TableVisor 2.0: Towards full-featured, scalable and hardware-independent Multi Table Processing

Stefan Geissler*, Steffen Gebert*, Stefan Herrnleben*, Thomas Zinner*, Robert Bauer‡, Michael Jarschel§
*University of Wuerzburg, Germany
Email: {stefan.geissler|steffen.gebert|stefan.herrnleben|zinner}@informatik.uni-wuerzburg.de
‡Karlsruhe Institute of Technology, Germany
Email: robert.bauer@kit.edu
§Nokia Bell Labs, Germany
Email: michael.jarschel@nokia-bell-labs.com

***Abstract*—Modern Software Defined Networking (SDN) applications rely on sophisticated packet processing. However, there is a mismatch between control plane requirements and data plane capabilities caused by increasing hardware heterogeneity. To overcome this challenge, we propose TableVisor, a proxy-layer for the OpenFlow control channel that enables the flexible and scalable abstraction of multiple physical devices into one emulated data plane switch that meets the requirements of the control plane application. TableVisor registers with the SDN controller as a single switch with use-case specific capabilities. It translates the instructions and rules from the control application towards the appropriate physical device where they are executed. In this paper, we present the updated architecture and functionality of TableVisor as well as first evaluation results based on testbed experiments.**

I. Introduction

The original idea of Software Defined Networking (SDN) is to control a network consisting of COTS hardware switches by programming their forwarding behavior via a standardized southbound interface such as OpenFlow [1]. However, this vision of a unified interface towards the data plane has not always been held up in practice. SDN-enabled switches are often shipped with different hardware capabilities and configurations, e.g., with a varying number of flow tables or support for certain data plane features [2], [3], [4], [5] for reasons of differentiation. Some switches, for example, do support hardware MPLS encapsulation or specific OpenFlow write actions while others do not. Some are limited by being equipped with a single TCAM chip that does not enable them to perform Multi-Table Processing (MTP). Unfortunately, this way the control plane developers currently have no other choice than to either restrict the compatibility of their control software to a set of well known devices or to provide solutions for most devices on the market. Both approaches essentially negate the original vision of SDN.

At the same time, many modern SDN applications rely on sophisticated packet processing. Therefore, missing hardware features and hardware heterogeneity is a growing problem for software-based networks. It can be characterized as a *general mismatch* between control plane requirements and data plane capabilities, e.g., an application needs flexible MTP with an arbitrary number of flow tables while a specific switch can only provide a fixed number.

Several approaches try to address this problem by introducing more flexibility into the data plane, e.g., programmable switches [6], OpenFlow Table Type Patterns [7], or high level programming languages for packet processing [8]. We refer to this as Programmable Data Plane (PDP) [9]. While PDP partially solves the mismatch, two fundamental problems remain: First, PDP based solutions are – by design – limited to the resources and capabilities of a single device. Control plane requirements going beyond what is provided by the device cannot be satisfied, even if two devices together could easily fulfill the request. Furthermore, real world SDN deployments are expected to be composed of heterogeneous devices [10]. SDN applications, however, are not normally designed to deal with this kind of heterogeneity. It should thus be possible to add new data plane features, e.g., by adding PDP-capable devices, and utilize them at application level without upgrading the entire infrastructure.

TableVisor is a novel approach for flexible and scalable pipeline processing that addresses the above problems and was originally introduced in [11]. TableVisor is based on the general idea, that multiple physical hardware switches can be pooled together to emulate a device with extended capabilities and capacity. It can be used, for example, to build scalable and feature-rich multi-table pipelines that are tailored to a specific use case, even if the switches in the infrastructure can only support a single flow table.

This paper extends our previous work on TableVisor to enable full-featured MTP and improve the overall usability of the approach, e.g., regarding configuration or wiring requirements. The main contributions of this paper can be summarized as follows:

- We extend TableVisor to include topology detection, meta-data support, pipeline processing and hardware table extension. We also provide TableVisor with a new *bypassing* feature to enable flexible go-to instructions without additional requirements to the wiring of physical switches.
- We include first evaluation results based on testbed experiments and show that the data plane overhead of Table-

Visor scales linear with the number of pipeline stages emulated by TableVisor (using single table hardware in the underlay). In addition we quantify the overhead on the control plane by evaluating flow mod processing times with and without TableVisor.
- We present a preliminary outline of an extended TableVisor architecture that can be flexibly integrated into an existing network infrastructure and discuss several advantages and disadvantages of our envisioned approach.

The remainder of this paper is structured as follows: In Section II, we discuss background and related work. Section III contains the TableVisor architecture and basic operation. Subsequently, in Section IV the proxy-layer functionality is presented. We then provide a discussion of TableVisor's advantages as well as challenges and possible extensions in Section V and give our concluding remarks in Section VI.

II. Background and Related Work

A. Multi Table Processing

Multi Table Processing (MTP) was introduced in OpenFlow 1.1 and provides a `goto-table` instruction which allows pipeline processing of packets inside the data path. As outlined by the Open Networking Foundation in [12], MTP is well suited to alleviate the flow table explosion problem common to various types of networking applications. For example in cases where matches and actions have to be considered independent of each other (e.g. traditional MAC address learning) or where some kind of two stage processing, like pre-classification, is required, MTP is a suitable solution. Many novel applications for software-defined networks also rely on MTP for similar reasons, e.g., source address validation [13], d-dimensional packet classification [14] or wildcard rule caching [15].

