Towards an Access Control Scheme for Accessing Flows in SDN

Felix Klaedtke, Ghassan O. Karame, Roberto Bifulco, and Heng Cui
NEC Laboratories Europe, Heidelberg, Germany
Email: firstname.lastname@neclab.eu

***Abstract*—Sharing network resources with user groups, divisions, or even other companies in software defined networking promises better network utilization. Resource sharing is effectively realized by empowering these tenants at the control plane with permissions for administrating network components. However, since the network resources at the data plane are shared and different tenants can have competing objectives, mechanisms are needed to protect the network resources from unauthorized access. In this paper, we propose mechanisms that focus on protecting the network flows, which are determined by the entries installed in the flow tables of the shared switches. To this end, we present an access control scheme, based on the OpenFlow model, for accessing the switches' flow tables and their entries. Our scheme accounts for various security requirements in multi-tenant networks, including requirements on sharing flow table entries for handling network flows, and the resolution of conflicts originating from the reconfiguration of network components.**

Keywords*—*Software defined networking, OpenFlow, Network flows, Access control, Reference monitor

I. Introduction

Managing and controlling network traffic is undergoing a major change. This change manifests itself in *software defined networking* (SDN) [1]. Google has, for example, successfully deployed the SDN technology in their wide-area network connecting their data centers [2]; further deployment of SDN technology is envisioned in the near future within fixed and mobile network operators [3].

At the core of SDN is the separation between the network's data plane and its control plane. At the data plane, network packets are physically shipped from one network component to the next until they reach their destination. The control plane comprises a logically centralized software entity—the controller—running on commodity hardware. Today's controllers like NOX [4] and Beacon [5] provide an abstract view on the network and an interface based on the programming model defined by the OpenFlow standard [6]. OpenFlow provides the means to program the switches, which the controller uses to interact with the data plane.

Network applications, e.g., for managing and analyzing the network, run on top of the controller. However, existing controllers do not expose a common interface for applications and applications need therefore to be redeveloped for different controllers. To overcome this limitation, organizations—such as the Open Networking Foundation (ONF)—started defining north-bound APIs, i.e., interfaces provided by controllers for interacting with applications [7]. The reliance on (de facto) standardized north-bound APIs is expected to shape the large-scale deployment of SDN, where multiple (third-party) applications interface with the same controller, providing better resource utilization and management.

A widely used north-bound API and the resulting deployment of various third-party applications pose new security threats in SDN. Indeed, "malicious" applications can leverage such an API and infiltrate the network. These threats become even more evident when the network owner "leases" network slices, which are administrated by the leasing tenants, which in turn might install their own, possibly third-party, applications on top of the network owner's controller. Since the leasing tenants might have competing objectives, access to the shared network resources must be appropriately handled and secured.

Note that by using network virtualization [8], [9] the network owner can provide an isolated view on the network to each leasing tenant. However, the different tenants still share the data plane's network resources, which they indirectly access whenever they access one of their virtual network components. Furthermore, accessing a virtual component might result in accessing several data plane components, e.g., when a virtual component comprises multiple components of the data plane as in the "one-big-switch" abstraction. The access translation can be non-trivial and the controller's compiler implementing the translation might be incorrect because of software bugs or controller misconfigurations. This in turn can lead to incorrect updates of the switches' forwarding logic and such vulnerabilities of the controller can be exploited to launch attacks to the network.

Enhancing the network owner's controller (i.e., the controller that directly interacts with the network components at the physical data plane) with mechanisms that control and restrict the access of the network users (i.e., the network owner and the leasing tenants) and the applications to the network components at the data plane would reduce the network's attack surface and protect the network resources. We argue that a reference monitor—similar as in an operating system—should be an integral part of a controller in SDN.

In this paper, we focus on protecting the network flows, which are determined by the entries installed in the switches' flow tables. We propose an access control scheme to express various policies about who can access the switches' flow tables and how. To this end, we introduce at the control plane counterparts of data plane entities, namely, counterparts for the switches' flow tables and their entries. These counterparts carry meta-information, which is managed and used by the controller's reference monitor. For example, flow rules are owned by network users and a flow rule can be linked to other flow rules. A policy might stipulate that only the rule's owner can modify it or the rule can only be linked to rules that have

978-1-4799-7899-1/15/$31.00

the same owner. Furthermore, we introduce flow spaces that allow one to hierarchically structure the switches' flow tables. Flow spaces are also owned by network users and can be used to restrict the kind of flow rules a network user can install.

