a suitable determination is challenging. While having more than two classes, using three classes allows for an approximate even number of samples per class. In order to preestimate the boundaries for different

[2]The dataset is available upon request.

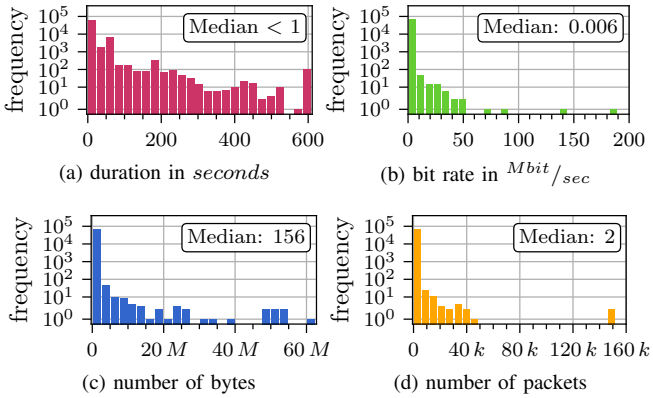


Fig. 8: Histograms of a block for all flow labels (log scale).

experiments with varying data distributions, an equal-depth frequency partitioning method (equal frequency binning) – smoothed by a bin's min. and max. value – is applied. Resulting boundaries ensure the best possible even distribution for each class, which is appropriate for DNN training, even though they have to be adapted for practical application.

Figure 9 illustrates the trend of the class distribution across all dataset blocks. A flow's bit rate is exemplarily used as label. The class boundaries are predetermined (in $^{bit}/_{sec}$) according to the above mentioned strategy: class 0 = [0, 4169[, class 1 = [4169, 12 288[, class 2 = [12 288, ∞]. While the proportion for each class remains approximately the same, there are minor deviations for individual classes. Additionally, class weighting methods are applied to address an imbalanced number of elements per class for training.

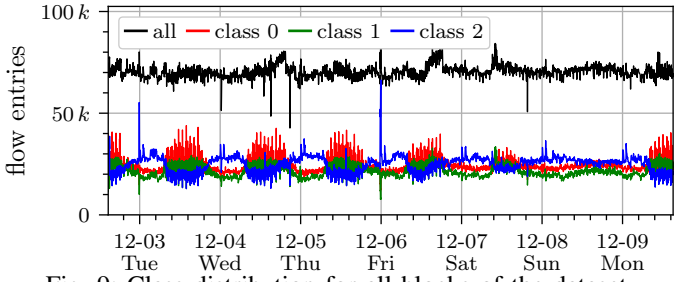


Fig. 9: Class distribution for all blocks of the dataset.

B. Structural Data Patterns

To discover and visualize structural patterns, we use t-SNE [**?**], which is based on an Euclidean distance metric and performs dimensionality reduction by mapping high-dimensional data to a lower space (2D) while preserving neighborhood relations as much as possible. To apply t-SNE on a block with adequate computational time, an optimized tree-based approximation [**?**] is used. Figures 10 to 12 share the same output for 10 000 flows with different context-related tags. Similarities are indicated by the relative distance between sample points, but absolute point positions are meaningless.

The differentiation of flow entries based on the transport protocol is shown in Figure 10. 85.5 % of the flow samples belong to UDP and 13.5 % to TCP traffic. The remaining proportion of 1.0 % describes communications using other protocols like ICMP. For each transport protocol, there are multiple accumulations of samples indicating feature similarities. Symmetric spots for each accumulation can be identified.

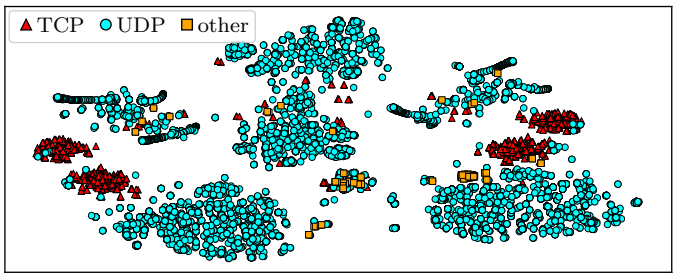


Fig. 10: Transport protocol-based tagging of the t-SNE results.

Figure 11 depicts the tagging of each data sample according to the communications' localities. Four communication types are distinguished based on the source and destination localities. 3.4 % of the 10 000 flow entries are related to communications between private systems, the other 96.6 % involve at least one publicly addressed system (i.e., communication to or from the Internet). While 25.2 % of the flows describe communications between publicly addressed systems, 36.6 % respectively 34.8 % belong to communications between an internal and external system. Again, symmetric accumulations for each communication type are visible. Considering the context of localities, symmetric spots within the t-SNE visualizations are related to different communication directions, i.e., belonging to the same session or conversation. WiFi traffic (46.3 %) is separately marked and strengthens this fact.

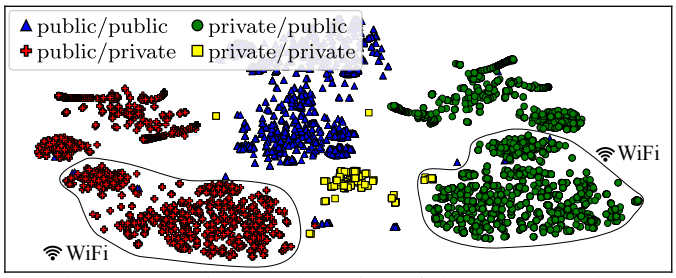


Fig. 11: Locality-based tagging of the t-SNE results.

The tagging of the t-SNE output with symmetric accumulations based on the application protocol is delineated in Figure 12. With 79.8 %, most of the flow entries are DNS traffic, which is interrelated to the huge proportion of UDP flows (Figure 10). 13.2 % of the flow entries belong to HTTP(S) traffic and another 7.0 % to other application protocols, e.g., SNMP, LDAP, SMTP.

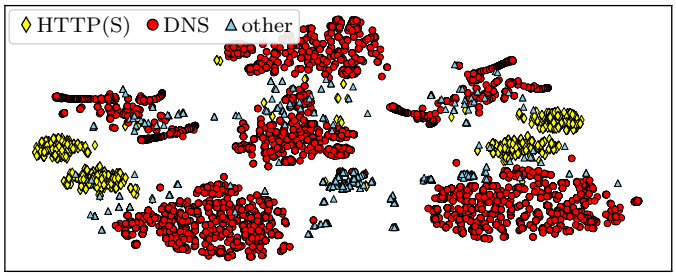


Fig. 12: Application-based tagging of the t-SNE results.