Hardware support for MTP, however, is difficult to achieve due to energy and cost constraints of the TCAM technology [16]. As a result, existing SDN-switches only support a small number of hardware flow tables and often do not allow to jump between them via the `goto-table` instruction. The NEC PF5240 used in this work has multiple hardware tables, but, with exception of specialized tables for MPLS support, does not allow jumping between these tables.

B. Hardware Abstraction

Several concepts deal with device heterogeneity by introducing a Hardware Abstraction Layer (HAL). Table Type Patterns [7] for OpenFlow are a prominent and recent example. ALIEN HAL [17] focuses on realizing OpenFlow capabilities on legacy network elements by introducing a reusable, two-layer abstraction platform with a unified control endpoint to common-of-the-shelf SDN controllers. FlowAdapter [18] is middle layer between the hardware and software data plane that provides flexible M-stage MTP support by properly mapping MTP rules onto existing hardware capabilities. Similar ideas of abstraction can be found in other architectural proposals as well [19], [20]. HAL-based solutions, however, are normally restricted to the hardware capabilities of the underlying network device. TableVisor can overcome this limitation by aggregating the capabilities of multiple and possibly heterogeneous hardware devices.

C. Programmable Data Plane

Programmable Data Planes (PDPs) are another important trend for SDN [21], [9]. The core idea here is to provide programmable network devices with freely definable packet processing pipelines, i.e., moving away from the fixed match-action paradigm currently found in hardware switches. The FlexPipe architecture of Intel's FM6000/FM7000 series [6], for example, allows programmable parsing of incoming traffic based on a TCAM/SRAM/MUX structure. [2] proposes a model for reconfigurable match tables and enables dynamic (in-field) reconfiguration of the data plane. Protocol Oblivious Forwarding [22] introduces a generic flow instruction set to make the data plane protocol-oblivious. As mentioned earlier in the introduction, PDPs together with high level programming languages for packet processing like P4 [8] or CNC [23] can help solving the mismatch between control plane requirements and data plane capabilities. They are, however, also limited by the capacity and functionality of the underlying hardware (similar to HAL-based solutions) and might even increase the problem of device heterogeneity in real world SDN deployments.

D. Proxy Layer Approaches

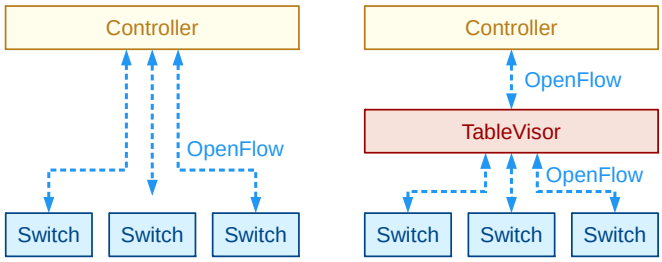
TableVisor is implemented as a proxy layer between the SDN controller and SDN hardware. OpenFlow protocol messages sent by either of those two entities are intercepted and modified by the TableVisor layer to realize the abstraction. The method of transparently processing OpenFlow messages is heavily used in related work, e.g., to perform network virtualization [24], [25], [26], to inter-operate with non-SDN legacy network equipment [27] or to transparently deal with flow table limitations [28].

III. TableVisor

This section covers the design concept behind TableVisor, presents the system architecture and concludes with a summary of functionality realized by the proxy layer.

A. Architecture

Figure 1 shows the control channel in a common OpenFlow setup compared to the intercepted communication through TableVisor as a proxy layer. In an OpenFlow setup without a proxy layer, messages between a controller and one or multiple switches are exchanged through a direct OpenFlow channel, as depicted in Figure 1a. To emulate a multi-table switch by the use of multiple hardware switches, TableVisor requires to intervene this communication channel in order to modify all incoming and outgoing OpenFlow message on the controller's Southbound API. Hence, TableVisor acts as an emulated switch, seen by the controller, and as a controller endpoint for the used switching hardware. By this, the received OpenFlow messages on the control channel can be adapted and



(a) Standard OpenFlow Control Channel.

(b) OpenFlow Control Channel using TableVisor.

Fig. 1: Control Channel Structure.

forwarded to the other party. The controller does not know anything about the composed multi-table pipeline.

The architecture of the TableVisor software is depicted in Figure 2. The implementation is based on LINC switch[1] and reuses the application framework, the provided OpenFlow libraries and the controller communication channel. The TableVisor proxy layer itself is comprised of three main logical layers, the switch endpoint, the message processing layer and the controller endpoint. This layer structure simplifies development, connection handling as well as debugging of errors. Finally, TableVisor can be configured via a configuration file that, e.g., allows to map a datapath ID to a table number and thus enables the configuration of the order in which switches are used in the switching pipeline.

