Enforcing Multilevel Security Policies in Database-Defined Networks

Ali Al-Haj and Benjamin Aziz
School of Computing, University of Portsmouth
Portsmouth PO1 3HE, United Kingdom
{ali.alhaj, benjamin.aziz}@port.ac.uk

***Abstract*—Despite the wide of range of research and technologies that deal with the problem of routing in computer networks, there remains a gap between the level of network hardware administration and the level of business requirements and constraints. Not much has been accomplished in literature in order to have a direct enforcement of such requirements on the network. This paper presents a new solution in specifying and directly enforcing security policies to control the routing configuration in a software-defined network by using row-level security checks. We show, as a first step, how a specific class of such policies, namely multilevel security policies, can be enforced on a database-defined network, which presents an abstraction of a network's configuration as a set of database tables. We show that such policies can be used to control the flow of data in the network either in an upward or downward manner.**

***Index Terms*—Software-Defined Networking, Database-Defined Networking, Information Flow Control, Row-Level Security, Security Policies, Multilevel Security**

I. Introduction

Complexity and robustness remain some of the main challenges that dominate the networking world [20], which are still frequently researched and thought over at the low level of the network hardware with little provision for establishing a direct relationship with business and application requirements and constraints. Moreover, nowadays the majority of the provided solutions are tightly restricted to vendor-specific hardware and hence, network administrators require extensive knowledge of the network technologies in order to enforce specific administration rules. By contrast, Software-Defined Networks (SDNs) [22] have emerged as a new paradigm based on the separation of network control plane from data plane and therefore facilitate a high-level management of the network in a direct manner. The control plane, being a logically centralized controller or a set of cooperating SDN controllers, is often implemented using standards such as OpenFlow protocol [28], which collect information from the data plane and offer a global view to the network operators. Moreover, all the management tasks are implemented as applications working on top of the controller.

More recently, a new approach to the implementation of SDNs has emerged aimed at simplifying the task of network administration through the introduction of further data-based abstractions of the control and data planes. This approach is called Database-Defined Networking (DDN) [33], which represents the entire network through standard relational databases. DDNs simplify the network management since the interface to its current state becomes purely database defined. Hence, the network can be *queried* and its configuration *updated* using standard data languages, such as Standard Query Language (SQL). Interestingly, it becomes straightforward to divide the network into multiple zones and enforce access rules on those zones using access control lists [18].

In this short paper, we demonstrate that recent security mechanisms in databases, particularly, row-level security, can be used to enforce more complex security policies, such as those using multilevel security [5], [6], [13] to control the flow of data. Row-level security has recently emerged as a feature in database management systems, allowing administrators to introduce security policy checks at the level of a row in a table. We demonstrate that changes in the network topology can be policy-controlled through the enforcement of policies on table rows such that only "good" topologies are deployed.

The rest of the paper is structured as follows: in Section II, we give an overview of the background relevant to the work of this paper discussing both database-defined networking and row-level security. In Section III, we discuss some related works in current literature. In Section IV, we give an overview of our proposed approach to the tackling of the problem of policy enforcement in database-defined networking. In Section V, we define a method for enforcing multilevel security policies in the routing of packets within a database-defined network using the row-level security feature in modern database systems. Finally, in Section VI, we conclude the paper and suggest some directions for future research.

II. Background

We give here a summary background on two of the relevant concepts that drive the work presented in this paper, namely database-defined networking and row-level security.

A. Database-Defined Networking

Database-Defined Networking (DDN) is a concept of using relational databases as an abstraction for managing an SDN, DDN is similar to Declarative Networking [25] but applied in the context of SDN. The concept of DDN has recently been introduced in RAVEL [33], which is a database-defined controller that represents the network using a standard relational database, e.g. PostgreSQL [19]. The architecture of RAVEL is shown in Figure 1.

