A fast method of verifying network routing with back-trace header space analysis

Toshio Tonouchi Satoshi Yamazaki Yutaka Yakuwa Nobuyuki Tomizawa †
Knowledge Discovery Laboratories, NEC
Kanagawa, Japan

***Abstract*— It is a tough job for operators to make perfectly accurate configuration of many network elements in large networks. Erroneous configurations may cause critical incidents in network, on which many ICT systems are running. It may also result in a security hole as well as system incidents. There has been much work on preventing erroneous configurations, but this has taken a lot of time to verify routing with large networks. We propose a new method of verifying network routing. It only focuses on verifying isolation and reachability, but it can verify these properties with $O(R^2)$, where R is the number of flow entries, while the performance of an existing method of verification is $O(R^3)$. We also provide a proof of the correctness of our method.**

***Keywords*— network configuration, verification, isolation, reachability, flow table, routing**

I. INTRODUCTION

Networks need to be stable and reliable because they are the infrastructures of ICT systems. Erroneous configurations may damage the stability and reliability of the network and, as a result, those of ICT systems running on them.

We handle two properties to retain in networks. The first is *reachability* where a packet with a given packet header from a given incoming port always reaches the correct outgoing port with the correct packet header. Reachability is broken if some packets with a header from an incoming port does not reach at the outgoing port with the proper header. It may cause a communication failure.

The second property is *isolation* where a packet belonging to a network slice cannot reach another network slice. A network slice is a kind of closed network, in which hosts can communicate with one another, but no hosts in a network slice can communicate with the hosts in other network slices. This is a security risk if the isolation is not satisfied. Malicious user can illegally access a closed network slice.

There has been much research on verifying reachability and isolation in the given network [1][2][4][5][6][7][8]. The technologies in this research read configurations, such as flow tables in network elements, and checks reachability, isolation and other properties, such as *NoForwardingLoops*, *DirectPaths*, *StrictDirectPaths* and so on[14]. However, the verification time takes time when a large network is verified. For example, a method of header space analysis (HSA) [1] is adapted to the network configuration of the Stanford University campus network, which has 750,000 flow entries [1]. Their researchers reduced 750,000 flow entries to 4,200 flow entries with Optimal Routing Table Constructor (ORTC) method [3], and they used HSA to check the reachability of a flow between a pair of operator zones (OZs) within an average of 13 seconds. There were 14 OZs at Stanford University, and we estimate the time of verifying the whole Stanford campus network to take $13\frac{14!}{2!(14-2)!} = 1{,}183$ [sec]. Although the HSA is one of the fastest methods [9], its time to verify the flows with one incoming port is $O(dR^2)$ [1] , where d is the diameter of the network and R is its number of the flow entries. The verification time for the whole network is $O(tdR^2)$, where t is the number of edge ports. The number of edge ports t is proportion to the number of flow entries R , and then $O(tdR^2) \propto O(dR^3)$. As mentioned in [1], $O(dR^3)$ is the worst case, and many effective techniques have been used to implement HSA, but more efficient verification algorithms are required to verify larger network, such as those for telecommunications.

We propose a new method of fast verification that is called back-trace header space analysis (B-HSA), in this paper. The method is a variation of the HSA. Although B-HSA only focuses on the verification of only reachability and isolation, it runs faster than the HSA.

The two main contributions of this paper are:

- We proposed a fast method of verification running with $O(R^2)$, whose performance we tested and confirmed by estimating and evaluating with a prototype verification tool.
- We provide a proof that the proposed method can correctly verify reachability and isolation.

The remainder of this paper is organized as follows. Section II explains our classification of the existing methods of verifying configurations and describes their characteristics and performance. Section III explains the algorithm for the HSA, which is the base algorithm for the proposed method. Further, Section IV describes the algorithm for B-HSA. We provide a proof of B-HSA in Section V and present the results we obtained from the evaluations in Section VI.

II. RELATED WORK

There are two categories for methods of verifying routing configurations, i.e. graph-based and logic-based approaches. Graph-based approaches create a graph that represents the flows in the target network. The graph is checked as to whether it has given properties, such as reachability or the isolation. HSA [1], VeriFlow [7], and NetPlumber [8] are graph-based approaches.

