Port Based Capacity Extensions (PBCEs): Improving SDNs Flow Table Scalability

Robert Bauer, Martina Zitterbart
Karlsruhe Institute of Technology
Karlsruhe, Germany
{robert.bauer, zitterbart}@kit.edu

Abstract—Software-defined networks (SDNs) come with great promises regarding flexible operation of networks. A key component within SDN-switches is the flow table which holds the rules that determine how data streams are handled. The flow table, however, is a scarce resource with a rather limited rule capacity. To soften this well-known hassle, we propose a novel delegation mechanism for OpenFlow-based SDNs called Port Based Capacity Extension (PBCE). PBCE provides the possibility to delegate flows from a switch with many flow table entries to another less loaded neighboring switch without breaking control plane transparency, i.e., without interfering with existing SDN applications. To do so, PBCE uses flow rule aggregation based on ingress ports and a small number of special rules for both switches. In this paper, we present the PBCE delegation middleware along with a prototypical implementation and first promising performance results that demonstrate the feasibility of the approach.

Index Terms—Software Defined Networking, Flow Table Scalability, Delegation, OpenFlow

I. INTRODUCTION

Software Defined Networking (SDN) is a popular and promising technology which enables more flexible control and easier management of forwarding devices. A key component within SDN-switches is the flow table which holds the rules that determine how data streams are handled. Because the flow table of a hardware switch is often implemented with TCAM, the available rule capacity is a scarce and expensive resource. Limited rule capacity of SDN switches is a well-known and intensively studied problem (e.g., [1], [2], [3]). While the realization of fine-grained network policies may require a large rule capacity, currently available hardware often only supports between 1000 and 10000 rules. However, the load of different switches under the control of a single SDN controller may vary, i.e., there might be switches with heavily loaded flow tables and others with spare capacity. We therefore propose an easy to implement delegation mechanism called Port Based Capacity Extension (PBCE) that copes with flow table scarcity by making use of such spare capacity. The core idea of PBCE is to move OpenFlow rules from a switch A (with many flow table entries) to another less loaded switch B. Traffic is then forwarded from A to B, where the fine grained rules are installed to perform the necessary packet processing and return the traffic back to B. Roughly summarized, switch A delegates part of the flow processing to switch B. For simplicity, we will refer to switch A as delegation switch (DS) and to switch B as extension switch (ES).

Assume an example with a heavily loaded switch DS (flow table utilization at 86%) and another switch ES (14%) that is directly connected to DS. The rigid TCAM limitations of DS can severely affect network performance while ES has plenty of unused flow table entries. To utilize this existing spare capacity, we install an eviction rule in DS to forward some of the flows to ES (see Section II). Given an evicted flow F that was originally intended for output port Q on DS, we install a rule in ES that performs the fine-grained packet processing. This rule returns processed packets back to DS, but with Q as a metadata item attached to each packet (say that we translate Q into VLAN field value V). Now it suffices to add another rule to DS that forwards all packets with VLAN==V to port Q. Note that this approach requires a small number of additional rules in DS (equal to the number of ports of DS) and overwrites existing header fields like VLAN or MPLS in order to transport metadata between the two switches.

Although we focus on flow table capacity delegation throughout this paper, the general concept can be adapted to further use cases as well. Examples are:

- Use spare capacity to improve existing services like heavy hitter detection and traffic engineering. Accuracy of SDN-based monitoring for example often scales with the number of TCAM entries [4] and could directly benefit from PBCE.
- Allow incremental hardware deployment. Due to the feature-richness of the OpenFlow specification and diversity in the switch vendor market not all features are available on every forwarding device. With PBCE, potentially heterogeneous devices could be used as a resource pool to incrementally integrate required features into the networking infrastructure.
- Use it as a lightweight alternative to Network Functions Virtualization and Service Function Chaining, if such solutions are unsuited, e.g., because of the general complexity, performance issues or the management and orchestration overhead.

The remainder of this paper is structured as follows: In section II we present the general architecture of PBCE. We then discuss some details of the prototypical implementation in section III and first evaluation results in section IV. Section V contains related work.

