©IEEE. Personal use of this material is permitted. Permission from IEEE must be obtained for all other uses, in any current or future media, including reprinting/republishing this material for advertising or promotional purposes, creating new collective works, for resale or redistribution to servers or lists, or reuse of any copyrighted component of this work in other works.

Network Function Offloading through Classification of Elephant Flows

Raphael Durner, Wolfgang Kellerer Chair of Communication Networks
Department of Electrical and Computer Engineering
Technical University of Munich, Germany
Email: {r.durner, wolfgang.kellerer}@tum.de

***Abstract*—With the move from traditional hardware appliance based network functions to Network Function Virtualization, software development is decoupled from the hardware. However, as a network function is no longer optimized for hardware, beneficial features of networking hardware may not be used any more. Solutions such as SDN or NIC offloading aim to overcome this antipodes by integrating networking hardware into the packet processing pipelines. On the one hand, offloading traffic of network functions to hardware can increase throughput and reduce resource consumption. On the other hand, the number of parallel flows in a network can be very high, exhausting the capacity of the tables of the networking hardware and force the system to fall back to software processing. Fortunately, it is known that a large portion of the flows in the internet are mice flows, whereas the majority of the traffic is constituted by elephant flows. If the elephant flows can be detected efficiently, the hardware tables can be used more efficiently as a larger share of the traffic can be offloaded. We introduce a machine learning based approach that takes its decision with the first packet of a flow. A fundamentally different approach is using packet sampling for the offloading decision. We are evaluating both approaches in terms of complexity, offloaded share of the traffic and table occupation. The results show that a machine learning based offloading decision is possible with the first packet. The sampling approach only reaches a comparable performance at very high sampling rates.**

***Index Terms*—Network function virtualization, Machine Learning, Software-defined networks, Performance management.**

I. Introduction

With the move towards Network Function Virtualization (NFV), network functions are realized in software, implemented on commodity hardware. This reduces costs and increases the flexibility, as virtualization techniques support sharing of hardware resources. On the downside, these novel Virtualized Network Functions (VNFs) may provide lower throughput when compared to hardware-based network functions, as they cannot benefit any longer of specific hardware features. Networking hardware commonly relies on hardware tables that match on packets, more specifically on packet headers. A hardware switch has a forwarding information base that matches on the MAC addresses of the packets, a router has its routing information base matching on the destination IP addresses. Traditionally, hardware tables are used in routers and switches as they are able to provide high performance. SDN and especially OpenFlow introduces means to program these hardware tables. Moreover, also other networking hardware such as Network Interface Cards (NICs) are providing programmable hardware tables nowadays.

VNFs solely rely on software but also use tables. For example a stateful firewall needs to hold a table that stores the mapping of connections. In contrast to a stateless firewall that only filters based on the packet header, a stateful firewall also considers the state of the connection. More specifically in case of TCP, at first only SYN packets are accepted and with the SYN packet an entry is added to the connection table. Other packets are only accepted if the corresponding entry is in the table and if TCP's state machine was followed. Likewise many other stateful network functions such as load balancers or Network Address Translators (NATs) maintain tables. Further, all these stateful network functions handle the first packet of a flow separately; As the first packet of a flow does not match the table, a corresponding entry has to be created in a table.

It has been shown in recent works that the usage of network hardware is beneficial to support VNFs. In the next section we show two detailed example that combine hardware tables and stateful VNFs to increase the performance. From the examples we can see, that networking hardware provides high throughput, but its capacity is limited in terms of table entries. As the number of connections can be very high, the applicability of hardware based approaches is limited. Software tables do not suffer from this limitation as they can theoretically be indefinitely large, but they provide worse performance.

From a performance point of view, it is desirable to use hardware tables for as much traffic as possible, while keeping the number of concurrent connections within the capacity of the hardware. Fortunately, it is known that few but large elephant flows comprise the bulk part of the internet traffic. Therefore we propose an offloading approach that uses offloading to hardware processing nodes for elephant flows. For efficiency it is desirable to classify the flows with the first packet, as the first packet is used to create a new entry in the connection table. In this way, the specific processing of the first packet can be used for the offloading decision at the same time. We evaluate the offloading decision based on the number of rules that are needed and how much of the overall data rate can be offloaded. The better the offloading decision is, the higher the rate that is handled in hardware while keeping the table size small enough. The main benefit of NFV offloading is that the resources that are consumed by the VNFs are reduced.

As the capacity of the VNFs is adapted to the demand by scaling up and down, a larger offloaded share decreases the consumption of resources in the NFV infrastructure.

The contributions of this work are as follows:

- Designing a classification based on machine learning for the offloading decision with only the first packet of a flow. Our NFV offloading solution includes identification of features and pre-processing of the data in order to improve the performance.
- Design of an improved heuristic for a simple sampling based approach for NFV offloading. We show that the performance of sampling based approaches heavily depends on the sampling rate.
- Comparison of both approaches and discussion of the pros and cons of each solution.

We belief that the results can be applied for a wide range of network functions and architectures. First, many network functions require matching on packets, two examples are given in the next section, nevertheless also other network functions such as billing functions need to match packets. Second, the offloading cannot only be done via SDN devices or NICs. The same decision mechanism can also be applied for other emerging programmable networking hardware such as, e.g., P4 enabled hardware.

The rest of this paper is structured as follows: Section II introduces two use cases for the presented offloading algorithms. In Section III related work in the fields of network function offloading and elephant flow detection is discussed. Section IV introduces the machine learning approach, the evaluated algorithms and shows a way how the data can be gathered. Afterwards the sampling based approach is introduced in Section V. In Section VI and VII both approaches are evaluated in terms of possible share of the offloaded rate and necessary table size. Furthermore different parameters are studied in order to fine tune the approaches. We discuss the findings and compare both approaches directly in Section VIII. Finally Section IX concludes the work.

II. Example use cases

In this work, we explore how to combine networking hardware tables and software network functions. We use an enterprise network firewall and a load-balancer architecture as examples to provide a more specific view on our problem scope.

A. *Firewall offloading*

The first use case scenario is depicted in Figure 1. The NFV Infrastructure (NFVI) is a data center like centralized infrastructure, hosting the virtual firewall (vFW) instances. If every connection is filtered using vFWs the capacity of the links to the NFVI must be high. Otherwise a few rate intense flows can lead to capacity bottlenecks and poor performance. One solution is to offload the connection filtering to the SDN switches and to use the direct path. Offloaded flows are filtered in SDN hardware, reducing the resource consumption in the NFVI and in the network by avoiding detours to the NFVI.

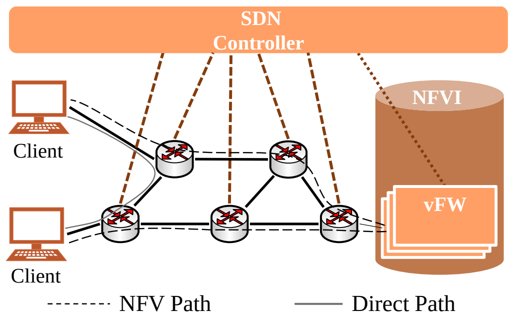


Figure 1. Network Function Offloading in an SDN/NFV environment. The virtual firewall (vFW) filters the packets. As the infrastructure is centralized, the NFV path takes a detour to the vFWs. To avoid such a detour, connections can be offloaded for filtering to the SDN switches on the left side and can then be routed on the Direct Path. However, only connections with high data rate should be offloaded to keep the rule count in the hardware tables of the switches small.

In [1] we showcased this solution with a proof-of-concept using Linux netfilter as a vFW. First the vFW acts like a normal stateful firewall. With the first packet an entry is added to the connection table. Subsequent packets are then checked if the protocol's state machine is followed. In the prototype the offloading decision was realized by detecting the current run time of a connection. If one connection has been active a certain duration, the vFW a flow is offloaded and a message containing the flow five-tuple is sent via the northbound API to the SDN controller. The SDN controller then installs the necessary rules in the SDN switches and the load on the vFWs is reduced. In the prototype setup the throughput could be increased by one magnitude using this approach.

This simple approach is clearly in need of improvement as it does neither consider the table size nor it is able to offload connections from the first packet. But it confirms that already a simple approach is of benefit with respect to the achievable data rate. Due to the high number of connections in the network and limited size hardware tables, not every connection can be offloaded. The presented offloading algorithms can be used to decide which connections should be offloaded to the SDN switches.

B. *NIC Offloading*

As a second use case, we want to highlight the utilization of Smart-NICs for offloading. Modern NICs provide packet matching capabilities that can be used to reduce the load on the CPUs of the NVFI. One example is the Intel FlowDirector technology that enables a programmable matching on the NIC. In this case, we are considering a stateful load-balancer. Load-balancers are used to distribute the packets between different endpoints. In current applications, e.g. web-apps, it is desirable that packets of one connection are always served by the same endpoint. Stateful load balancers ensure this connection consistency by storing the connection to endpoint mapping in a table.

The architecture of such a solution is given in Figure 2. The first packet is always matched in software and the mapping is stored in a software table. The load balancer installs the mapping in the NIC as well. Subsequent packets are then matched by the hardware. Although the NIC cannot directly

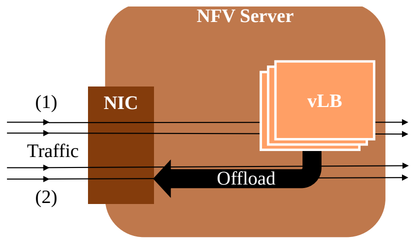


Figure 2. Offloading packet matching to the NIC. Modern NICs provide matching capabilities that can be used to reduce the load on the CPU when using a virtualized load-balancer (vLB). The first packet is always handled in software subsequent packets can be matched in hardware. The table capacity is limited, thus only elephant flows shall be offloaded.

rewrite the packets it can forward them to a specific queue. The load-balancer software can then directly rewrite the packets without any lookup. We have shown in [2] that this approach reduces the load on the CPU and increases the throughput by up to 50%.