† Nobuyuki Tomizawa resigned from NEC.

Logic-based approach creates Boolean expressions that represent the conditions of packet transfer defined in flow tables. A satisfiability (SAT) solver or a satisfiability modulo theorem (SMT) solver checks whether there is a solution that satisfies the Boolean expressions. FlowChecker [2], Anteater [4], Flover [5], and DNA [6] are logic-based approaches. For example, FlowChecker is an extension of ConfigChecker [13] to OpenFlow. It represents a packet header and the location of the packet as Boolean expressions, and a packet transfer as the relation between Boolean expressions before the transfer and those after the transfer. It translates these relations of Boolean values into Binary Decision Tree (BDD) expressions and it uses symbolic model checking with an SMT solver in order to detect a violation of properties given in Computation Tree Logic (CTL) expression.

Graph-based approaches are generally faster than those that logic-based approach [9]. The most well-known graph-based approach is HSA, but it still has problems with scalability, as was explained in Section I.

Some differential methods of verification have been proposed, such as VeriFlow [7] and NetPlumber [8], that are faster than HSA. They monitor update operations of configurations, and check whether these operations violate properties or not. Although differential verifications can verify configurations in real time, they have to assume that the configurations are correct at the initial time, and they are gradually updated like those in the OpenFlow network. They cannot effectively work in networks that have conventional network elements. This is because the configurations for conventional network elements are initially set once, and they are seldom changed.

III. Algorithm for HSA

In this paper, we use mathematical variables in TABLE 1.

HSA considers the attributes of a packet header as a sequence of bits. It uses a ternary logic, in which a bit can be zero, one, or **T**. **T** can take a value of either zero or one. This bit sequence is called a *header space* $h \in \mathrm{H}$. We use $\perp$ if the calculation of header spaces results in an empty set. This means that a packet does not reach any place if we encounter$\perp$ in the process of calculating the header space.

HSA represents a set of the flow tables in the target network as a *transfer function* $\phi_e: H \to H$. Transfer function ϕ_e returns a header space of the output packet from the given header space of packet matched with Flow Entry e. .

In order to define properties to be kept in this paper, we first define that a *terminal point*, $t \in (H \times P)$, is a pair of the port of the edge switch in the network and the header space of a flow at the port. We define a *network slice* as being a group of terminal points that can communicate with one another but not with terminal points in different network slices.

We define an *isolation property* as being satisfied if and only if terminal points in different network slices cannot communicate with one another. We expect that the verification of isolation shall judge whether isolation holds or not and that, in case of the violation, it shall show all header spaces violating the isolation.

We define a *reachability property* as being satisfied if and only if all terminal points in the same network slice can communicate with one another.

We show these properties with an example in Fig. 1. Terminal Point t_i and t_o belong to the same Slice S and packets between t_i and t_o should be reachable. On the other hand, Terminal Point t_i and Terminal Point t_o' in another Slice S' should not communicate. If it could, this is isolation violation.

Fig. 2 indicates how HSA verifies the network has been isolated. Firstly, HSA calculates an outgoing header space $h^{\mathrm{out}}_{p_5@S}$ at Port p_5 when a flow starting at Port p_0 in Slice S is applied to sequence of Flow Entries e_1, e_2, and e_3 as Eq. (1).

$$h^{\mathrm{out}}_{p_5@S} = \phi_{e_3} \circ \phi_{e_2} \circ \phi_{e_1}\left(h_{\mathbf{T}}\right), \tag{1}$$

where all bits in Packet Header $h_{\mathbf{T}}$ are **T**s. This is the forward phase in Fig. 2.

In backward phase in Fig. 2, Header Space $h^{\mathrm{in}}_{p_0@S}$ at Port p_0 can be calculated with Eq. (2) by using the calculation result $h^{\mathrm{out}}_{p_5@S}$ of Eq. (1).

$$h^{\mathrm{in}}_{p_0@S} = \phi^{-1}_{e_1} \circ \phi^{-1}_{e_2} \circ \phi^{-1}_{p_3}\left(h^{\mathrm{out}}_{p_5@S}\right), \tag{2}$$

where $\phi^{-1}_e: H \to H$ is the inverse function of ϕ_e. In other word, ϕ^{-1}_e is defined as: $\phi^{-1}_e(h) \stackrel{\text{def}}{=} \{h_0 | \phi(h_0) = h\}$.

