Flow-based Throughput Prediction using Deep Learning and Real-World Network Traffic

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

***Abstract*—We present a processing pipeline for flow-based throughput classification based on a machine learning component using deep neural networks (DNNs) that is trained to predict the likely bit rate of a real-world network traffic flow ahead of time. The DNN is trained and evaluated on a flow data stream as well as on a reference dataset collected from a university data center. Predicted bit rates are quantized into three classes instead of the common binary classification into "mice" and "elephant" flows. An in-depth description of the data acquisition process, including preprocessing steps and anonymization used to protect sensitive information, is given. We employ t-SNE (a state-of-the-art data visualization algorithm) to visualize network traffic data, thus enabling us to analyze and understand the characteristics of network traffic data and relations between communication flows at a glance. Additionally, an architecture for flow-based routing utilizing the developed pipeline is proposed as a possible use-case.**

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

I. Introduction

Traffic Engineering has been an ongoing field of research since the early days of computer networks [**?**], [**?**]. Recent advancements in machine learning have led to new approaches that improve network and service management in this area, 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 Multiple Paths (ECMP) [**?**]. A significant effort has been put in the classification of "elephant" and "mice" network flows to allow for an even utilization of individual links, as, e.g., described in [**?**].

When a link or path is selected for a flow, the traffic amount it will produce is not known a priori. As flows cause different traffic volumes, two key problems arise: First, using only a single optimal path (e.g., the shortest) while alternative paths exist can result in high load or congestion on this route leading to unequal utilization. Second, multiple paths with equal cost have to be shared equally to balance the load properly. Because individual flows typically have to be forwarded over the same path during their lifetime, reactive load distribution of network flows across multiple links or paths is not always possible. For example, this is caused by stateful network components like firewalls and middleboxes in current networks [**?**], [**?**]. Hence, 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 of network flows based on the prediction of flow metadata.

In this paper, we focus on a *flow data stream pipeline* to train and deploy a deep learning based prediction model for the forecasting of network flow features. Additionally, a high-level architecture that uses the pipeline for flow routing is proposed. Compared to existing traffic classification mechanisms (e.g., distinguishing elephant from mice flows [**?**]), we introduce a bit rate estimation categorized into multiple traffic classes. This allows for a more fine-grained load balancing and traffic engineering of flows within a network. 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 data for machine learning. To this effect, a systematic collection of network traffic metadata was conducted in our university campus network. To address the constant change of collected traffic characteristics (i.e., concept drift), we use a stream processing approach over consecutive intervals to achieve continuous learning and adaptation of the prediction model. In this setting, an efficient implementation of the processing pipeline is required to keep up with the incoming stream of collected network flow data. Besides these challenges, an obstacle to process flow data and to obtain a realistic dataset is privacy. The collected flows contain IP addresses, which can be viewed as sensitive information. Accordingly, anonymized data is used at all stages of data processing to ensure privacy protection.

II. Related Work and Contributions

A survey of techniques for traffic classification using machine learning is given in [**?**], [**?**] and [**?**]. A recent example for the use of machine learning to classify flows and their bandwidth in data center networks is presented in [**?**]. This study is based on a simulation of data center networks. In contrast, network traffic flow characteristics in real-world networks and data centers are often fluctuating and complex, as, e.g., observed in [**?**]. Other related work in the area of using machine learning techniques for routing can be found in [**?**]. [**?**] presents an SDN-based adaptive flow traffic engineering framework. Like multiple other publications, these papers focus on two traffic classes of large (elephant) and small (mice) flows. A generalized approach to use machine learning for

978-3-903176-24-9

routing can be found in [**?**]. This publication proposes the use of deep reinforcement learning, but it is again not using real-world network traffic data. [**?**] introduces the combination of learning from existing Dijkstra-based routing algorithms and imitating it with higher performance using a dynamic routing framework for SDN. This framework focuses on the optimization of network throughput. It also uses traffic data from a simulated instead of a real-world network topology.

Our developed flow data stream pipeline can be combined with network management and monitoring, e.g., as proposed for knowledge-defined networking [**?**] or cognitive network management [**?**], e.g., joined with existing SDN [**?**] and Network Functions Virtualization [**?**] solutions.