978-0-9883045-1-2/16 $31.00 © 2016 ITC
II. PBCE: ARCHITECTURE

Before discussing individual elements of the PBCE architecture, we first present the high level idea of the OpenFlow rules that are necessary to realize delegation (see Fig. 1). Within PBCE, the ports involved with delegation are named Delegation Port (DPort) and Extension Port (ExPort). A DPort is always associated with exactly one ExPort and all packets sent out via the DPort are received at the ExPort and vice versa. Delegation inside PBCE is based on a few simple SDN primitives:

1) Dynamically chosen eviction rules are low priority rules that match only on a specific ingress port of DS. Packets that arrive at this ingress port but do not match any other rule are forwarded to ES, i.e., the output action of the rule is set to a DPort. Because of the low priority, only new flows are affected by eviction rules. Ingress ports whose unmatched traffic is redirected to a DPort are referred to as Eviction Ports (EvPorts).

2) Delegated rules are flow rules initiated by an SDN application that are currently stored inside ES. Packets that match on a delegated rule are always sent back to the ingress port (ExPort) with forwarding metadata attached. Delegated Rules are created by the PBCE middleware.

3) Backflow rules are static rules inside DS that can read the forwarding metadata inserted at ES to forward packets on the associated egress port. If, for example, DS has n physical ports, n different backflow rules are required in order to cover all egress ports.

If these three types of rules are set up properly, packets entering DS via EvPort will follow the path depicted in Fig. 1 (blue numbering) and packet handling takes a detour via the extension switch (this is explained in more detail later). Switches can act as delegation and extension switch at the same time and multiple EvPorts can be mapped to the same DPort.

A. Components of the PBCE architecture

We now discuss the components of the PBCE architecture and the steps necessary in order to realize delegation. The architecture is shown in Fig. 2 and consists of three main building blocks:

- **Monitoring and Configuration:** The configuration component is required to set up the initial behavior of the middleware, e.g., selecting appropriate thresholds for the delegation mechanism. The monitoring component collects data about the current situation in the network which is required to detect bottlenecks or determine the excess capacity of extension switches.

- **Delegation Decision:** The PBCE decision engine uses the data collected by the monitoring component. It keeps track of active delegations and is responsible for triggering and revoking the actual delegation mechanism.

- **Delegation Mechanism:** The PBCE delegation mechanism controls the flow table utilization of a delegation switch and consists of two parts, port eviction and flow migration. Port eviction is responsible for traffic redirection and rule delegation (affects multiple flows linked to an EvPort). Flow migration, on the other hand, is a fine granular mechanism used to further optimize flow table utilization and the link utilization between delegation and extension switch.

1) Monitoring: Two categories of monitoring data relevant to PBCE can be distinguished. First, basic monitoring of the physical devices, which includes the current and maximum flow table utilization of the switches in the network and existing interconnections between them. The monitoring component can easily gather such information from the SDN controller or directly from the switches in a standardized way (e.g., using OpenFlow). The other category consists of extended monitoring information that is not easily available, e.g., the flow-to- ingress-port mapping (FIPM). This mapping describes the relationship between a set of flow rules $R$ inside the flow table and the associated group of ingress ports $P$, i.e., packets originated from $p \in P$ are matched by a rule $r \in R$. The FIPM is of particular importance to PBCE, because it can be utilized to determine the amount of flow rules that can be easily migrated in case of delegation – especially if $|P|$ is small, which seems a reasonable assumption for many SDN scenarios. The FIPM can be derived from the global network view, e.g., by using a collector inside the SDN controller that continuously monitors Packet-In events (with the assumption that follow-up packets for a newly programmed flow will enter the switch at the same port). Note that the FIPM is an optional

![Fig. 1. High level overview of the delegation mechanism](image)

![Fig. 2. PBCE architecture](image)
metric, i.e., the basic operability of PBCE and especially the delegation mechanism does not depend on it.