We opted for an access control scheme that is simple, focused, and close to the south-bound API of the controller, which interfaces directly with the switches using OpenFlow. The rational behind this design decision is as follows. First, it supports one to build a tamperproof and verifiable reference monitor. This is based on the scheme's simplicity and since it is focused. Furthermore, since the controller only communicates via OpenFlow messages with the switches, we obtain complete mediation by permitting or denying OpenFlow messages by the reference monitor before they are sent. These are essential principles for a reference monitor [10]. Second, the switches' flow tables are one of the most sensitive resources in a multi-tenant network. Their entries determine how the network handles the traffic. Moreover, they are shared between the tenants and their capacities are scarce. Controlling the access to them protects the network flows. Finally, we expect that future north-bound APIs in SDN will support multiple different abstractions of the network at the control plane. Any such interface will be built on top of the interface provided by OpenFlow, which directly interacts with the network components. Access control at higher layers will utilize our access control scheme and complement it.

In summary, the main contribution of this paper is a powerful, yet simple access control scheme for controllers for protecting the network flows in a SDN network. Thus, our work adds a fundamental security concept to controllers, which is essential for the development of mature SDN network operating system for multi-application and/or multi-tenant settings.

The remainder of this paper is organized as follows. We describe our access control scheme for SDN controllers in the Sections II to Section IV. In more detail: in Section II, we describe the part for accessing the switches' flow tables and its entries, in Section III, we describe mechanisms to protect entries by relating them to other entries, and in Section IV, we describe the part for messages that are initiated by a switch, including how to resolve conflicting reactions to such messages. We discuss related work in Section V and we draw conclusions in Section VI.

II. Accessing Flow Tables

In this section, we describe the part of our access control scheme for protecting the switches' flow tables and its entries from unauthorized access. As in most access control schemes, the entities of the scheme are classified as subjects and objects. In the proposed controller extension, policy enforcement is done at the control plane, close to the interface between the control plane and the network components (cf. Figure 1). In particular, the reference monitor permits or denies access to the objects; it might stop some OpenFlow messages to be sent to the data plane. The network components of the data plane are not affected by the extension. We start with a brief refresher on OpenFlow followed by a description of the entities of the scheme.

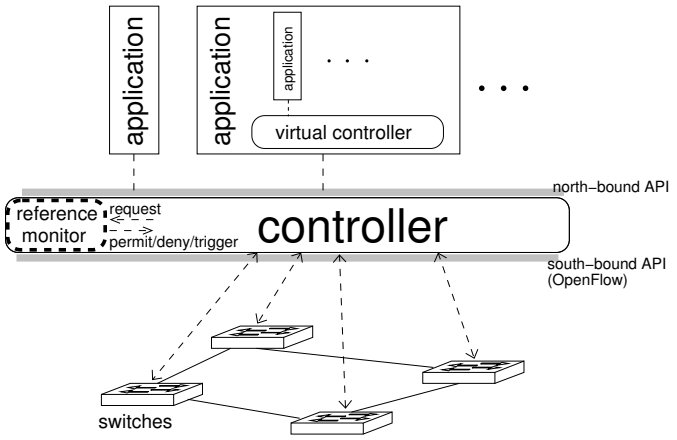

Fig. 1. Security-enhanced SDN architecture with reference monitor.

A. Preliminaries

OpenFlow is one of the most prominent instantiations of the SDN principles. Due to the lack of space, we refer the readers to [6] for details on OpenFlow.

