Taking the Sting out of Flow Update Peaks in Software-Defined Service Chaining

Jeremias Blendin, Julius Rückert, Sascha Bleidner, and David Hausheer
Peer-to-Peer Systems Engineering Lab, Technische Universität Darmstadt
Email: {jblendin,rueckert,sascha.bleidner,hausheer}@ps.tu-darmstadt.de

***Abstract*—Dynamic network service chaining allows network operators to apply network services to customer traffic on demand and in a highly flexible manner. SDN and particularly OpenFlow-based approaches have been presented that exploit the flexibility and feature richness of OpenFlow for service chaining in data center settings. These systems often use network architectures similar to MPLS, where network traffic is processed at the network edges and the network core conducts simple packet forwarding only. This approach is suitable for many use cases due to the complementary characteristics of the hardware devices used in the core and the virtual switches used at the edge on virtualization hosts. Yet, this leads to a concentration of processing flow updates solely at the network edges, which can cause lowered QoS during flow update peaks, e.g. when a virtualization host fails. To tackle this problem, in this paper a concept is presented and evaluated that offloads OpenFlow rule updates from software-based edge switches to exploit hitherto unused resources on hardware switches in the core of the network. Furthermore, an analytical model is presented that allows describing the expected gain of the approach based on characteristics of the used OpenFlow devices.**

I. Introduction

Dynamic network service chaining allows network operators to apply network services to customer traffic on demand and in a highly flexible manner [1]. Service chains are constructed by steering network connections through chains of service instances. Each service instance implements a part of a network service and is often operated in a VM running on a virtualization host, which is operated in a data center environment [2]. The optimization of OpenFlow-based network service chaining approaches relies on the different performance characteristics of hardware switches and software switches, which are all assumed to be OpenFlow-enabled. In a typical data center environment, software switches, termed virtual switches, run on the virtualization hosts, where they provide connectivity for VMs [3]. For this, the virtualization hosts are interconnected by hardware switches.

OpenFlow relies on rules for implementing network behavior; for each incoming packet, the corresponding rule is selected by matching bits in the packet header with a bit matcher stored in each rule. On virtual switches, OpenFlow rule tables are implemented by optimized data structures, while on hardware switches these tables are often implemented by specialized memory called TCAMs. TCAMs match incoming network packets with stores rules in one clock cycle, which enables high speed packet forwarding, but comes at the cost of slow updates [4]. In contrast, OpenFlow rule table implementations in software offer lower lookup performance, but a much higher flow update rate than hardware implementations. This is one reason why many SDN forwarding approaches rely on concepts similar to MPLS, with simple and fast forwarding in the network core and classification and routing decisions at the edges [5]. Therefore, the most OpenFlow rules and updates are required at the edge switches where the processing capabilities are high, while very few rules as well as virtually no rule updates are required in the core of the network. Approaches for optimizing OpenFlow-based service chaining systems [6] as well as data center traffic in general [3] rely on this concept. As demonstrated in literature, this approach works well and is a good match for the respective capabilities of hardware and software switches. However, the entire workload for rule changes in the network lies on the software switches, while the resources available at the hardware switches are largely unused. When peaks in the flow rule update load occur, the software switches can be overloaded, leading to increased flow rule update latencies and, as a consequence, to a lowered QoS. Such peaks can be caused by a large number of changes in the service chaining configuration e.g. by flash crowds and the migration of OpenFlow rules from one virtualization host to another. The latter occurs when one virtualization host in operation fails and the affected users and their OpenFlow rules are moved to a new virtualization host.

To soften the effects of rule update peaks, in this paper a system is proposed that exploits hitherto unused resources in core switches to reduce the load on the edge switches. The described approach transfers compatible rules from software switches on the virtualization hosts to hardware switches in the core of the network. The impact on the failure time in case of a failing virtualization host is analyzed in a testbed with three virtualization hosts and a Broadcom ASIC-based OpenFlow hardware switch (*HwSwitch*). Furthermore, an analytical model is derived that describes the reduction in the failure time for a given OpenFlow rules installation performance ratio between software and hardware switches. To the best knowledge of the authors, an investigation of the peak rule update load of OpenFlow-based dynamic network service chaining systems has not been reported yet. Many of the existing approaches are expected to benefit from the findings presented in this paper.