2) Delegation Decision: The PBCE Decision Engine is responsible for making the decision whether to establish or revoke a delegation between two switches. This decision is executed only once per decision cycle (freely definable, e.g., once per second). The process involves three steps: (1) determine a set of EvPorts, (2) map the selected EvPorts to a suitable DPort and (3) trigger the delegation mechanism. Every port of a delegation switch can be selected as an EvPort, as long as it is not part of any other delegation. Three important parameters affect the selection of a suitable EvPort, as long as it is not part of any other delegation. Three important parameters affect the selection of a suitable EvPort: the amount of flow rules that are currently stored in the flow table of DS where the FIPM points to $p_e$, the estimated arrival rate of new flows that enter DS via $p_e$, and the amount of traffic the two aforementioned groups of flows carry. The first and the second parameter determine the delegation potential of $p_e$, i.e., the amount of rules that could be transferred from DS to ES if $p_e$ is selected for eviction. The third parameter determines the delegation cost of $p_e$, i.e., the amount of traffic that is redirected towards ES in case of eviction. The decision engine therefore should select a set of EvPorts with high delegation potential and low delegation cost. Given that step one of the decision process has – without loss of generality – selected $p_1$ and $p_2$ as EvPorts, the decision engine then selects a DPort based on the free flow table capacity of possible extension switches, the delegation cost of $p_1$ and $p_2$ and the link utilization of the connection to the particular extension switch (step two). In the third step, the decision engine triggers the delegation mechanism explained in the next paragraph. An in-depth analysis of suitable heuristics for the decision process is beyond the scope of this paper. We do, however, present some details on the heuristic currently applied in the prototype in section III.

3) Delegation Mechanism: The delegation mechanism consists of two parts, port eviction and flow migration. Port eviction is required to evict a subset of the flows towards ES. A common way to achieve this is flow rule aggregation, i.e., install coarse-grained rules covering a set of more fine-grained rules. This approach is utilized by various scalability solutions [5], [3] and could very well be adopted by PBCE. Flow rule aggregation, however, has two major drawbacks. It requires a sophisticated and possibly hard to solve algorithm for dependency resolution (flow rules inside a single flow table with intersecting matches, see [3]). And more importantly, it is hard to establish a mapping between ingress port and the covering aggregation rule, because one aggregation rule can be associated with multiple ingress ports (which makes it difficult to inform ES about the original ingress port).

Therefore, we propose a slightly modified aggregation scheme where dynamically installed eviction rules and static backflow rules are used to redirect traffic. The eviction rule is a low-priority rule covering all packets that match solely on a specific ingress port with all the other match values wildcarded. The priority of the eviction rule is set to a value that is above the priority of the default OpenFlow table miss entry (a fully wild-carded rule that is covering all flows with minimum priority) and below the priority of the rules installed by SDN applications. The eviction rules, a static set of backflow rules is installed on DS. Because most of the treatment is taken care of at ES – flow specific matching, header modifications – only a small and static amount of backflow rules is needed, more precisely a maximum of one rule for every possible egress port and a special rule for flooding. In addition to the setup of eviction and backflow rules, the delegation mechanism has to cope with the problem of metadata transport between DS and ES. For PBCE, two metadata items are required. One metadata item is written by the eviction rule in DS and contains the original ingress port $L$ (EvPort in Fig. 1). Without this item, an SDN application could not match on $L$ if a flow gets evicted, because the new ingress port would be an ExPort and not $L$. The second – more important – metadata item is written inside ES and contains the forwarding information $Q$ (c.f., the same variable $Q$ in the introduction). $Q$ is determined by the SDN application and represents the intended output action for a flow on DS. For this to work, the PBCE middleware has to intercept all Packet-In messages sent to the controller if the ingress port is an ExPort and overwrite the ingress port value that is seen by the SDN application with $L$. In addition, every fine-grained rule installed on ES has to be modified in a way, that the output port is changed to the ExPort (or IN_PORT in terms of OpenFlow) and $Q$ is added to the packet for the forwarding decision in DS. The modified rules in ES are called delegated rules.

Metadata transport could be realized by overwriting existing header fields, i.e., use VLAN, MPLS, Flow Label or DSCP to transport $Q$ and $L$ between DS and ES. If this is not feasible – e.g., because the headers are used otherwise – PBCE would have to further encapsulate the packets and use a proprietary header to encode the metadata (we used an overwrite approach and DSCP for the evaluation presented in Section IV). Note that a single eviction rule is not sufficient because the ingress port mapping is lost and delegation would be limited to a single DPort. With the eviction scheme described here, on the other hand, it is possible to adapt the number of ports that are evicted in order to scale the amount of flow rules that are delegated towards an extension switch.