The table capacity of the NIC is limited, e.g. for the utilized Intel X550 NIC to 8000 entries. Thus only elephant flows shall be offloaded. In case of traffic that has more concurrent connections than available entries, an offloading decision has to be taken. Even though the use case and the hardware utilized are different from the first example, this decision should follow the same principles: the maximization of the offloaded packet rate with the available table capacity in focus.

III. Related Work

In this section we give an overview of the related work. It can be divided in two fields: Works that introduce techniques for elephant flow detection and works that mainly consider network function offloading.

A. *Network function offloading*

SciPass [3] use an OpenFlow switch for the offloading of an institutional firewall in a science network. The system consists of a 10G Firewall, a 100G OpenFlow switch and the Bro Intrusion Detection System that is used to identify the flows that can be offloaded. The evaluation shows that a firewall bypass can significantly improve performance of the network as the OpenFlow switch has a much higher data rate. In contrast to our approach, SciPass is using a specialized Intrusion Detection System for identification. The work also focuses on the very specific use case of a science DMZ.

NFShunt [4] is a prototype implementation for firewall offloading, realized with a Linux Netfilter based software firewall and a OpenFlow switch, that is used as hardware accelerator. The paper details the use-case of a science DMZ and is showing implementation details of the prototype. In contrast to our analysis, the authors are using static rules for the offloading decision.

Heimgaertner et. al. [5] study firewall offloading that is specifically designed to avoid congestion at the firewall. The authors are using two different algorithms for the offloading decision: A random decision chooses random flows for offloading and a so called intelligent algorithm decides for offloading based on the byte count of the flow. The results show that the bypassing of the firewall can significantly improve performance. Furthermore, it can be seen that the decision algorithm is important, as more load is bypassed using fewer rules in the OpenFlow switch with the intelligent algorithm than with the random algorithm.

There are also several existing works on NIC offloading that show that such approach can increase performance: Firstly NIC offloading can be used for firewall offloading. Authors in [6] utilize 5-tuple filtering in the NIC and show that offloading firewall functions to NIC can improve both CPU utilization as well as packet throughput.

Further, with NDN-NIC [7] the authors propose NIC offloading for name-based filtering. They utilize bloom-filters to filter incoming packets to specific names. The pre-filtering of the NIC reduces CPU overhead and the energy consumption of the solution.

Finally, in a recent work Microsoft present their NIC Offloading solution for Azure [8]. They utilize custom NICs with built-in FPGAs to realize network functions such as tunneling. This solution frees CPU resources and increases their revenues by selling this resources to customers.

None of the works on NIC offloading consider limitations in the size of the NIC table. Instead, the above works concentrate on the benefits of the solution and the respective implementation aspects.

B. *Elephant flow detection*

In general there is a lot of work on elephant flow detection, mostly focusing on detecting elephant flows, when they are already elephants. This means, many works concentrate on detecting the flows that at a certain time constitute already most to the overall network traffic, i.e. the redirection comes too late. The overall problem is that the number of network packets is too large in modern networks to be analyzed packet by packet. There are a number of different approaches to tackle this issue:

Many approaches require specific functionality in the data plane: Widely used approaches such as NetFlow [9] and FlowRadar [10] are using hash based approaches in the data plane.

Other works are tracking the size of all flows and are then choosing the heavy hitters from all flows [11], [12], [13]. This is particularly useful for routers with a slow path and a fast path. The flows that use the fast path are then chosen using the largest flow from all flows. However in a centralized approach like SDN, this is quite demanding in terms of statistics collection. It has been found that handling all flows in the control plane causes a high overhead [14]. To avoid this, DevoFlow [14] describes three different options to detect elephant flows. First a sampling based approach is suggested. Further the use of triggered reports and finally approximate counters are proposed, the two last methods are not available in current hardware and are hence out of scope.

Besides DevoFlow other sampling based approaches were introduced [15], [16], [17], [5]. The underlying idea of these approaches is that the number of packets of an elephant flow is much higher than for a mice flow. Therefore, the probability to miss an elephant flow is low, while the complexity of the detection can be greatly reduced. In Section VII, we investigate and compare to the sampling based approach and show that the detection algorithm is of less importance than the sampling rate.

In addition, there are approaches like Mahout [18] and Hedera [19] that are using end-host based elephant flow detection. However, in contrast to the presented approaches in this work, [18] and [19] require modifications at the end-hosts, which may be feasible in data centers but not in most of the other networks.

Further there are also a number of works studying machine learning approaches for elephant flow detection. Chabra and Kiran [20] are using clustering algorithms to classify flows retrospectively into mice and elephant flows. Therefore they are researching the problem from a different angle, namely: How to define an elephant flow.

Pouper et al. [21] propose the use of neural networks, Gaussian mixture models and Gaussian process regression for the prediction of the flow size. Xiao et al. [22] use C4.5 decision trees and Viljoen et al. [23] are using a neural network to classify flows into mice and elephant. Chao et al. [24] uses a 2-stage detection scheme with C4.5 trees as a first stage and stream mining with Hoeffding trees in the second stage. All works above show the feasibility of machine learning for elephant flow detection, however none aims at a classification with the first packet as proposed in our paper. Further their evaluation is focusing on traffic engineering use cases, i.e. they do not consider table size restrictions.

IV. Offloading with the first packet

In order to overcome the limitations in the state of the art with respect to flow classification, we introduce a classification system that decides with the first packet if a flow is worth to be offloaded to hardware or not. The envisioned use cases provide stateful network functions; stateful network functions handle the first packet of a flow separately. E.g., a stateful firewall tracks the state of each connection and adds a new entry in its state table with the first packet. In our approach we make use of this specific processing path to forward the features of the first packet from the VNF to a central entity. This central entity e.g. an SDN controller can then classify the flow and take the offloading decision.

This classification is done with a model that is trained using machine learning.

Figure 3 illustrates the steps of the classification system:

1. In a first step the traffic is recorded by a monitoring system. This step is described in more detail in the next section.
2. From this raw data, a ground truth is derived: All flows that are bigger than a threshold are labeled as 1 all others are labeled with 0. This means the class is binary where 0 represents no offloading and 1 offloading.
3. The weights of the classes are equalized by weighting some flows higher than others. This is necessary as much more flows are of class 0 and the resulting models would consequently also be biased towards class 0.

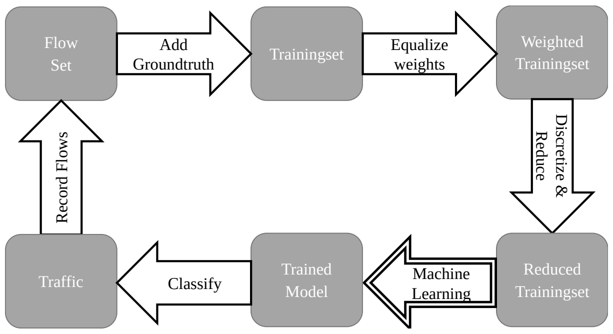


Figure 3. The classification system: The packets from the trace resp. on the line are recorded as flows. The ground truth is added depending on the gathered statistics. After that some filters are applied aiming to build a basis for the machine learning algorithm. The algorithm uses the set to train a model that can be used to classify new connections. These connections are recorded again for the next iteration.

4. The feature first packet size is discretized. Infrequent nominal values of all features are merged.
5. The model is trained by the respective machine learning algorithm.
6. The trained model is used by the offloading function to decide if new flows are offloaded or not.

For the evaluation in this work Step 1 is replaced by a packet-trace parsing script. Step 2-6 are implemented using Weka [25]. Steps 1-5 have to be repeated regularly to retain an up to date model for the classification.

A. Gathering the training-data

Gathering of the training-data is not in the focus of this work. Nevertheless we want to outline an architecture that could perform this necessary task. In particular, it is about the realization of the measurement points. At the measurement points the packets have to be grouped to flows and the features described in IV-B are extracted from the first packet of a flow. Therefore the measurement points have to be able to group the flows using their five tuple, gather statistics and to extract the size of the first packet. After the first packet, the accumulated size of the transmitted data of the flow has to be gathered, this information is needed to get the ground truth. In order to provide valid data all the network traffic has to be analyzed and not only a subset. This is especially important for a system with deployed NFV hardware offloading. As the offloading bypasses the VNFs, in turn these VNFs cannot be used alone for the data gathering.

A centralized network monitoring system that merges the measurements from hardware and software systems can solve this problem. Figure 4 depicts such an architecture. By default, offloading is not used, then all statistics, i.e. the size of the first packet and the total size of the flow can be easily extracted in a statistics network function (Statistics NF) realized in software. Additionally the hardware has to be capable to count the transmitted data of the offloaded flows. This is possible with current networking hardware that provides statistics via OpenFlow or sFlow. Finally, statistics from hardware and software have to be merged together by a central entity, e.g. an SDN controller or a network management system. The resulting data set can then be used for labeling the ground

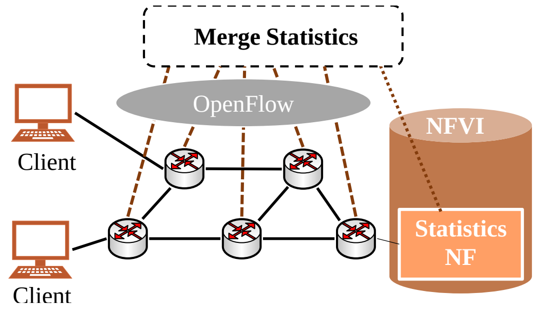


Figure 4. Gathering the necessary statistics using multiple points in the network. In a normal case packets are processed using a function chain located at the NFVI, in this case statistics can be gathered using a statistics NF. If a flow is offloaded the statistics have to be gathered by the hardware. All statistics have to be consolidated in the end, e.g. using an SDN controller or a network management system.

truth. All flows that have a total size of more than a threshold Θ_f are labeled with class 1, all others with class 0. How the threshold influences the offloading decision is shown in Section VI-B.