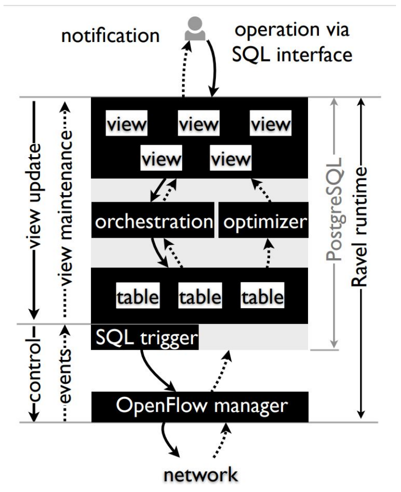

Fig. 1. RAVEL Architecture [34]

The design of RAVEL is driven by a separation of data and control planes, which revolves around data representation. In RAVEL, to merge multiple views into single coherent forwarding behavior, the system automatically orchestrates the abstractions thereby allowing multiple simultaneous abstractions to collectively drive control [33].

Ravel's database view has numerous benefits as the ad-hoc programmable abstractions are facilitated through the database views. This enables the construction and building of new applications on one another. Another positive feature is the ability to express integrity constraints above views, where high-level policy constraints are expressed through the use of SQL statements, and the controllers are constructed dynamically while running. Additionally, RAVEL provides an SQL interface through which the state of the network can be viewed and its configurations be updated. This will allow application developers and network operators to control and query the network through a SQL interface. As a result, RAVEL represents a network in a flat manner exposing the topology and forwarding information as three tables:

- `tp`: this is the network topology table representing pairs of connected nodes. This table is defined by the type:
  ```
  tp(sid integer, nid integer, ishost
   ↪  integer, isactive integer, bw
   ↪  integer)
  ```
  where `sid` is an integer representing the identity of the current switch (node), `nid` is an integer representing the identity of the next hop node, `ishost` is a 0/1 integer to denote whether the current node is a host (1) or not (0), `isactive` is a 0/1 integer to denote whether the current link between `sid-nid` is online (1) or not (0) and finally, `bw` is an integer denoting the bandwidth of the link between `sid-nid`.
- `cf`: this is a table representing per-switch configurations. This table is defined by the type:
  ```
  cf(fid integer, pid integer, sid
   ↪  integer, nid integer)
  ```
  where `fid` is an integer representing the identity of a flow (path), `pid` is an integer representing the identity of the previous hop node, `sid` is an integer representing the identity of the current switch (node) and finally, `nid` is an integer representing the identity of the next hop node. It is this table that defines how a packet is routed over the network.
- `tm`: this is an end-to-end reachability matrix describing properties of the current source-to-destination flow (path). This table is defined by the type:
  ```
  tm(fid integer, src integer, dst
   ↪  integer, vol integer, FW integer,
   ↪  LB integer)
  ```
  where `fid` is an integer representing the identity of the current flow, `src` is an integer representing the IP address of the source node in the flow, `dst` is an integer representing the IP address of the destination node in the flow, `vol` is an integer representing the amount of traffic volume allocated for the path, `FW` is a 0/1 integer denoting whether the data flow of the path has to pass through a firewall (1) or not (0) and finally, `LB` is a 0/1 integer denoting whether the flow in the path has to be load-balanced (1) or not (0).

Additionally, there are other node tables, which include a host and a switch table as well as a generic node table containing the identities and names of all the nodes in the network.

B. *Row-Level Security*

Recently, Microsoft [27] and PostgreSQL [30] released a new feature called Row-Level Security (RLS), that integrates with their database management systems. Whilst previously, security was enforced on a whole table, RLS introduces fine-grained security policy checks at the level of *single rows* in a database table. Under an RLS policy, users are able to create security conditions on tables that they own to grant access to other subjects. Each security policy consists of:

- A subject to whom the policy grants access.
- An SQL expression that evaluates to true for all records the subject is able to access in the table.
- An SQL command `(Select, Insert, All, etc)` or role that the security policy applies to.

To create a policy, the following syntax is used:
```
CREATE POLICY name ON table_name
 [ FOR { ALL | SELECT | INSERT | UPDATE |
  ↪  DELETE } ]
 [ TO { role_name | PUBLIC | CURRENT_USER |
  ↪  SESSION_USER } [, ...] ]
 [ USING ( using_expression ) ]
 [ WITH CHECK ( check_expression ) ]
```
Various security policies can be defined for a table by users, these policies can be easily granted, revoked, and modified. Moreover, anytime users issue a query against a table, the SQL expressions of all security policies defined on the table that match the SQL command issued by the subject or the subject's role will be evaluated as predicates on the issued query. This ensures that the query executed by the subject only applies to records in the table that render the expressions true. Each

existing table row is checked against the expression specified in `USING`, while new rows that would be created via `INSERT` or `UPDATE` are checked against the expression specified in `WITH CHECK`. Both `USING` and `WITH CHECK` accept only SQL conditional expressions returning a Boolean value (True, False, Null). In order to created policy to be enforced, RLS should be enabled using the following command:

```
ALTER TABLE sometable ENABLE ROW LEVEL
 ↪  SECURITY;
```