The **main contributions** of this article are as follows:

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

III. Architectural Overview

A closed-loop architecture in which the proposed prediction of flow characteristics can be utilized is depicted in **??**. The network topology is required to be built of (hybrid) SDN-enabled devices (e.g., P4 [**?**], OpenFlow [**?**]) that forward network traffic within the topology. These devices are able to share their state and configuration with a centralized control instance (e.g., via SNMP [**?**], NETCONF [**?**]) that also receives exported flow information from these devices (e.g., via NetFlow [**?**], IPFIX [**?**]). 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 trained Deep Neural Network (DNN). All required operations are realized through a flow data stream pipeline that performs data collection, preparation and machine learning (see **??**). The *Topology Monitor* continuously maintains a state model of the routing topology, e.g., path capacities and load. To route individual flows, a network device sends a routing request to the central *Routing Engine* that queries the Flow Processing Engine with the corresponding 5-tuple (source/destination IP address, source/destination port and transport protocol) describing the flow. It receives a forecasting of one or more relevant flow characteristics, i.e., the amount of transferred bytes and/or packets, the duration as well as the bit rate. Knowing a flow's bit rate and/or duration at the beginning of a communication can be used for resource-efficient routing, e.g., to equally spread the traffic load across multiple links/paths. The Routing Engine evaluates the topology's state and, considering the predicted flow characteristics, selects an appropriate path within the topology. Afterwards, the path is programmed on affected network devices.

While the topology monitoring and the discovery mechanism as well as the flow export are used to gather relevant information, the Routing Engine is responsible for proactively controlling the flow forwarding in the underlying topology (dashed arrows in **??**). It is crucial that individual routing for each flow results in significant challenges, especially w.r.t. performance, which can (partly) be addressed by some form of aggregation, e.g., considering IP flows or entire IP subnets.

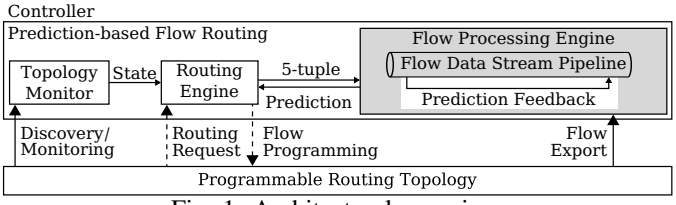


Fig. 1: Architectural overview.

Besides a centralized approach that leverages a global topology view, a distributed or hybrid solution is also conceivable. In a distributed scenario, each router has to implement the monitoring as well as the Flow Processing and Routing Engine individually, with only a local topology view being available to it. Local results can be used to share traffic load across directly connected links (e.g., improved ECMP). Another distributed alternative would require a network protocol that distributes prediction results of edge devices for path programming.

The focus of this paper is placed on the centralized Flow Processing Engine (highlighted gray in **??**), which provides the essential basis for the conception of the Routing Engine that will be implemented as a next step. The same applies to the Topology Monitor as well as the *Prediction Feedback* mechanism improving the forecasting. Effectiveness is evaluated by comparing a prediction with the corresponding flow export after the transmission of the flow is completed. Next to flow characteristics, the prediction of topology features (e.g., link or path latency) is considered as a further extension. A (weighted) combination of both result types can be leveraged for selecting adequate routing paths.

IV. Flow Data Stream Pipeline

The Flow Processing Engine (see **??**) includes a flow data stream pipeline (see **??**) that is responsible for data collection and preparation as well as learning from flow data. The codebase is publicly available in a Git repository[1].

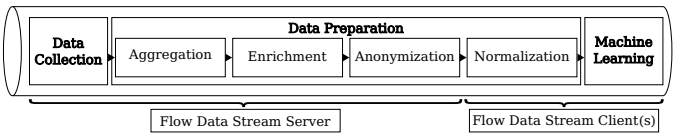


Fig. 2: Flow data stream pipeline.

A. Data Collection

Network Architecture Data collection takes place in a real production network at Fulda University of Applied Sciences. The university has 26 individual local buildings and one off-campus location. The network is designed according to the classic core-distribution-access model, in which each building represents an element in the distribution layer, being connected to two core routers as shown in **??**. Two data centers are also

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

connected to the core layer, providing all central IT services (e.g., directory, file, email, DNS servers). Through the use of CAPWAP [**?**] for the provision of the wireless infrastructure, any WiFi traffic is passing the routers as well.

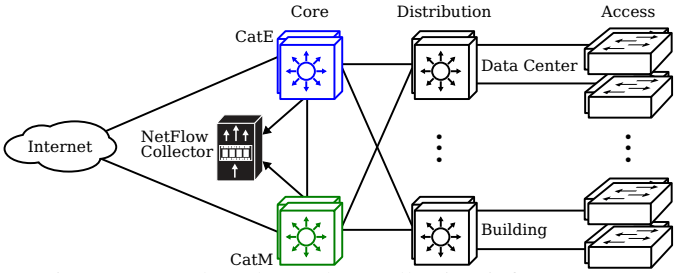


Fig. 3: Network and NetFlow collection infrastructure.

This architecture allows us to export traffic characteristics and metadata from both central core routers using the NetFlow protocol, catching any WiFi, data center, internal and external traffic, except for packets that get routed within a single building at 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 [**?**], an extension to the NetFlow protocol [**?**].