According to the accumulations of individual data samples and their symmetric pair spots, the t-SNE results show feature similarities for several flow entries. The context-based tagging helps to clarify the existing structural patterns in the data.

Besides the fact that almost every communication requires a domain name lookup, the reason for the huge proportion of flow communications that are related to DNS lies in the network architecture and the protocol operation of DNS itself. Requests sent to the internal DNS resolver and those addressed to external DNS servers pass through the central network devices that export flow data. External requests also include queries that cannot be handled by the local DNS resolver. Hence, they are externally forwarded. Flows that describe DNS communications normally have either one exchanged packet or both a small number of transferred bytes and a short duration (short-term). While ≈70 % of all flows belong to DNS, their volumetric proportion is small (Table VI). Thus, these flows and similar ones can be excluded as the prediction is practically not relevant.

V. Flow Prediction Experiments

Subsequently, insights into the experimental setup are given and the results of our streaming experiments are presented.

A. Experimental Setup

In order to perform 240 prediction experiments on the stream of flow data that last for one week, experiments are conducted simultaneously by about 180 computational nodes with different hardware specifications (Table III).

TABLE III: Used computational resources.

#	GPU Type	CUDA Cores	Memory (GB)	Clocking (MHz)	CPU (# × GHz)	RAM (GB)
1	TITAN Xp	3840	12	1582	24 × 2.3	64
1	GTX 1080 Ti	3584	11	1480	24 × 2.3	64
9	RTX 2080	2944	8	1515	8 × 3.4	16
2	GTX 980 Ti	2816	6	1000	24 × 2.3	64
20	Quadro P5000	2560	16	1600	16 × 3.2	32
40	RTX 2060 Super	2176	8	1470	8 × 3.6	16
20	Quadro K2200	640	4	1046	24 × 2.4	32
90	Quadro P620	512	2	1354	16 × 3.6	48

Since static DNN models are used the GPU memory usage is constant for all trained models (about 400 MB). As GPU utilization depends on the clocking and number of CUDA cores, the value ranges from 30 % to 50 % during model training and testing. Data preparation steps are processed parallel by several processes whose number depends on the available cores. Each experiment uses up to 2 GB of RAM. On average, data preparation steps on the flow data streaming server need about 12.20 seconds (aggregation: 7.65 s, enrichment: 4.50 s, anonymization: 0.05 s) while using 8 parallel processes and about 700 MB of RAM. Data preparation steps on the flow data streaming client averagely require 2.13 seconds (labeling: 0.01 s, normalization: 2.12 s) while using 16 parallel processes.

In order to eliminate the computational imbalance of heterogeneous hardware, we chose a slow (Quadro P620) and a fast (RTX 2080) GPU to measure the influence on the training. Both nodes are connected to the flow data streaming server, and two DNNs with identical hyper-parameters are trained for one week. The comparison of performed training iterations or epochs on each block, respectively, yields a constant factor of 2.75. The number of epochs (Figure 13) depends inversely on the number of received flows per second in order to ensure real-time processing. Consequently, training and testing are performed for fewer epochs during the day than at night.

On average, the different iteration counters result in an accuracy deviation of about ±1 % (Figure 14). To exclude the influence of varying training iterations, counters are fixed for all experiments based on the slowest GPU.

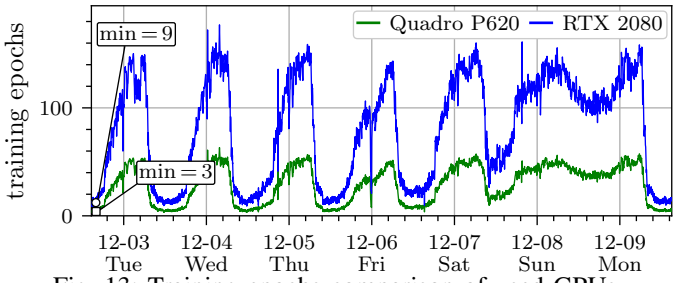


Fig. 13: Training epochs comparison of used GPUs.

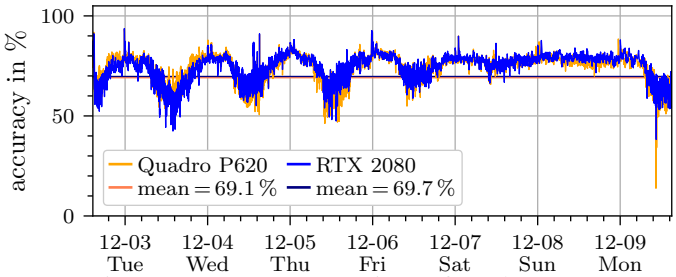


Fig. 14: Accuracy comparison of used GPUs.

B. Deep Neural Network Architecture

Fully-connected DNN (FC-DNN) classifiers with fixed hyper-parameters are used to predict various flow labels. DNN parameters are selected based on a previously performed parameter optimization [**?**] (grid search) using a reference dataset: 5 layers with 1 000 neurons each and a learning rate of 0.001. We chose FC-DNNs because they are efficiently (re-)trainable in a streaming setup (online learning support without resource-intensive pruning) and can represent more complex problems than sparser connected models. During DNN training on a single data block, standard cross-entropy loss is minimized by Stochastic Gradient Descent. Rectified Linear Unit is applied to each hidden layer as transfer function, and the output layer utilizes a softmax function. While flow data is continuously streamed, sequentially prepared and processed (Section III), each block is fed mini-batch-wise (batch size of 100) to the DNN until a specified number of training iterations is reached or the next prepared block becomes available.

A flow's number of packets and bytes as well as its duration and bit rate are selected as class labels. Because regression is more challenging for unevenly distributed data, the prediction is treated as a multi-class classification problem. Training (90 %) and test data (10 %) are obtained by splitting a data block, while respecting chronological order, and are individually shuffled. This ensures that training and test data are separated in time preventing overly optimistic test results due to correlated timestamps, which could be used as features by DNNs. Standard class weighting methods are applied to handle an imbalanced number of data samples for each class. This means that a weight factor based on the proportion of data samples is determined for each class. Weighting is performed block-wise for the training and testing phase.