III. Related Work

Network management has developed and becomes more vibrant, with the SDN paradogm emerging in recent years [23]. The integration of the Internet, software technologies and traditional telecommunication technologies has brought challenges to network and service operators as far as service deployment and management [10], [11], [17] are concerned. One particular challenging area is the management of security policies. We outline below a few related works in this area that have been proposed and implemented recently.

The authors in [16] proposed *Frenetic*, an OpenFlow-based network programming language, which provides an interface to query traffic information and create policies to react to network events. Simplification of network event programming and retrieval of traffic information is the main focus of Frenetic, though it does not provide alternative mechanisms for handling events sent by network switches. *Procera*, another high-level language proposed in [21], allows administrators to define policies and deploy in SDN networks. A dynamic network reconfiguration is required for this framework since it focuses on event-driven networks. According to [1], in order to validate the Procera framework, the scalability of the number of rules and the performance related to the time of translation of these rules to OpenFlow rules remains to be evaluated.

Fresco [31] is another OpenFlow-based security framework, where the security modules are exposed to external users giving them the ability to define and enforce security policies. Definition of the types, input/output parameters, actions and events are all required information for using Fresco. Fresco can be compared to Procera and Frenetic, in terms of allowing network events to be manipulated and in handling them through predefined modules.

Ponderflow [4] uses the Ponder language [12] for managing an OpenFlow network. The main drawback of *Ponderflow*, however, is that it lacks policy conflict resolution mechanisms. In addition to that, no experiments were made by the authors, for translating the proposed Ponderflow language to OpenFlow rules, to validate their approach within a real-world scenario.

OpenSec [24] is another policy-based network security management system, in which the authors focused on simplifying how network security policies are implemented and how they can respond to system alerts. OpenSec implements network policies in a simple language, which is then automatically converted into a set of rules that are set up into the network devices' level. OpenSec allows administrators to define a flow in terms of OpenFlow matching fields and identify which security properties should apply to that flow. Additionally, administrators can specify security levels that determine how OpenSec would react to malicious traffic, if detected.

PolicyCop proposed in [3] as, an autonomic framework for Quality-of-Service (QoS) policy enforcement in SDNs, by which service-level agreements are allowed to be specified for implementation and enforcement of QoS policies in an OpenFlow-based network. PolicyCop converts QoS policies into flow rules.

Recently, the authors in [32] proposed a network policy chain criteria based on the Database-Defined Networks approach, they employ the database integrity constraints to provide a logical framework to describe network policies. Moreover, the core idea behind thier work is the semantic modelling of network policies as integrity constraints that is managed by relational database.

Finally, the work most relevant to the work presented in our paper here is the information-flow type system defined by [8]. The type system ensures secure flow of information according to some high- and low-level classification relation. However, their treatment is restricted to the preservation of the secrecy of data, and it requires the implementation of a secure controller system that will replace the standard SDN controller, rendering their approach more complex than ours.

IV. The Proposed Approach

Our main approach is to control the manner in which network routing configurations are generated such that only those configurations, (in RAVEL, defined by the `cf` table), that confirm to some predefined security policy are allowed to be deployed upon the underlying network architecture. In this sense, we maintain secure network flow of data as defined by the security policy being enforced.

As shown in Figure 2, our approach currently uses RLS to enforce Multi-Level Security (MLS)-based policies. The interaction of the business applications with the network tables is limited through the policy specification and enforcement layers.