The remainder of this paper is organized as follows: The system design is presented in Section II. The prototype, its evaluation and results are presented as well as discussed in Section III. Related works are discussed in Section IV. Finally, a conclusion in drawn in Section V.

II. System Design

A typical usage environment for service chaining systems today is the Gi-LAN, which is the reference point in the

978-3-901882-77-7

architecture of cellular networks, where mobile networks are interconnected to the packet networks [2]. Figure 1 depicts a conceptual service chaining network. Mobile users, which are denoted by H_1 and H_2 in the figure, initiate connections to hosts on the Internet. Two example hosts are denoted as H_3 and H_4 in the figure. Network connections are automatically steered through a subset of the available service instances and denoted by $SI_1 - SI_4$ as well as *DPI* and *Firewall* in the figure, as specified by the service chain selected by the mobile user. Mobile users are connected through the mobile network to the edge switch of the dynamic service chaining system. The same is true for the Internet link. Inside the service chaining system, the edge switches are connected to core switches that interconnect not only the edge switches but also the virtualization hosts. On the virtualization hosts, the interconnection is provided by virtual switches, denoted by VS_1 and VS_2 in the figure. All switches in the service chaining system are managed by an OpenFlow controller. The core switches are assumed to be hardware devices, with a high port density and a high forwarding capacity but with a low OpenFlow rule update performance. The virtual switches inside the virtualization hosts are software switches. Although their forwarding performance is lower than that of hardware switches [7], it is sufficient for interconnecting the locally attached service instances. The edge switches are not the focus of the investigation in this paper, they can either be hardware or software switches.

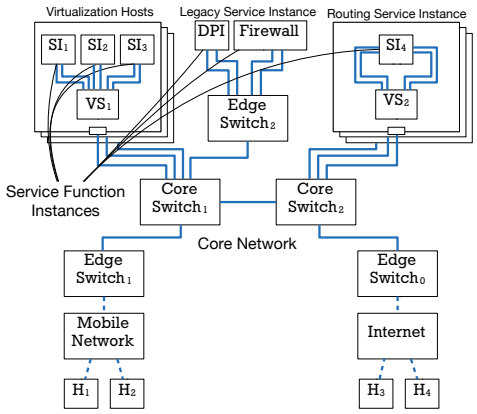

Fig. 1. Dynamic Network Service Chaining and its Entities

The dynamic service chaining system is based on the approach described in [2], a previous work by the authors. It is not described in detail in this document, for a complete description please refer to [2]. The approach aims at easing the deployment of service instances compared to other approaches and exhibits a less complex approach to managing rules and user flows compared to StEERING [6] and SIMPLE [8]. Each service instance is connected by at least two ports to the virtual switch running on the virtualization host. The service instances are isolated from the rest of the network, enabling to use the same network settings on each service instance, with the only exception being the MAC address of the interfaces. However, as the forwarding system presented in [2] is not optimized, an improved approach is introduced. It is based on SwitchReduce [3] and uses a source routing approach, where packets are steered through the network by a stack of VLAN tags in each packet header. On the switches, a VLAN tag is popped from the stack whose ID encodes the output port that the packet is sent to. Thereby, each core switch contains exactly one rule per network port in use. At the edge switch, users are identified by the IP address of their mobile device. The OpenFlow controller retrieves the service chain selected by the user and calculates the path to the first service instance in the chain. Then, the VLAN tags corresponding with the output ports on each switch on the path to this service instance are pushed to the stack and the packet is forwarded to the next core switch. When the packet arrives at the virtualization host, the last VLAN tag is popped, the required packet modifications are applied and the packet is forwarded to the service instance. After the service instance has completed the processing, the packet is forwarded back to the virtual switch. There, it is processed again and the path to the next service instances is encoded as VLAN tags in the packet header. The service chain for each user is implemented individually by a chain of point-to-point connections.