In the sequel, we denote by *flow table entries* the rules that specify how network packets must be handled. They are stored in the switches' *flow tables*. Each such entry consists of a *match set* that determines to which packets the rule applies, an *action set* that defines how matching packets are processed, a *priority* field, and possibly hard or idle *timeout values*. The installation, modification, and deletion of an entry is done by sending an OpenFlow message of type `OFPT_FLOW_MOD` from the controller to the switch. Note that this message has several parameters that specify the flow table entries and how the flow table of a switch should be modified. Switches can also send messages to the controller. For instance, a switch can be configured to send an OpenFlow message of type `OFPT_PACKET_IN` to the controller whenever it receives a packet that has no matching flow table entry. The controller then triggers an application, running on top of the controller, to react to this message.

B. Subjects and Objects

The *subjects* of the access control scheme are the entities that manipulate objects, namely, users. Usually, there is a special subject, the network administrator, which has complete control over the network. We remark that applications are not subjects; they are executed on behalf of users (i.e., the subjects) and the user running an application manipulates the objects. They have therefore the permissions of the users. For example, the same monitoring application can be executed by different users, with different permissions. The *objects* of the scheme are as follows.

Flow Rules: The first kind of objects are flow rules, which can be understood as annotated flow table entries. Formally, a *flow rule*, FR for short, is a quadruple $(match, act, attr, info)$, where $match$ is a set of packet headers against which incoming packets are matched, act is a finite set of actions that a switch performs when this rule applies, $attr$ are the attributes of the FR, and $info$ contains meta-information about the FR. Attributes appearing in the OpenFlow standard for flow table entries are an entry's priority, its timeout values,

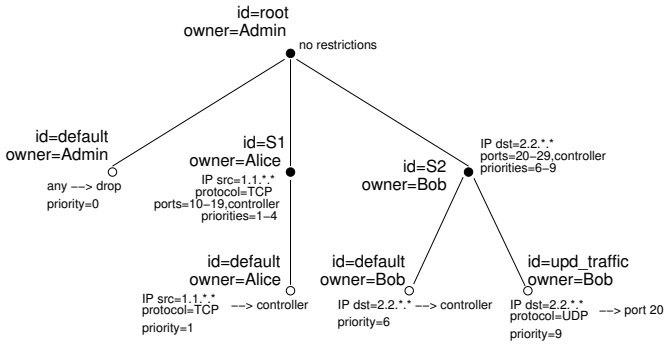

Fig. 2. A flow directory.

and whether a message must be sent to the controller when the entry is removed from the flow table. In Section III, we consider additional attributes. Examples of meta-information contained in the $info$ component of a FR is an identifier of the FR and the owner of the FR. In the following, we introduce more meta-information, which the controller's reference monitor uses for policy enforcement. To simplify matters, we use the terms flow rule and flow table entry interchangeably, assuming that each flow rule corresponds to exactly one flow table entry.

Flow Directories: The second kind of objects are flow directories. They share some similarities with directories in an operating system, where flow rules correspond to files and flow spaces (see below) correspond to subdirectories. In particular, a flow directory allows us to hierarchically structure a flow table. To define flow directories, we first need to define flow spaces.

A *flow space*, abbreviated by FS in the following, is a quadruple $(heads, acts, attrs, info)$, where $heads$ is a set of packet headers, $acts$ is a set of actions, $attrs$ is a set of attributes of FRs, and the $info$ component contains—as for FRs—meta-information. With $attrs$ we can, e.g., specify ranges of valid priorities and timeout values; only FRs with priorities in this range can be assigned to the FS.

The FR $r = (match, act, attr, info)$ is *valid* for the FS $S = (heads, acts, attrs, info)$ if $match \subseteq heads$, $act \subseteq acts$, and $attr \in attrs$. The FS S_1 is a *subspace* of the FS S_2 if S_1's $heads$, $acts$, and $attrs$ components are pointwise included in S_2's corresponding components. The FSs S_1 and S_2 are *overlapping* if $heads_1 \cap heads_2 \neq \emptyset$, where $heads_i$ is the first component of S_i, for $i \in \{1, 2\}$.

A *flow directory*, FD for short, is a tree, where each inner node is annotated with a FS and each leaf is annotated with either a FS or a FR such that for every inner node N with FS S, it holds that (i) r is valid for S if r is the FR of a child of N and (ii) S' is a subspace of S if S' is the FS of a child of N.