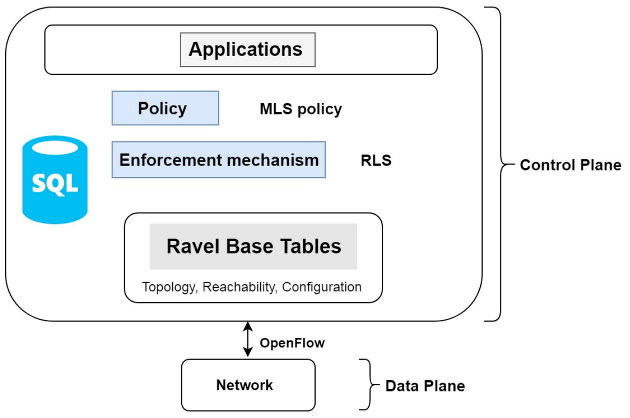

Fig. 2. Overview of our Approach

V. Enforcing MLS Policies

We introduce the idea that multilevel security policies can be used to control the flow of packets and data in a DDN using the RLS mechanism. We give first a short introduction to multilevel security, afterward we demonstrate how upward and downward routing of data packets can be achieved.

A. Multilevel Security

Multilevel Security (MLS) models are built based on the classifications expressed by security levels of the system components. Data objects have security levels and users have clearance levels. The security levels of objects are known as *security labels*. A security label can contain one security level or a list of levels. Formally, an MLS labelling system can be defined as a bounded lattice, $\mathit{Lattice}_S$ of security labels [15]:

$$\mathit{Lattice}_S = (\mathcal{L}_S, \leq_S, \top_S, \bot_S)$$

where $\mathcal{L}_S$ is a set of labels representing security sensitivity or classification levels, ranged over by ℓ_i, where $i \in \{1, \ldots, n\}$ for some value of n. This set is accompanied by the partial ordering relation $\leq_S$, which is reflexive, antisymmetric and transitive. The structure becomes a lattice when for any two levels, $\ell_1, \ell_2 \in \mathcal{L}_S$, then both ℓ_1 and ℓ_2 would have a least upper bound ($\top_{\{\ell_1,\ell_2\}}$) and a greatest lower bound ($\bot_{\{\ell_1,\ell_2\}}$) within $\mathcal{L}_S$. The highest label in the lattice is called $\top_S$ and the lowest level $\bot_S$. An example of a lattice is the lattice constructed from the power set of a set and ordered by the non-strict subset inclusion relation, $(\wp(\mathit{Set}), \subseteq, \mathit{Set}, \{\})$.

A special case of a lattice is one in which the relation $\leq_S$ is a total ordering, where there are no incomparable elements:

$$\forall \ell_i, \ell_j \in \mathcal{L}_S : \ (\ell_i \leq_S \ell_j \ \vee \ \ell_j \leq_S \ell_i)$$

An example of such a total order would be the chain of document security classifications:

$$\mathit{public} \leq_S \mathit{confidential} \leq_S \mathit{secret} \leq_S \mathit{top\ secret}$$

Next, we consider the problem of encoding lattices as tables in order for the information contained in them to be enforceable within the environment of relational databases. A lattice of security labels can be encoded as a matrix using ζ matrices [2] as follows:

$$\zeta(\ell_i, \ell_j) = 1 \ \mathit{iff}\ \ell_i \leq_S \ell_j$$

for any two security labels, $\ell_i, \ell_j \in \mathcal{L}_S$. Otherwise, $\zeta(\ell_i, \ell_j) = 0$. Furthermore, it is possible to model a ζ matrix as a table with the following type, and where $\mathcal{B} = \{0, 1\}$:

$$\mathit{Table}_S : \ \mathcal{L}_S \times \mathcal{L}_S \times \mathcal{B}$$

In SQL, this can be implemented with the following command:

```
CREATE TABLE mls_lattice (label_i int, label_j
 ↪  int, zetavalue_ij int)
```

For example, consider the following lattice:

$$(\{\ell_1, \ell_2, \ell_3, \ell_4, \ell_5\}, ((\ell_2 \leq_S \ell_1), (\ell_3 \leq_S \ell_1), (\ell_4 \leq_S \ell_2), (\ell_4 \leq_S \ell_3), (\ell_5 \leq_S \ell_4)), \ell_1, \ell_5)$$