Flow migration is the second part of the delegation mechanism. Migration in this context means that existing rules in the flow table of ES are moved to the flow table of DS or vice versa. Two types of migration can be distinguished: Forward-migration and backward-migration. Forward-migration is applied to flow rules in DS, e.g., to delegate already existing rules to ES after an eviction rule was installed. This is required, because port eviction only affects new flows entering DS. Flows that already existed before the eviction rule was set up will have a higher priority and the corresponding traffic would not be redirected by only using eviction rules (which is intended). Forward-migration is done by adding an already
Because of the high delegation potential, $p_2$ and $p_4$ are selected as EvPorts.

- Because ES has plenty of excess flow table capacity (current utilization is 29%) and the link between ES and DS is not saturated, $p_0$ is selected as a single DPort. $p_{24}$ of ES thus becomes an ExPort.

After the decision process, PBCE initiates the delegation mechanism, which in turn installs eviction rules for $p_2$ and $p_4$ and seven backflow rules to cover the return traffic – one for each physical port of DS (including DPorts and EvPorts) and one to handle flooding. Note that the 852 flows pointed out earlier are not yet redirected to ES because of the low priority of the eviction rules. This is done by forward-migration, which installs altered versions of the rules in ES (forwarding metadata, output action set to ExPort) and deletes them at DS. The bottom of Fig. 3 shows the new situation in the example scenario after PBCE has performed the changes described above. The 852 flows are now stored in the flow table of ES and a small amount of eviction and backflow rules are added to the flow table of DS (labelled with $\circled{5}$). Flow table utilization of ES is reduced from 93% to 50%, utilization of ES is increased from 29% to 50%.

Packet forwarding and reactive flow setup is now handled as follows: If a packet arrives on an EvPort at DS and no matching rule is found in the flow table, it is forwarded using the DPort because of the eviction rule (illustrated by $\circled{4}$ in Fig. 3). Metadata about the ingress port (EvPort) is added to every single packet. If the packet arrives at ES and a delegated rule is present, it is directly forwarded back to the ExPort with attached forwarding metadata. Without such a rule, the default rule of ES will forward the packet to the controller, where the PBCE middleware detects a Packet-In coming from an ExPort. The new flow is then not installed as a normal rule in DS but as a delegated rule in ES. Either way, packets are sent back to DS ($\circled{6}$), where the forwarding metadata and corresponding backflow rules are used to forward the packet to the originally intended egress port ($\circled{6}$). Backward-migrations are applied if one of the delegated flows requires too much bandwidth on the DPort-ExPort connection (by installing a new rule for the flow in DS with higher priority and deleting the delegated rule).

### B. Delegation Workflow

Finally, this section gives an example for the interaction of the different PBCE building blocks and the resulting delegation workflow. Fig. 3 shows an example scenario with six OpenFlow-enabled switches (DS, ES, A-D) with and without delegation. 852 rules in the flow table of DS are associated with flows originating from A and B, i.e., 43% of the available rule capacity is occupied by flow rules where the FIPM points to $p_2$ and $p_4$. These 852 rules are highlighted by $\circled{1}$ and $\circled{2}$ in Fig. 3 and represent the delegation potential of $p_2$ and $p_4$ in terms of possible forward-migrations. As soon as PBCE monitoring detects the high flow table utilization of DS (93%), the decision engine determines both, an appropriate set of EvPorts and one or more DPorts with sufficient resources:

- Because of the high delegation potential, $p_2$ and $p_4$ are selected as EvPorts.

---

**Fig. 3. Example without (top) and with (bottom) PBCE enabled.**
the core functionality of the controller and individual SDN applications only have to change their super class to the middleware wrapper. The main driving factor behind this design decision was that a middleware can easily memorize the ingress port of packets responsible for the installation of new flows which is required for an estimation of the FIPM.