The **Switch Endpoint** is responsible for establishing a connection to the controller. This layer allows TableVisor to be completely transparent towards the controller and act like a single OpenFlow switch regardless of the number of actual hardware switches connected to TableVisor itself.

Following, the **Message Processing** handles all OpenFlow messages coming from either the controller or one of the hardware switches. Thereby, the contents of the messages are adapted in such a way, that regular OpenFlow messages send by the controller can be forwarded distributed to the underlying switching hardware. As the rewriting process represents the fundamental functionality of the shim layer, it is described in Section III-B in detail.

Finally, the **Controller Endpoint** is the layer of TableVisor switches connect to. The IP address of the actual controller therefore needs to be replaced with the address of the host running TableVisor. The switches then see the controller endpoint as a regular OpenFlow controller. The switches themselves are in this case not aware of the multi-table architecture created by TableVisor, as every switch represents one table in the architecture. The controller then instructs TableVisor to use multiple tables via the `goto-table` instruction which is processed by the message processing layer and passed to the respective switch as an output action. In addition to this, the controller endpoint is able to automatically detect the underlying switch topology using LLDP.

[1]https://github.com/FlowForwarding/LINC-Switch/

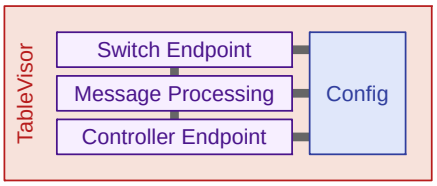


Fig. 2: Logical Layers of TableVisor.

B. Message Processing

In order to realize the transparent proxy layer TableVisor introduces between the OpenFlow controller and involved switching hardware, OpenFlow messages need to be processed. This is, as already mentioned, performed by the message processing layer of TableVisor. To achieve this, messages between the controller and switching hardware are intercepted by the TableVisor proxy layer which processes these messages and then forwards or distributes them to the switches or the controller respectively.

The controller sends messages via the southbound API to the data plane, assuming that a single hardware switch will process them. TableVisor needs to inspect these messages and has to decide about rewriting and forwarding or direct replying to them. The connection establishment and maintenance messages are examples of messages, which will be handled directly by TableVisor without rewriting and forwarding them to the switches. If messages require to be forwarded to the switches, TableVisor needs to perform the following major tasks:

1) If the message is related to a specific table, TableVisor has to determine the responsible switch based on the table ID targeted by the controller.
2) If the message includes a target table ID, TableVisor has to rewrite the table ID to the ID the switch uses to address this table internally. Especially many hardware switches use table IDs different from zero for their tables, stored in TCAM memory. Without this functionality, TableVisor would be limited to table 0 of all switches.
3) If the message includes a `goto-table` instruction, the instruction has to be replaced by an `output` action to pass the packet via Ethernet connection to another switch representing the target table.

Messages from switches to the controller need to be handled in a similar way. The switches do not know anything about the multi-table implementation. They send their messages to the controller endpoint of TableVisor. Messages for connection establishment and maintenance are handled directly by TableVisor, while other messages, like packet-in, have to be rewritten and forwarded. For the messages that have to be forwarded, TableVisor performs the following major tasks:

1) Replace the table ID sent by the switch with the multi-table related ID, based on the socket that identifies the switch in TableVisor.

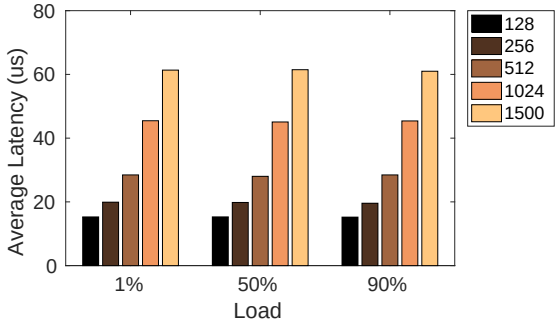


Fig. 3: Data plane delay of an emulated multi-table switch for different frame lengths and load levels.

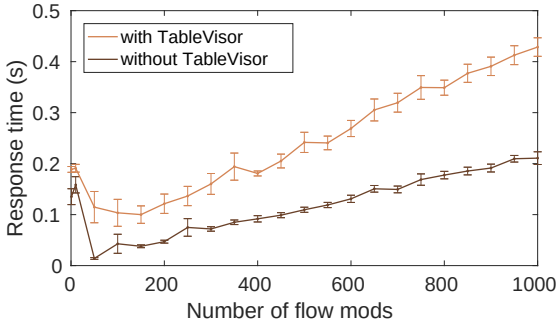


Fig. 4: Setup times for 1 to 1000 flow mods with and without TableVisor.

2) If the message includes an `output` action to pass packets to another switch representing a subsequent table, this has to be replaced with a `goto-table` instruction, as this is what the controller expects.

C. Impact on Control and Data Plane

Obviously, the introduction of an additional layer to the OpenFlow control channel, as depicted in Figure 1b, as well as the concatenation of switches affects the control and data plane respectively. In order to analyze the influence the TableVisor software itself as well as the multi-switch pipeline have on SDN performance, we conduct a measurement study regarding two important performance indicators for SDN enabled networks.