Flow Export Flow information is continuously exported by the core routers in the data center. These central devices connect most of the campus infrastructure. Hence, the network traffic includes a variety of realistic traffic patterns. Collected flow information describes internal and external as well as client-to-server and server-to-server communications.

A configured flow record selects flows to export based on a set of matching criteria (i.e., the 5-tuple). Additional information is added to the flows before they get exported to the collector (collect criteria). This metadata consists of timestamps for the flow's start and end time, the number of packets and bytes transmitted, as well as a union of all TCP flags found in any packet belonging to the flow. A flow monitor is defined, which references the flow record as well as the configured flow collector. Specified timeouts ensure that flow information is exported at regular intervals, even if individual flows last for a long time. An active timeout of 300 s and an inactive timeout of 30 s are used.

To avoid duplicated exports from one router, flow records are exported only for ingress traffic. The collector handles exported flows (∅ ≈2 000 records per second). Because only unicast flows are considered, multicasts and broadcasts are filtered out. After receiving a collection of about 100 000 consecutive flow records (*block*), data preparation is initiated.

Furthermore, depending on the routing, a flow can pass through both routers. As both devices export data, the same information is potentially collected multiple times. These duplicate flow records are filtered as part of the aggregation stage in the flow data stream pipeline (see **??**). Thus, only unique records are further processed.

B. Data Preparation

To prepare flow information for the training performed in the machine learning module, the following operations 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 realize 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 a dataset. The streaming clients normalize the received flow data and assign class labels for classifier training (see **??**). All data preparation operations are parallelized on both the 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 maintain *flow entries* that represent an entire communication, flow records are aggregated block-wise. The applied method is based on the 5-tuple, timestamps and flags. Flow properties like timestamps, the duration, the number of packets and bytes as well as the bit rate are updated. Because of the timestamp resolution (ms), 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 internal and global topology (e.g., private prefixes, VLANs and public prefixes, ASNs), based on a flow's source and destination IP address. IPAM exports as well as a lookup service are used as data sources. Data enrichment enables the consideration of different communication contexts and levels.
Moreover, dynamically chosen ports $\geq 2^{15}$ are replaced by 0.

3) Data Anonymization: To guarantee 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.

4) Data Normalization: To feed the model (e.g., the DNN) with suitable inputs, a normalization and/or data format transformations are performed. Three formats are supported: *float*, *bit pattern*, *one-hot* (e.g., see **??**). 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 1.0.

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 **??**. For each available feature, the supported formats with the size of the resulting output vector are stated. Data transformation types marked in gray are used for the experiments. The origin 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			Origin
	Float	Bit	One-hot	
month	1	4	12	DC
day	1	5	31	
hour	1	5	24	
minute	1	6	60	
second	1	6	60	
protocol	1	8	✗	
address ⇄	4	32	✗	
port ⇄	1	16	✗	
network ⇄	4	32	✗	DE
prefix_len ⇄	1	5	✗	
asn ⇄	1	16	✗	
longitude ⇄	1	✗	✗	
latitude ⇄	1	✗	✗	
country_code ⇄	1	8	240	
vlan ⇄	1	12	✗	
locality ⇄	1	1	2	

DC = Data Collection; DE = Data Enrichment

C. Machine Learning

This module implements the training and inference stages of a Deep Neural Network (DNN). Various online and offline (i.e., not operating on live data streams) experiments can be performed to evaluate the validity of different machine learning models (e.g., DNNs or Support Vector Machines), their hyper-parameters (e.g., number and size of layers for DNNs) and/or data normalization and enrichment strategies. When doing so, either for model training or inference, the prepared flow data is always processed in a block-wise fashion.

Since data normalization and machine learning are performed on the client side, sub-datasets can be extracted from a block of data and used as training and test set (splitting and shuffeling). The slicing of flow information (feature selection) as well as the filtering of flow feature values (feature filtering) are supported, e.g., excluding WiFi traffic or considering specific transport protocols. Each flow entry is assigned a class using predefined class boundaries for the selected flow property, e.g., the bit rate. Flow properties that merit being predicted by the model are termed *labels* in the subsequent text. The `number of bytes`, the `number of packets` as well as a flow's `duration` and `bit rate` are supported as class labels. All labels are determined during the data collection respectively updated in the data aggregation stage and afterwards transformed into the one-hot format. The machine learning stage outputs the predictions of the trained machine learning model for individual network communications.

V. Network Flow Analysis

By monitoring the number of current cache entries, as well as the high watermark values, we can ensure that the devices are capable of exporting all flow information without hardware restrictions like cache sizes being the limiting factor. The CPU and memory utilization do not increase significantly while exporting flow information. **??** shows the trend of the NetFlow cache sizes on both switches for a selected week (2019-06-03 16:50 to 2019-06-10 16:50), with no significant rise on 2019-06-10, which was a national holiday.