Assume that an operator defines that Terminal Port $t_{p_0} = \left(h^{\mathrm{in}}_{p_0@S}, p_0\right)$ and Terminal Port $t_{p_5} = \left(h^{\mathrm{out}}_{p_5@S}, p_5\right)$ are in Slice S. HSA detects the isolation violation in S if Header Space h_{p_5} does not include Header Space $\phi_{e_3} \circ \phi_{e_2} \circ \phi_{e_1}\left(h^{\mathrm{in}}_{p_0@S}\right)$. In short, $h^{\mathrm{out}}_{p_5@S} \not\supset \phi_{e_3} \circ \phi_{e_2} \circ \phi_{e_1}\left(h^{\mathrm{in}}_{p_0@S}\right)$.

Consider the case of the isolation violation, $h^{\mathrm{out}}_{p_5@S} \not\supset \phi_{e_3} \circ \phi_{e_2} \circ \phi_{e_1}\left(h^{\mathrm{in}}_{p_0@S}\right)$. Here, $h^{\mathrm{out}}_{p_5@S} - \phi_{e_3} \circ \phi_{e_2} \circ \phi_{e_1}\left(h^{\mathrm{in}}_{p_0@S}\right)$ is not an empty set. We define this header space as $h^{\mathrm{out}}_{p_5@\bar{S}} \stackrel{\text{def}}{=} h^{\mathrm{out}}_{p_5@S} - \phi_{e_3} \circ \phi_{e_2} \circ \phi_{e_1}\left(h^{\mathrm{in}}_{p_0@S}\right)$. In short, Header Space $h^{\mathrm{out}}_{p_5@\bar{S}}$ is the remaining set of the possible header space at Port p_5 of the flow initiating from Port p_0 with Header Space h_{p_0} in Network Slice *S*. We then calculate the header space, at Port p_0, of the flow terminating with $h^{out}_{p_5@\bar{S}}$ at Port p_0. We define this header space as $h^{\mathrm{in}}_{p_1@\bar{S}}$, and calculate it with Eq. (3).

$$h^{\mathrm{in}}_{p_0@\bar{S}} = \phi^{-1}_{p_1} \circ \phi^{-1}_{p_2} \circ \phi^{-1}_{p_{4\_1}} \circ \phi^{-1}_{p_5}\left(h^{\mathrm{out}}_{p_5@\bar{S}}\right), \tag{3}$$

Notice that HSA can detect isolation violation in the forward phase, but it needs the backward phase if the header space with which a packet violates the isolation.

HSA has two main advantages. First, it can calculate all the header space $h^{\mathrm{in}}_{p_0@\bar{S}}$ that violates isolation. A ping test, which is usually used in network test, cannot check all the header space because it can only carry out the tests by using sample data. Second, HSA can be used in networks with many kinds of network elements (e.g. switches, routers, and firewalls). This is because the routing configurations of many kinds of network elements can be modeled as a flow table.

TABLE I. VARIABLES

Variable	explanation
$p \in P$	Physical port of switches. It is identical in the network.
S	Network Slice
$h^{\text{in}}_{p@S} \in H$ $h^{\text{out}}_{p@S} \in H$	Header space of an incoming packet $h^{in}_{p@S}$ in Slice S at Port p and that of outgoing packet ($h^{out}_{p@S}$).
$\phi_e: P \to P$	HSA transfer function which calculates the output header space when the input header space is given and is applied to Flow Entry e.
$\phi_e^{-1}: H \to H$	Inverse function of ϕ_e. It calculates a header space at an input port $e.p_{in}$ when the header space of the output packet is given, and it is the applied result of Flow Entry e.
$\zeta_e: H \to H$	B-HSA backward transfer function which calculates a header space at Input Port $e.p_{in}$ when the header space of the output packet is given, and it is the applied result of Flow Entry e.

However, HSA has one major disadvantage in that it lacks scalability as was mentioned in Section I.