First, we evaluate the influence of the multi-switch pipeline on the data plane. To do so, we measure the end-to-end delay of the emulated pipeline using a Spirent C1 Testcenter as the traffic generator. As expected, the end-to-end delay of the pipeline is simply the sum of delays introduced by the individual hardware appliances.

Figure 3 shows the delay in a pipeline comprised of four identical HP 2920-24G switches regarding different load levels and frame lengths. It can be seen that the delay is independent of the load and is constant for all three load levels. The frame length, on the other hand, has significant influence on the end-to-end delay, which indicates, that a large portion of the time is used to write the information to the transmission media while the processing of packets is independent of the packet size.

Second, in order to evaluate the impact of TableVisor on the control plane, we measure flow setup times using one of the HP2920-24G switches. The measurements have been conducted in a dedicated testbed comprised of a single physical host running the Ryu SDN controller and TableVisor and the aforementioned hardware switch. Figure 4 shows the measurement results which have been obtained by sending between 1 and 1000 randomly generated flow mod messages, shown along the x axis, followed by a barrier request. Thereby, the scenario in which the controller is directly connected to the switch is shown in orange while the brown line depicts the measurement results including the TableVisor proxy layer. The time between the barrier request and the corresponding barrier reply, shown along the y axis, is considered to be the flow mod response time. The whiskers represent the 95% confidence interval of 30 measurement repetitions. This measurement methodology has been evaluated in detail in [29].

It can be seen that the response time grows linearly in both cases, while the absolute values in the TableVisor scenario are roughly twice as high. This additional delay results from the message processing performed by TableVisor in order to map incoming OpenFlow messages transparently onto the underlying switching hardware.

IV. Proxy Layer Functions

The following section presents the functionality realized by the current implementation of the TableVisor proxy layer solution and points out possible use cases.

A. Pipeline Processing

The first functionality provided by TableVisor we focus on in this work is *Pipeline Processing*. Thereby, the involved hardware switches are used as a queue all packets have to pass through. Packets are then iteratively processed by the switches. This allows the combination of different hardware appliances in order to realize complex scenarios that are not possible using a single hardware switch due to table sizes or supported functionality. Figure 5 shows an exemplary use case for pipeline processing using multiple, different hardware switches.

In this scenario, we realized an MPLS label edge router using four separate hardware switches[2]. The first switch represents an access control list and matches on destination MAC address as well as the MPLS Ethernet. As this switch does not support MPLS actions, a second switch that is able to pop the MPLS label is required. This is done by specifying the `output` action which sends the packet out to the second switch in the pipeline. The flow rule for this behavior on the switch was intended as a `goto-table` instruction by the controller and was rewritten to an `output` action by TableVisor. In the second switch the MPLS label is removed and the packet is passed to the next switch in the pipeline

[2] 1 × NEC PF5240, 3 × HP 2920-24G

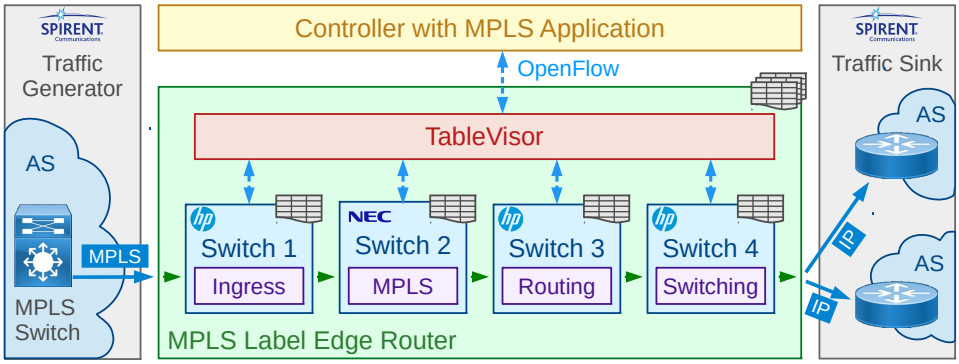


Fig. 5: MPLS Label Edge Routing.

which performs the routing actions. Therefore, it matches on the destination IP prefix of the packet and sets the destination mac address correspondingly. Finally, the last switch in the pipeline performs the switching step in which the packets are forwarded to the respective destination autonomous systems.

This example shows one possible use case that can be realized using the pipeline processing feature of TableVisor. As of writing this paper, no switch that supports pipeline processing in hardware is known to the authors.

While in this scenario all packets need to be processed by all of the involved switches, there are other use cases for multi-table processing in which, depending on a multitude of factors, packets may not require to be processed in each single table of the emulated multi-table switch. In these cases, tables can be skipped by either using the previously described meta-data feature provided by TableVisor to carry over meta-data between tables, or an additional physical link between switches can be used to jump between tables without having to pass through unused intermediary tables. This *bypassing* feature further increases flexibility and can reduce the additional data-plane delay introduced by the multi-switch pipeline. A detailed analysis of the additional delay is performed in Section III-C.