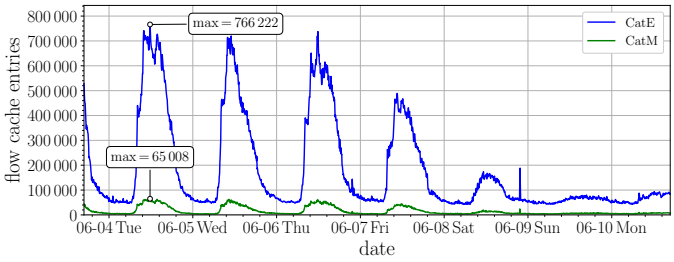


Fig. 4: Flow cache entries for both central network switches (CatE, CatM) for one week (5 min resolution).

The incoming traffic volume for both switches is depicted in **??**, and one can observe that the trend of the amount of summed-up traffic correlates with the trend of the number of cache entries. Outgoing traffic shows the same pattern, but the rates are much less and roughly equal for both routers. For simplicity's sake, their depiction has been omitted.

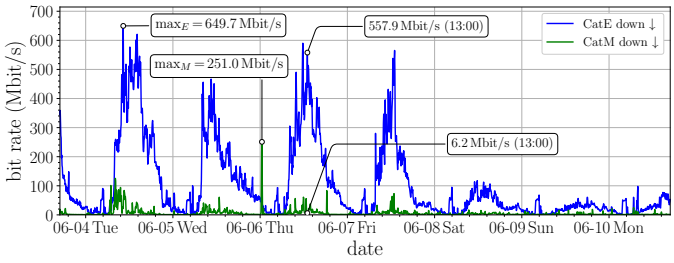


Fig. 5: Monitored network traffic (downstream) for both network switches (CatE, CatM) for one week (5 min resolution).

?? outlines the average number of flow records received by the collector for each block during a continuous interval of one week. As the collection time for a block is directly related to the number of received flow records, the corresponding trend can also be derived from the same figure.

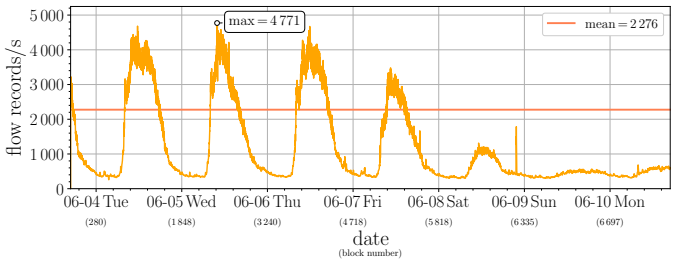


Fig. 6: Average number of flow records per second for each data block during a continuous week.

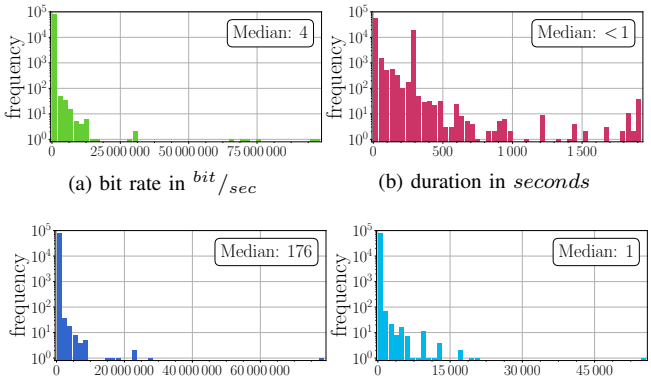
As expected, the trends in **??????** show that there is significantly less traffic at night and on the weekend.

VI. Traffic Analysis Experiments

The data distribution was investigated based on the chosen flow labels. For the traffic analysis, unless otherwise stated, we selected the training proportion from a block (5413) of aggregated flow entries (2019-06-07 at about 14:00) out of our stored reference dataset[2]. The dataset contains about 7 000 blocks with overall approximately 525 million flow entries collected for a continuous week (2019-06-03 16:50 to 2019-06-10 16:50). Existing structural patterns within the flow data are visualized, while, for the sake of clarity, only the first 1 000 flow entries of the selected block are used. Both steps help to provide a better understanding of the data. Regarding the analysis' results, variations were recognized for previous or following blocks, but the trend remained the same when considering day and night time separately. Self-generated traffic patterns were used to verify the pipeline operations and supported the validity of the analysis' results.

A. Label-based Data Distribution

?? outlines the data distribution within the complete selected block for each flow feature suitable for class labeling. Considering both the bit rate and duration of a flow (see **????**), most flow communications are active for a relatively short time and/or transmit very few data while being active. The same findings apply to the observed number of transferred bytes and/or packets (see **????**), which are both relatively small.