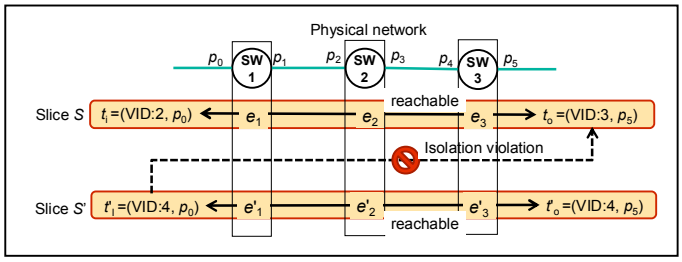

Fig. 1. Isolation Property and Rechability Property

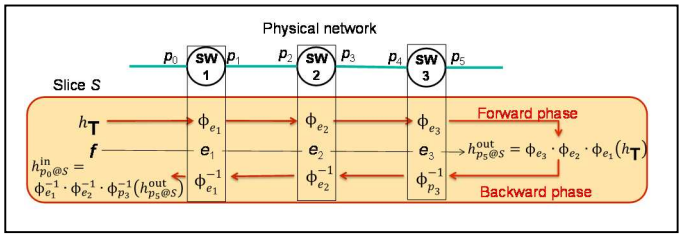

Fig. 2. Header Space calculation with HSA

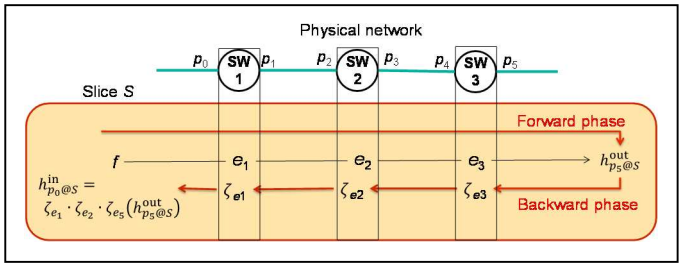

Fig. 3. Header Space calculation with B-HAS

IV. BACK-TRACE HEADER SPACE ANALYSIS

A. Overview

We propose a back-trace header space analysis (B-HSA), which can be applied to large scale networks. Fig. 3 has an overview of B-HSA. Consider that Flow f is being initiated from Port p_1. B-HSA traces a path for Flow f from Port p_1, in the forward phase, by referring to the incoming port and the outgoing port of flow entries. For example, the thin blue lines in Fig. 2 indicates a path (SW1→SW2→SW3), and Flow f finally reaches Port p_5. B-HSA remembers the sequence of Flow Entries e_1, e_2, and e_3 that are used in Flow f. Notice that B-HSA only refers to the incoming port and the outgoing port of the flow entries in this phase, but it does not perceive the matching rule or actions of the flow entries. While HSA calculates a header space in this forward phase, B-HSA does not.

Second, in the backward phase, B-HSA backwardly traces the path that is found in the forward phase. B-HSA in Fig. 3 calculates header spaces by referring to flow entries in order of e_3, e_2, and e_1. Assume that an operator defines that Terminal Port $t_{p_0} = \left(h^{\text{in}}_{p_0@S}, p_0\right)$ and Terminal Port $t_{p_5} = \left(h^{\text{out}}_{p_5@S}, p_5\right)$ are in Slice S. B-HSA uses a back-trace function $\zeta_e: H \to H$, in this calculation, which is defined in Subsection IV.D, and it calculates header space $h'^{\text{in}}_{p_0@S}$ at Port p_0 of Slice S with Eq. (4).

$$h'^{\text{in}}_{p_0@S} = \zeta_{e_1} \circ \zeta_{e_2} \circ \zeta_{e_5}\left(h^{\text{out}}_{p_5@S}\right) \tag{4}$$

B-HSA can detect an isolation violation if $h^{\text{in}}_{p_0@S}$ calculated from $h^{\text{out}}_{p_5@S}$ has an intersection of the complement set of $h^{\text{in}}_{p_0@S}$. In another word B-HSA checks if $\left(h'^{\text{in}}_{p_0@S} - h^{\text{in}}_{p_0@S}\right) \neq \phi$. An incoming packet from Port p_0 with a header in Header Space $\left(h'^{\text{in}}_{p_0@S} - h^{\text{in}}_{p_0@S}\right)$ violates the isolation of Slice S.