This can be represented as the following ζ matrix:

$$\mathrm{Ex}_S = \begin{array}{c|ccccc} & \ell_1 & \ell_2 & \ell_3 & \ell_4 & \ell_5 \\ \hline \ell_1 & 1 & 0 & 0 & 0 & 0 \\ \ell_2 & 1 & 1 & 0 & 0 & 0 \\ \ell_3 & 1 & 0 & 1 & 0 & 0 \\ \ell_4 & 1 & 1 & 1 & 1 & 0 \\ \ell_5 & 1 & 1 & 1 & 1 & 1 \end{array} \qquad (1)$$

Where the diagonal running from the top left to the bottom right of the matrix is filled with 1s due to the reflexive nature of $\leq_S$. The matrix is then encoded as an SQL table with the values $(\ell_i, \ell_j, \zeta(\ell_i, \ell_j))$ for all $i, j \in \{1, 2, 3, 4, 5\}$. Another approach for encoding a lattice as a table was given by Denning [14] using flow-preserving mappings.

As we discussed earlier and as demonstrated by the RAVEL system, a DDN facilitates the notion of the management of network topology and routing using a purely database-oriented approach. Row-level security further controls the manner in which the basic network tables should be updated and viewed. In the case of MLS-based policies, it is possible to combine the above representation of MLS systems with network management, for example, how packet routes should be defined in a DDN. We shall demonstrate next how variants of two well-known MLS-based policies, the Bell-La-Padula (BLP) [5] and the BIBA [6] policies, can be encoded in RLS to enforce upward or downward flows of data. Furthermore, the security labeling of network components will allow end-to-end assurance of integrity and/or confidentiality, based on the security policy type that is enforced. For instance, the authors in [29] categorised the network into various zones (red, gray and black) on the basis of data sensitivity that need to be carried over the network. Similarly, our approach associates each node (e.g. switch and router) in the network with a security label and allows policy-controlled traffic among the classified network.

B. Enforcing Upward Flow of Data

We start first by defining the `cf` table as a function:

$$\mathit{cf} : \mathcal{N} \rightarrow ((\mathcal{N} \times \mathcal{N} \times \mathcal{N}) \rightarrowtail \mathcal{N})$$

which takes as input the path's number (i.e. `fid`) and returns as output an injective function, which can provide a sequence of triples representing the numbers of nodes forming the path, defined as the number of the current switch (i.e. `sid`), the previous (i.e. `pid`) and next (i.e. `nid`) nodes. This triple is an expression of the configuration of the various switches on the network that the path traverses. The fact that each triple is itself mapped to a number renders the triple as an ordered sequence.

For simplicity, and for a given path f_{id}, we refer to the first node that appears in the path as the source src_{fid}:

$$src_{fid} = \mathit{fst}((\mathit{cf}(f_{id}))^{-1}(1))$$

where fst is a special function that returns the first element in a tuple t, assuming $n \geq 1$:

$$\forall t, t = (e_1, \ldots, e_n) : \ \mathit{fst}(t) = e_1$$

We can similarly define the second and third elements, assuming consecutively that $n \geq 2$ and $n \geq 3$:

$$\forall t, t = (e_1, \ldots, e_n) : \ snd(t) = e_2$$

$$\forall t, t = (e_1, \ldots, e_n) : \ thd(t) = e_3$$

We consider here a policy that states that packets (i.e. data) must only travel upward, akin to the properties of a BLP policy [5] preserving the secrecy of data. We define the following logical formula to formalise this kind of data flows:

$$(\theta(fst((cf(f_{id}))^{-1}(i))) \ \leq_S \ \theta(snd((cf(f_{id}))^{-1}(i)))) \quad \wedge \\ (\theta(snd((cf(f_{id}))^{-1}(i))) \ \leq_S \ \theta(thd((cf(f_{id}))^{-1}(i)))) \quad (2)$$

where i is ranged over the number of entries a particular path f_{id} has in a routing table, equivalent to the size of the injective function $cf(f_{id})$:

$$i = 1 \ldots |cf(f_{id})|$$

Formula (2) states that packets should only be routed on switches of increasing security classifications and therefore, packets (and hence data) are always travelling in an upward direction in the security lattice.

The security label for a node itself is returned via the special function θ, defined as:

$$\theta : \mathcal{N} \to \mathcal{L}_S$$

which takes a number identifying some node and returns the security label for that node.