TABLE I. OPENFLOW RULE METRICS

Metric	Switch Type	Number of Flow Rules
Max. number of rules per switch	Edge switch	$2 \times$ (Num. users in the system)
	Core switch	(Num. local ports in use)
	Virtual switch	$4 \times \Sigma_{i \in \{\text{Local SI}\}} \text{SI}^i$ (Num. users)
Max. rule updates per user modification	Edge switch	$2 \times$ (Num. users in the system)
	Core switch	None
	Virtual switch	$4 \times$ (Num. service instances of user)

Performance metrics related to the OpenFlow rule usage of this approach are listed in Table I. On the core switches, the maximum number of rules per switch and the number of rule updates per user modification are fixed values and do not change with the load in the system. These corresponding rule update capacities are free to be used for other purposes. Rule update peaks for virtual switches are identified by comparing the update rate with a configurable threshold. When a rule update peak occurs, the rule update load is automatically distributed between the virtual switch and the directly connected hardware switch. The distribution of the load is configured based on the flow update performance ratio of virtual and the hardware switch. The selection of an appropriate value is investigated in Section III. However, not every rule can be moved from a virtual switch to a core switch. The movement of rules to another switch should not impact the performance of the whole service chaining system. Hence, rules that forward packets that stay inside the same virtualization host are not moved. Only rules, which are involved in forwarding packets from one virtualization host to another, and therefore pass through a core switch anyway, are eligible for moving. A rule is installed on each virtual switch that forwards all traffic that is not matched by a local rule to the next core switch. On the core switch the packets are identified by the IP address of the user and the source MAC address, which identifies the packets source port. Thereby, the rules can be used in a similar manner than on the virtualization host. Matching on incoming ports is replaced by matching the corresponding MAC address.

III. PROTOTYPE & EVALUATION

The approach is evaluated using a testbed with three servers as virtualization hosts and a Broadcom ASIC-based OpenFlow hardware switch termed *HwSwitch*. The failure of a virtualization host in the service chaining system is investigated with different shares of the rule update load moved from the

software switch to the *HwSwitch*. The goal of the evaluation is to analyze the reduction of the total service outage for the affected users by utilizing hitherto underused resources.

A. Scenario & Workload

The service chaining topology used is depicted in Figure 2. It consists of a representative, yet small part of a service chaining system. The edge of the service chaining network is not relevant for the analysis; therefore a virtual switch running on the same server with the emulated users is used. The core network of the service chaining system is represented by *HwSwitch*, which interconnects the virtualization nodes 1 and 2 with the edge switch and users to the two virtualization hosts. Several service instances, denoted by SI in the figure, are hosted on each of the two virtualization hosts. To emulate a realistic service chaining system, both systems carry load, there are no dedicated virtualization nodes on standby. The focus for the investigation is the OpenFlow rule installation performance; a realistic traffic load on the network is not necessary. Therefore, using 1GbE interfaces as done in the testbed is appropriate, even though in a real service chaining system 10GbE interfaces are state of the art.

The node failure is emulated by disabling the network connection between Virtualization Host 1 (VH1) and the core switch. This event triggers the installation of all rules that were used on Virtualization Host 1 before on Virtualization Host 2 (VH2), creating a spike in the OpenFlow rule installation load. Before the failure event, both virtualization hosts are used for user traffic. One half of the users, termed User Group A, use four service instances on Virtualization Host 1 and 2 each. The other half of the users, termed User Group B, are assumed to use four service instances on Virtualization Host 2 only. The time from a failure of Virtualization Node 1 until all members of User Group A are successfully moved to Virtualization Node 2 is referred to as failure time. The objective of the evaluation is to investigate if moving parts of the OpenFlow rule update load from the virtual switch to the adjacent core switch reduces the failure time for affected users.

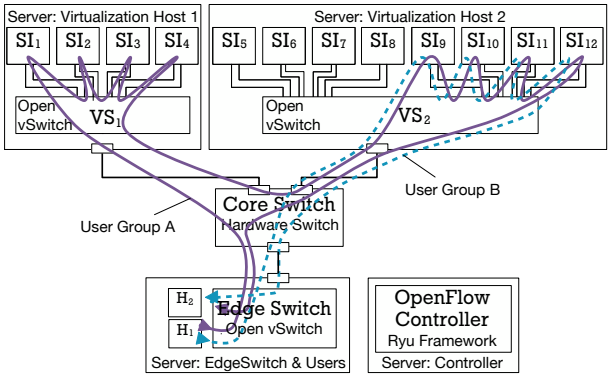

Fig. 2. Testbed Topology and the Evaluated Service Path

To the best knowledge of the authors, no data sets of real service chaining systems are available for research. Therefore, assumptions are made for determining the load in the testbed for evaluation. A maximum of 3,000 users per server in normal operations and 6,000 in case of a failover are used. User Group A use a service chain containing eight service instances, User Group B uses four network services as depicted in Figure 2. The failing Virtualization Host 1 runs four service instances, the other one eight. The service instances are assumed to be shared between all users. The resulting number of used OpenFlow rules for a single virtual switch can be derived by the formula given in (1).