(a) bit rate in $^{bit}/_{sec}$ (b) duration in *seconds*

(c) number of bytes (d) number of packets

Fig. 7: Histograms of the selected block for all considered flow labels (logarithmic scale).

With regard to **??**, flow data is unevenly distributed (clarified by median values). As the definition of class boundaries that are used for labeling each flow entry depends on the observed flow data, an appropriate determination is challenging. Hence, ad-hoc boundaries for three classes matching the current data distribution are used for our experiments, even though they have to be adapted for a practical application.

?? illustrates the trend of the class distribution across all blocks of the reference dataset. Here, a flow's bit rate is considered as label and predefined exemplary class boundaries in $^{bit}/_{sec}$ are used: class 0 = [0, 50[(red), class 1 = [50, 8 000[(green), class 2 =[8 000, ∞] (blue). While the flow data proportion for each class remains approximately the same, there are minor deviations for individual classes. Therefore, class weighting methods are also applied to address an imbalanced number of elements per class within a block.

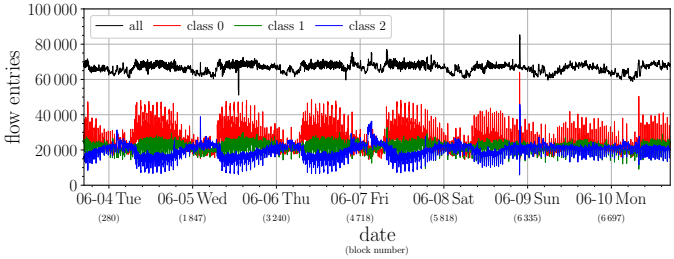


Fig. 8: Class distribution for all blocks of the reference dataset.

B. Structural Data Patterns

t-distributed Stochastic Neighbor Embedding (t-SNE) [**?**] is used to discover and visualize structural data patterns based on an Euclidean distance metric. t-SNE performs dimensionality reduction by mapping high-dimensional data to a space with lower dimension (2D or 3D space) while preserving neighborhood relations as much as possible. **??????** share the same t-SNE output. The tagging is context-related. Similarities between flows (multi-dimensional feature vectors) are indicated by the relative position of each data sample, though the absolute positions can be neglected.

The differentiation of analyzed flow entries based on the used transport protocol is shown in **??**. 82.6 % of the flow data samples belong to UDP (●) and 17.2 % to TCP (▲) traffic. The remaining proportion of 0.2 % (■) describes communications for other protocols like ICMP. For each transport protocol, there are multiple accumulations of data samples, which indicate feature similarities. Symmetric spots for each accumulation can be identified.

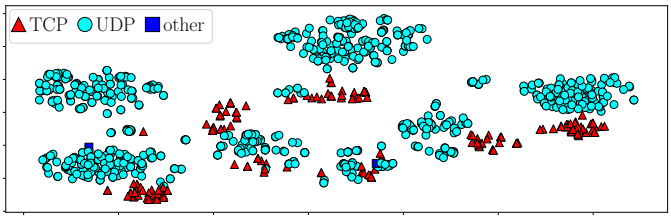


Fig. 9: Transport protocol-based tagging of the t-SNE results.

?? depicts the tagging of each data sample according to the communications' locality. Based on the locality of the source and destination, four different communication types are distinguished. 6.4 % of the 1 000 flow entries are related to communications between private systems (□), the other 93.6 % involve at least one public system (i.e., communication to or from the public Internet). While 39.4 % (▲) of the flows are describing communications between publicly addressed systems, 27.0 % (●) respectively 27.2 % (✚) belong to a communication between an internal and external system, i.e., inverse directions. Again, symmetric accumulations for each communication type can be determined. 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

[2]The dataset is available upon request.

traffic (33.1 %) is separately marked and strengthens this fact.

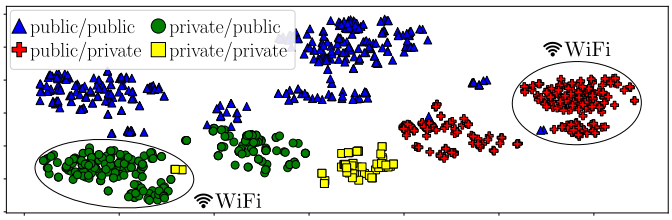


Fig. 10: Locality-based tagging of the t-SNE results.

The tagging of the t-SNE output with symmetric accumulations based on the application protocol for each flow entry is delineated in **??**. With 77.0 %, most of the flow entries are classified as DNS traffic (●), which is interrelated with the huge proportion of UDP flows (see **??**). 13.8 % of the flow entries belong to HTTP(S) traffic (◆) and another 9.2 % to other application protocols, e.g., SNMP, LDAP, SMTP (▲).