The functionality realizable using pipeline processing depend solely on the supported features of the underlying hardware. If support for a certain protocol or functionality is required, it can simply be realized by adding a hardware switch that implements the required feature.

B. Metadata

The OpenFlow protocol specifies that packets may be equipped with a mask-able register that allows to carry so called *Metadata* from one table to another. This enables, depending on the support from switches, more complex actions as packets can be marked by an initial table and then processed accordingly in a later table. As the meta-data is meant to be used within a single integrated hardware switch, the information is generally lost with the transmission of the packet. In order to allow these more complex multi-table scenarios in a multi table emulation scenario using TableVisor, we implemented the ability to transmit meta-data between the involved hardware switches.

In order to achieve this, TableVisor exploits the source and destination MAC address fields of the Ethernet header. As these fields are not required to allow proper transmission of packets between switches aggregated using TableVisor we decided to store the meta data in either the source or destination MAC header field. Each of these fields allows for 48 bits of meta-data to be transferred between switches. Moreover, the number of switches that support writing and reading the MAC address fields is far larger than the number of switches that support the internal usage of meta-data. This results in the possibility to use switches that do not support native OpenFlow meta-data to realize more complex scenarios if used in combination with the TableVisor proxy layer.

This functionality is realized by the *Message Processing* layer described in Section III-A by translating an OpenFlow `Write-Metadata` action into a `Set-Field-Action` which then writes the meta-data into the packet header instead of the switch internal meta-data register. TableVisor ships with three meta-data providers that use different header fields to store the meta-data. Supported fields are source and destination MAC and VLAN ID.

C. Table Extension

The second use case provided by the current implementation is the extension of hardware tables, as these are often very limited in size. Since the required TCAM (Ternary Content-Addressable Memory) storage is very complicated and expensive to build, hardware manufacturers limit the provided size in order to provide appliances at competitive prices. The special feature of TCAM storage is that it is not only able to search for words consisting entirely of 0s and 1s, but instead can match on a third state X or *don't care*, which allows for wildcard matching. This enables matching on specific bits of a packet header, like used in IP address prefix matching.

In the previous pipeline processing scenario, TableVisor maps a single switch via its data path ID to a specific table in the emulated multi-table switch. This behavior needs to be changed in order to combine the TCAM storage of multiple switches into one larger hardware accelerated flow table since now a table, as seen by the controller, can span multiple underlying hardware switches. To achieve this, two crucial

requirements have to be met. First, TableVisor has to ensure that packets are only matched once, even if the virtual table has multiple matching entries distributed over more than one switch. This is done via the previously described meta-data functionality that allows TableVisor to maintain meta-data between hardware switches. Thereby, packets are marked by the first rule that matches. Following switches belonging to the same virtual table are configured to ignore marked packets. Hence, every packet gets only matched exactly once per virtual table.

Second, a mechanism is required to distribute flow rules evenly among all switches involved in the table size aggregation. This distribution is performed through a combination of priority boundaries and hashing. Due to the fact that TableVisor configures switches involved in a virtual multi-switch table to ignore packets that have already been matched by an earlier switch, we had to ensure that rules with high priority are installed in the first switch of the composite. On the other hand, flow rules with low priority need to be installed in the last switch of the aggregation. To realize this, a fixed priority or priority interval is specified in the TableVisor configuration for each of the used hardware switches. It is possible to share one priority between multiple switches to distribute flows. When installing such flows, a hash value, based on the match fields, is calculated and the flow is forwarded to the switch associated with this hash value. This results in an even distribution of flow rules of this priority among all involved switches while at the same time allowing the controller to replace flow rules since the hash value is definite and based on the match fields. Finally, all flow rules with higher or lower priority are simply installed in the first or last switch respectively. This behavior results in a fully transparent and OpenFlow conform hardware accelerated flow table spanning multiple hardware switches.

V. Discussion

TableVisor can be seen as a first step to conceptually address the mismatch between control plane requirements and data plane capabilities. This section briefly discusses general advantages and challenges of the TableVisor approach and provides several pointers for future work.

A. Advantages

One of the main advantages of TableVisor is increased flexibility. More specifically: the possibility to realize powerful proxy layer services (like flexible pipeline processing) even if the underlying hardware is limited with respect to functionality or capacity. The current TableVisor prototype, for example, enables full-featured MTP while the underlying hardware only has a single flow table and does not support the `goto-table` instruction. Because TableVisor is implemented as a software layer and provides a fully transparent view towards control applications, the approach can be easily integrated into existing software-defined networks (no changes are required to existing hardware and software). The aspect of increased flexibility is especially important since current state-of-the-art SDN switching technology often lacks support for complex pipeline processing features or optional OpenFlow actions. And even if new technology is available at some point in the future – e.g., COTS switches with a programmable data plane – it can be reasonably assumed that control plane requirements remain that cannot be fulfilled by a single network device.

TableVisor can overcome this conceptual limitation by introducing a proxy layer that emulates a feature-rich multi-table pipeline using several hardware switches. This approach is most applicable to scenarios where (a) specific data plane capabilities are otherwise unavailable, (b) already existing hardware has to be utilized, e.g., because of economic efficiency and (c), it is viable to accept the tradeoff between improved flexibility and reduced performance (slightly increased control and data plane latency).