C. Learning on a Streaming Interval

240 streaming experiments for various communication contexts are performed, in which DNNs (Section V-B) are trained and tested on a flow dataset collected for a week (2019-12-02 15:00 to 2019-12-09 15:00). For each context, individual sub-datasets and class label boundaries are determined. Only intermediate values are given, whereby 0 as the lower and ∞ as the upper bound are used for each entry (Table IV). Features are grouped and their importance is analyzed (Table V).

TABLE IV: Contexts/feature boundaries for the experiments.

Communication Context	Feature Boundaries																	
	KBytes				Packets				Duration (*s*)				Bit Rate (*Kbit/s*)					
all contexts	0.11	≤	0.35	≤	1	≤	3	≤	0.14	≤	0.53	≤	4.1	≤	12.2	≤		
only WiFi	0.15	≤	0.45	≤	2	≤	3	≤	0.21	≤	1.67	≤	4.2	≤	10.8	≤		
exclude WiFi	0.09	≤	0.33	≤	1	≤	2	≤	0.11	≤	0.24	≤	4.1	≤	12.9	≤		
exclude DNS	0.51	≤	2.94	≤	5	≤	12	≤	0.18	≤	2.38	≤	3.5	≤	24.1	≤		
only TCP	1.50	≤	4.49	≤	9	≤	15	≤	0.25	≤	4.92	≤	6.3	≤	45.8	≤		
only UDP	0.09	≤	0.17	≤	1	≤	2	≤	0.12	≤	0.57	≤	4.0	≤	9.7	≤		
multi-packet flows	0.28	≤	1.41	≤	2	≤	8	≤	0.21	≤	1.39	≤	5.6	≤	20.8	≤		
public/public	0.09	≤	0.28	≤	1	≤	2	≤	0.11	≤	0.23	≤	4.4	≤	12.0	≤		
private/private	0.09	≤	0.44	≤	1	≤	4	≤	0.10	≤	0.28	≤	3.9	≤	14.4	≤		
private/public	0.01	≤	0.50	≤	1	≤	3	≤	0.17	≤	0.92	≤	4.0	≤	12.8	≤		

TABLE V: Feature groups for the prediction experiments.

Feature Group	Features in the Data Stream	Inputs
all	each flow feature from Table II	247
5-tuple	IP address ⇄, port ⇄, protocol, timestamp	109
internal	network + prefix length ⇄, VLAN ⇄	98
external	network + prefix length ⇄, ASN ⇄, country code ⇄, geo coordinates ⇄	112
5-tuple + internal		207
5-tuple + external		221

In summary, each prediction experiment evaluates an individual combination of a communication context, feature group and selected flow label. Table VI summarizes the results, whereby maximum, mean and median accuracy calculated based on the maximum value of each block are given. The forecasting of a flow's number of packets and bytes is less challenging compared to the throughput and duration.

Table VI also shows the protocol proportion and median values of flow volume metrics that are determined based on the first ≈4 000 blocks. Later blocks are excluded as flows with a negative duration are contained (exporter issue, Section III-A). Besides TCP and UDP, DNS is listed because it is the predominant application protocol in collected data.

Eight experiments are selected and the accuracy trends are shown in Figure 15. All are related to the same communication context. The achieved accuracies for using all features or a flow's 5-tuple are compared for different labels. The differentiation between day and night has a significant accuracy influence. This is due to more network dynamics at daytime, e.g., a varying number of student mobile devices. Generally, the enrichment improves results by about +2 % and stability of the inference stage especially during the day.

VI. Discussion of Results

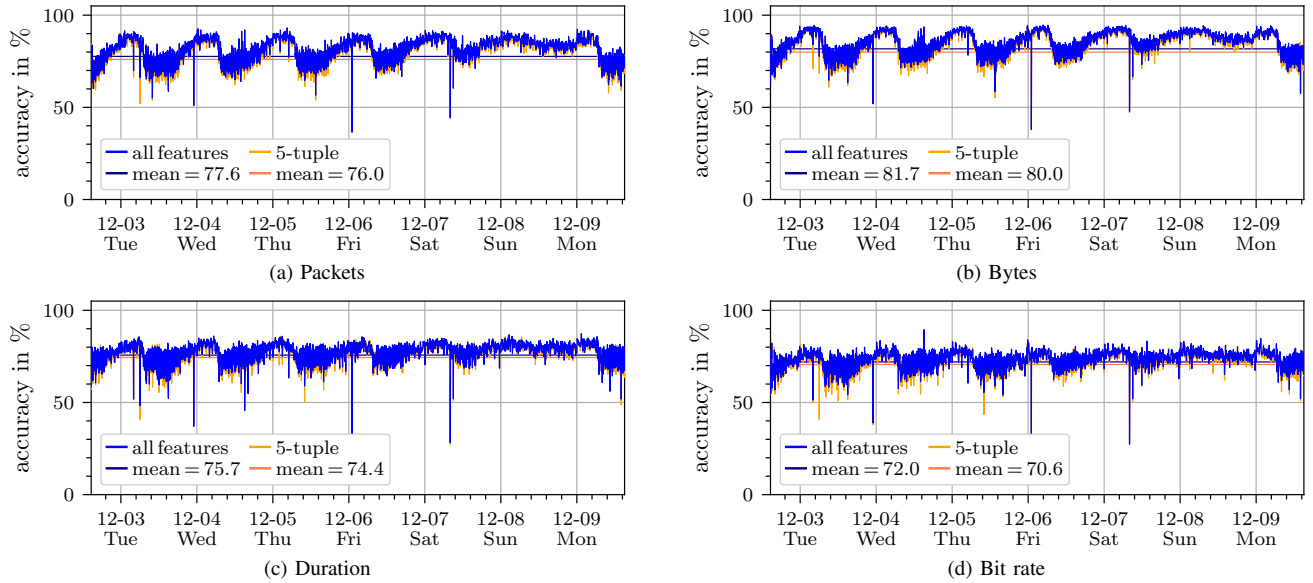
In the following, analysis results for collected flow traffic and findings from the streaming experiments are discussed.