$$\text{OpenFlow Rules}_{\text{VS}} = 2 \times \Sigma_{u \ in \ \text{users}}(1 + (\text{Num. SI}^{\text{u}}_{\text{local}})) \quad (1)$$

When Virtualization Host 1 fails, users of Group A are relocated to use additional service instances on Virtualization Host 2. It is assumed that all same service instances that are available on Virtualization Host 1 are also available on Virtualization Host 2. Between 5 and 50% percent of user rules are moved from the virtual switch to the core switch during rule installation. However, as not all rules are eligible for movement, two rules per user, the ones from and to the edge switch, are moved. All parameters used in the evaluation are listed in Table II.

TABLE II. PARAMETER VALUES USED IN THE EVALUATION

Parameter	Investigated Values			
Users in the service chaining system	1,000	2,000	5,000	6,000
Users in Group A that are affected by the node failure	500	1,000	2,500	3,000
Total number of rules installed during failover	5,000	10,000	25,000	30,000
Total number of rules installed during failover & eligible for movement	1,000	2,000	5,000	6,000
% of total number of rules installed during failover & eligible for movement & moved to the core switch	5, 10, 15, 20, 25, 30, 35, 40, 45, 50			

B. Measurement Methodology

Measuring time in OpenFlow networks requires detailed investigation of the components and consideration of potential sources of errors. All time measurements are conducted on the same server to ensure accurate results. The completion time of the OpenFlow rule updates is measured through the Barrier command of the OpenFlow protocol. The OpenFlow controller sends an OpenFlow barrier request after the transmission of the rule installation commands. The point in time when the resulting barrier reply is received by the Controller Server is used as completion time of the rule installation process. Since the barrier request is known to not be processed according to the OpenFlow standard by all implementation, the behavior of the *HwSwitch* and Open vSwitch is investigated to this end. The results for the *HwSwitch* are depicted in Figure 3. The barrier replies are measured using the tcpdump tool to ensure the results are not skewed by added processing overhead introduced in the OpenFlow controller. In parallel, ping requests are send through the switch every 500ms. They are crafted to be forwarded only when the new OpenFlow rules are installed. The results indicate that the device behaves according to the OpenFlow standard. The discrepancies in the data for 3,000 and 4,000 users can be explained by measurement artifacts that are caused by the small durations of the measurement. While high-precision measurements with earlier versions of Open vSwitch revealed discrepancies between the barrier reply and the passing of the first packet through the switch [9], the results at the measurement granularity of 500ms did not reveal significant differences. An investigation of the OpenFlow rule installation characteristics of both Open vSwitch and hardware

switch reveal that using different OpenFlow rule priorities leads to a signification rule installation performance reduction on the *HwSwitch*. Therefore, all OpenFlow rules used use the same priority and some matching network traffic is applied during the evaluation.

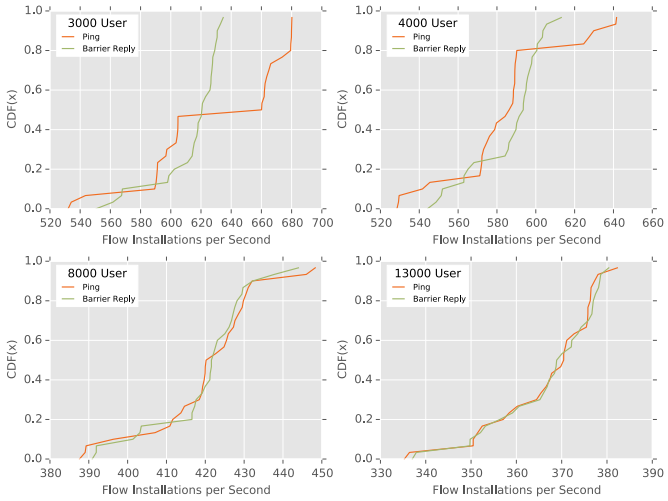

Fig. 3. Comparison of the Barrier Reply Time and the Effective Flow Rule Installation Time on the *HwSwitch*