In the presented work, we used a Python script to parse the network traces that are shown in Section VI-A. Dumping the network traffic to a file could also be used in a real system. Though it might be problematic in practice, due to high overhead for storing and recording such a trace.

B. Features

Our classification approach essentially tries to separate mice from elephant flows. As the classification is used as an input to the hardware offloading of network functions, the decision should be available at the start of a flow. This is why only features available with the first packet are used. Secondly, as more and more traffic is encrypted, only features that can be directly deduced from the packet header should be used. Especially upcoming standards like QUIC and TLS 1.3 reduce the clear-text parts of the packets even further compared to current standards. The chosen features that are shown in Table IV-B are available with the first packet even when TLS encryption is used.

IP source and destination combine both IPv6 and IPv4 addresses. L4 source and destination port are both UDP and TCP ports, for other protocols this feature equals 0. Our notion of flows is bidirectional, this means the first combination of a five tuple is saved as a flow according to this packet's headers. If a packet of the same connection is seen in the opposite direction it is counted for this flow. The five-tuple features IP source, IP destination, IP protocol number, source port and destination port are nominal or categorical features, as the information depends on the specific number and not on the range. In order to avoid over-fitting of the model and reduce the complexity of both training and classification we merge all nominal feature values with a frequency of less than f into one value. An evaluation of parameter f is given in Section VI-D

For each flow the size of the first packet in Bytes is stored during flow recording. Figure 5 shows a histogram of the size of the first packet at each flow. As can be seen from the figure, the frequency of small packets with a size close to the

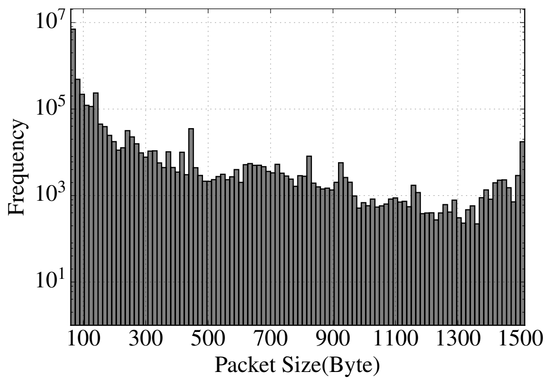


Figure 5. Histogram of the size of the first packet from each flow for the Wide A trace. The distribution shows high frequency for small packets.

Feature	Type
IP source	Nominal
IP destination	Nominal
IP protocol number	Nominal
L4 source port	Nominal
L4 destination port	Nominal
Size of first packet	Nominal (Discretized)

Table I
FEATURES USED FOR MACHINE LEARNING. ALL FEATURES ARE AVAILABLE WITH THE FIRST PACKET OF A FLOW. THIS ENABLES EARLY CLASSIFICATION AND A LARGER GAIN COMPARED TO OTHER APPROACHES.

minimum MTU of Ethernet is very high. These small packets can be either e.g. ICMP packets that constitute a mice flow, or possibly a TCP-Syn resulting in an elephant flow. Bigger first packets can be part of a small or large flow as well (e.g. a Syn with TCP Fast Open [26] or a DNS request). Therefore using the packet size as a numeric feature will not give the best classification results. Consequently, the size of the first packet is discretized. We are using equal frequency binning that creates bins with an equal number of instances, instead of regular discretization where the bins cover the continuous numeric space equally. This takes into account the differences regarding the first packet frequencies and results in more bins for small packet sizes and fewer bins for larger packet sizes. We performed studies that show a significantly increased performance with equal frequency binning compared to the numeric feature.

C. Machine Learning Algorithms

In this section we briefly introduce the employed machine learning algorithms.

NaiveBayes is a simple algorithm for creating a classification model. It is based on Bayes' theorem. To compute the probabilities, NaiveBayes uses the assumption that the value of one feature given the class is independent of the values of the other feature. For our real world problem this is an assumption that will not hold as for example the port and IP of a server are coupled and are clearly not independent. Nevertheless it has been shown in [27] that NaiveBayes still works well for many real world cases even if the assumptions of Bayes' theorem do not hold.

On the one hand, a machine learning algorithm should be able to build a model that is accurate, i.e. has a high classification performance. On the other hand it should also build a model that is fast in classifying new instances and also understandable to support debugging. Decision tables can support these concerns. For classifying the instances, the table is searched for an exact match. If no match is found, the majority class is returned. For building the table we used the IDTM algorithm presented in [28]. Note that not all features are included in the table. The feature subset is chosen using BestFirst heuristic according to [28]. On the other hand, decision tables are known to have a tendency to over-fit. This means that the learned table fits very well to the training set but is bad for predicting new unseen instances.

This can be solved by using tree algorithms, which can be tuned in a way that avoids over-fitting. Additionally, the complexity for the prediction in the offloading logic is low when trees are used. J48 is the Java implementation of the C4.5 algorithm presented in [29]. This algorithm generates a decision tree from the training data, which is used for the classification. Every decision node in the tree takes a decision based on one attribute. J48 uses the entropy to decide the distance to the root of the different attributes in the decision tree. Each leaf of the tree identifies a class value, in our case as the class is binary, 0 or 1. Therefore, the maximum number of decisions that have to be made for a classification is p+1, where p is the number of attributes. For our experiments, we set the algorithm such that each leaf has to contain at least 2 instances. Additionally, we used pruning to reduce the tree size and avoid over fitting with a pruning confidence of 0.5, following best practice.

One way to improve classification performance is to use ensemble methods. Ensemble methods combine the prediction of multiple other learning algorithms. The drawback is that complexity increases, as multiple base models have to be trained or evaluated. We used two popular ensemble methods namely Random Forest [30] and Adaptive Boost Algorithm or shortly AdaBoost [31].

The Random Forest algorithm generates an ensemble of trees, the decision is then taken by voting for the most popular class. Random Forest is designed to improve classification accuracy while being robust against outliers and noise. It uses bagging together with random feature selection for creating the trees. Bagging is a method to combine different models created from samples of the training set. We used a bag size of 5 % and a minimum number of instances of 100 per leaf. Each tree uses $F = log_2 p + 1$ randomly selected features from a total of p features.

Random Forest is specifically created to be used with tree algorithms, AdaBoost on the other hand can be used in conjunction with many other algorithms. It combines team of multiple inner models in a weighted sum, which is then used for predicting the class. The AdaBoost algorithm chooses the inner algorithms systematically to find a good classifier for all instances in the learning set. For each iteration one inner model is chosen such that the team with the new inner model performs better than the old team. In this work we are using the M1 variant of the algorithm with 10 iterations. As inner classifiers we are using DecisionStump. DecisionStump builds a decision tree based on one attribute with only one level, i.e. it consists of the tree root and the leaves. Nevertheless in conjunction with an ensemble learner like AdaBoost, the DecisionStump algorithm can be used for building well performing classifiers [31].

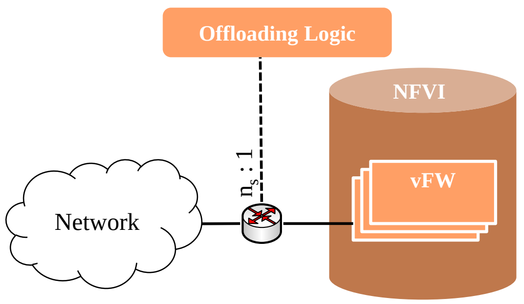


Figure 6. Sampling based Offloading approach: Every n_s packet is forwarded to the Offloading logic. If the decision is for offloading the flow of the corresponding packet is offloaded by the SDN Controller.

V. Sampling based approach

An alternative approach for NFV offloading uses sampling: Instead of being based on the first packet of a flow, the offloading decision is based on randomly sampled packets. Sampling is a common method for monitoring networks and is realized e.g. by sFlow [32]. We compare our machine learning algorithm with algorithms based on sampling.

In general, the sampling based offloading approach shown in Figure 6 works as follows: The sampled packets are forwarded to the decision logic. The logic decides if the flow corresponding to this packet should be offloaded or not, based on the packet and the internal state of the logic.

If the decision is for offloading, a new flow is installed in the hardware table and the load on the network function is reduced. Our employed decision algorithms are modified versions of Heimgaertner's [5] algorithm. Heimgaertner's approaches are designed for imminent congestion and therefore not directly comparable. The results show that the offloading algorithm itself is, somehow surprisingly, of minor importance for the sampling based approach, if real network traces are used. Instead, we show in Section VII that the sampling rate is the most important parameter. Therefore, we only show two rather simple algorithms for the offloading decision in the following.

A. Baseline Algorithm

One very simple yet effective offloading algorithm is to always decide on offloading for every sampled packet. We call this algorithm Baseline as it achieves the maximum reachable offloaded rate for a certain sampling rate. Due to the sampling, not every packet and thus not every flow is offloaded. Thus, the maximum offloaded rate is limited. The only parameter of this approach is the sample rate n_s, i.e. only one out of n_s packets is forwarded to the offloading algorithm.