A. Traffic Analysis and Data Distribution

Although the flow data has no inherent metric, t-SNE is a suitable tool for the visualization of structural patterns. The

TABLE VI: Overview of the flow prediction experiment results (left part) and flow volume metrics (right part) for each context.

Communication Context		Accuracy for Feature Groups and Labels in %																						Flow Volume Metrics (Median)				
		all				5-tuple				internal				external				5-tuple + internal				5-tuple + external						
		Bytes	Packets	Duration	Bit Rate	Bytes	Packets	Duration	Bit Rate	Byte	Packets	Duration	Bit Rate	Bytes	Packets	Duration	Bit Rate	Bytes	Packets	Duration	Bit Rate	Bytes	Packets	Duration	Bit Rate	Proportion (%)	KBytes	Packets	Duration (s)	Bit Rate (Kbit/s)
all contexts	max	100	100	90	93	100	100	90	93	98	99	99	98	98	99	95	91	100	100	90	94	100	100	90	92	TCP 21	2.6	12	1.0	14.9
	mean	96	98	67	74	95	97	67	73	85	88	61	67	85	88	61	67	95	97	67	73	95	97	67	74	UDP 79	0.1	1	0.2	6.1
	median	97	98	68	75	95	98	68	73	85	88	61	67	85	88	61	67	95	98	68	74	95	98	68	74	DNS 71	0.1	1	0.2	6.0
only WiFi	max	100	100	96	95	100	100	94	95	98	100	93	95	98	100	97	95	100	100	96	95	100	100	94	95	TCP 24	3.3	13	3.3	9.0
	mean	96	99	67	69	94	98	64	66	76	81	58	62	76	81	59	62	94	98	65	68	94	98	66	68	UDP 75	0.2	2	0.2	6.2
	median	97	99	69	69	95	98	66	66	82	86	60	62	82	86	60	62	95	98	67	68	95	98	68	68	DNS 73	0.2	2	0.2	5.9
exclude WiFi	max	100	100	94	96	100	100	93	95	98	99	88	90	98	99	88	92	100	100	93	96	100	100	93	96	TCP 19	2.2	11	0.8	17.7
	mean	97	98	71	77	96	98	71	76	86	88	65	68	86	88	65	69	96	98	71	76	96	98	71	76	UDP 81	0.1	1	0.1	6.1
	median	98	99	72	77	97	99	72	76	86	89	66	69	87	89	66	69	97	99	72	76	97	99	72	76	DNS 70	0.1	1	0.1	6.0
exclude DNS	max	96	95	91	90	95	95	90	89	93	92	88	88	93	92	88	88	95	95	91	90	95	95	90	90	TCP 70	2.6	12	1.0	14.9
	mean	85	84	72	72	84	82	71	71	74	71	64	65	74	71	65	65	84	82	71	71	84	82	71	71	UDP 27	0.1	1	0.1	4.9
	median	85	84	73	73	83	81	72	71	74	70	64	65	74	71	65	65	83	81	72	71	83	82	72	72	DNS 0	0.0	0	0.0	0.0
only TCP	max	95	93	87	89	94	92	87	89	92	89	83	89	93	90	83	90	94	92	87	90	94	92	87	89	TCP 100	2.6	12	1.0	14.9
	mean	82	78	76	72	80	76	74	71	70	65	67	65	71	65	68	65	80	76	75	71	80	76	75	71	UDP 0	0.0	0	0.0	0.0
	median	80	77	76	72	79	75	74	71	70	64	67	64	70	64	67	65	79	75	75	71	79	75	75	71	DNS 0	0.0	0	0.0	0.0
only UDP	max	97	100	100	99	97	100	100	99	96	100	100	97	97	100	98	97	97	100	100	99	97	100	100	100	TCP 0	0.0	0	0.0	0.0
	mean	75	83	56	54	75	83	56	55	71	80	54	52	71	80	54	53	75	83	56	54	75	83	56	54	UDP 100	0.1	1	0.2	6.1
	median	74	82	56	53	74	83	56	53	71	80	54	51	71	80	54	51	74	82	56	53	74	82	56	53	DNS 90	0.1	1	0.2	6.2
multi-packet-flows	max	98	97	93	93	97	96	92	94	95	93	90	91	94	94	91	91	97	96	92	94	97	97	93	94	TCP 40	2.6	12	1.1	16.5
	mean	90	91	76	76	88	89	75	74	78	78	67	67	78	78	68	68	88	89	75	74	88	89	76	75	UDP 60	0.2	2	0.2	6.7
	median	89	91	77	76	88	89	76	74	78	77	68	67	78	78	68	67	88	89	76	75	88	89	76	75	DNS 54	0.2	2	0.3	6.0
public/ public	max	100	100	99	98	100	100	99	98	98	100	96	95	99	100	97	95	100	100	99	98	100	100	99	98	TCP 11	2.2	11	0.9	18.0
	mean	97	99	78	74	96	99	78	72	86	94	72	61	87	94	73	62	96	99	78	73	96	99	78	73	UDP 88	0.1	1	0.1	6.2
	median	98	100	79	75	97	100	79	73	88	95	72	61	89	95	73	62	97	100	79	73	97	100	79	74	DNS 87	0.1	1	0.1	6.2
private/ private	max	100	100	97	98	100	100	97	97	99	99	97	97	99	99	97	97	100	100	97	98	100	100	97	98	TCP 25	1.2	7	0.2	29.0
	mean	93	96	63	76	92	95	62	76	78	80	56	68	78	80	56	68	92	95	63	76	92	95	63	76	UDP 71	0.1	1	0.1	4.9
	median	94	97	63	77	93	96	62	76	79	80	55	68	79	80	55	68	93	96	63	76	93	96	63	76	DNS 7	0.7	6	5.7	2.9
private/ public	max	100	100	94	94	100	100	91	93	96	99	90	91	96	99	88	91	100	100	90	97	100	100	98	94	TCP 25	3.1	13	2.5	10.1
	mean	95	98	68	71	94	97	67	70	82	86	59	64	82	87	59	65	94	97	68	71	94	97	68	71	UDP 74	0.2	2	0.2	6.1
	median	96	99	70	72	95	98	69	70	83	87	60	64	83	87	60	64	95	98	69	71	95	98	69	71	DNS 71	0.2	2	0.2	6.0



(a) Packets (b) Bytes (c) Duration (d) Bit rate

Fig. 15: Accuracy trend for only TCP flows and different labels.

t-SNE visualizations clarify data similarities and provide a better understanding of the flow data in our university network.