Metadata transport is required for both, eviction rules in DS (otherwise the original ingress port would be lost) and delegated rules in ES (forwarding metadata). Because of the limited range of values (port-count of DS) the prototype uses the DSCP header field, which is sufficient for switches with a port count smaller than 63. The DSCP field is cleared inside the backflow rules.

The delegation decision of the prototype is primarily based on port specific flow arrival rates. We denote $\delta_j(t_1, t_2)$ as the number of flows arrived on ingress port $p$ between $t_1$ and $t_2$. The current delegation potential is then defined as

$$w_t(p) = (1 - k) \cdot \delta_p(t, t - 1) + k \cdot h_t(t, t - n)$$

where $n$ is the number of considered samples in the near past and $k$ is a weighting factor. We found $n = 10$ and $k = 0.5$ to be acceptable values for our scenario after a (small) series of experiments with different values for $n$ and $k$. For the decision process, $w_t(p) \forall p \in \text{Ports} \setminus \text{(DPorts} \cup \text{EvPorts})$ is determined and the port with maximum $w_t(p)$ is chosen as EvPort. For DPort selection, the prototype selects the port with minimum link utilization. Two thresholds determine whether the delegation mechanism is triggered (executed once per decision cycle): The delegation threshold defines the upper limit for flow table utilization of DS, i.e., as soon as this threshold is exceeded, a new EvPort is determined. If the utilization falls below the revocation threshold, delegation is canceled for one EvPort. In addition, the decision process is connected to a grace period to avoid overregulation. This is done by incrementing a counter every time an EvPort is selected or revoked. The counter is decremented linearly and the delegation decision is delayed if the counter is greater than zero (otherwise, too many EvPorts may be selected).

IV. EVALUATION

We now evaluate the feasibility and performance of the prototypical PBCE implementation outlined in the previous section with a series of experiments. The key findings are:

- PBCE is able to efficiently control the average flow table utilization of a delegation switch.
- Control plane overhead in terms of CPU consumption and control traffic is dominated by monitoring and scales well with increasing delegation workloads. For smaller workloads, PBCE induces low overhead.
- Only 0.1ms of additional end-to-end delay is added for delegated flows in the data plane.

A. Experimental Setup

To cover a wide area of different evaluation scenarios, we first introduce a generic workload indicator called table ratio (TR). Given the average flow table utilization of the delegation switch ($T_{DS}$) and the extension switch ($T_{ES}$), we define TR as $T_{ES}/T_{DS}$. Because we only allow delegated rules in ES (for simplicity of the evaluation), this parameter gives us a good indication on how much delegation work the PBCE middleware has to accomplish. For $TR = 0$, no flow is delegated to ES. If over-utilization occurs infrequently, the average number of rules in the flow table of ES is going to be small (compared to DS) and the value for TR is low. In case of constant and long-term over-utilization, on the other hand, the value for TR is much higher (possibly $TR = 1$ indicating that – on average – half of the flows inside DS are delegated to ES).

Up to 200 instances of iperf3 with individual settings (bandwidth, flow duration, number of parallel sessions) inside a mininet environment are used for evaluation. We consider flow arrival rates between 50 and 350 flows/second and delegation thresholds in the range of 50-1600 to create various table ratios and flow table utilization patterns. The CDFs in Fig. 4 further characterize the experiments used for evaluation throughout this section. A single experiment lasts for 400 seconds and is executed in the testbed depicted in Fig. 5, which consists of three physical nodes. Each node is equipped with an Intel(R) Xeon(R) E5-2640 v2 processor (2.50 GHz, 2 sockets, 4 cores/socket) and separate physical 1Gbit/s networks for control and data plane traffic. One node runs DS (Open vSwitch v2.4.0) and the mininet tool. Note that the virtual hosts are not directly attached to DS to allow flexible scale up of the traffic generation endpoints without changing the port count. ES (another Open vSwitch) is placed on a physically separated node to enable measurements with real network limitations (link capacity, latency). The Ryu controller runs a reactive SDN application that installs forwarding flows based on TCP ports. Destinations for generated traffic are selected randomly.