C. Implementation & Testbed

The relevant parts of the service chaining systems are implemented using on the Ryu OpenFlow controller. The VLAN-based forwarding mechanism could not be used with the *HwSwitch*, because it does not support adding and removing VLAN tags. Although not identical to the VLAN-based approach, a MAC address-based mechanism is used similar to PortLand [10] that is equivalent for the evaluated scenario; modifying MAC addresses is supported by the *HwSwitch*. Instead of a VLAN tag stack, the MAC address is used as an array of tags. Figure 4 depicts an example tag, which stores the user ID, the output port on the edge switch, as well as the output port on the core and the virtual switch. The drawback of this encoding scheme is that it does not support paths with more than three hops. However, such paths are not required for the evaluation.

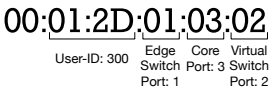

Fig. 4. Destination MAC Path Information Encoding

The topology of the testbed as well as its components is depicted in Figure 2. Information on the soft- and hardware used for the components is listed in Table III. The hardware used for the virtualization hosts is considered low end compared to a modern server CPU. However, the main task of the virtualization host is to run service instances and forward packets. Therefore, only a fraction of a state-of-the-art server CPU is likely to be dedicated to the process that processes the rule updates; the performance level of the servers in the testbed is considered similar to this fraction of processing resources. A small amount of traffic is passed through the system to ensure it is working. The performance required for the forwarding of this traffic through the virtual switch and the service instances is negligible.

TABLE III. HARDWARE INFORMATION OF THE PROTOTYPE

Node	Hardware	Software
Edge Switch & Users	Intel Pentium G640 CPU, 8GB RAM, Intel 82579LM NIC	Ubuntu 12.04, Open vSwitch 2.3.1
Virtualization Host	Intel Pentium G640 CPU, 8GB RAM, Intel 82579LM NIC	Ubuntu 12.04, Open vSwitch 2.3.1
Core Switch	Broadcom ASIC-based Hardware Switch (*HwSwitch*), 48 1GbE & 4 10GbE ports	–
Controller	Intel Core i5-3470 CPU, 32GB RAM, Intel 82579LM NIC	Ubuntu 12.04, Ryu 3.19

D. Results & Analysis

A comparison of the OpenFlow rule installation performance of Open vSwitch 2.3.1 and *HwSwitch* is given in Figure 5. For this measurement, both switches are investigated individually. For both switches, the performance is not dependent on the total number of OpenFlow rules installed on the devices. The number of flow installations per second is, with about 800, more than an order of magnitude slower on the core switch than on the virtual switch with about 10,000. The large performance difference is surprising; a previous investigation with Open vSwitch version 1.7.0 running on an Intel Xeon 3210 resulted in about 400 flow updates per second [11].

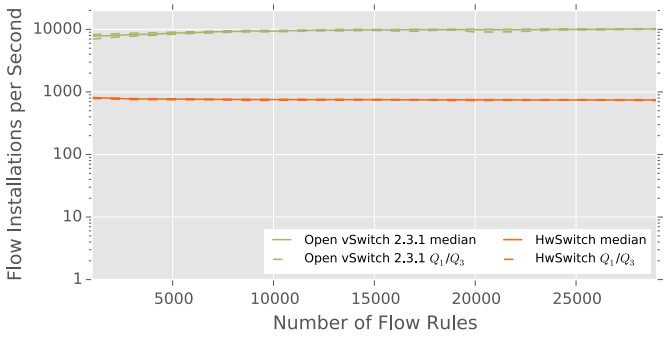

Fig. 5. Comparison of the OpenFlow Rule Installation Performance of Open vSwitch 2.3.1 and the *HwSwitch*

The results of varying the percentage of users whose eligible rules are moved from the virtual switch to the core switch are depicted in Figure 6 and Figure 7 for 3,000 users affected by the node failure. The percentage of users whose eligible flows are moved is varied in 5% steps between 0 and 50%. When the eligible flows of 50% of the affected users are moved to the core switch, the total number of newly installed OpenFlow rules on the virtual switch decreases by 10% from 30,000 to 27,000. The median of the total installation time for all rules drops from nearly 4s to 3.75s. However, the drop in installation time is less than the 10% reduction in flow rules. The reduction of the number of rules is deterministic, the resulting time reductions feature some variation, which is denoted by the Q_1 and Q_3 quartiles. The receiver of the moved flow rules, the core switch, increases the number of installed flow rules from 0 to 3,000. The total rule installation time on *HwSwitch* increases to more than 6s.