Regardless of selected labels, the data is unevenly distributed w.r.t. classes. Hence, the definition of class boundaries is challenging. In this article, classes are statically defined so that an on average approximately equal number of samples for each class is obtained, which is required for DNN training. Small differences in class frequencies are compensated by a class-dependent weighting factor in the loss function. At the same time, sufficient data samples for each class are required.

We chose a shortcut and analyze the label distribution for all training data (e.g., not just the current block) in order to define appropriate class boundaries. In a streaming setting where only the current block is available, this shortcut will have to be replaced by a more general solution.

The treatment of classification problems with unbalanced classes is difficult with DNNs [**?**] when class imbalances are too high. Roughly, there are three ways to deal with imbalances: First of all, there is the method we have adopted here, which simply increases the DNN learning rate for less frequent classes (analogous to oversampling). A downside of this approach is that there is no way of telling if this modified learning rate is too high, which could break down the training process completely. Secondly, one might simply discard samples from more frequent classes in order to ensure balanced classes. However, this would ignore available data, which is inacceptable to us. And lastly, one could store or generate a set of "holdout" samples that is used to bolster less frequent classes. While this would be a robust solution, it means that learning is partly performed on data that have no connection to the current learning task. Thus, performance could suffer enormously.

Network traffic contains a huge proportion of flows that do not necessarily require a prediction, e.g., short-term flows, and, thus, can be excluded or handled in a differentiated manner.

B. Results of the Experiments

Our experiments show that multi-class bit rate prediction for streamed flow data is feasible with DNNs. Compared to binary classification, a multi-class problem is more challenging. Considering only two classes, our experiments also achieve more than $90\,\%$ accuracy.

Regarding the investigation of different communication contexts, feature groups and flow labels, the prediction results vary. Whereas the proposed enrichment strategy can improve and stabilize the accuracy, using a flow's 5-tuple only provides slightly worse results (about $-2\,\%$). Forecasting a flow's number of packets and bytes is less challenging than predicting its throughput and duration. Because the latter and the number of bytes are used to calculate the bit rate, a fine-grained duration prediction is of high interest. This calls for a more precise export of flow timestamps.

Comparing the results of a previously performed feature importance analysis [**?**] to the results presented in Section V-C, learning on a stream rather benefits from the enrichment but is context-dependent. While the initial feature importance experiments from [**?**] only consider a short time interval (10 blocks, 5 - 10 minutes), the streaming experiments are based on a week's worth of flow data. Due to more network dynamics during the day, the accuracies are considerably less favorable than at night. The results do not seem to be directly related to the number of training epochs, which strengthens this fact.

We initially employed the Adam optimizer in 240 performed streaming experiments, in which DNN models were trained for a continuous week. A significant proportion of the experiments resulted in accuracy instabilities. All experiments provided stable results for the first 444 blocks ($6\,\mathrm{h}$). The instabilities we observed started at different times and followed no simple condition or obvious pattern like having no sufficient number of flow data samples for one class. However, this may be due to the varying data distributions (e.g., class imbalances) in consecutive blocks and the resulting impact on the used optimizer. As a more comprehensive analysis of the optimizer issue is required, we used vanilla Stochastic Gradient Descent for our DNNs. Compared to the previous experiments, no unstable accuracy results were detected.

A statement on the generalizability of our approach remains an open issue, because no comparable datasets exist as a basis for further evaluation. In contrast to using synthetic data, which depends on the generators quality, a real-world network scenario is the most credible setup. Although other environments have different traffic patterns or characteristics, a campus network with a huge amount of fluctuating and heterogeneous systems is challenging. Thus, obtained results may be transferable to similar networks. Nevertheless, reproducible simulations for practical transferability are needed to evaluate the actual benefits in real scenarios.

VII. Flow Routing Scenarios

First, the forecasting of flow characteristics can be used to enhance a flow's performance (i.e., latency, packet loss, throughput). A suitable path for a flow communication in the network topology can be determined based on the predicted flow characteristics. Second, flow-based prediction offers potential for an optimized or equal utilization of multiple paths (e.g., multi-pathing or ECMP), whereas a round-robin or hash-based path selection can result in uneven resource utilization or congestion. As both scenarios require an advanced traffic routing, a proactive flow steering based on predicted flow metadata is suggested. Additionally, an architectural proposal for a centralized and distributed approach is presented.

A. Use-Cases

In the following, different use-cases for the application of prediction results to network routing are presented.

1) Throughput Prediction for Path Selection: A proactive path determination based on the prediction of a flow's likely throughput is one approach to address the challenges related to the examples given in Section I. On the one hand, using a single optimal path for multiple flows between two routers may result in a high load or congestion. Meanwhile, alternatives that have higher costs, but are less loaded, can exist (Figure 1). On the other hand, multiple paths of equal costs have to be utilized equally (Figure 2). Knowing a flow's approximate throughput ahead of time allows for an advanced distribution of flows across paths. This minimizes negative impacts on individual flows and optimizes the MLU/MPU in the topology.

In relation to the first example given in Paragraph I-1a, Figure 16 shows three simple paths between the routers R_1 and R_7 with a uniform capacity of $100\,\%$: P_1 as the shortest one (three router hops) as well as P_2 and P_3 as two alternatives with higher costs (four hops). Predicting the required throughput for the three flows f_1, f_2 and f_3 that need to be routed while

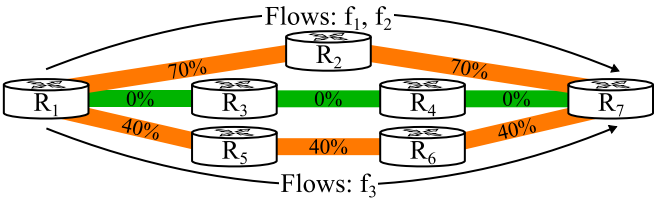


Fig. 16: Preventing path congestion by throughput prediction.

causing varying loads and considering the forecasting for path selection enables the avoidance of congestion:

1) f_1 is forwarded via P_1 ($L(P_1) = B(f_1) = 40\,\%$).
2) f_2 is routed via P_1 ($L(P_1) = B(f_1) + B(f_2) = 40\,\% + 30\,\%$).
3) As forwarding f_3 via P_1 would cause congestion, an alternative path is selected, e.g., P_3 ($L(P_3) = B(f_3) = 40\,\%$).