Example: Before we proceed, we illustrate flow rules, flow spaces, and flow directories. Consider the FD in Figure 2. A filled circle is annotated with a FS and a non-filled circle is annotated with a FR.

The root of the tree, owned by the system administrator, is annotated with the FS that does not impose any restrictions on the flows, i.e., its $heads$ set is the set of all packet headers, its $acts$ set is the set of all actions that can be performed by a rule on a matching packet, and its $attrs$ set is the set of all attributes that can be assigned to a rule.

The left most successor of the root, also owned by the system administrator, is annotated with the FR that specifies the default for incoming packets. It matches any packet header and the performing action is to drop a matching packet. Its priority is low so that it can be overwritten by other rules.

The other two successors of the tree's root are annotated by the FSs S_1 and S_2. S_1 is owned by the subject Alice and comprises packet headers with IP source 1.1.*.* and the protocol TCP. S_2 is owned by the subject Bob and comprises packet headers with IP destination 2.2.*.*. The allowed actions of FRs in these FSs are to forward packets to the controller, and to the ports 10 to 19 and 20 to 29, respectively. The priority for FRs in S_1 must be in the range of 1 to 4. For S_2, the priorities must be in the range 6 to 9. Note that S_1 and S_2 are subspaces of the root's FS. Furthermore, S_1 and S_2 overlap. Both Alice and Bob have their own default rules for handling packets. In addition, Bob has a FR that forwards UDP packets to port 20. All the FRs are obviously valid for the respective FSs. Also note that neither Alice's FS nor Bob's FS can contain a FR that drops packets.

We remark that FSs with different owners in a FD owners might overlap. We do not exclude overlapping FSs in the first place. For example, the FSs might have been obtained from virtualized networks in which the FSs' owner operate. Mapping these virtualized networks to the physical network might result in overlapping FSs. If overlapping FSs between different owners are considered as undesirable then the network administrator needs to prevent this.

Another remark concerns the FSs and its FRs in a FD. Note that the matching set $match$ of an FR might repeat values that can be inherited from the rule's FS. For example, in the FD shown in Figure 2, for the FR with id default in the FS S_1 owned by Alice we might want to inherit the IP source 1.1.*.* from the FS. This has the advantage that if a subject modifies the FS S_1's $heads$ set the default FR still covers all IP sources of the FS S_1. Such an extension is rather straightforward and for the sake of readability and space, we omit its details.

C. Permissions

For each switch, the controller maintains a FD in which objects—FRs and FSs—are assigned to its nodes. The semantics of a FD is given by the flow table that contains all its FRs. We require that overlapping FRs (i.e. FRs for which the intersection of their $match$ sets is non-empty) have different priorities—a condition that can be checked by the reference monitor whenever a FR is installed or modified. Each object in a FD is owned by a subject. Usually, the creating subject is the owner. However, ownership can also be transferred. The object owner can set the permissions for accessing the object. For delegation, the owner can assign permissions to other subjects. This information must be maintained, e.g., stored in ACLs, by the controller and is used by the controller's reference monitor for policy enforcement.

We have two permissions for accessing an object and for manipulating a FD, namely, read and modify, which the object owner assigns to the subjects. The read and modify permissions are as follows.

– The read permission allows one to read the fields of the object, e.g., the priority of a FR. However, information about the object like the assigned permissions of other subjects is only visible to the object owner. For FSs, the read permission also allows one to see the object's successor nodes in the FD, including, e.g., the object's identifier and its owner.
– The modify permission allows one to make certain modifications to the object. It is, e.g., necessary for a subject to delete the object. In case the object is a FS, modifications also comprise the creation of successor nodes. In particular, this is necessary for a subject to install a FR. Note that the FS itself imposes restrictions on the FRs that a subject can install in this FS. Namely, the FR must be valid for the FS. Similarly, when creating a successor node for a new FS then this new FS must be a subspace. However, note that the modify permission does not allow a subject to modify the FS's fields. For this, the subject needs the modify permission at the parent node, at which the object was created. In case the object is a FR, modifications comprise the changing of the rule's fields. However, the modified rule must still be valid for the FS at the parent node.