B. Table restricted approach Sample+

In order to restrict the table size necessary for the offloading, the algorithm can be improved. The following algorithm is based on Heimgaertner's algorithm. In contrast to our approach, Heimgaertner's algorithm is designed for Offloading during immanent congestion and assumes that this only lasts for a fraction of the time. Furthermore, it does not adapt the offloading probability to the number of appearances of a flow.

We denote our advanced algorithm shortly as *Sample+*. The target offload rate r_{off} should be as large as the inverse of the usage time of one entry t_{reuse} multiplied by the target number of entries in the table n_{by}.

$$r_{off} = \frac{n_{by}}{t_{reuse}} \tag{1}$$

If Equation 1 holds, then the number of rules in the table is exactly as large as targeted. The reuse time is not known beforehand, it consists of the remaining active time of the flow after offloading it t_{rct} and the soft timeout t_{out} of the OpenFlow switch table.

$$t_{reuse} = t_{rct} + t_{out} \tag{2}$$

Therefore it holds $t_{out} < t_{reuse}$ and we can upper-bound $r_{off} < \frac{n_{by}}{t_{out}}$. We denote the rate of new flows seen by the algorithm as r_f . It is estimated by the offloading logic using UTEMA method [33]. Further we can then upper bound the offloading probability p_{off} as follows:

$$p_{off} = \frac{r_{off}}{r_f} \leq \frac{n_{by}}{t_{out} \cdot r_f} \tag{3}$$

In order to compensate that Equation 3 gives only an upper bound, we use a decreasing probability if the table is nearly full. The offloading probability p_{off} is then defined as:

$$p_{off} = min\left(\frac{n_{by}}{t_{out} \cdot r_f}, \frac{n_{by}^{rem}}{t_{out} \cdot r_f \cdot th}\right) \tag{4}$$

n_{by}^{rem} denotes the number of remaining entries, i.e. the number of used entries subtracted from n_{by}. The threshold th specifies the border between the first and the second term in the minimum. In our experiment we have set $th = 0.1$, i.e. for a table occupation of 90% both terms in Equation 4 are equally large.

VI. Evaluation of the Machine Learning approach

For the evaluation of the machine learning approach we initially present the results using stratified cross-validation. This approach delivers multiple outcomes for one data set and allows a better grading of the algorithms than a single result for each data set.

A. Data Sets

The machine learning approach tries to detect patterns in the network traffic. One such pattern could be an IP address of a popular video service, connections to this IP would be most likely elephant flows. These patterns might be very diverse in nature and therefore hard to model. As a consequence simulated or emulated traffic cannot be used for the evaluation and we have to use real traces.

Algorithm 1: Sample+ offloading algorithm

```
Input: Sampled Packet, t
Output: Offload Decision
1  $5t$ = get5tuple(packet);
2  if $5t \in T$ then
3      $T_c[5t] = T_c[5t] + 1$;
4  else
5      $r_f = updateFlowRate(r_f, t)$;
6      $T_c[5t] = 0$;
7  end
8  $p_{off} = min\left(\frac{n_{by}}{t_{out} \cdot r_f}, \frac{n_{by}^{rem}}{t_{out} \cdot r_f \cdot th}\right)$ ;
9  if $random[0,1] < p_{off} \cdot T_c[5t]$ then
10     return True;
11 else
12     return False;
13 end
```

	WIDE A	WIDE B	WIDE IX-24h	CAIDA 1	CAIDA 2
Duration	15 min	15 min	24 h	15 min	10 min
Flows	8M	8.6M	344M	45.6M	40.6M
Packets	99M	113M	15965M	977M	874M
Total Traffic	70 GiB	91 GiB	19 TiB	716 GiB	631 GiB
TCP flows share	85.8 %	77.6 %	46.0 %	83.6 %	80.6 %
UDP flows share	12.4 %	22.1 %	48.33 %	16.2 %	19.2 %

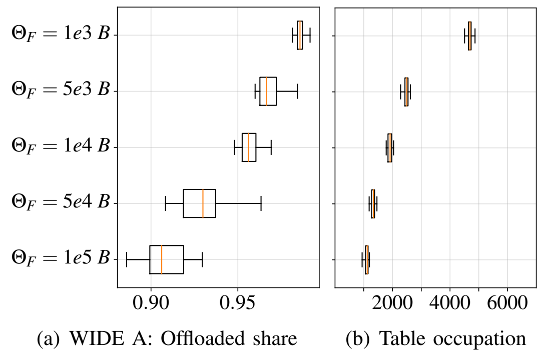
Table II
DATA SETS USED IN THIS WORK

We have chosen multiple traces among the publicly available traces to evaluate our results on diverse data sets in terms of link capacity, average flow size and composition. We use five publicly available data sets from three different measurement points. Main parameters of the traces are shown in Table II.

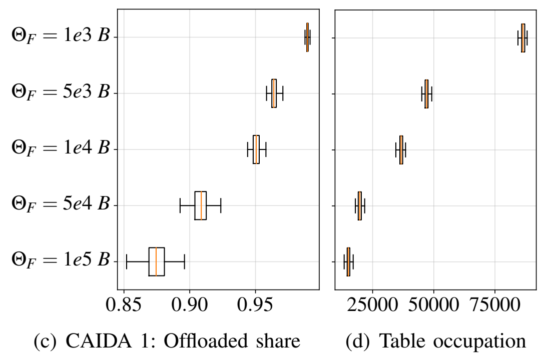
The WIDE data sets were retrieved from the MAWI Working Group Traffic Archive [34]. The traces WIDE A and WIDE B are collected from the 1 Gbps transit link of WIDE to their ISP. The traces were gathered at two consecutive days, Thursday and Friday. They were chosen out of the daily traces from the MAWI Working group. The daily traces are collected every day at 14:00 to 14:15. Furthermore we use a 24 hours trace (WIDE IX 24h) to study temporal effects of the machine learning solution. This trace was gathered from the main link of WIDE to the internet exchange point DIX-IE on a Tuesday. The WIDE IX 24h trace has a significantly larger share of UDP flows mostly due to a large number of DNS flows.

The CAIDA data sets were retrieved from the Center for Applied Internet Data Analysis [35]. The traces were collected at a data center's link to the backbone of a Tier 1 provider between 13:00 and 13:15. The link has a maximum data rate of 10 Gbps, this yields a much higher data rate and a larger number of parallel flows.

All traces where anonymized from the respective organizations. As statistical features were retained by the anonymization, our results are not influenced.



(a) WIDE A: Offloaded share (b) Table occupation



(c) CAIDA 1: Offloaded share (d) Table occupation

Figure 7. Data Labeling: Offloaded share and table occupation if the labels are used for the decision directly. For a small threshold Θ_F the table occupation is quite high even in this all-knowing case. But it quickly decreases. We chose $\Theta_F = 1e4\,B$ as a compromise between table occupation and offloaded rate.

B. Data Labeling

In this work, we are using supervised learning algorithms. This class of algorithms needs labeled data. Our classification problem is only binary with the class being the offloading decision that can be True or False. We labeled the data using a threshold with the following rule:

$$Offload = \begin{cases} True & L_F > \Theta_F \\ False & L_F \leq \Theta_F \end{cases}$$

Where L_F is the total size of a flow, meaning the sum of the lengths of the packets belonging to one flow. Figure 7 shows the offloaded share and the table occupation for different thresholds Θ_F in the WIDE A and the CAIDA 1 data set. The results in the figure show the table occupation and offloaded share if the labels are used directly for the decision. As this requires a system with global knowledge, this approach is not possible in reality.

Nevertheless, the results show the trade-off between required table size and offloaded share. As can be seen in the figure, there is not one single best decision for Θ_F.

Small thresholds yield a high table occupation obviously. On the other hand, the results show that rate share and table occupation are not proportional. E.g., in the WIDE A set changing the threshold Θ_F from $1e4$ to $5e4$, reduces the offloaded rate share by only 1.1%, while reducing the necessary table size by 11%. In contrast to that, the table occupation for the CAIDA 1 set is much higher in the range of 40000 entries. The threshold should be chosen such that the hardware table is fully utilized but not over-utilized. In our testbed we have an OpenFlow enabled NEC PF5240 switch with a table capacity of 64K entries suitable for the CAIDA data set in the SDN Offloading use case. On the other hand the FlowDirector enabled Intel X590 NIC has a capacity of 8K rules only. Following these guidelines, we decided to use $\Theta_F = 1e4$ as a threshold in the following reaching an offloading rate of 95% for both data sets.

	WIDE A	WIDE B	WIDE IX-24h	CAIDA 1	CAIDA 2
IP source	0.009	0.009	0.042	0.030	0.029
IP destination	0.007	0.007	0.029	0.021	0.019
IP protocol number	0.001	0.001	0.017	0.002	0.002
L4 source port	0.001	0.001	0.001	0.006	0.006
L4 destination port	0.009	0.010	0.037	0.008	0.008
Size of first packet	0.009	0.009	0.027	0.023	0.022

Table III
INFORMATION GAIN OF THE ATTRIBUTES IN BIT

C. Feature selection

Next we evaluate the information gain of the features, in order to understand the main drivers of a good classification scheme. Additionally, the results can be used to reduce the complexity of training and classification, by omitting features with low gain.