Although the utilization of P_2 is low ($L(P_2) = 0\,\%$), instead of having one overloaded path, both P_1 and P_3 are utilized to a medium level ($70\,\%$ / $40\,\%$). Because there can be multiple paths to choose from (i.e., P_2 and P_3), different selection strategies are possible: choose the less loaded path or evaluate path metrics (e.g., latency/load or router resources like queue utilizations) to calculate and compare their weighted sums.

According to the second example given in Paragraph I-1b, Figure 17 depicts the simple paths P_1 and P_2 between the routers R_1 and R_4, which are of equal cost and have a capacity of $100\,\%$. Four different flows with varying throughput

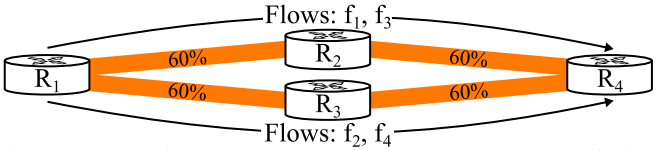


Fig. 17: Preventing uneven path load by throughput prediction.

requirements have to be forwarded. The prediction of a flow's throughput allows an even utilization of the paths and helps to avoid congestion:

1) f_1 is routed via P_1 (randomly chosen, $L(P_1) = B(f_1) = 55\,\%$).
2) f_2 is routed via P_2 ($L(P_2) < L(P_1)$, $L(P_2) = B(f_2) = 55\,\%$).
3) f_3 is forwarded via P_1 (randomly selected because P_1 and P_2 are equally utilized, $L(P_1) = B(f_1) + B(f_3) = 60\,\%$).
4) f_4 is forwarded via P_2 because P_2 is less loaded than P_1 ($L(P_2) = B(f_2) + B(f_4) = 60\,\%$).

As a result, P_1 and P_2 are equally utilized to a medium load level of $60\,\%$. Besides a randomized selection of evenly utilized paths, an advanced mechanism that selects a path based on the evaluation of prespecified path or router statistics and their weighted combination is possible. While this analysis is either based on the current or past state, a prediction of path attributes, e.g., the observed latency, can serve as a criteria.

At this point, the examples assume that the available capacities satisfy the required resources. Discussed flows can also be interpreted as an aggregation of a given set of communications.

2) Duration Forecast for Predictive Traffic Load Matrices: When a flow routing decision is required, the predicted flow throughput must be compared to the monitored network state. Link or path loads can be organized in link-level traffic load matrices based on past or current load data. The use-cases in Section VII-A only use the current load levels $L(P_x)$. If all flows are routed based on the forecasting results, the traffic load matrices correspond to the sum of predicted throughputs. To evaluate a predictive link/path state, not only the likely throughput $B(f_x)$ but also the probable duration $D(f_x)$ of a flow is required. Predicting both characteristics answers the question of how long and resource intensive a flow will be.

For example, considering two flows f_1 (start t_1, $B(f_1)$, $D(f_1)$) and f_2 (start $t_2 = t_1 + \frac{1}{2} D(f_1)$, $B(f_2)$, $D(f_2) = D(f_1)$) that are routed over the same path P_x results in the following facts regarding the likely link loads of P_x in a predictive load matrix:

- $B(f_1)$ for the time interval from t_1 to $t_1 + \frac{1}{2} D(f_1)$
- $B(f_1) + B(f_2)$ for the time interval from t_2 to $t_2 + \frac{1}{2} D(f_2)$
- $B(f_2)$ for the time interval starting from $t_2 + \frac{1}{2} D(f_2)$

3) Flow and Topology Predictions for Path Selection: Next to organizing predictive path states based on the likely throughput and duration of flows, the prediction of topology characteristics can serve as the basis for evaluating the topology's state. For example, the average load or related latency of a single link or comprehensive path $L(P_x)$ can be predicted for a given time interval T. Hence, not only the current topology state or, e.g., an average of a past time interval, but also the likely future state can be considered. Based on the example given in Paragraph I-1a, Figure 18 depicts a related scenario.

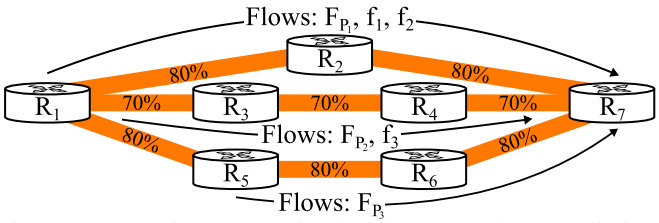


Fig. 18: Preventing congestion by flow + topology prediction.

In a hybrid scenario, there are flows that get routed using a classic routing algorithm (F_{P_x}) and selected ones f_x for which prediction-based flow routing is applied. The former flows traverse P_x or a part of it. Over time, F_{P_x} flows cause different load on various paths that are monitored in a differentiated manner (e.g., excluding prediction-based routed flows). They are used to train a model based on path loads to predict the load of each link/path for a specified time interval T. Combining the predicted throughput $B(f_x)$ and duration $D(f_x)$ with the forecasted path load $L(P_x)$, high load/congestion on the optimal path between R_1 and R_7 can be proactively avoided. Three flows f_1, f_2 and f_3 with $B(f_1) = B(f_3) = 40\,\%$ and $B(f_2) = 30\,\%$ need to be routed from R_1 to R_7 with the expected path loads $L(P_1) = 10\,\%$, $L(P_2) = 30\,\%$ and $L(P_3) = 60\,\%$ for T. All flows f_x start and end in T ($D(f_x) \leq T$).

1) f_1 is forwarded via P_1 ($L(P_1) + B(f_1) < 100\,\%$).
2) f_2 is routed via P_1 ($L(P_1) + B(f_1) + B(f_2) < 100\,\%$).
3) Because f_3 would cause congestion on P_1, an alternative path (P_2 or P_3) must be selected.
4) While $D(f_3) \leq T$, and the expected path load is $L(P_2) < L(P_3)$, P_2 is chosen. This is due to a classically routed flow that starts after f_3 and is routed from R_5 via R_6 to R_7 and likely causes load on the path.