For policy enforcement, the reference monitor must be invoked when a subject wants to access an object in a FD, e.g., for changing the object's permissions. The reference monitor must also be invoked when a subject wants to send an OpenFlow message that can have an effect on the flow table of a switch. In particular, an OpenFlow message of the type `OFPT_FLOW_MOD` might get blocked by the reference monitor since the subject is lacking modify permissions for installing, deleting, or modifying a flow table entry. If the message is permitted, the FD must be appropriately updated. Note that for performing such a check, the reference monitor must be aware of the subject that wants to send the message, and it must know the subject's permissions for the objects that are affected by this message. This includes the FS in which the subject wants, e.g., to install the new FR. In case the reference blocks a message, it notifies the subject that the requested access is not permitted. This way, the subject is aware that the network configuration has not changed and might request another reconfiguration.

For simplicity, we assumed in the above described invocation of the reference monitor that a subject's action comprises only a single OpenFlow message. However, handling, e.g., a network flow might involve the sending of multiple `OFPT_FLOW_MOD` messages to different switches, which should be all permitted or all denied be the reference monitor. If only some of them are sent to the switches, the network might get misconfigured. Hence, a subject can bundle messages and the reference monitor permits the bundle if the subject possesses the permission for sending each individual message.

Also reply messages from the switches might get blocked or sanitized by the reference monitor in case they contain information about the installed flow table entries for which the subject is lacking read permissions. In Section IV, we discuss extensions where the reference monitor needs to be invoked on further OpenFlow messages of other types.

Note that the capacity of the flow tables in today's physical switches is a scarce resource. To account for this, we can equip each FS with a quota that limits the number of FRs that can be contained in the FS. The number of FRs in a FS at a node in a FD is the number of nodes in the subtree of the FS's node that are labeled by a FR. Note that we assume here that a FR corresponds to a single flow table entry. If this is not the case, e.g., since the FR expands into multiple entries in a TCAM, we can equip a FS with different quotas, in particular, one for the maximal number of TCAM entries. The reference monitor needs to keep track of the current numbers and must deny the installation or modification of FRs if the quota of a FS is exhausted.

III. Relating Flow Table Entries

The permissions in Section II restrict how individual objects can be accessed. In particular, the modify permission for a FR is concerned about how this rule can be modified. However, a rule's modification can affect other rules, which together handle the flows in the network. In this section, we complement the permissions in Section II by restricting the access to a FR depending on how it relates to other FRs. To express relations between FRs, we introduce new flow rule attributes, in addition to the OpenFlow attributes (priority, timeout values, etc.). Due to space limitations we restrict ourselves in the following to only a few new attributes. However, our list of attributes is not meant to be complete by any means.

Overwriting Flow Rules: The first attribute in our list is the overwrite attribute. It prevents the installation of certain FRs with a lower priority. The parameter of a rule's overwrite attribute specifies the FRs (with higher priority) that are allowed to be installed or modified and for which the intersection of the match sets is non-empty. For example, one can specify that the FR can only be (partially) overwritten by FRs that have the same owner as the FR. This condition can either be relative to a specific FS or all rules in the switch's flow table. For policy enforcement, the reference monitor must check, whenever a switch's flow table is reconfigured, whether the conditions specified by the overwrite attributes of the installed FRs are satisfied.

Cascading Flow Rules: The second attribute in our list is the cascade attribute. It is inspired by data integrity constraints across multiple tables in relational databases. There, one links rows from different tables and, for example, requires the automatic deletion of a row whenever a row linked to that row is deleted. In our setting, the cascade attribute allows one to relate FRs that are installed in flow tables from different switches. The intuition is that related FRs form a larger entity for handling certain network flows. For example, a FR in the "one-big-switch" abstraction corresponds to such a larger entity since it is compiled down to multiple FRs.