Table III shows the information gain of each attribute p:

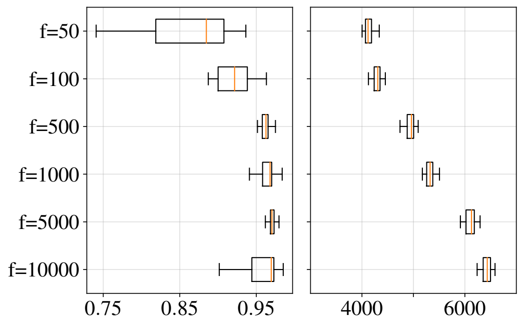
$$InfoGain(Class, p) = H(Class) - H(Class|p)$$

where $H(X)$ is the entropy of random variable X. From the table it can be seen that the information gain of each feature is small (maximum 0.042 bit for IP source and WIDE IX-24h) and no single attribute can be used for a decision. Additionally, the gain of attributes depends on the network conditions, as the size of the first packet has a much higher gain for the CAIDA data sets than for the WIDE data sets. Only the IP protocol number shows little information gain and could be omitted for performance reasons. As the notion of 5-tuples is very common in packet processing systems we keep the feature for the rest of the evaluation nevertheless. On the other hand using only the 5-tuple features would cause a loss in information for the classification and will therefore reduce the precision. This complicates data gathering a little as for example sFlow cannot be used to record the size of the first packet.

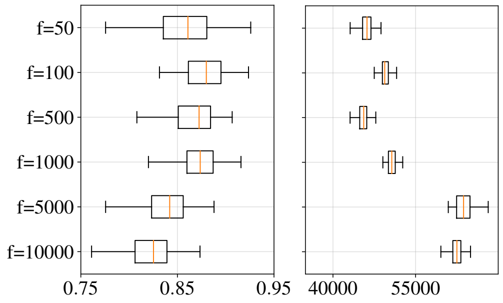
D. Merging infrequent nominal values

The employed features are mainly nominal, e.g. IP addresses which are numerically close to each other do not necessarily have similar effects on the classification. Consequently the models have to use single nominal values to give a prediction. As the learning sets can be built of a large number of flows the value sets for nominal features can quickly become very large. This is especially a problem for the IP addresses. Even if only one packet is sent from one IP, the models have to keep track of that.

In order to overcome this issue infrequent nominal feature values are merged to one value. This is done by merging all values that did occur in the training set less often than the frequency f into one single nominal value. We apply merging to all nominal features with the same f. Figure 8 shows the trade-off between table occupation and offloaded



(a) WIDE A: Offloaded share (b) Table Occupation



(c) CAIDA 1: Offloaded share (d) Table Occupation

Figure 8. Merging of infrequent values: The Figure shows the performance of the offloading algorithm with J48 as machine learning algorithm on the WIDE A and CAIDA 1 data sets. Nominal features with a lower frequency than the minimum frequency f are merged. f=500 provides a good compromise between table occupation and offloaded share.

bitrate that occurs with this parameter. A low f causes over-fitting and elephant flows are not recognized well. On the other hand, a high f under-fits and the table occupation necessary in turn increases. Additionally the complexity of training and classification largely depends on f.

A frequency of f=500 provides a good compromise between over- and under-fitting for both data sets, even though they are quite different. Thus f=500 is used for all other results.

E. Algorithm evaluation

1) Precision: The precision of the classification is defined as usual:

$$Precision = \frac{True\,Positives}{True\,Positives + False\,Positives}$$

As we want to select those flows that are worth offloading in hardware, we want a high true positive rate for class 1. On the other hand, if we select too many flows for the offloading, we need a large hardware forwarding table and might exceed its capacity. Figure 9 shows the results for the different data sets and both classes. The boxes are ranging from the lower to the upper quartile of the outcomes. The whiskers mark the full range of the outcomes. The line in the box represents the mean.

The mean precision ranges between 0.85 and 0.95 depending on data set and algorithm. J48 and RandomForest show the best classification precision in general. Especially J48 shows a high precision for both classes. The precision of the DecisionTable classifier is below 85 % for some folds in the CAIDA data sets, on the other hand the results for the WIDE data sets are unremarkable. Another outstanding result is the precision of the boosted DecisionStump algorithm in the CAIDA 1 data set. The model almost always decides for class 0, this results in precision of almost 100 % for class 0 and close to 50 % (out of the scale of the figure) for class 1. For the other data sets the performance is clearly worse than that of other algorithms. Consequently we did not consider boosted DecisionStump further in the following.

2) Offloaded bit rate vs table requirement: One main drawback of networking hardware compared to software solutions, is the limited hardware resources especially the tables. In the ideal case we are only using hardware offloading for large flows and consequently can treat the lion's share of the total data rate in hardware using few matching rules. As the machine learning algorithms are imperfect, we have to either sacrifice data rate or use a larger hardware table.

For this evaluation, we made the simplifying assumption that the flow can be offloaded from the first packet. As we cannot always sense the end of a flow, we always assumed that one rule has to be kept in the hardware table for additional three seconds after the last packet. This matches the soft timeout mechanism used in OpenFlow devices, but could also be used for other acceleration technologies. In Figure 11, we present the mean table occupation for the offloading solution, Figure 10 shows the offloaded data rate. Specifically for the CAIDA data sets more than 80% and for the WIDE data sets even more than 90% of the data rate can be handled in hardware. On the other hand a fairly large hardware table is necessary, 8000 entries in the WIDE case and 45000 in the CAIDA case. This corresponds to roughly 20 % of all flows handled in hardware. This is surely a significant number, but as regular Binary Content Addressable Memory (BCAM) can be used instead of expensive Ternary Content Addressable Memory (TCAM) it is still feasible with the hardware available today.

F. Classification Complexity

From the results shown before, we can see that especially the tree algorithms, J48 and RandomForest, are always in the group of the best performing algorithms. This is why we want discuss the aspect of the classification complexity. J48 builds a decision tree which makes a fast classification possible. RandomForest is more complex in general as multiple trees are evaluated and weighted. As RandomForest does not show a superior performance, J48 seems to be a better choice. The other algorithms require the evaluation of tables (NaiveBayes and DecisionTables). This means their classification process is more complex in general.

In case of J48, the complexity of the decision depends on the depth of the leaves and the degree of the decision nodes. As this metric is not easy to tackle we evaluated the decision tree of the J48 learned model with the tree size. The tree size is the number of all nodes in the tree.

We reduced the tree size (and consequently the decision complexity) by allowing only leafs in the tree that are matching a minimum number of weighted training instances. Figure 12 shows the tree size on the left axis and the necessary table size on the right axis for a growing number of minimum

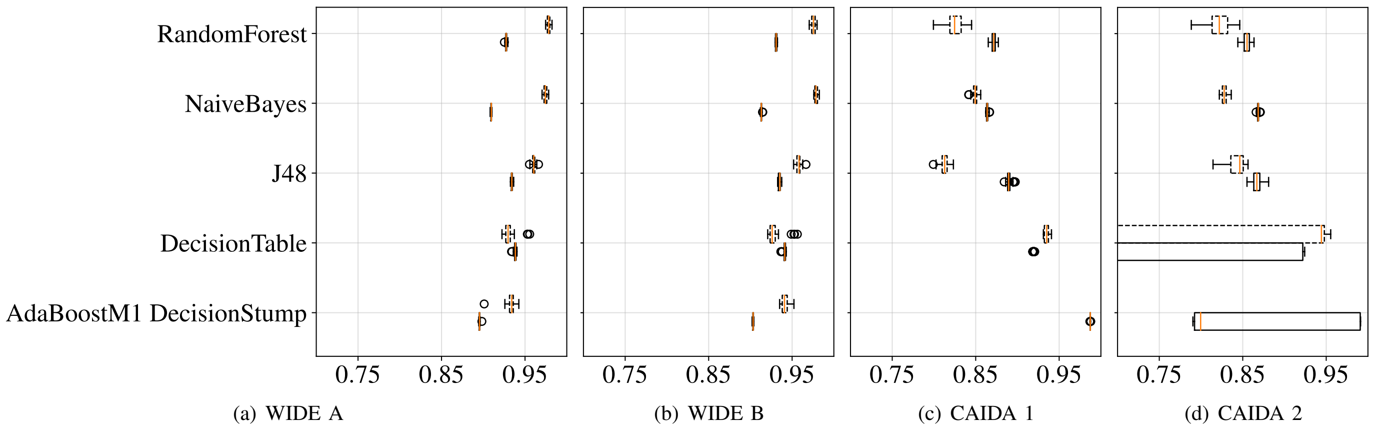


Figure 9. Precision of the selected algorithms with the different data sets. The mean precision for class 0 (offload=No) ranges between 0.84 and 0.96 depending on data set and algorithm. The precision for class 1 is worse in general. J48 and RandomForest show the best classification precision in general. The Decision Table classification precision is differing largely for the CAIDA data sets. Boosted DecisionStump is showing decent performance for the WIDE data sets but has only low accuracy in the CAIDA data sets.