B. Differences between HSA and B-HSA

HSA mainly calculates header spaces in the forward phase, but B-HSA only calculates them in the backward phase. This is why B-HSA is faster than HSA. B-HSA can re-use the calculated results that have been calculated along the way of other back-trace paths. See Fig. 4. Consider that B-HSA calculates Header Space $h^{\text{in}}_{p_1@S}$ in Flow f_a initiating from Port p_1 with Eq. (5).

$$h^{\text{in}}_{p_1@S} = \zeta_{e_{1\to3}} \circ \zeta_{e'} \circ \cdots \left(h^{\text{out}}_{p_e@S}\right) \tag{5}$$

where $h^{\text{out}}_{p_e@S}$ is the outgoing header space at the terminal port p_e of Flow f in Fig. 4. In other word, B-HSA also calculates Header Space $h^{\text{in}}_{p_2@S}$ Flow f_b initiating from Port p_2 with Eq. (6).

$$h^{\text{in}}_{p_2@S} = \zeta_{e_{2\to3}} \circ \zeta_{e'} \circ \cdots \left(h^{\text{out}}_{p_e@S}\right) \tag{6}$$

In this case, B-HSA can re-use the calculated results $\zeta_{e'} \circ \cdots \left(h^{\text{out}}_{p_e@S}\right)$ in Eq. (5) for the calculation of $h^{\text{in}}_{p_2@S}$ in Eq. (6).

HSA cannot re-use the calculated results in the backward phase. See Fig. 5. HSA calculates the header space Port p_1 of Flow f_a with Eq. (7).

$$h^{\text{in}}_{p_1 \to p_e} = \phi^{-1}_{e_{1\to3}} \circ \phi^{-1}_{p_{e'}} \circ\cdots\circ \phi_{e_{1\to3}} \circ\cdots (h_T) \qquad (7)$$

It calculates the header space at Port p_1 of Flow f_b with Eq. (8).

$$h^{\text{in}}_{p_2@S} = \phi^{-1}_{e_{2\to3}} \circ \phi^{-1}_{p_{e'}} \circ\cdots\circ \phi_{e_{2\to3}} \circ\cdots (h_T) \qquad (8)$$

Notice that Sub-equation $\phi^{-1}_{p_{e'}} \circ\cdots\circ \phi_{e_{1\to3}} \circ\cdots (h_T)$ in Eq. (7) and $\phi^{-1}_{p_{e'}} \circ\cdots\circ \phi_{e_{2\to3}} \circ\cdots (h_T)$ in Eq. (8) are not the same. This is the reason why HSA cannot reuse the calculation results in backward phase while B-HSA can.

HSA has a similar technique of optimization, called "Bookmarking Applied Transfer Function Rules" [1]. It memorizes a transfer function used in a process of the forward phase, and refers to it in the corresponding process in the backward phase. However, it only skips the recalculation of a transfer function, which is used in calculating its inverse function. B-HSA can re-use the result values calculated with the back-trace functions.

However, B-HSA has disadvantages: It cannot detect a loop route, which is a critical failure in the network. This is because a loop route defined in packet forwarding with flow entries, and the backward trace cannot detect it.