More formally, a *cascaded flow rule*, CFR for short, is a finite DAG, where its nodes are labeled by FRs. Note that since a FR's action set can contain actions for forwarding a packet, e.g., to two different ports, we use DAGs instead of sequences here. We also remark that we do not impose any requirements on how the FRs' matching sets in a CFR must fit together. Such requirements would result in additional checks for the reference monitor, e.g., when a FR of a CFR is modified. Although such checks are possible [11], [12], they might be prohibitive in practice in terms of computational costs. Instead, we assume here that FRs are assembled meaningfully into CFRs in terms of the traffic they handle and focus on maintaining the

knowledge in which CFRs a FR is part of. As a consequence, the controller only needs to maintain pointers between the FRs to quickly access the FRs of a CFR. Since a FR can be a part of multiple CFRs, we equip FRs with vectors of pointers.

The deletion of a FR that is part of a CFR requires the modify permission of all the FRs that are part of the CFR. The cascade attribute of a FR r triggers on r's deletion the deletion of each FR t in r's CFRs, unless t is also a part of another CFR (in this case only t's pointers for r's CFRs are removed). This can be understood as some sort of garbage collection for flow tables and can be implemented efficiently by following the pointers attached to the FRs in a CFR. Modifying a FR r also requires the modify permission for all FRs t in which r is part of. Furthermore, the cascade attribute only allows one to install a FR that (partially) overwrites r if r is the first rule of a CFR. Similarly, when installing the FRs of a CFR then they must not interrupt the flow determined by an already installed CFR, assuming that the FRs' matching sets fit together.

Connecting Flow Rules: The third attribute in our list is the connect attribute. It complements the cascade attribute. The connect attribute differs from the cascade attribute in that it does not relate specific FRs. Instead, it allows one to specify conditions on a FR what FRs can send traffic that is then handled by the FR. Such a condition is, e.g., that the FR must only handle traffic that comes from FRs that are owned by the subject Alice. A special case of the connect attribute is the service attribute that stipulates that packets processed by the FR must only be processed by other FRs with the same service number, assuming that each FR is equipped with such a number. Alternatively, if the service numbers are (partially) ordered, we can, e.g., require that the service numbers of FRs that process a packet must increase.

As already remarked for the cascade attribute, checking that a FR only handles network flows of a certain type is non-trivial in general [11], [12]. The reference monitor would have to perform such a check online whenever a subject requests to modify or install a FR. To decrease the computational burden, we can restrict ourselves to pairs of FRs from two connected switches. Furthermore, we can ignore the priority of the FRs so that we do not need to account for FR that are (partially) overwritten by others.

IV. Triggering Applications

In this section, we extend the access control scheme to also account for packet-in events, i.e., for OpenFlow messages of the type `OFPT_PACKET_IN`, which also concern the network flows. Recall that these messages are sent from a switch to the controller. Subjects can subscribe applications that are triggered when the controller receives such a message. Subscriptions are relative to a flow space and require the subscribe permission. For example, assume that Bob has the subscribe permission in the FS S_2 in the FD depicted in Figure 2 for packet-in events. This allows Bob to subscribe, e.g., his routing application. The application is triggered when a switch receives a packet that does not match any installed flow table entry and the packet header falls into the FS S_2. Note that in case multiple applications are subscribed in the same FS or in overlapping FSs, all these applications are triggered. Note that also OpenFlow messages of the type `OFPT_FLOW_REMOVED` and `OFPT_PORT_STATUS`, which are sent by a switch when an entry is removed from its flow table and when the status of one of its ports changes, respectively, concern the network flows. As for packet-in events, we can subscribe permissions for these events.

Triggering multiple applications becomes problematic when the applications' reactions are conflicting. For instance, for a packet-in event, assume that one application wants to drop the packet, while another one wants to install a FR that forwards the packet to some port. These two applications might even be subscribed in different but overlapping FSs. Such conflicts must be prevented or resolved quickly. We assume here that the controller can correlate an application's reaction with the event that caused it, e.g., by including identifiers to messages.

Conflicts can be resolved statically or dynamically. In the static approach the reference monitor checks, before permitting the subscription of an application, whether the application's reactions might conflict with the reactions of the already subscribed applications. To perform this check, we require that the subscriber specifies the reactions of the subscribed application (unspecified reactions of an application will be ignored by the controller). Although this prevents conflicts, it might be too conservative. A dynamic approach is more liberal. Here, we prioritize the applications' reactions and only carry out the ones that are not in conflict with a reaction of a higher priority. Priorities for resolving conflicts can, e.g., be determined by the subscribers or the FSs if the trigger event is originating from a FR (the network administrator assigns priorities to the subjects or FSs). Priority can also be given to FSs deeper or higher in the FD. Conflicts can also be first resolved locally, i.e., within each FS, before resolving the conflicts of applications subscribed to different FSs. The method for resolving local and global conflicts does not need to be the same.