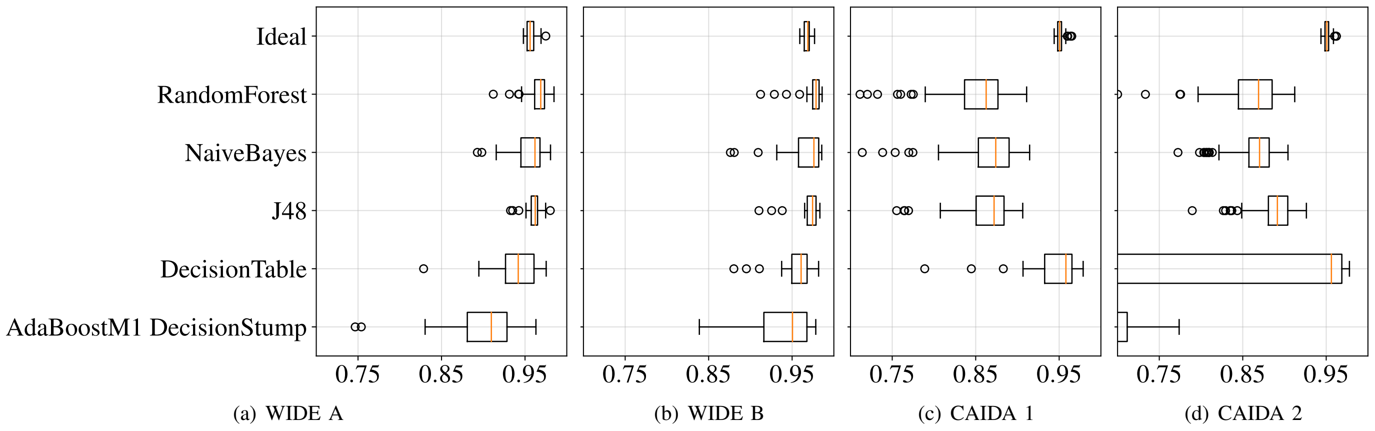


Figure 10. Offloaded data rate using the different learning schemes. The bit rate that is processed by hardware is approximately as good as with the all-knowing ideal scheme. This is due to the much higher number of processed flows compared to the ideal case. In general the lion's share of the rate can be processed in hardware.

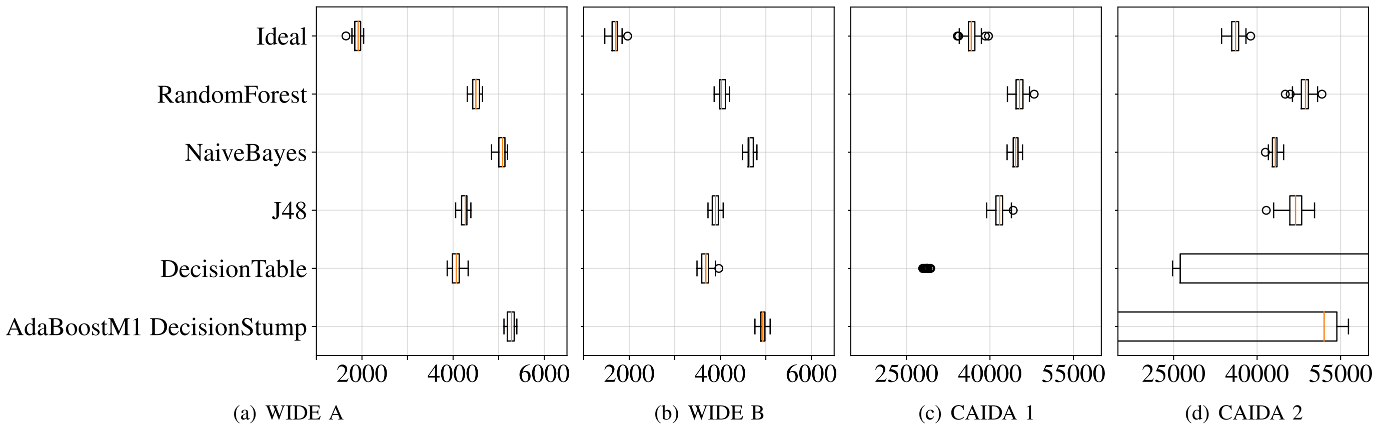


Figure 11. Mean table occupation. One major limitation is the number of rules in the hardware table. J48 requires the lowest number of table entries. On the other hand the overhead compared to the ideal case is quite large for all learning schemes.

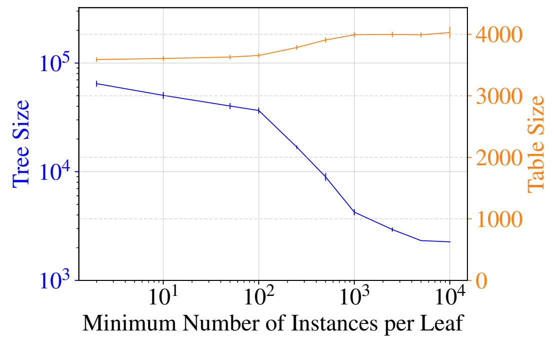


Figure 12. Decision tree size and table size when increasing the minimum number of instances per leaf. The Figure shows the results using WIDE B data set. A reduction of the tree by one magnitude only increases the table size necessary by 12.5 %.

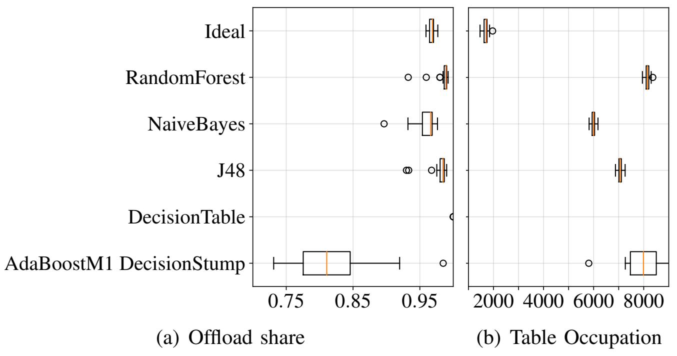


Figure 13. Temporal Stability: Offloading performance when using the Model trained with WIDE A data set for classifying WIDE B data set. DecisionTable, NaiveBayes and boosted NaiveBayes perform worse in this scenario. J48 and RandomForest algorithm show less degradation. The offloaded rate is still in the same range as for the ideal decision, the table occupation is ∼20 % higher when compared to the model learned by cross validation.

instances per leaf. As the main effect of a modification in the learning algorithm is visible in the necessary table size only this metric is shown here. The results show that a reduction of the tree by one magnitude increases the table size necessary by 12.5 %. We can therefore reduce the complexity significantly while only increasing the necessary table size gradually. Nevertheless, in a deployment scenario it could be more pressing to reduce the table size while scarifying the decision complexity.

G. *Temporal stability*

In this work, we are using an offline learning approach. One question that is arising is how stable the model is regarding a longer period of time between the time the training set was recorded and the time the model is used. Figure 13 shows the results in this case: We have used the WIDE A data set for training the model and applied it then to the WIDE B data set. We again stratified the test set to end up with results that can be compared. This way the test sets are the same as in Figure 10(b), only the training sets differ. The results show that the offloaded bit rate does not change much with the delay between training and classification. Only the offloaded bit rate using DecisionTable classification is clearly lower. This can be explained by over fitting the training data.

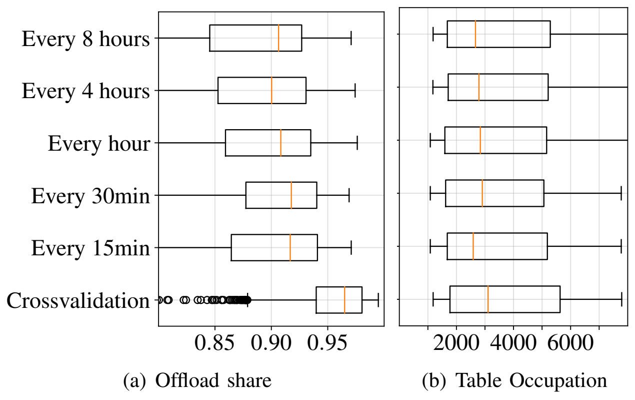


Figure 14. Temporal Stability: Offloading performance for the WIDE 24h data set and J48. Performance is not impacted largely when different learning intervals are used.

Both tree algorithms, J48 and RandomForest, show a slightly higher offloaded rate when compared to NaiveBayes.

A more in depth view can be gained if we apply the approach to a longer trace. Thus we also applied it to the WIDE IX-24h data set and used different training intervals. Figure 14 shows the results for different training intervals, e.g., every 15min means that the model is retrained every 15 minutes. We use always data from 15 minutes for training. The results show that the performance is slightly reduced due to the time shift. A more frequent training does not yield significantly better results. Further, the spread of both offloading share and table occupation is quite large.

We see the reason for this result in Figure 15. For this figure we split the data into 15 minutes chunks. The figure shows the mean offloading share of all combinations of training and test sets. From the figure it can be seen that there are more and less challenging times of the day. While the performance in the morning (until 12:00 pm) and in the night (after 23:00 pm) is above 90% for most of the training sets, it is different for a short interval around 13:00 pm and especially in the evening after 18:00 pm. The analysis of the data shows that during these short intervals the average size of the flows is smaller than at all other times of the day. During these short intervals a 20-50% (not shown in the figure) decreased flow size is consistently visible. The reduction is caused by a larger number of mice flows. This shift makes correct classification more challenging.

It should be noted that the pattern reverts to its normal behavior after 23:00 as can be seen by the high offloading shares even with training sets in the early morning. On the other hand, training more frequently does not significantly improve the performance, as the pattern is changing quite often at this time of the day (visible by the stripes in the upper right corner of Figure 15).

This means that it is sufficient to retrain the model every day without losing too much performance. We imagine to do this as a regular job that is executed during less busy hours in a real deployment scenario.