A simpler and less restrictive variation of Formula (2) would have been to say that only the source of the packet is less than or equal in its security classification than other nodes in the path on which the packet will travel:

$$\forall n \ \in \ cf(f_{id})^{-1}(i) \quad : \quad \theta(src_{fid}) \quad \leq_S \quad \theta(n)$$

again where $i = 1 \ldots |cf(f_{id})|$. This variation is less restrictive as it allows more freedom for the packets to travel within a section of the network, as long as the section itself is of higher classification than the classification of the source node.

1) Implementation in SQL: The upward flow of data condition as formulated in Formula (2) can be enforced as an RLS policy defined in the following manner:

```
CREATE POLICY UpwardFlow ON cf FOR ALL TO
 ↪  PUBLIC USING ((select zetavalue_ij from
 ↪  mls_lattice where pid=
label_i and sid=label_j)=1 and (select
 ↪  zetavalue_ij from mls_lattice where
 ↪  sid=label_i and nid=label_j)=1)
WITH CHECK ((select zetavalue_ij from
 ↪  mls_lattice where pid=label_i and
 ↪  sid=label_j)=1 and (select zetavalue_ij
 ↪  from mls_lattice where sid=label_i and
 ↪  nid=label_j)=1);
```

The condition on both the `USING` and `WITH CHECK` parts is similar; this is to ensure that the upward flow of data condition is preserved both in the current and next state of the `cf` table after the changes have been applied to it. Both conditions ensure that $\zeta(\theta(\mathtt{pid}), \theta(\mathtt{sid})) = 1$ and $\zeta(\theta(\mathtt{sid}), \theta(\mathtt{nid})) = 1$ to ensure that data are always being routed to switches classified at higher levels.

Figure 3 shows an example of a routing configuration that will preserve upward routing of packets (á la Formula (2)).

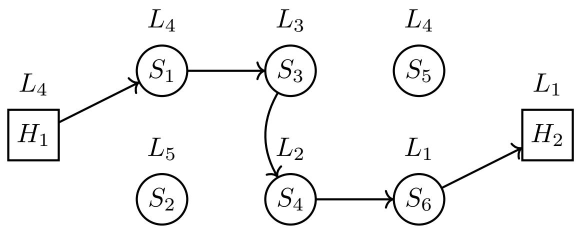

Fig. 3. Example of Routing Following Upward Flow

C. Enforcing Downward Flow of Data

The next policy we consider is the reverse of the previous one. It requires packets to travel downward only in terms of security labels. This policy is similar to the BIBA policy [6], where data are deemed to be potentially dangerous to the (integrity of the) entities consuming them. As such, by reversing the direction of ordering in Formula (2), one can obtain an integrity-preserving policy:

$$(\theta(thd((cf(f_{id}))^{-1}(i))) \ \leq_S \ \theta(snd((cf(f_{id}))^{-1}(i)))) \quad \wedge \\ (\theta(snd((cf(f_{id}))^{-1}(i))) \ \leq_S \ \theta(fst((cf(f_{id}))^{-1}(i)))) \quad (3)$$

again, where $i = 1 \ldots |cf(f_{id})|$. This formula ensures that only high-level entities can forward data to the lower-level ones. It is also possible to obtain its less restrictive variation as follows:

$$\forall n \ \in \ cf(f_{id})^{-1}(i) \quad : \quad \theta(n) \quad \leq_S \quad \theta(src_{fid})$$

Which only compares the level of the source node with a section of the network on which packets will travel, and does not require that the flow is continuously going downwards.

1) Implementation in SQL: The BIBA integrity condition is easy to implement and enforce in SQL as an RLS policy, as follows, in order to enforce Formula (3):

```
CREATE POLICY DownwardFlow ON cf FOR ALL TO
 ↪  PUBLIC USING ((select zetavalue_ij from
 ↪  mls_lattice where pid= label_i and
 ↪  sid=label_j)=0 and (select zetavalue_ij
 ↪  from mls_lattice where sid=label_i and
 ↪  nid=label_j)=0) WITH CHECK ((select
 ↪  zetavalue_ij from mls_lattice where
 ↪  pid=label_i and sid=label_j)=0 and (select
 ↪  zetavalue_ij from mls_lattice where
 ↪  sid=label_i and nid=label_j)=0);
```

where this time, $\zeta(\theta(\mathtt{pid}), \theta(\mathtt{sid})) = 0$ and $\zeta(\theta(\mathtt{sid}), \theta(\mathtt{nid})) = 0$ to ensure that data are always being routed to switches classified at lower levels. Figure 4 shows a different example of a configuration that will preserve in this case downward routing of packets.