B. Functional Evaluation

For the functional evaluation, we first demonstrate that flow table capacity delegation reduces the flow table utilization of
This section analyzes the flow table utilization (sum of flow table sizes exceeding the threshold). Fig. 7 therefore shows four experiments where the amount of delegated flow rules by adapting the delegation threshold without further port evictions until the load on DS increases again at t=105 and the amount of EvPorts is raised to 11. After 130 seconds, utilization falls below the delegation threshold (100) and two delegations are revoked (i.e., the eviction rules for two EvPorts are deleted). This shows that the PBCE delegation mechanism is able to cope with flow table over-utilization by dynamically adding and removing EvPorts depending on the current situation at DS.

As a second step, we illustrate that PBCE can control the amount of delegated flow rules by adapting the delegation threshold. Fig. 7 therefore shows four experiments where the flow table of DS has to store up to 800 concurrent rules between the 150 and 200 second mark. The leftmost experiment can be seen as a baseline, i.e., PBCE is disabled and all rules are stored in the flow table of DS. The three experiments on the right were conducted using the exact same traffic pattern but with different delegation thresholds (600, 400 and 200). We see that the flow table utilization of DS is reliably kept below the configured threshold and more flow rules are stored at the flow table of ES (which results in an increased table ratio). It should be noted, however, that the accumulated flow table utilization (sum of flow table sizes from DS and ES) is slightly higher (896-917) compared to the values measured at the baseline experiment (831). The reasons for this are twofold: First, PBCE requires a static amount of backflow rules (17 in this case) that are not required if PBCE is disabled. The same applies for eviction rules. And second, the prototype currently relies on the OpenFlow idle timeout (set to 5 seconds) to delete flows in ES leading to duplicate rules in both flow tables for a short time frame. Note that this behavior can be prevented in exchange for additional controller communication by explicitly deleting migrated rules in ES.

In the last part of the functional evaluation, we show that PBCE is capable of providing effective countermeasures for flow table over-utilization. We therefore analyze the flow table utilization of DS for all the experiments where PBCE is enabled. The results in Fig. 8 (left plot) show that the prototypical implementation is able to keep the maximum average utilization (i.e., the maximum value for average utilization measured within all experiments for the given threshold, printed as a red line with square markers) below the intended value independently of delegation workloads and flow arrival rates. Note that the flow table utilization of DS will always exceed the delegation threshold for a short time frame, because the delegation mechanism needs to be executed and new flows may arrive in the meantime, which is especially true for scenarios with high and bursty flow arrival rates. This effect is illustrated by the utilization peaks exceeding the delegation threshold highlighted in the upper left corner of Fig. 6 (black arrows). In order to evaluate efficiency, it is important to analyze these peaks. This is done by looking at the peak duration, i.e., the duration where the flow table utilization of DS stays above the delegation threshold after the threshold is exceeded. The CDF at the right side of Fig. 8 characterizes all peak durations for all experiments. Duration is measured in decision cycles, i.e., the minimum frequency with which the decision engine selects new EvPorts (1 second). The results show that peak utilizations above the delegation threshold can be resolved within 6 cycles in 84.6% of the cases. Less than 2% of the cases require 10 or more cycles to push the flow table utilization back below the delegation threshold.

**C. Control Plane Implications**

1) **Computational Overhead:** This section analyzes the overhead related to CPU consumption. We show that PBCE scales well with different delegation workloads and computational overhead is below 5% for smaller table ratios.
calculate the overhead of a PBCE-enabled experiment, we determine the CPU time (seconds) of the Ryu controller process for the total runtime of that experiment and compare it to an appropriate baseline experiment. The baseline experiment has (approximately) the same average flow arrival rate but PBCE and all monitoring is disabled, except for aggregate flow table statistics. Note that we cannot calculate the flow table utilization of the baseline experiment without aggregate flow table statistics. Because the monitoring frequency has a great influence on CPU consumption and control traffic, we consider experiments with two different frequencies m=3 and m=6 (i.e., gathering of individual flow statistics every m seconds). The results in the top left of Fig. 9 show that the amount of CPU time increases linearly with the average flow arrival rate and higher delegation workloads require more CPU time. The plot in the top right of the same figure shows CPU consumption for table ratios between 0 and 14, divided into four different flow arrival rate classes c1-c4 where c1 represents experiments with an average arrival rate of 50-80 flows/sec (c2=80-130 flows/sec, c3=130-180 flows/sec and c4=180-250 flows/sec). While higher flow arrival rates lead to higher CPU consumption, the differences in CPU consumption for increasing table ratios inside one flow arrival rate class are quite small and follow a linear trend.