We can save computational power by not triggering all subscribed applications. With prior knowledge on how applications react to events (similar to the static approach), we only trigger the applications that do not conflict with the application that has the highest priority. Note that determining the application with the highest priority can be done by traversing the FD.

V. Related Work

A preliminary version of our access control scheme was presented as a poster [13]. Prior work on controlling access to network components in SDN also appears in FlowVisor [9], FortNOX [14], PANE [15], PermOF [16], and YANC [17].

FlowVisor can been seen as a controller that allows one to run other controllers as applications. It primarily targets network slicing and network virtualization. It also provides some basic means for restricting the access to network resources. A slice can be understood as a flow space in our access control scheme. Each controller running on top of FlowVisor can have read or write permissions on a network slice. FlowVisor does not provide means, as done by the attributes of FRs in our scheme, to relate flow table entries. Furthermore, FlowVisor provides no support for resolving conflicts. FortNOX's extends the NOX controller [4] with an access control mechanism that targets the installation of flow table entries and handles conflicts of a specific type. In particular, applications, according to an assigned priority, may not overwrite certain

flow table entries. This is related to the attribute overwrite in our access control scheme. The controller PANE targets participatory networking [18]. PANE provides a north-bound API to applications like Skype to request network resources. Its access control scheme does not account, e.g., for trigger events. The proposed access control scheme PermOF allows the network administrator to assign permissions for different types of OpenFlow messages to applications. Flow tables entries cannot be hierarchically structured and related. YANC is a controller that is integrated into Linux. It inherits some features of the operating system's discretionary access control scheme. However, the details concerning YANC's access control scheme are not discussed in [17].

Policy network languages like the ones presented in [12], [19] allow one to specify access control policies with respect to the forwarding of network packets. In contrast, policies of our access control scheme concern the administration of the network components.

VI. Concluding Remarks

We presented a discretionary access control scheme for SDN controllers for protecting the network flows in a multi-application/multi-tenant setting. It includes mechanisms for restricting the access to the switches' flow tables and its entries and for relating flow table entries. The mechanisms are inspired by access control schemes commonly found in operating systems and in database management systems. The proposed scheme is low-level in the sense that it is close to the south-bound API of the controller, which interfaces directly with network components using OpenFlow. We expect that future north-bound APIs in SDN will support multiple different abstractions of the network at the control plane. Furthermore, any such interface will be built on top of the interface provided by OpenFlow that directly interacts with the network components. We claim that building a tamperproof, verifiable reference monitor with complete mediation for an access control scheme at a higher level will be more complicated. Moreover, such a scheme will be complemented by ours.

Variants and extensions of the described access control scheme are possible. For example, the scheme can be complemented by a mandatory access control scheme, where the system administrator assigns permissions to subjects for accessing the objects. This would allow the network owner to enforce some general rules that apply to all or some network users. The scheme can also be extended to account for network components like the ports of a switch in addition to flow tables and its entries. For such an extension, one needs to introduce permissions of how a port can be accessed. One canonical choice is to have the read permission for reading the port's current status and modify permission for changing its status.

In terms of future work, we plan to develop a reference monitor for the proposed access control scheme, deploy it into SDN controllers, and evaluate its performance in practical settings. This includes the design and implementation of algorithms to efficiently carrying out the checks by the reference monitor. We also plan to develop access control schemes that support more abstract views of the network.

References

[1] N. McKeown, T. Anderson, H. Balakrishnan, G. Parulkar, L. Peterson, J. Rexford, S. Shenker, and J. Turner, "OpenFlow: Enabling innovation in campus networks," *ACM SIGCOMM Computer Communication Review*, vol. 38, no. 2, pp. 69–74, 2008.