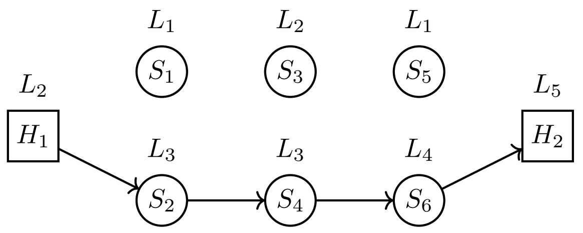

Fig. 4. Example of Routing Following Downward Flow

VI. Conclusion

Since the emerging SDN technology attracts more and more attention and applications with its benefits in flexibility and programmability, controlling the network correctly and efficiently with the new architecture is challenging. The concept of a DDN, as abstraction of an SDN, offers a great opportunity to simplify the network management. Secure flow of information in networks will continue to be a challenge to the network operators due to the increasing demands on the security of data. We presented in this paper a novel approach in specifying and directly enforcing security policies to control the routing configuration in a DDN by using row-level security checks. We showed how multilevel security policies, *upward* and *downward*, can be enforced on a DDN using the proposed approach, therefore maintaining a secure flow of data throughout.

For future work, we plan to investigate how other models of security policies, e.g. event-condition-actions, contextual and obligation-based policies [26], Clark Wilson [9] and Chinese Wall [7] can be managed and enforced within our approach in order to achieve other aspects of information flow security. Moreover, we plan to investigate how our approach can be used alongside other dataflow confidentiality and integrity mechanisms, such as cryptography, to combine different security approaches.

References

[1] Aschoff, R., Rosendo, D., Machado, M., Santos, A., Sadok, D.: A network access control solution combining orbac and sdn (2017)

[2] Ballantine, C.M., Frechette, S.M., Little, J.B.: Determinants associated to zeta matrices of posets. Linear Algebra and its Applications **411**, 364 – 370 (2005)

[3] Bari, M.F., Chowdhury, S.R., Ahmed, R., Boutaba, R.: Policycop: An autonomic qos policy enforcement framework for software defined networks (2013)

[4] Batista, B.L.A., Fernandez, M.P.: Ponderflow: A new policy specification language to sdn openflow-based networks (2014)

[5] Bell, D.E., LaPadula, L.J.: Secure computer systems: Mathematical foundations. Tech. rep., MITRE CORP BEDFORD MA (1973)

[6] Biba, K.J.: Integrity considerations for secure computer systems. Tech. rep., MITRE CORP BEDFORD MA (1977)

[7] Brewer, D.F., Nash, M.J.: The chinese wall security policy. In: Security and privacy, 1989. proceedings., 1989 ieee symposium on. pp. 206–214. IEEE (1989)

[8] Chalyy, D., Nikitin, E., Antoshina, E.J.: A simple information flow security model for software-defined networks (2015)

[9] Clark, D.D., Wilson, D.R.: A comparison of commercial and military computer security policies. In: Security and Privacy, 1987 IEEE Symposium on. pp. 184–184. IEEE (1987)

[10] Clayman, S., Clegg, R., Mamatas, L., Pavlou, G., Galis, A.: Monitoring, aggregation and filtering for efficient management of virtual networks (2011)

[11] Clegg, R.G., Clayman, S., Pavlou, G., Mamatas, L., Galis, A.: On the selection of management/monitoring nodes in highly dynamic networks. IEEE Transactions on Computers **62**(6), 1207–1220 (2013)

[12] Damianou, N., Dulay, N., Lupu, E., Sloman, M.: The ponder policy specification language. Policy **1**, 18–38 (2001)

[13] Denning, D.E.: A lattice model of secure information flow. Communications of the ACM **19**(5), 236–243 (May 1976)

[14] Denning, D.E.: On the Deriviation of Lattice Structured Information Flow Policies. Tech. Rep. CSD TR 180, Purdue University (Mar 1976)