B. Challenges and Future Work

There are several open challenges connected to the TableVisor approach. First of all, the true potential of TableVisor ultimately depends on the proxy layer services that can be realized. Section IV already gives two important examples (pipeline processing and table extension), but there are other use cases that could benefit from an additional proxy layer:

Feature Pooling: As mentioned earlier, state-of-the-art SDN hardware cannot normally provide all the features required from the control plane. The OpenFlow specification, for example, lists various features as optional. Adding new features normally requires installing new hardware. Upgrading the entire infrastructure, however, is a time consuming, error prone and expensive procedure, especially if certain features are only required for a small fraction of flows. The TableVisor proxy-layer allows incremental hardware acquisition by pooling several switches together. This way, a certain feature can be used by the control plane as long as it is supported by one of the switches in the pool.

Local Repair: The control delay between switches and controller is a well-known problem for repair and recovery operations. Reactive repair that involves the controller, for example, is often too slow to provide carrier grade requirements (less than 50ms). The TableVisor proxy layer can be used to handle local repair operations without including the SDN controller. If the proxy layer is executed in near vicinity of the switching hardware (e.g., based on Network Functions Virtualization), a low delay control loop could be realized.

Transparent Optimizations: Because the TableVisor proxy layer works independently of the SDN controller, it can perform transparent optimizations. This includes, for example, the rate at which new flows can be installed into a hardware switch or parallel execution of different pipeline stages.

For future work, we plan to implement and evaluate additional proxy layer services. Note that this may require other conceptual considerations besides the TableVisor proxy layer, e.g., mechanisms similar to what is described in [28].

Dynamic provisioning is another important challenge, i.e., online configuration of TableVisor based on specific control-plane requirements. To reduce complexity, the current Table-

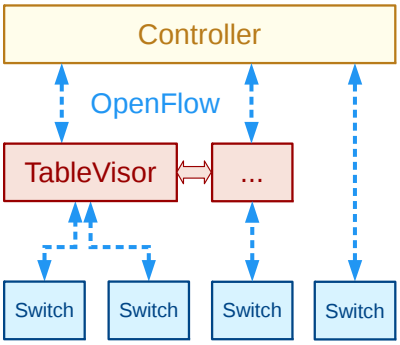


Fig. 6: Vision of an advanced control channel structure with logical TableVisor instances for increased flexibility.

Visor architecture enforces a strict separation of concerns in two dimensions: (a) between SDN controller and TableVisor and (b) between multiple instances of TableVisor within the same network. While the former is crucial for control plane transparency, the ladder can be softened to enable dynamic provisioning and further improve flexibility.

In this regard, we plan to investigate alternative control channel structures. Fig. 6 shows an example where multiple logical TableVisor instances are organized as a middle-ware layer attached to the SDN controller. This way, the organization of the underlying switches of the proxy layer can be configured and changed on-demand which allows for a much more flexible deployment.

In addition, the proxy layer between controller and switches introduces an additional layer of complexity. While this complexity is hidden from the controller, it can still affect the overall operation of the network, e.g., regarding robustness and security. Future work therefore includes a thorough investigation of these aspects (switch and link failures, proxy layer failures, security implications).

VI. Conclusion

In this paper, we present the second major version of TableVisor, a transparent proxy layer software tool that allows pipeline processing and enables the extension of hardware flow table sizes using multiple hardware switches. Today, most OpenFlow-enabled switches are either limited to a single hardware table or, if multiple tables are available, do not allow jumping between these tables. Hence, the control plane is constrained by the capabilities provided by the underlying switching hardware. To alleviate this restriction, the most recent version of the TableVisor software provides two new major features.

First, the pipeline processing functionality of TableVisor, as presented in Section IV-A, emulates a single multi-table switch using multiple hardware switches. Multi-table processing is an essential feature for many OpenFlow use cases. MTP can counter the flow table explosion problem, since in a multi-table scenario the number of flow entries grows linearly instead of quadratically.

The second major feature provided by TableVisor is the emulation of large hardware tables by combining the TCAM storage of multiple switches, as described in Section IV-C. This feature increases the often limited memory storage capacity of OpenFlow switches.

Beyond the introduction of these new features, we evaluate the performance impact of the TableVisor shim-layer regarding data plane and control plane. Measurements show that the additional data plane delay introduced by pipeline processing using multiple switches scales linearly with the number of used hardware switches. The available bandwidth is not affected by the concatenation of hardware switches and latency is limited by the slowest device of the composite.

Regarding the influence on the control plane, we measured flow setup times with and without the TableVisor software in between the controller and switching hardware. The results show that the introduction of TableVisor into the control channel increases the response time of the control plane by roughly factor 2. This, however, is not a significant problem in real world scenarios, since in most use cases, flow rules are installed proactively and the control plane in general has only weak constraints regarding the response time of switches. Furthermore, we are confident that the control plane performance of TableVisor can be further optimized in future versions.