[2] S. Jain, A. Kumar, S. Mandal, J. Ong, L. Pouttievski, A. Singh, S. Venkata, J. Wanderer, J. Zhou, M. Zhu, J. Zolla, U. Hölzle, S. Stuart, and A. Vahdat, "B4: Experience with a globally-deployed software defined WAN," in *Conference on Applications, Architectures, and Protocols for Computer Communications (SIGCOMM)*, 2013.

[3] A. Gudipati, D. Perry, L. Li, and S. Katti, "SoftRAN: Software defined radio access network," in *Workshop on Hot Topics in Software Defined Networking (HotSDN)*, 2013.

[4] N. Gude, T. Koponen, J. Pettit, B. Pfaff, M. Casado, N. McKeown, and S. Shenker, "NOX: Towards an operating system for networks," *ACM SIGCOMM Computer Communication Review*, vol. 38, no. 3, pp. 105–110, 2008.

[5] D. Erickson, "The Beacon OpenFlow controller," in *Workshop on Hot Topics in Software Defined Networking (HotSDN)*, 2013.

[6] "OpenFlow switch specification—version 1.0.0," Open Networking Foundation, 2009.

[7] S. Raza and D. Lenrow, "North bound interface working group charter," Open Networking Foundation, 2013, www.opennetworking.org/images/stories/downloads/working-groups/charter-nbi.pdf.

[8] M. Casado, T. Koponen, R. Ramanathan, and S. Shenker, "Virtualizing the network forwarding plane," in *Workshop on Programmable Routers four Extensible Services of Tomorrow (PRESTO)*, 2010.

[9] R. Sherwood, G. Gibb, K.-K. Yap, G. Appenzeller, M. Casado, N. McKeown, and G. Parulkar, "Can the production network be the testbed?" in *Symposium on Operating Systems Design and Implementation (OSDI)*, 2010.

[10] J. P. Anderson, "Computer security technology planning study," US Air Force Electronic Systems Devision, Tech. Rep. ESD-TR-73-51, 1972.

[11] A. Khurshid, X. Zou, W. Zhou, M. Cesar, and P. B. Godfrey, "VeriFlow: Verifying network-wide invariants in real time," in *Symposium on Network Design and Implementation (NSDI)*, 2013.

[12] P. Kazemian, M. Chang, H. Zeng, G. Varghese, N. McKeown, and S. Whyte, "Real time network policy checking using header space analysis," in *Symposium on Network Design and Implementation (NSDI)*, 2013.

[13] F. Klaedtke, G. O. Karame, R. Bifulco, and H. Cui, "Access control for sdn controller," in *Workshop on Hot Topics in Software Defined Networking (HotSDN)*, 2014, poster presentation.

[14] P. Porras, S. Shin, V. Yegneswaran, M. Fong, M. Tyson, and G. Du, "A security enforcement kernel for OpenFlow networks," in *Workshop on Hot Topics in Software Defined Networking (HotSDN)*, 2012.

[15] A. D. Ferguson, A. Guha, C. Liang, R. Fonseca, and S. Krishnamurthi, "Participatory networking: An API for application control of SDNs," in *Conference on Applications, Architectures, and Protocols for Computer Communications (SIGCOMM)*, 2013.

[16] X. Wen, Y. Chen, C. Hu, C. Shi, and Y. Wang, "Towards a secure controller platform for OpenFlow applications," in *Workshop on Hot Topics in Software Defined Networking (HotSDN)*, 2013, poster presentation.

[17] M. Monaco, O. Micher, and E. Keller, "Applying operating system principles to SDN controller design," in *Workshop on Hot Topics in Networks (HotNets)*, 2013.

[18] A. D. Ferguson, A. Guha, J. Place, R. Fonseca, and S. Krishnamurthi, "Participatory networking," in *Workshop on Hot Topics in Management of Internet, Cloud, and Enterprise Networks and Services (Hot-ICE)*, 2012.

[19] T. Nelson, A. Guha, D. Dougherty, K. Fisler, and S. Krishnamurthi, "A balance of power: Expressive, analyzable controller programming," in *Workshop on Hot Topics in Software Defined Networking (HotSDN)*, 2013.