The result is a medium load for all three simple paths $L(P_1) = 80\,\%$, $L(P_2) = 70\,\%$ and $L(P_3) = 80\,\%$.

The same procedure can help to optimize the example given in Paragraph I-1b. Figure 19 shows an exemplary scenario.

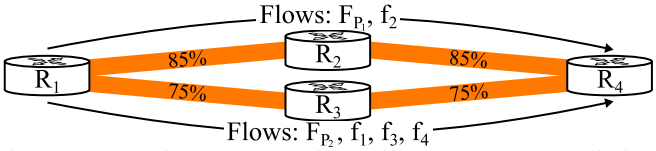


Fig. 19: Preventing unequal util. by flow + topology prediction.

Four flows f_1, f_2, f_3 and f_4 with $B(f_1) = B(f_2) = 55\,\%$ as well as $B(f_3) = B(f_4) = 5\,\%$ need to be forwarded from R_1 to R_4 with the expected path loads $L(P_1) = 30\,\%$ and $L(P_2) = 10\,\%$ in T. The initial difference results from varying classically routed flows. All flows f_x start and end in T ($D(f_x) \leq T$).

1) f_1 is routed via P_2 ($L(P_2) < L(P_1)$) and at the same time, $L(P_2) + B(f_1) < 100\,\%$).
2) f_2 is forwarded via P_1 ($L(P_1) < L(P_2)$ and at the same time, $L(P_1) + B(f_2) < 100\,\%$).
3) f_3 and f_4 are routed via P_2 ($L(P_2) < L(P_1)$ and $L(P_2) + B(f_3) + B(f_4) < 100\,\%$).

The result is a medium load for the two paths ($85\,\%$ and $75\,\%$).

The prediction of path loads is used for the representation of the topology state in both examples. Other path information, e.g., the latency or various router resources like CPU or memory and their weighted combination can be considered as well. If $D(f_x)$ does not fall completely within T, the programmed flow steering needs to be reevaluated and reactively adapted.

B. Architectural Overview

1) Centralized Approach: A closed-loop architecture in which the proposed prediction of flow characteristics can be utilized for various flow routing scenarios (Section VII-A), is depicted in Figure 20. The network topology is build of SDN-enabled devices (e.g., OpenFlow [**?**] or P4 [**?**]) that forward network traffic within the topology. These devices are able to share their state and configuration with a centralized control instance, e.g., by SNMP [**?**], NETCONF [**?**] or network telemetry. The controller receives exported flow information from SDN devices, e.g., via NetFlow [**?**], IPFIX [**?**] or supported by programmable data planes (P4).

The *Controller* implements a prediction-based flow routing with three main components. A *Flow Processing Engine* is responsible for handling collected network flow data that will be fed to a continuously (re)trained ML model. All required operations are executed through a flow data stream pipeline that performs data collection, preparation and machine learning (Section III). The *Topology Monitor* continuously maintains a state model of the routing topology, e.g., path capacities and current load. To route individual flows, a network router sends a routing request to the central *Routing Engine* that queries the Flow Processing Engine with the corresponding 5-tuple describing the flow. It receives a forecast of one or more relevant flow characteristics. While considering the predicted flow characteristics, the Routing Engine evaluates the topology's state to select an appropriate path within the topology. Afterwards, the path is programmed on affected network devices and the Routing Engine maintains the proactive distribution of flow communications.

Next to a fully SDN-enabled solution where a complete path is programmed on each relevant network router, a source

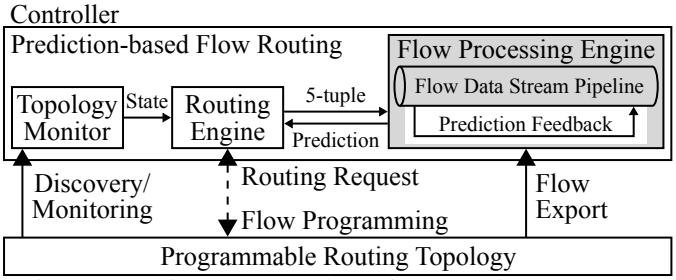


Fig. 20: Architectural overview of a centralized approach.

or segment routing based approach is also possible. Thereby, flow-based predictions are requested from a source edge or border router, and the centralized controller provides a suitable path for a flow communication which is embedded in the flow's packets and evaluated at each intermediate router on the path to forward the packets to the destination.

While the topology monitoring, discovery mechanism and the flow export are used to collect information, the Routing Engine is responsible for proactively controlling the flow forwarding in the underlying topology (dashed arrow in Figure 20). It is crucial that individual routing for each flow results in significant challenges, especially w.r.t. performance that must be considered separately. For instance, there is a delay of the first packet of each flow that is defined by the time required for performing a prediction and determining the routing decision. While using all supported flow features and the DNN architecture described in Section V-B, we measured the elapsed time for a prediction. Measurements without any optimization for 100 000 single flows and batches (100) on a node with an RTX 2080 GPU (Section V-A) resulted in a median delay of $\approx 0.5\,\mathrm{ms}$ and $\approx 0.7\,\mathrm{ms}$, respectively.

2) Distributed Approach: A distributed scenario is also conceivable. Each router serves as a *Prediction-Based Flow Router* (PBR) that implements the Topology Monitor, the Flow Processing and the Routing Engine (Figure 21). Thereby, only a local topology view is available. Each PBR can either use local flow-based prediction results to share traffic load resulting from multiple flows across connected links, e.g., for improved ECMP. Alternatively, prediction results from other PBRs are distributed and used for path programming. In both cases, a Flow Routing Protocol is required that shares the local state on a global level or the network state plus flow-based prediction results to combine them at each router. In addition, there is a delay that affects the first packet of each flow.