The resulting failure times for both switches, the virtual switch on Virtualization Host 2 and the Core Switch are

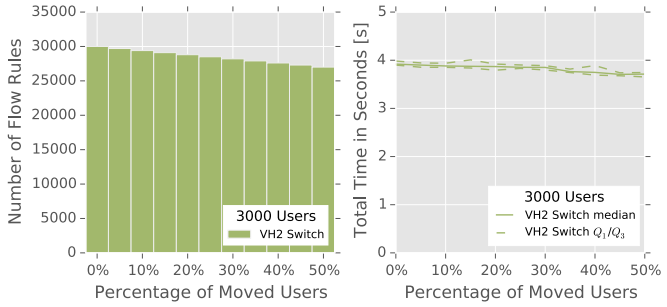

Fig. 6. Flow Rule Installation Performance of Open vSwitch

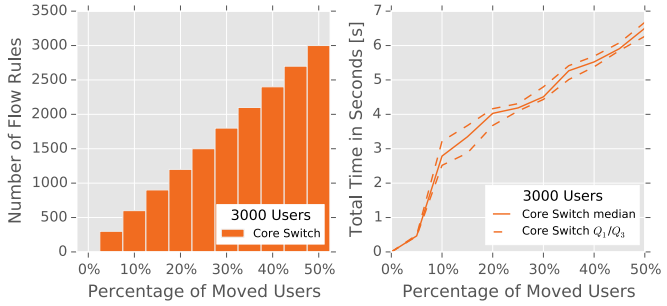

Fig. 7. Flow Rule Installation Performance of the Broadcom-based Hardware Switch

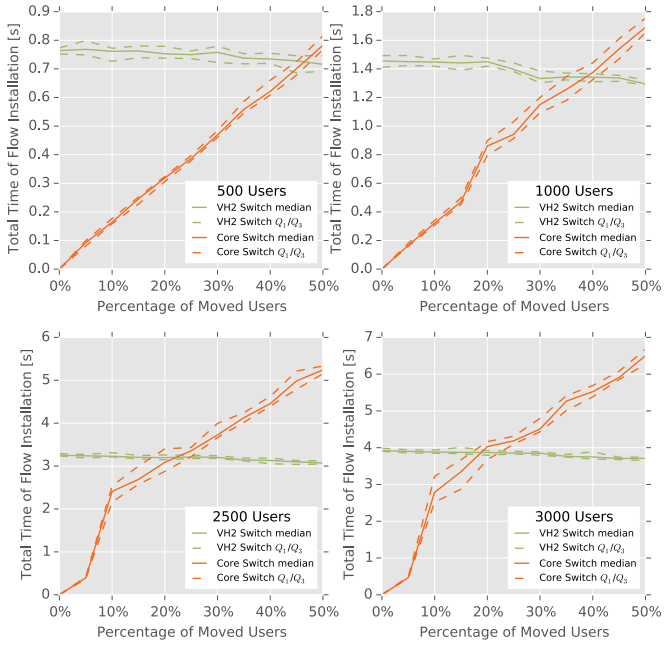

Fig. 8. Failure time for 500, 1,000, 2,500, and 3,000 users affected by the node failure

depicted in Figure 8 for 500, 1,000, 2,500, and 3,000 users affected by the node failure. The total failure time is the maximum of the flow update completion time of the virtual switch and the Core Switch. The solid line depicts the median of the completion times; the dotted lines depict the Q_1 and Q_3 quartiles. The rules that are moved to the Core Switch quickly become the bottleneck as the number of moved users increases. While for 500 and 1,000 affected users the decrease in the total failure time seems at least signification with the respect to the 25% and 75% percentiles, it becomes barely visible for the scenarios with 2,500 and 3,000 users. It can be concluded that the flow installation performance gap between the *HwSwitch* and an Open vSwitch 2.3.1 on the server hardware used in the evaluation testbed to too big for the rule movement to show and significant reduction of failure time. The performance gap was not expected to be smaller based on literature; the performance of Open vSwitch has been greatly improved in the latest releases of the software. Nevertheless, the measurement shows the potential of the presented service chaining concept.