The current version of TableVisor is thus able to alleviate some of the problems current OpenFlow switching hardware has. By emulating multi table hardware through the combination of, potentially heterogenous, single table switches, TableVisor is able to partially solve the mismatch between control plane requirements and underlying data plane capabilities. In addition, the combination of multiple switches to emulate a single, large hardware flow table solves the problem of limited TCAM storage in current switching hardware. Although the impact TableVisor has on control and data plane have to be considered, our measurements have shown, that the trade-off between more powerful switching hardware through TableVisor emulation and additional data plane delay or control plane performance is justifiable.

Acknowledgment

This work has been performed in the framework of the CELTIC EUREKA project SENDATE-PLANETS, and it is partly funded by the German BMBF. The authors alone are responsible for the content of the paper.

References

[1] The Open Networking Foundation, “OpenFlow Switch Specification (Version 1.5.0),” Dec. 2014.

[2] P. Bosshart, G. Gibb, H.-S. Kim, G. Varghese, N. McKeown, M. Izzard, F. Mujica, and M. Horowitz, “Forwarding metamorphosis: Fast programmable match-action processing in hardware for sdn,” in *Proceedings of the ACM SIGCOMM 2013 Conference on SIGCOMM*, ser. SIGCOMM ’13. New York, NY, USA: ACM, 2013, pp. 99–110. [Online]. Available: http://doi.acm.org/10.1145/2486001.2486011

[3] A. Lazaris, D. Tahara, X. Huang, E. Li, A. Voellmy, Y. R. Yang, and M. Yu, "Tango: Simplifying sdn control with automatic switch property inference, abstraction, and optimization," in *Proceedings of the 10th ACM International on Conference on Emerging Networking Experiments and Technologies*, ser. CoNEXT '14. New York, NY, USA: ACM, 2014, pp. 199–212. [Online]. Available: http://doi.acm.org/10.1145/2674005.2675011

[4] C. J. Casey, A. Sutton, and A. Sprintson, "tinynbi: Distilling an api from essential openflow abstractions," in *Proceedings of the Third Workshop on Hot Topics in Software Defined Networking*, ser. HotSDN '14. New York, NY, USA: ACM, 2014, pp. 37–42. [Online]. Available: http://doi.acm.org/10.1145/2620728.2620757

[5] M. Kuźniar, P. Perešíni, and D. Kostić, *What You Need to Know About SDN Flow Tables*. Cham: Springer International Publishing, 2015, pp. 347–359. [Online]. Available: http://dx.doi.org/10.1007/978-3-319-15509-8_26

[6] R. Ozdag, "Intel Ethernet Switch FM6000 Series - Software Defined Networking," 2013.

[7] The Open Networking Foundation, "OpenFlow Table Type Patterns (Version 1.0)," Aug. 2014.

[8] P. Bosshart, D. Daly, G. Gibb, M. Izzard, N. McKeown, J. Rexford, C. Schlesinger, D. Talayco, A. Vahdat, G. Varghese, and D. Walker, "P4: Programming protocol-independent packet processors," *SIGCOMM Comput. Commun. Rev.*, vol. 44, no. 3, pp. 87–95, Jul. 2014. [Online]. Available: http://doi.acm.org/10.1145/2656877.2656890

[9] D. Perino, M. Gallo, R. Laufer, Z. B. Houidi, and F. Pianese, "A programmable data plane for heterogeneous nfv platforms," in *2016 IEEE Conference on Computer Communications Workshops (INFOCOM WKSHPS)*, April 2016, pp. 77–82.

[10] Z. Bozakov and A. Rizk, "Taming sdn controllers in heterogeneous hardware environments," in *2013 Second European Workshop on Software Defined Networks*, Oct 2013, pp. 50–55.

[11] S. Gebert, M. Jarschel, S. Herrnleben, T. Zinner, and P. Tran-Gia, "Table visor: An emulation layer for multi-table open flow switches," in *2015 Fourth European Workshop on Software Defined Networks*, Sept 2015, pp. 117–118.

[12] The Open Networking Foundation, "The Benefits of Multiple Flow Tables and TTPs," Feb. 2015.

[13] B. Liu, J. Bi, and Y. Zhou, "Source address validation in software defined networks," in *Proceedings of the 2016 Conference on ACM SIGCOMM 2016 Conference*, ser. SIGCOMM '16. New York, NY, USA: ACM, 2016, pp. 595–596. [Online]. Available: http://doi.acm.org/10.1145/2934872.2960425

[14] C. R. Meiners, A. X. Liu, E. Torng, and J. Patel, "Split: Optimizing space, power, and throughput for tcam-based classification," in *Proceedings of the 2011 ACM/IEEE Seventh Symposium on Architectures for Networking and Communications Systems*, ser. ANCS '11. Washington, DC, USA: IEEE Computer Society, 2011, pp. 200–210. [Online]. Available: http://dx.doi.org/10.1109/ANCS.2011.36

[15] B. Yan, Y. Xu, H. Xing, K. Xi, and H. J. Chao, "Cab: A reactive wildcard rule caching system for software-defined networks," in *Proceedings of the Third Workshop on Hot Topics in Software Defined Networking*, ser. HotSDN '14. New York, NY, USA: ACM, 2014, pp. 163–168. [Online]. Available: http://doi.acm.org/10.1145/2620728.2620732