[15] Dilworth, R.P.: Review: G. birkhoff, lattice theory. Bull. Amer. Math. Soc. **56**(2), 204–206 (03 1950)

[16] Foster, N., Harrison, R., Freedman, M.J., Monsanto, C., Rexford, J., Story, A., Walker, D.: Frenetic: A network programming language (Sep 2011). https://doi.org/10.1145/2034574.2034812, http://doi.acm.org/10.1145/2034574.2034812

[17] Galis, A., Rubio-Loyola, J., Clayman, S., Mamatas, L., Kukliński, S., Serrat, J., Zahariadis, T.: Software enabled future internet–challenges in orchestrating the future internet (2013)

[18] Glaeser, N., Wang, A.: Access control for a database-defined network (2016)

[19] PostgreSQL Global Development Group: PostgreSQL. http://www.postgresql.org (2008)

[20] Jammal, M., Singh, T., Shami, A., Asal, R., Li, Y.: Software defined networking: State of the art and research challenges. Computer Networks **72**, 74–98 (2014)

[21] Kim, H., Feamster, N.: Improving network management with software defined networking. IEEE Communications Magazine **51**(2), 114–119 (2013)

[22] Kreutz, D., Ramos, F.M., Verissimo, P.E., Rothenberg, C.E., Azodolmolky, S., Uhlig, S.: Software-defined networking: A comprehensive survey. Proceedings of the IEEE **103**(1), 14–76 (2015)

[23] Lange, S., Gebert, S., Zinner, T., Tran-Gia, P., Hock, D., Jarschel, M., Hoffmann, M.: Heuristic approaches to the controller placement problem in large scale sdn networks. IEEE Transactions on Network and Service Management **12**(1), 4–17 (2015)

[24] Lara, A., Ramamurthy, B.: Opensec: Policy-based security using software-defined networking. IEEE Transactions on Network and Service Management **13**(1), 30–42 (2016)

[25] Loo, B.T., Condie, T., Garofalakis, M., Gay, D.E., Hellerstein, J.M., Maniatis, P., Ramakrishnan, R., Roscoe, T., Stoica, I.: Declarative networking. Commun. ACM **52**(11), 87–95 (Nov 2009). https://doi.org/10.1145/1592761.1592785, http://doi.acm.org/10.1145/1592761.1592785

[26] Lupu, E.C., Sloman, M.: Conflicts in policy-based distributed systems management. IEEE Transactions on Software Engineering **25**(6), 852–869 (Nov 1999)

[27] Macauley, E., Hamilton, B., West, R., Byham, R., Guyer, C.: Row-level security (2017), https://docs.microsoft.com/en-us/sql/relational-databases/security/row-level-security

[28] McKeown, N., Anderson, T., Balakrishnan, H., Parulkar, G., Peterson, L., Rexford, J., Shenker, S., Turner, J.: Openflow: enabling innovation in campus networks. ACM SIGCOMM Computer Communication Review **38**(2), 69–74 (2008)

[29] NSA/CSS: Multi-Site Connectivity (MSC) Capability Package V1.0. Capabilities Directorate (2017), https://www.nsa.gov/resources/everyone/csfc/capability-packages/assets/files/msc-cp.pdf

[30] PostgreSQL: Row security policies (2017), https://www.postgresql.org/docs/current/static/ddl-rowsecurity.html

[31] Shin, S., Porras, P., Yegneswaran, V., Fong, M., Gu, G., Tyson, M.: Fresco: Modular composable security services for software-defined networks (2013)

[32] Wang, A.: Database criteria for network policy chain. In: Proceedings of the 2018 ACM International Workshop on Security in Software Defined Networks & Network Function Virtualization. pp. 49–54. SDN-NFV Sec'18, ACM, New York, NY, USA (2018)

[33] Wang, A., Mei, X., Croft, J., Caesar, M., Godfrey, B.: Ravel: A database-defined network. In: Proceedings of the Symposium on SDN Research. pp. 5:1–5:7. SOSR '16, ACM, New York, NY, USA (2016)

[34] Wang, A., Mei, X., Croft, J., Caesar, M., Godfrey, B.: Ravel: a database-defined network (2016), http://ravel-net.org/docs/SOSR16slide2.pdf, sOSR '16