The total failure time of a hardware and a software switch combination in relative terms depends only on the relative flow rule installation performance for the hardware switch P_{hw} and the software switch P_{sw}. For $P_{hw} \leq P_{sw}$, the total time reduction for rule update operations is determined by the percentage M of the moved flow operations and the relative performance of P_{hw} in terms of P_{sw} as described in (2).

$$\frac{P_{hw}}{P_{sw}}(M) = \frac{1}{\frac{1}{M} - 1} \tag{2}$$

The relation is depicted in Figure 9 as well as the values for the *HwSwitch* and the NoviFlow NoviSwitch 1132 [12]. The latter is an example for a commercial hardware OpenFlow switch that is advertised with an OpenFlow rule update performance of 3200 flow operations per second, which is very high compared to the performance of *HwSwitch*. The measurements conducted with the *HwSwitch* and the Open vSwitch 2.3.1 showed that a performance level of less than 10% of the core switch in comparison to the virtual switch is too small to improve the total failure time significantly. However, new devices promise an increase OpenFlow rule update performance. Furthermore, the presented system design introduces no costs other than a single additional OpenFlow rule per virtual switch.

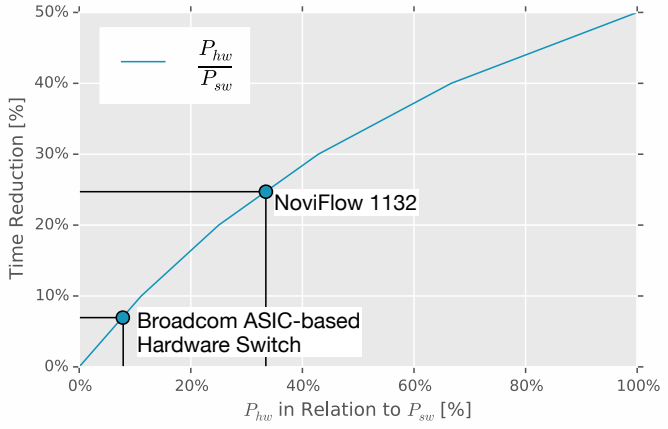

Fig. 9. Failure Time Reduction for when using Open vSwitch as Edge switch on the Evaluated Testbed with Different Core Switches based on (2)

IV. Related Work

One of the first works to optimize SDN-based service chaining is StEERING [6]. StEERING utilizes OpenFlow rules in a highly optimized manner on the virtualization hosts, where the service instances are running. SIMPLE [8] proposes to implement service chaining behavior by using tunnels between the network nodes where service instances are attached. While both papers introduce relevant optimization concepts, they do not investigate the behavior of the respective system under a peak rule update load. However, the approaches use a forwarding architecture that is similar to the concept presented in this paper. Therefore, the flow rule update hotspots are similar and thus, both approaches will likely benefit from the load shifting concept introduced in this paper. Besides SDN-based approaches, which do not require the cooperation of service instances, Network Service Headers (NSH) [13], based on a new network protocol have been proposed. However, these approaches require the virtualization hosts and the service instances to support the NSH protocol.

CacheFlow [14] introduces the idea of combining the capabilities of hardware and software switches to improve the performance of SDN networks. The approach supplements hardware switches with their limited TCAM memory with software switches to relieve them from a large number of small, low-bandwidth flows. The approach described in this paper uses a similar approach in the opposite direction: unused resources on hardware switches are used to supplement software switches during rule update peaks.

V. Conclusion

An OpenFlow-based dynamic service chaining system with OpenFlow rule installation offloading during rule update peaks was presented in this paper. The concept combines the scalability of the MPLS-like forwarding approach of SwitchReduce [3] with the ease of service instance deployment of [2]. The rule installation offloading mechanism lowers the load on the edges of the network, which host most of the forwarding logic, to help them coping with rule update peaks during node failures or other exceptional events. The approach relies on the exploitation of hitherto unused resources, by moving eligible OpenFlow rules from the edge of the network to a core switch. Thereby, the approach does not incur any performance and resource costs, while it helps to use available resource more efficiently. The presented approach has not been investigated in literature before. Still, other SDN-based service chaining approaches are likely to benefit from it as well.