[16] K. Kannan and S. Banerjee, "Compact tcam: Flow entry compaction in tcam for power aware sdn," in *Distributed Computing and Networking*, ser. Lecture Notes in Computer Science, D. Frey, M. Raynal, S. Sarkar, R. Shyamasundar, and P. Sinha, Eds. Springer Berlin Heidelberg, 2013, vol. 7730, pp. 439–444. [Online]. Available: http://dx.doi.org/10.1007/978-3-642-35668-1_32

[17] D. Parniewicz, R. Doriguzzi Corin, L. Ogrodowczyk, M. Rashidi Fard, J. Matias, M. Gerola, V. Fuentes, U. Toseef, A. Zaalouk, B. Belter, E. Jacob, and K. Pentikousis, "Design and implementation of an openflow hardware abstraction layer," in *Proceedings of the 2014 ACM SIGCOMM Workshop on Distributed Cloud Computing*, ser. DCC '14. New York, NY, USA: ACM, 2014, pp. 71–76. [Online]. Available: http://doi.acm.org/10.1145/2627566.2627577

[18] H. Pan, H. Guan, J. Liu, W. Ding, C. Lin, and G. Xie, "The flowadapter: Enable flexible multi-table processing on legacy hardware," in *Proceedings of the Second ACM SIGCOMM Workshop on Hot Topics in Software Defined Networking*, ser. HotSDN '13. New York, NY, USA: ACM, 2013, pp. 85–90. [Online]. Available: http://doi.acm.org/10.1145/2491185.2491209

[19] B. Sonkoly, R. Szabo, D. Jocha, J. Czentye, M. Kind, and F. J. Westphal, "Unifying cloud and carrier network resources: An architectural view," in *2015 IEEE Global Communications Conference (GLOBECOM)*, Dec 2015, pp. 1–7.

[20] E. Haleplidis, K. Pentikousis, S. Denazis, J. H. Salim, D. Meyer, and O. Koufopavlou, "Software-Defined Networking (SDN): Layers and Architecture Terminology," RFC 7426 (Informational), Internet Engineering Task Force, Jan. 2015. [Online]. Available: http://www.ietf.org/rfc/rfc7426.txt

[21] M. Shahbaz and N. Feamster, "The case for an intermediate representation for programmable data planes," in *Proceedings of the 1st ACM SIGCOMM Symposium on Software Defined Networking Research*, ser. SOSR '15. New York, NY, USA: ACM, 2015, pp. 3:1–3:6. [Online]. Available: http://doi.acm.org/10.1145/2774993.2775000

[22] H. Song, "Protocol-oblivious forwarding: Unleash the power of sdn through a future-proof forwarding plane," in *Proceedings of the Second ACM SIGCOMM Workshop on Hot Topics in Software Defined Networking*, ser. HotSDN '13. New York, NY, USA: ACM, 2013, pp. 127–132. [Online]. Available: http://doi.acm.org/10.1145/2491185.2491190

[23] C. Schlesinger, M. Greenberg, and D. Walker, "Concurrent netcore: From policies to pipelines," *SIGPLAN Not.*, vol. 49, no. 9, pp. 11–24, Aug. 2014. [Online]. Available: http://doi.acm.org/10.1145/2692915.2628157

[24] R. Sherwood, G. Gibb, K. kiong Yap, M. Casado, N. Mckeown, and G. Parulkar, "Flowvisor: A network virtualization layer," Tech. Rep., 2009.

[25] R. D. Corin, M. Gerola, R. Riggio, F. D. Pellegrini, and E. Salvadori, "Vertigo: Network virtualization and beyond," in *2012 European Workshop on Software Defined Networking*, Oct 2012, pp. 24–29.

[26] A. Al-Shabibi, M. De Leenheer, M. Gerola, A. Koshibe, G. Parulkar, E. Salvadori, and B. Snow, "Openvirtex: Make your virtual sdns programmable," in *Proceedings of the Third Workshop on Hot Topics in Software Defined Networking*, ser. HotSDN '14. New York, NY, USA: ACM, 2014, pp. 25–30. [Online]. Available: http://doi.acm.org/10.1145/2620728.2620741

[27] I. Alawe, B. Cousin, O. Thorey, and R. Legouable, "Integration of legacy non-sdn optical roadms in a software defined network," in *2016 IEEE International Conference on Cloud Engineering Workshop (IC2EW)*, April 2016, pp. 60–64.

[28] R. Bauer and M. Zitterbart, "Port based capacity extensions (pbces): Improving sdns flow table scalability," in *28th International Teletraffic Congress (ITC 28)*, Wuerzburg, Germany, 2016.

[29] A. Nguyen-Ngoc, S. Lange, S. Gebert, T. Zinner, P. Tran-Gia, and M. Jarschel, "Performance evaluation mechanisms for flowmod message processing in openflow switches," in *Communications and Electronics (ICCE), 2016 IEEE Sixth International Conference on*. IEEE, 2016, pp. 40–45.