VII. Evaluation of the sampling approach

In order to evaluate the sampling approach we applied both algorithms described in Section V for the data sets WIDE A

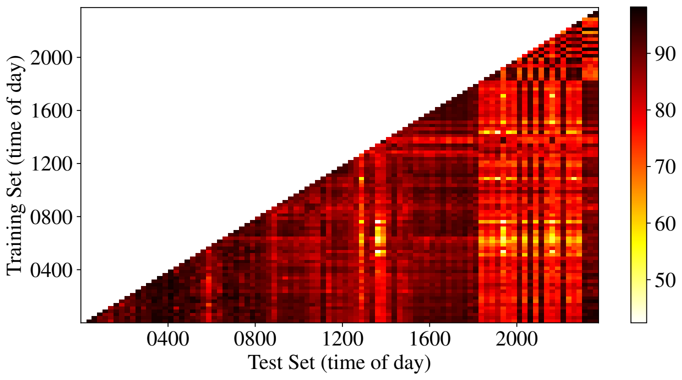


Figure 15. Temporal Stability: Offloading share for the WIDE 24h data set in a heat map. This view on the results shows that some times of the day have a different traffic pattern than others.

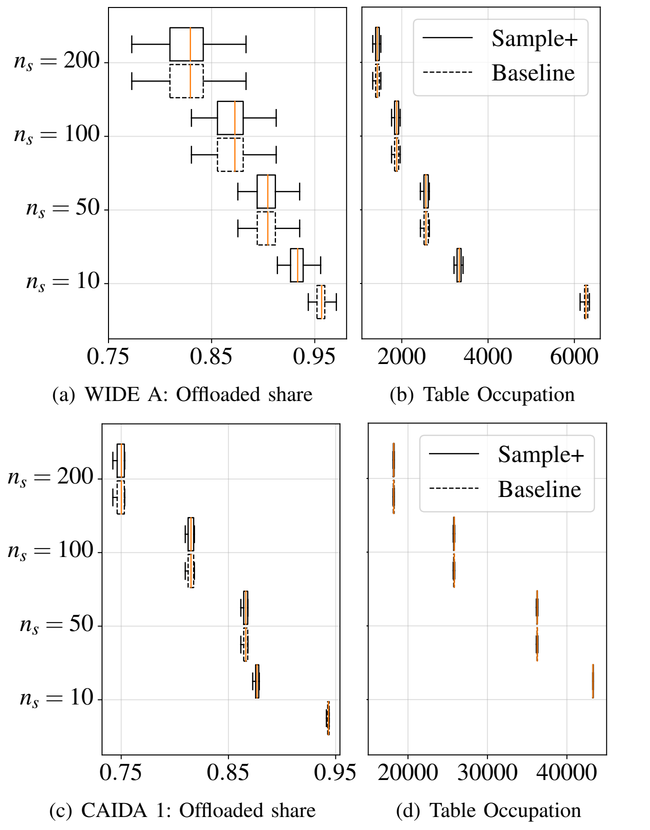


Figure 16. Offloading using Sampling: For lower sampling rates all flows that belong to sampled packets are offloaded, thus Sample+ and Baseline algorithm perform equally. Only for higher rates we can see that Sample+ is much more economically regarding table occupation.

and B and both CAIDA data sets. One main parameter of both algorithm is the sampling rate n_s, we used different settings for n_s, starting with moderate sampling of $n_s = 200$ up to a high sampling rate with $n_s = 10$. This is the only parameter of the baseline algorithm, from runs with this algorithm we can derive how many table entries are necessary in the restricted case. We have chosen a table size of 4000 for the data set WIDE A and 45000 for CAIDA 1, this allows for a high offloading rate in both cases.

Figure 16 shows the results for both algorithms. It can be seen that for lower rates up to $n_s = 50$ both algorithms have the exact same performance and the table is not filled to the targeted value. This is due to the fact that many flows are not even seen by the offloading algorithms as no packet of these flows is sampled. As the heuristic tries to use the table up to the allowed level it behaves equal to the baseline algorithm. On the other hand it can be seen that the offloaded rate is quite high if compared to the table occupation. This is due to the fact that sampling itself already filters the flows as large flows have a higher probability to be sampled, this directly resembles the fact that elephant flows makeup the major share of the complete traffic.

The downside of the sampling based approach is that the overhead is quite high. While the sampling itself can be done in hardware that is deployed quite often nowadays, the rate that has to be processed by the algorithm is still high. E.g. if the share of the offloaded traffic should reach 90% it is necessary to set the sampling rate to $n_s = 50$, this results in a packet rate of $110 \cdot 10^3$ packets per second (pps) for the WIDE A data set. Even though it is feasible to process such a number, it is demanding and can be infeasible for scenarios with higher rates.

VIII. Discussion

In this section, we are summarizing the findings and assess them in context of the NFV acceleration approach.

In Section IV, we are introducing a classification model that is used to build a model for the classification of the flows in small flows and large flows used for offloading. We present a monitoring approach for gathering the necessary training data. We argue that a combined approach that uses statistics from a NFV based tailored statistics network function for the first packet size and a conventional monitoring system using OpenFlow statistics or sFlow can be used.

Furthermore we introduce the features that are used for training and how they are preprocessed. The preprocessing is based on domain knowledge and is an important step for a good classification performance.

The preprocessing uses the parameters offloading threshold Θ_F and the minimum frequency f of the nominal features. We tuned the parameters independent of each other using the data sets CAIDA A and WIDE A. The results are similar for both data sets even though the data sets are quite different, as CAIDA A has much higher data rate and many more concurrent flows. Further, we use the same parameters for the data sets CAIDA B, WIDE B and WIDE IX-24 with good results. This implies that the parameters hold for typical internet traffic. However, for different network conditions such as, e.g., an industrial network, the optimal parameters might be different. Nevertheless, we argue that the pipeline itself can be applied to other conditions with minor or no changes.

In Section VI, we apply different machine learning algorithms using the presented classification system. We rely on publicly available data sets. Figures 9, 10 and 11 show the results. It can be seen that DecisionTable and boosted DecisionStump algorithm do not provide a reliable performance with the investigated data sets. Therefore we argue that they are not suitable for the presented application. Overall RandomForest and J48 achieve high offloaded rates with

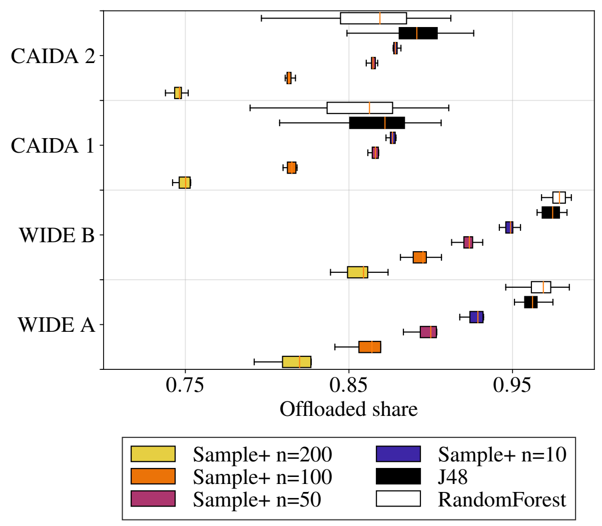


Figure 17. Overview of the offloaded share for all algorithms and data sets. In order to reach the performance of the Machine Learning algorithms a high sampling rate is necessary.

slightly lower table occupation. We argue that J48 has a lower complexity than RandomForest and analyze this property more deeply for J48 in Section VI-F.

The model is employed for classifying new flows with the first packet and the acceleration use cases as shown in Section II can be used. As the model is network specific and not static it has to be retrained regularly. In Section VI-G, we have shown that a daily retraining is sufficient for the evaluated network. As the old model can still be used while the new one is trained we do not have any interruptions of the acceleration.

Section V introduces two sampling based approaches that can be used alternatively for the offloading decision. The baseline algorithm always decides for offloading. Due to the sampling not all flows are offloaded, as it can happen that all packets of a flow are not part of the sampled subset. The table restricted approach, *Sample+* takes a maximum allowed table size as input and keeps the number of rules below this threshold. Notably the most important parameter of the sampling approach is the sampling rate, for low to medium sampling rates it does not even matter if the offloading algorithm decides on offloading all the time, as shown with the Baseline algorithm.

Figure 17 shows an overview of the results for both approaches. We have chosen the best performing machine learning algorithms namely J48 and RandomForest here. It can be seen that the offloaded share for the WIDE data sets is significantly higher compared to the CAIDA data sets. This indicates that large flows have a smaller share in this network. In general both presented learning algorithms perform better and especially more stable for the WIDE sets. We can conclude that the algorithms are more suitable for small scale networks like campus networks. All results support the finding that the sampling based algorithms require a large sampling rate to reach the performance of the Machine Learning algorithms: For the WIDE data sets we do not even meet the performance with a sampling rate of 1:10, for the CAIDA data sets still a sampling rate of 1:50 is necessary. A high sampling rate requires significant compute resources as all the sampled packets have to be processed. That is with the presented sampling rate and the WIDE A data set a packet rate of $110 \cdot 10^3$ pps.

Even though the sampling rate for the CAIDA traces can be lower, the resulting packet rate that must be processed by the sampling approach is higher: $1 \cdot 10^6$ pps.

The traces differ a lot in terms of table occupation. The traces that were retrieved from the WIDE traffic archive require table sizes in the range of a few thousand entries. In contrast to that the traces retrieved from CAIDA require much bigger tables of several ten thousand entries. This observation holds for both approaches as can be seen from Figure 11 and Figure 16. This difference in the order of magnitude stems from the fact, that the WIDE traces are gathered from a 1G uplink, while the CAIDA traces are gathered at a 10G backbone link. This indicates that different hardware, in terms of size of the hardware table, is necessary, depending on how aggregated a link is. On the one hand, highly aggregated links require acceleration hardware providing large tables. On the other hand, less aggregated links like those in the WIDE case suffice with smaller tables.