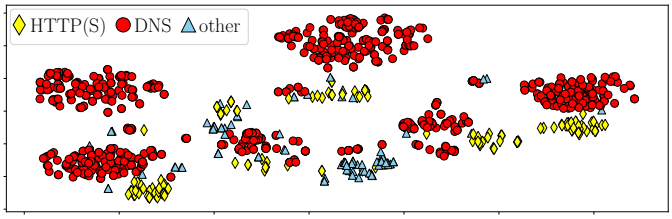


Fig. 11: Application-based tagging of the t-SNE results.

According to the accumulations of individual flow data samples and their symmetric pair spots, the t-SNE result shows feature similarities for several flow entries. The context-based tagging illustrated in **??????** helps to further specify respectively clarify the existing structural patterns in the data.

Besides the fact that many communications require 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 and hence are externally forwarded. Flows that describe DNS communications normally have either only one exchanged packet or both a small number of transferred bytes and a short duration (short-term). Thus, those flows and similar ones can be excluded because the prediction is practically not relevant.

?? gives an overview of flow-based volume characteristics for all data blocks. While approximately 69 % of all flows belong to DNS, their volumetric proportion is small.

TABLE III: Volume metrics for all blocks (mean/median).

Protocol (flow proportion)	**KBytes**	**Packets**	**Duration** (seconds)	**Bit rate** (Kbit/s)
TCP (19.4%)	155.4/84.7	172.8/93.6	23.7/2.0	192.9/5.1
UDP (79.9%)	6.4/ 3.8	16.0/12.8	22.5/0.1	14.3/2.3
DNS (69.4%)	0.2/ 0.2	1.4/ 1.5	17.7/0.1	5.6/2.2

C. Verification of the Flow Data Stream Pipeline Stages

To verify the operation of the data collection and each stage in the preparation phase (see **??**), self-generated flow communications were injected into the network, e.g., using iperf, and retraced. Our strategy considered various application and transport protocols, varying bit rates and durations (either including interruptions for predefined time intervals or not) for both internal and external communications.

The completeness of collected flow data was also exemplarily checked. Therefore, the amount of incoming public traffic was compared to the summed up traffic data of related flow entries for a specific time interval (about 5 minutes, 2019-06-06 at 13:00). The overall average bit rate for flow entries (starting between 12:55 and 13:00), calculated based on the accumulated number of bytes for each entry and the given time interval in seconds, is approximately 450 Mbit/s. The added up bit rate for both central routers (see additional markers in **??**) is 564 Mbit/s, which roughly matches the determined value for the blocks. As expected, the observed traffic volume differs from our accumulated flow entries, because of the time offset for the flow export. Sensitive networks were filtered out beforehand and protocol overheads were not considered which also explains the lower value.

VII. Flow Prediction Experiments

In our experiments, fully-connected DNN classifiers with varied hyper-parameters (e.g., the number and size of layers) are used to forecast a selected flow label. We choose DNNs because they are efficiently (re-)trainable in a streaming setup (support for online learning). During DNN training on a single data block, standard cross-entropy loss is minimized based on stochastic gradient descent using the "Adam" optimizer. Whereas a ReLU transfer function is applied to each hidden layer, the output layer utilizes a softmax function.

We select a flow's bit rate as class label. While only a single flow label is used here, other flow features or feature combinations are conceivable as well. 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, which prevents overly optimistic test results due to correlated timestamps, which could be used as features by the DNN. Class weighting methods are applied to handle an imbalanced number of data samples for each class. To achieve this, either standard class weighting or under-sampling is used. The latter reduces the number of data elements in each class to the minimum for all classes. With standard class weighting, a weight factor based on the proportion of data samples is determined for each class. Weighting is considered for the training as well as for the testing phase block-wise.

A. Feature Importance

A parameter optimization of roughly 6 000 experiments using another reference dataset [**?**] is performed to evaluate different DNN configurations. The dataset contains 8 h of flow data with about 53 million flow entries that were collected during a normal weekday (2019-02-15 09:00 to 17:00). Tested

DNN parameters include the number of layers $\{3, 4, 5\}$ with different sizes $\{200, 400, 600, 800, 1000, 1500\}$. The learning rate $\{0.001, 0.0001, 0.00001\}$ is varied and the optional application of dropout (probabilities: $0.9/0.8$ for hidden and $0.6/0.5$ for the input layer) is considered. Each DNN is trained sequentially for 10 epochs on the first 10 blocks of the mentioned reference dataset using the training proportion of a data block. Blocks are fed mini-batch-wise to the DNN (100 samples per batch). The prediction accuracy is evaluated based on test data for each block. Also, to investigate the relevance of supported features, feature combinations as well as their pairing with the 5-tuple are extracted from a data block as sub-datasets and fed to the DNN. Considered features respectively pairs are shown in **??**. Features with a source and a destination information are marked with ⇄. The size for the input vector for the DNN is also outlined.