A more detailed CPU overhead analysis can be seen in Fig. 10. The CDFs in the top contain the deviation of CPU consumption between PBCE-enabled and baseline experiments.

Overhead is expressed as a percentage, grouped by table ratio TR and monitoring frequency m. The results show that low delegation workloads induce low computational overhead, which seems reasonable because PBCE is inactive most of the time. For m=6 and TR < 0.1 the overhead is below 2.98% for 95% of the experiments and below 1.53% for 80% of the experiments. Higher workloads and higher monitoring frequency increase the computational overhead, e.g., for m=3 and TR between 0.5 and 1, overhead is below 17.25% for 95% of the experiments. Extreme delegation workloads with table ratios greater than 5 induce more, but still acceptable overhead (27.85% for m=3 and 18.75% for m=6 in 99% of the cases).

2) Control Traffic Overhead: Similar to computational overhead, the overhead in terms of control traffic is low for smaller table ratios. We use the same methodology and input data and determine the total amount of exchanged control traffic between SDN controller and switches for PBCE-enabled and baseline experiments (in MBytes). Control traffic is partitioned into receiving direction (RX) and transmitting direction (TX). RX consists of the control traffic sent from the switches and received by the controller and is dominated by monitoring results, e.g., aggregated and detailed flow statistics that are queried periodically. TX covers only the control traffic sent from the controller and consists of OpenFlow messages to install and delete rules (and monitoring queries). The two plots in the bottom of Fig. 9 show that especially the overhead for the receiving direction (Ctrl RX) increases with higher table ratios. For high flow arrival rates and high delegation workloads, we see 15 MBytes of additional received control traffic compared to less than 3 MBytes of additional traffic in the transmitting direction. This can be explained with the considerable amount of monitoring data required by PBCE, which is disabled for baseline experiments.

The detailed overhead analysis for control traffic is shown in the bottom of Fig. 10. For m=6 and TR < 0.1, control traffic overhead is below 5.03% for receiving direction and below 0.9% for transmitting direction (95% of the experiments). While the overhead for the transmitting direction stays low in case of higher delegation workloads (always less than 10%, even for m=3 and TR > 5), the overhead for the receiving direction increases significantly and reaches 86.91%
for extreme table ratios and high monitoring frequency. Note that PBCE can utilize already existing sources for monitoring data to reduce the overhead in the receiving direction, e.g., routinely executed monitoring necessary to establish the global network view of SDN.

3) Monitoring Tradeoff: We now show that the monitoring frequency is closely linked to computational overhead and the amount of traffic that must be redirected to ES. We therefore execute another experiment with static table ratio (TR = 1) and average flow arrival rate (100 flows/second) and gradually reduce the monitoring frequency. The results in Fig. 11 show that the computational overhead is significantly lower if the frequency is reduced. Less monitoring, however, affects traffic optimizations on top of the physical link between ES and DS because high bandwidth flows are not detected quickly enough. This opens up the field for various tradeoffs and optimizations, e.g., by integrating approaches like PayLess [7] or OpenSample [8].