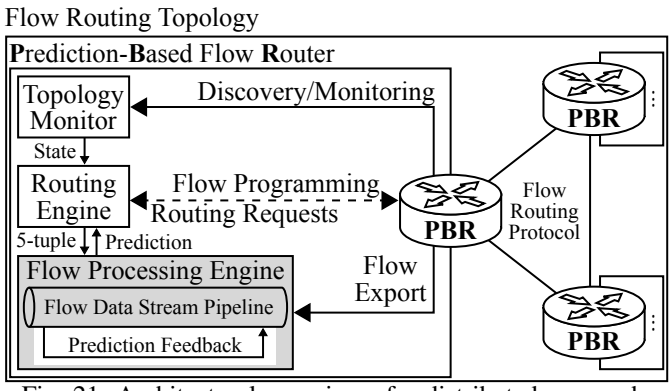


Fig. 21: Architectural overview of a distributed approach.

A hybrid approach, in which distributed and centralized entities form a comprehensive solution, e.g., organizing the topology state in a centralized controller and implementing the other modules in a distributed manner, is possible as well.

The Flow Processing Engine (highlighted gray in Figures 20 and 21) provides the basis for the conception of the Routing Engine. The *Prediction Feedback* mechanism aims at the improvement of the forecasting. Effectiveness is evaluated by comparing a prediction with the corresponding flow export after the transmission of the flow is completed. A further optimization is possible by applying reinforcement learning in the context of a routing decision and the topology's state.

VIII. Conclusion and Future Work

We propose a flexible flow data stream processing pipeline to train machine learning models (here DNNs) on real-world network flow data for the prediction of flow characteristics that can be used for network traffic optimization.

On the one hand, analyzing the observed flow data stream from a productive campus network reveals an unequal data distribution for all considered flow labels, which makes the definition of application-oriented classes with a balanced number of flow data samples challenging. On the other hand, feature similarities between various flow samples are discovered by applying t-SNE to flow data. Thereby, context-related tagging provides a better understanding of the network data.

Different communication contexts (e.g., only TCP or exclude WiFi), feature combinations (e.g., 5-tuple or all features) and flow labels (e.g., throughput or duration) are investigated. While 240 experiments, each lasting one week, show that the forecasting is feasible for streamed flow data, the benefit of our proposed enrichment strategy like an improved or stabilized accuracy depends on the setup. Our findings show that the prediction accuracy varies for different flow labels and depends both on the selected communication context and the available features. This limits the individual operational capability for certain application scenarios.

Furthermore, we found that using the Adam optimizer instead of vanilla Stochastic Gradient Descent is problematic for a continuous learning on flow data (e.g., accuracy instabilities). Instead of only differentiating between small (mice) and large (elephant) flows, we use a multi-class model. This allows for a more fine-grained classification for an improved traffic engineering, especially in combination with network automation.

The codebase for the streaming setup and a reference dataset are available to reproduce the experiments.

We see the following points as avenues for further work:

- *Class Definition*: A process for the definition of classes (number, boundaries) that is based on application requirements rather than pure data statistics needs to be developed.
- *Dealing with Class Imbalance*: We plan to evaluate data generation and oversampling methods to compensate the imbalanced number of examples for each class, and to ensure that suddenly occurring class imbalances do not disrupt the training of DNN classifiers.
- *Robust Continuous Adaptation*: A continuous adaptation of the prediction model based on network changes or seasonal effects is essential to ensure a sustained prediction accuracy. Training several classifiers at different time scales might be beneficial, since this provides fallbacks if a classifier trained on short-term information only is degraded by a short-term change in data statistics. This was observed for several of the long-term experiments we conducted for this article. Lastly, it is planned to investigate other online machine learning models that do not share the limitations of DNNs w.r.t. incremental learning capacity and class imbalance or that can handle imbalanced regression problems. For these investigations, it is imperative to systematically consider datasets that are even larger and more extended in time.
- *Stochastic Gradient Descent Optimizer Investigation*: A detailed analysis of the accuracy instabilibities observed when utilizing the Adam optimizer for the DNN models remains an open issue. In order to investigate the effect and impact of different strategies for a continuous learning on flow data, a more comprehensive study of optimizers proposing enhancements for Stochastic Gradient Descent is necessary.
- *Generalizability Evaluation*: To study the generalizability of our proposed approach, we aim at getting access to flow data from other network setups, e.g., data centers, or the evaluation of other data sources like Internet traces.
- *Integrating Forecasting Results in Flow-based Routing*: Forecasting results can serve as basis for a prediction-based flow routing. To evaluate this approach, a network emulation environment that contains the architecture given in Section VII-B is under development. Besides an adaptive routing, this includes network monitoring mechanisms like (in-band) telemetry or data plane supported probing for maintaining the state of the network topology.
- *Programmable Data Planes for Flow Export*: Switches with a programmable data plane (e.g., P4) are planned to be evaluated as flow exporters. This allows for a flexible export of features and can provide more fine-grained flow information like timestamps. Additionally, the switches might implement a data enrichment strategy at the network level.

Acknowledgements

We thank Sven Reißmann from the university data center for assistance with data collection and data preparation.

Christoph Hardegen received his B. Sc. and M. Sc. degrees in applied computer science from Fulda University of Applied Sciences in 2015 and 2018. He is currently pursuing his Ph. D. degree at Fulda University of Applied Sciences. His research interests include traffic engineering, especially the application of machine learning results to routing in programmable networks.

Benedikt Pfülb received his B. Sc. and M. Sc. degrees in applied computer science from Fulda University of Applied Sciences in 2015 and 2018. He is currently pursuing his Ph. D. degree at Fulda University of Applied Sciences. His research interests include neural networks and artificial intelligence, especially incremental and life-long learning.

Sebastian Rieger received his Diploma degree in computer science from Fulda University of Applied Sciences in 2003. In 2007 he received his Ph. D. degree from the University of Göttingen. He was a research assistant at Gesellschaft für wissenschaftliche Datenverarbeitung mbH Göttingen and Steinbuch Centre for Computing. Since 2012 he holds a full professorship for Multimedia Communication Networks at Fulda University of Applied Sciences. His research interests include network management and virtualization, automation and cognitive management and data center networks.

Alexander Gepperth received his Diploma in physics from LMU Munich in 2002. In 2005, he received his Ph .D. from Ruhr-Universität Bochum. He worked at Honda Research Institute (Offenbach, Germany) as a Senior Scientist from 2005 to 2010, and as an assistant professor at ENSTA ParisTech until 2016. Since 2016, he holds a full professorship for Machine Learning at Fulda University of Applied Sciences. His research interests include continual machine learning as well as cognitive architectures.