TABLE IV: Importance of feature pairs and combinations.

Feature(s)	**Mean Accuracy in %** (chance improved)			
	Single Feature	Inputs	5-tuple plus Feature	Inputs
all features	**79.3** (+46.0)	247	-	-
5-tuple features	77.4 (+44.1)	104	-	-
ip address ⇄	75.1 (+41.8)	64	-	-
network prefix ⇄	73.8 (+40.5)	74	77.6 (+44.3)	178
VLAN ⇄	71.7 (+38.4)	24	77.7 (+44.4)	128
locality ⇄	63.6 (+30.3)	2	77.3 (+44.0)	106
protocol	62.7 (+29.4)	16	-	-
ASN ⇄	62.3 (+29.0)	32	77.5 (+44.2)	136
geo coordinates ⇄	60.9 (+27.6)	4	77.7 (+44.4)	108
start timestamp	56.3 (+23.0)	5	**78.8** (+45.5)	109
transport port ⇄	43.1 (+ 9.8)	8	-	-
country code ⇄	33.3 (+ 0.0)	2	77.6 (+44.3)	106

For a classification task with an evenly distributed number of elements using three classes, chance is about $33.3\,\%$. Choosing an individual feature pair that is not part of the 5-tuple, e.g., the VLAN tag or prefix information, an improvement of up to $\approx +40\,\%$ can be achieved. Using only the 5-tuple, the prediction achieves an average of about $77\,\%$. The combination of all features from the data enrichment with the 5-tuple information leads to a further enhancement of $\approx +2\,\%$.

Due to accuracy irregularities for some individual features in different blocks, the average and not the maximum value is used to compare feature importance. The combinations of the 5-tuple with each individual feature only exhibit minor accuracy deviations for the investigated blocks.

B. Learning on a Streaming Interval

Four DNNs are trained and tested simultaneously for one week (2019-06-03 16:50 to 2019-06-10 16:50). Flow data is continuously collected as well as sequentially prepared and processed (see **??**). DNN parameters are selected based on a previously performed parameter optimization (see **??**, [**?**]) using the reference dataset mentioned in **??**. As undersampling performs worse on average than class weighting, we exclude it from further considerations. Each block is fed mini-batch-wise (batch size of 100) to the DNN until the next prepared block is available.

?? shows the maximum accuracy per data block using class weighting as balancing method. As network traffic is more dynamic during daytime, the accuracy drops accordingly. The achieved maximum/average accuracies for the experiment with all flow features are $94.1\,\%/82.1\,\%$. In comparison, the ones for the 5-tuple experiment are $91.9\,\%/61.2\,\%$. Regarding the means, a significant improvement, which results from the data enrichment, is recognizable ($\approx +20\,\%$).

TABLE V: Hyper-parameters for the streaming experiments.

Features	**Layers**	**Sizes**	**Learning Rate**	**Dropout Probability**
all	3	1 000	0.0001	(1.0, 1.0)
5-tuple	5	1 000	0.001	(0.9, 0.6)

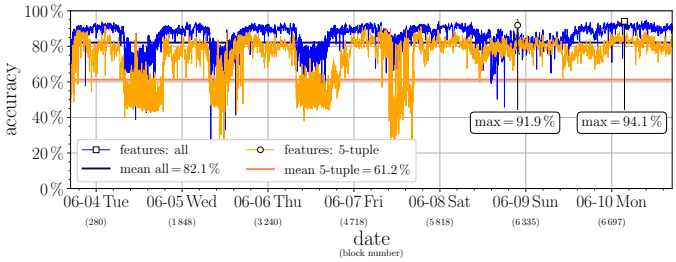


Fig. 12: Maximum test accuracies for each block and overall mean for a continuous streaming interval of one week.

The number of training epochs for each block (see **??**) depends inversely on the number of received flows per second in order to ensure real-time processing capability. Consequently, training and testing are performed for considerably less epochs during the day than at night.

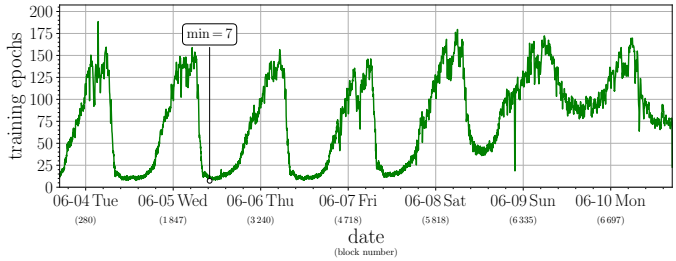


Fig. 13: Number of training epochs for each block for a continuous streaming interval of one week.