Finally, the analysis of the data set shows that the approach performs better if the data set is less noisy. Especially a very high number of small micro flows, like in the CAIDA sets or during some hours in the WIDE IX-24h set can cause some performance degradation.

IX. Conclusion and Outlook

In this work, we explore the capabilities of different algorithms for deciding if a flow should be offloaded to hardware or not. We consider two fundamentally different approaches: First, offloading with the first packet of a flow using machine learning. The first packet is sent by the VNF to the algorithm. Secondly, offloading with sampling, where all packets are sampled and the sampled packets are used for the decision.

The results show that the lion's share of the data rate can be handled in hardware using our approaches. The drawback is that the necessary hardware table is comparably large, as a fairly big number of flows are classified for offloading falsely. As regular BCAM can be used for matching, the approach is feasible nevertheless.

The results show that the J48 algorithm has the best properties of all the investigated machine learning algorithms. It combines low table occupation with a high offloaded data rate and additionally has a low complexity of the classification. On the other hand, other algorithms like RandomForest and NaiveBayes are only slightly worse and could also be used.

The sampling based approach can also reach a high offloaded share, however it requires a quite high sampling rate to meet the performance of the machine learning approach. If the overhead of sampling the complete traffic should be avoided, offloading with the first packet is a viable alternative to sampling.

Acknowledgment

The authors would like to thank the anonymous reviewers for their important feedback during the review process. This work has been partially funded by the German Research Foundation (DFG) under the grant number KE 1863/8-1 and by the Federal Ministry of Education and Research Germany (BMBF) under grant number 16KIS0260. The authors alone are responsible for the content of the paper.

References

[1] B. Pfaff, J. Scherer, D. Hock, N. Gray, T. Zinner, P. Tran-Gia, R. Durner, W. Kellerer, and C. Lorenz, "SDN/NFV-enabled Security Architecture for Fine-grained Policy Enforcement and Threat Mitigation for Enterprise Networks," in *Proceedings of the SIGCOMM Posters and Demos*, 2017, pp. 15–16.

[2] R. Durner, A. Varasteh, M. Stephan, C. Mas Machuca, and W. Kellerer, "Hnlb: Utilizing hardware matching capabilities of nics for offloading stateful load balancers," in *IEEE International Conference on Communications (ICC '19)*, 2019.

[3] E. Balas and A. Ragusa, "Scipass: a 100gbps capable secure science dmz using openflow and bro," 2014.

[4] S. Miteff and S. Hazelhurst, "NFShunt: A Linux firewall with OpenFlow-enabled hardware bypass," *IEEE Conference on Network Function Virtualization and Software Defined Network (NFV-SDN), 2015*.

[5] F. Heimgaertner, M. Schmidt, D. Morgenstern, and M. Menth, "A Software-Defined Firewall Bypass for Congestion Offloading," in *International Conference on Network and Service Management (CNSM)*, 2017.

[6] Y. Weinsberg, E. Pavlov, Y. Amir, G. Gat, and S. Wulff, "Putting it on the nic: A case study on application offloading to a network interface card (nic)," in *3rd IEEE Consumer Communications and Networking Conference (CCNC)*, 2006.

[7] J. Shi, T. Liang, H. Wu, B. Liu, and B. Zhang, "Ndn-nic: Name-based filtering on network interface card," in *Proceedings of the 3rd ACM Conference on Information-Centric Networking*, 2016.

[8] D. Firestone, A. Putnam, S. Mundkur, D. Chiou, A. Dabagh, M. Andrewartha, H. Angepat, V. Bhanu, A. Caulfield, E. Chung *et al.*, "Azure accelerated networking: Smartnics in the public cloud," in *15th USENIX Symposium on Networked Systems Design and Implementation (NSDI 18)*, 2018.

[9] B. Claise, "Cisco systems netflow services export version 9," Tech. Rep., 2004.

[10] Y. Li, R. Miao, C. Kim, and M. Yu, "Flowradar: a better netflow for data centers," in *13th USENIX Symposium on Networked Systems Design and Implementation (NSDI)*, 2016.

[11] L. Che and B. Qiu, "Landmark LRU: An efficient scheme for the detection of elephant flows at internet routers," *IEEE Communications Letters*, vol. 10, no. 7, 2006.

[12] R. B. Basat, G. Einziger, R. Friedman, and Y. Kassner, "Optimal elephant flow detection," in *IEEE Conference on Computer Communications(INFOCOM)*, 2017.

[13] N. Sarrar, S. Uhlig, A. Feldmann, R. Sherwood, and X. Huang, "Leveraging Zipf's law for traffic offloading," *ACM SIGCOMM Computer Communication Review*, 2012.

[14] A. R. Curtis, J. C. Mogul, J. Tourrilhes, P. Yalagandula, P. Sharma, and S. Banerjee, "Devoflow: Scaling flow management for high-performance networks," *ACM SIGCOMM Computer Communication Review*, vol. 41, no. 4, pp. 254–265, 2011.

[15] T. Mori, M. Uchida, R. Kawahara, J. Pan, and S. Goto, "Identifying elephant flows through periodically sampled packets," in *Proceedings of the 4th ACM SIGCOMM conference on Internet measurement*, 2004.

[16] K. Psounis, A. Ghosh, B. Prabhakar, and G. Wang, "Sift: A simple algorithm for tracking elephant flows, and taking advantage of power laws," in *43rd Allerton Conference on Communication, Control and Computing*, 2005.

[17] Y. Afek, A. Bremler-Barr, S. Landau Feibish, and L. Schiff, "Sampling and large flow detection in sdn," *ACM SIGCOMM Computer Communication Review*, vol. 45, no. 4, pp. 345–346, 2015.

[18] A. R. Curtis, W. Kim, and P. Yalagandula, "Mahout: Low-overhead datacenter traffic management using end-host-based elephant detection," in *IEEE Conference on Computer Communications(INFOCOM)*, 2011.

[19] M. Al-Fares, S. Radhakrishnan, B. Raghavan, N. Huang, A. Vahdat *et al.*, "Hedera: dynamic flow scheduling for data center networks." in *7th USENIX conference on Networked systems design and implementation (NSDI)*, 2010.

[20] A. Chhabra and M. Kiran, "Classifying elephant and mice flows in high-speed scientific networks," in *Proceedings of INDIS*, 2017.

[21] P. Poupart, Z. Chen, P. Jaini, F. Fung, H. Susanto, Y. Geng, L. Chen, K. Chen, and H. Jin, "Online flow size prediction for improved network routing," in *International Conference on Network Protocols (ICNP)*, 2016.

[22] P. Xiao, W. Qu, H. Qi, Y. Xu, and Z. Li, "An efficient elephant flow detection with cost-sensitive in sdn," in *1st International Conference on Industrial Networks and Intelligent Systems (INISCom)*, 2015.

[23] N. Viljoen, H. Rastegarfar, Mingwei Yang, J. Wissinger, and M. Glick, "Machine learning based adaptive flow classification for optically interconnected data centers," in *18th International Conference on Transparent Optical Networks (ICTON)*, 2016.

[24] S. Chao, K. C. Lin, and M. Chen, "Flow classification for software-defined data centers using stream mining," *IEEE Transactions on Services Computing*, vol. 12, no. 1, pp. 105–116, 2019.

[25] F. Eibe, M. Hall, and I. Witten, "The WEKA Workbench. Online Appendix for "Data Mining: Practical Machine Learning Tools and Techniques"," *Morgan Kaufmann*, 2016.

[26] Y. Cheng, J. Chu, S. Radhakrishnan, and A. Jain, "Rfc 7413 - tcp fast open," 2014. [Online]. Available: http://tools.ietf.org/html/rfc7413

[27] H. Zhang, "The optimality of naive bayes," *AA*, vol. 1, no. 2, p. 3, 2004.

[28] R. Kohavi, "The power of decision tables," in *8th European Conference on Machine Learning*. Springer, 1995, pp. 174–189.

[29] R. Quinlan, *C4.5: Programs for Machine Learning*. San Mateo, CA: Morgan Kaufmann Publishers, 1993.

[30] L. Breiman, "Random forests," *Machine Learning*, vol. 45, no. 1, pp. 5–32, 2001.

[31] Y. Freund and R. E. Schapire, "Experiments with a new boosting algorithm," in *Thirteenth International Conference on Machine Learning*. San Francisco: Morgan Kaufmann, 1996, pp. 148–156.

[32] P. Phaal and M. Lavine, "sFlow Specification Version 5," 2004.

[33] M. Menth and F. Hauser, "On moving averages, histograms and time-dependentrates for online measurement," in *Proceedings of the 8th ACM/SPEC on International Conference on Performance Engineering*, 2017, pp. 103–114.

[34] WIDE MAWI Working Group, "Wide traffic archive," http://mawi.wide.ad.jp/.

[35] Center for Applied Internet Data Analysis, "Caida Anonymized Internet Traces 2016," http://www.caida.org/data/passive/passive_2016_dataset.xml.

Raphael Durner received the M.Sc. degree in electrical engineering from the Technical University of Munich, Munich, Germany, in 2014. In November 2014, he joined the Chair of Communication Networks, Technical University of Munich as a member of the research and teaching staff. His research focuses on network softwarization approaches and their impacts on network performance.

Wolfgang Kellerer Wolfgang Kellerer (M'96, SM'11) is a Full Professor with the Technical University of Munich (TUM), heading the Chair of Communication Networks at the Department of Electrical and Computer Engineering. Before, he was for over ten years with NTT DOCOMO's European Research Laboratories. He currently serves as an associate editor for IEEE Transactions on Network and Service Management and on the Editorial Board of the IEEE Communications Surveys and Tutorials.