Open vSwitch is used as virtual switch software on the virtualization hosts and a Broadcom ASIC-based OpenFlow hardware switch (*HwSwitch*) as core switch. An important finding is that the OpenFlow rule installation performance gap between the two switches is an order of magnitude bigger than described in literature before. The main reason for this is a huge increase in the OpenFlow rule installation performance from version 1.7.0, with about 400 flow modifications per second [11], to version 2.3.1, which was measured to perform around 10,000 flow modifications per second. Due to the large performance gap between the hardware and the software switch, the approach on smoothing rule update peaks during node failure does not significantly reduce the failure time in the testbed used in this paper. However, an analytical approach is presented that shows that newer, high-performance devices promise to offer a better performance compared to Open vSwitch and therefore significantly reduce the failure time during a virtualization host failure. The evaluation shows that the OpenFlow rule update performance of the core switch should be at least 20% of the virtual switch for significant improvements.

Potential next steps are the investigation of a high-performance core switch in the testbed and the investigation of bigger topologies. The rule movement in the failover case could be split to multiple core switches and other virtualization hosts that have available resources. Therefore, an approach for efficiently distributing the flow rules on multiple devices should be investigated.

Acknowledgments

This work has been funded in parts by the European Union (FP7/#317846, SmartenIT and FP7/#318398, eCOUSIN) and the German Research Foundation (DFG) as part of project C03 of the Collaborative Research Center (CRC) 1053 – MAKI.

References

[1] W. John, K. Pentikousis, G. Agapiou *et al.*, "Research Directions in Network Service Chaining," in *IEEE SDN for Future Networks and Services (SDN4FNS)*, 2013.

[2] J. Blendin, J. Rückert, N. Leymann *et al.*, "Position Paper: Software-Defined Network Service Chaining," in *European Workshop on Software Defined Networks (EWSDN)*, 2014.

[3] A. S. Iyer, V. Mann, and N. R. Samineni, "SwitchReduce: Reducing switch state and controller involvement in OpenFlow networks," in *IFIP Networking Conference*, 2013.

[4] M. Kuzniar, P. Peresini, and D. Kostic, "What You Need to Know About SDN Flow Tables," in *Passive and Active Measurements Conference (PAM)*, 2015.

[5] M. Casado, T. Koponen, S. Shenker, and A. Tootoonchian, "Fabric: a retrospective on evolving SDN," in *ACM SIGCOMM Workshop on Hot Topics in Software Defined Networking (HotSDN)*, 2012.

[6] Y. Zhang, N. Beheshti, L. Beliveau *et al.*, "StEERING: A Software-Defined Networking for Inline Service Chaining," in *IEEE International Conference on Network Protocols (ICNP)*, 2013.

[7] P. Emmerich, D. Raumer, F. Wohlfart, and G. Carle, "Assessing Soft- and Hardware Bottlenecks in PC-based Packet Forwarding Systems," in *International Conference on Networks (ICN)*, 2015.

[8] Z. A. Qazi, C.-C. Tu, L. Chiang *et al.*, "SIMPLE-fying Middlebox Policy Enforcement Using SDN," in *ACM SIGCOMM*, 2013.

[9] C. Rotsos, N. Sarrar, S. Uhlig *et al.*, "OFLOPS : An Open Framework for OpenFlow Switch Evaluation," in *Passive and Active Measurements Conference (PAM)*, 2012.

[10] R. N. Mysore, A. Pamboris, N. Farrington *et al.*, "PortLand : A Scalable Fault-Tolerant Layer 2 Data Center Network Fabric," in *ACM SIGCOMM*, 2009.

[11] D. Y. Huang, K. Yocum, and A. C. Snoeren, "High-fidelity Switch Models for Software-defined Network Emulation," in *ACM SIGCOMM Workshop on Hot Topics in Software Defined (HotSDN)*, 2013.

[12] NoviFlow, "NoviSwitch 1132 Data Sheet," http://noviflow.com/wp-content/uploads/2015/07/NoviSwitch-1132-Datasheet-V2.0.pdf (accessed July 27, 2015).

[13] P. Quinn and U. Elzur, "Network Service Header," IETF, https://tools.ietf.org/html/draft-ietf-sfc-nsh-01 (accessed July 27, 2015), 2015.

[14] N. Katta, O. Alipourfard, J. Rexford, and D. Walker, "Infinite Cacheflow in Software-Defined Networks," in *ACM SIGCOMM Workshop on Hot Topics in Software Defined (HotSDN)*, 2014.