VIII. Discussion of Results and Use-Cases

Flow Data Acquisition An issue is the collection and preparation of real-world flow data, which requires an understanding of the network topology (e.g., the behavior of flow exporters). While anonymization ensures privacy protection for processing sensitive information in our experiments and for publishing a dataset, it is not a mandatory functional operation for the proposed solution. Nonetheless, our strategy allows a consistent transformation for consecutive data acquisitions.

Aggregation and filtering methods ensure that flow characteristics are not collected multiple times and only complete flow communications are considered. Due to the block-wise processing of data, the aggregation in turn causes a loss of flow information because flow records not belonging to complete conversations are dropped. Long-term flows ($> 5\,\mathrm{min}$) can suffer from this problem but the model impact needs to be further investigated. We plan to develop a solution, e.g., a window- or cache-based mechanism. To also calculate the duration of short-term flows correctly, a higher timestamp resolution for the flow export would be required.

Moreover, the amount of flow data that needs to be processed in a stream represents another challenge. To handle flow data in time, a parallelization of data preparation steps is necessary. Our solution is vertically scalable and requires a central point for the data collection. Larger topologies with higher data volumes would need an approach that is horizontally scalable. At the same time, it must be ensured that model training is done sufficiently on the observed data.

All pipeline stages support IPv4. While the used flow collection also supports IPv6, all IP address operations need to be modified to handle both address types.

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

Based on all selected labels available for prediction, the data is very unevenly distributed. Hence, the definition of class boundaries (including the number of classes) is challenging. Classes are defined based on the observed data distribution to have an approximately equal number of samples for each class, which is suited for DNN training. To ensure applicability in practice, e.g., for adaptive flow routing, the adaption of the boundaries is necessary. At the same time, sufficient data samples for each class are required.

Network traffic contains a huge proportion of flow communications that do not necessarily require prediction (e.g., short-term flows like DNS) and thus can be excluded.

Results of the Experiments Our experiments show that a multi-class bit rate prediction for streamed flow data is feasible with DNNs. Compared to binary classification, a multi-class problem is more challenging. In contrast to using synthetic data, a real-world network scenario represents a more credible setup. Considering the prediction as two-class problem, our exemplary experiments reach more than $90\,\%$ accuracy.

Regarding the investigation of feature importance, processing only the 5-tuple for the prediction achieves similar results compared to considering all features. Here, the data enrichment is less important (about $+2\,\%$) but can provide an increased accuracy in other scenarios. For example, in our streaming experiments, an average improvement of up to about $+20\,\%$ is achieved through the data enrichment.

Comparing the results, learning on a stream profits more from enrichment. While the initial feature importance experiments 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 than at night. These results are related to the number of encountered training epochs, which depends on the amount of received flows.

In general, our pipeline setup 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.

A statement about the generalizability of our approach remains an open issue, because no comparable datasets exist with which our solution can be further evaluated. Additionally, reproducible simulations for practical transferability are needed to evaluate the actual benefits in real environments.

Use-Cases The architecture in **??** can be deployed for various flow routing scenarios, e.g., in network automation platforms. First of all, forecasting can be used to enhance the performance of a flow respectively maximizing its required resources. A suitable path for a communication in the topology can be determined based on the predicted flow characteristics. Furthermore, flow-based prediction offers potential for the optimization of link or path utilization in network devices capable of using multiple outgoing links/paths (e.g., using routing or multi-pathing), whereas a round-robin or hash-based selection of the outgoing link to forward a flow's packets can result in uneven resource utilization respectively congestion.

Forecasting results can also be employed to implement policing, shaping or other QoS-related constraints on the flows.

IX. 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 a flow-based bit rate prediction that can be used for throughput optimization, improved network utilization or flow routing. In our experiments, the forecasting of a flow's bit rate achieves an average accuracy of $82\,\%$ within a continuous interval of one week. To reproduce the experiments, the codebase for the streaming setup and a reference dataset are available. Instead of only differentiating between large (elephant) and small (mice) flows, we use a multi-class model. Generally, this allows a more fine-grained classification for an improved network traffic engineering, especially in combination with SDN.

To enhance the practical applicability, an appropriate definition of classes (number and boundaries) that are currently specified based on the data distribution and not on topology information like link/path capacities is required. We plan to evaluate data generation methods, e.g., Generative Adversarial Nets [**?**], to compensate the imbalanced number of examples for each class. A continuous adaption of the prediction model based on network setup changes or seasonal effects is essential to ensure a sustained prediction accuracy. Next to DNNs, other online machine learning models need to be evaluated.

Forecasting results can serve as the basis for a prediction-based flow routing. To evaluate this approach, a network emulation environment based on Mininet [**?**] that contains the architecture and the proposed controller components given in **??** is under development. Besides an adaptive routing, this includes advanced network monitoring mechanisms like network telemetry. The consideration of classical and programmable routing topologies allows the comparative evaluation of both strategies.