D. Data Path Implications

Finally, we show that the redirection to a neighboring extension switch does not severely degrade the performance inside the data path. Therefore an experiment was conducted that examines the additional delay induced by delegation. The Distributed Internet Traffic Generator (D-ITG, [9]) is used in this experiment to provide accurate statistics with packet level granularity. The detailed setup is as follows: 300 consecutive D-ITG flows (TCP, constant packet rate of 30Mbit/s) are created between two virtual hosts h1 and h2 where every D-ITG flow lasts for one second before the next one is started. After ten seconds, the port that connects h1 and h2 is selected as an EvPort and the delegation is revoked after another ten seconds. This continues periodically until all 300 flows are processed. Fig. 12 shows the average delay for the 300 D-ITG flows starting with the first flow at index 0. Without delegation, the average end-to-end delay is measured as approximately 0.058 milliseconds, which is reasonable, because forwarding is handled locally inside the Open vSwitch (OVS). This delay is increased to a value between 0.15 and 0.2 milliseconds if delegation is enabled. Note that, in the latter case, packets are transmitted via a physical link to the OVS on a remote server and then back via the same link to DS (cf. test setup in Fig. 5). 0.1ms of additional delay seems acceptable for many scenarios. We compared the results with tests on a physical SDN switch, where the delay is even smaller (less than 0.13ms of total end-to-end delay for delegated flows on a Brocade ICX 6610), because the forwarding is done in hardware and not in software.

![Fig. 9. CPU consumption and control traffic divided in receiving (RX) and transmitting (TX) direction for all PBCE-enabled experiments for flow arrival rates between 100 and 300 flows/sec. In addition, the plot in the top right shows the CPU consumption for 1795 experiments with various table ratios grouped by four average flow arrival rate classes from c1=50-80 flows/sec (blue line in the bottoms) to c4=180-250 flows/sec (red line in the top).]

![Fig. 10. CPU and control traffic overhead for the experiments shown in Fig. 9 grouped by table ratio (TR) and monitoring frequency (m). Control traffic overhead is subdivided in receiving (RX) and transmitting (TX) direction.]

![Fig. 11. Tradeoff between monitoring overhead and utilization of delegation ports (100 flows/second, TR=1).]
V. RELATED WORK

Using flow aggregation or flow table decomposition to achieve scalability for software defined environments is a well-known research area [1], [2], [10], [11], [12]. Within this scope, DIFANE and CacheFlow are two prominent examples closely related to our solution. In DIFANE [5], all switches are equipped with coarse-grained, low priority partition rules that can encapsulate and redirect incoming traffic to authority switches where the appropriate packet handling is provided. Because the partition rules cover the whole flowspace, packets never have to leave the data plane and powerful partitioning and rule caching is required to assure efficiency. While DIFANE tries to optimize the network as a whole, PBCEx explicitly focuses on localized scalability improvements and can be deployed without hardware modifications. CacheFlow [3], on the other hand, presents a scalability solution where large flow tables of software switches are utilized as co-located extension devices. Because it also wants to preserve existing OpenFlow semantics, CacheFlow has to deal with similar problems and we therefore borrowed some of their ideas (e.g., the general idea of the ingress-port-tagging mechanism for backflow rules). We see the main difference between CacheFlow and our solution as twofold: We want to provide greater flexibility by not relying solely on specialized extension devices but by exploiting excess capacity of the existing network infrastructure. And, secondly, our port-based eviction scheme could be a valid alternative to the CacheFlow algorithm (for future work, we plan to implement and compare both approaches). Similar to CacheFlow, [13] also utilizes software switches to improve performance but focuses on faster flow table entry installation. Various other SDN (control plane) scalability approaches exist [1], [14], [15] but – to the best of our knowledge – capacity delegation as it is described here was not yet intensively studied.

VI. CONCLUSIONS AND FUTURE WORK

This paper introduces a novel architecture for port based capacity extensions (PBCEx). We propose a port based rule aggregation scheme to redirect traffic to extension switches and present an easy to implement delegation mechanism for OpenFlow-based SDNs where excess capacity of neighboring switches is utilized to dynamically improve scalability of the flow table. First evaluation results with a prototypical implementation show that the PBCEx middleware can efficiently and reliably control the flow table utilization of a delegation switch. Control plane overhead is low for smaller delegation workloads (well below 5%) and only 0.1ms of additional delay is added to the data path (for delegated flows). Important areas for future work include architectural extensions to support light-weight service chaining, delegation of other resources (e.g., optional OpenFlow features or monitoring capabilities) and an in-depth analysis of efficient delegation heuristics. We also plan to evaluate the approach using more complex and realistic topologies and compare the efficiency of PBCEx with existing approaches like CacheFlow.

REFERENCES


Fig. 12. Average delay for 300 flows where delegation is disabled and enabled every 10 seconds.