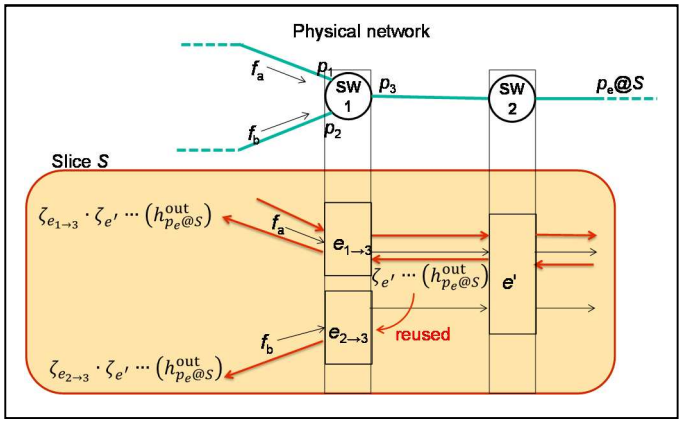

Fig. 4. B-HSA can re-use $\zeta_{e'} \circ\cdots (h_{p_e@S_1})$

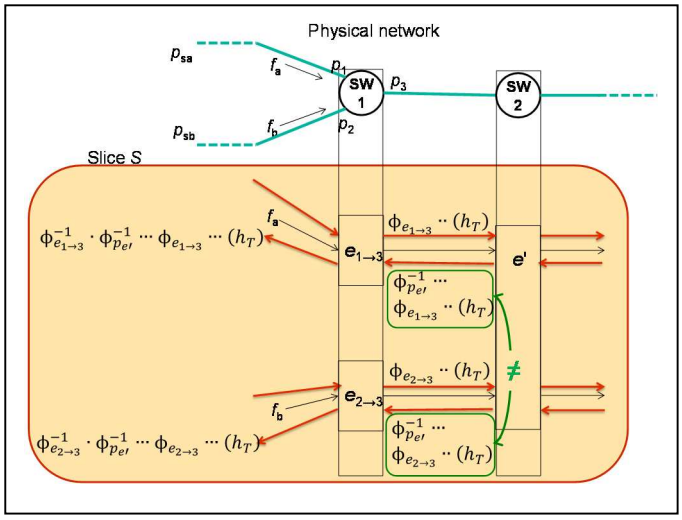

Fig. 5. HSA cannot reuse the calculation result in another flow.

Another disadvantage is that B-HSA does not calculate the header space at outgoing ports, while HSA does. Some operators want to know the header space at the outgoing port. In that case, he/she has to apply the HSA transfer function after the B-HAS calculates the header space at the incoming port.

The operator should choose HSA or B-HSA properly depending on the situation.

C. *Detailed algorithm: assumption*

We will first describe assumptions before providing details on the algorithm for B-HSA.

We assume a packet has k attributes. An attribute value is a direct value (e.g. an integer value, an IP address, …). A flow table installed in a network element has flow entries. We assume that a flow entry is a form of OpenFlow v1.0 because it is a general form of a flow entry in many kinds of network elements. Flow Entry e consists of a quadruplet, i.e. incoming port $e.p_{in} \in P$, outgoing port $e.p_{out} \in P$, matching rule $e.\mathrm{m} \in M$, and action $e.\mathrm{a} \in A$.

A matching rule has k attributes that correspond to the attributes of a packet. An attribute value is a direct value, a value space (e.g. an IP subnet address, a differential set value defined after, …), or **T**, which is matched with any attribute value in a packet.

An action has also k attributes. An attribute value in an action is a direct value, or **T**.

A direct value v′ of a packet is matched with a direct value v of a matching rule if $\mathrm{v}' = \mathrm{v}$ A direct value v′ of a packet is matched with a value space p if $\mathrm{v}' \in \mathrm{p}$.

A packet coming from Incoming Port $e.p_{in}$ of a network element is checked whether it is matched with Flow Entry e in the flow table installed in the network element. The packet is matched with a flow entry if every attribute of the packet is matched with the corresponding attribute of Flow Entry e.

If the network element finds Flow Entry e with which the incoming packet is match, the matched packet is applied to the action $e.\mathrm{a}$ of the flow entry. If an attribute value of the action is **T**, the corresponding attribute value of the packet is not changed. If an attribute value of the action is a direct value v", the corresponding attribute of the packet is changed to v".

After that, the packet leaves Outgoing Port $e.p_{out} \in P$ of the network element.

Notice that a packet and a matching rule have k attributes, and a network element independently matches each attribute of the packet and that of the matching rule. We can therefore consider an application of a matching rule or an action to a packet header as being as independent application of an attribute of the matching rule or the action to the corresponding attribute of the packet, without loss of generality.

A flow table in a network element has plural flow entries matched with Incoming Port p_{in}. We assume each entry has a totally ordered priority. A packet coming from Incoming Port p_{in} is matched with the flow entry with the highest priority in flow entries that can be matched with the packet.

Consider that a packet coming from Incoming Port p_{in} and several flow entries e_i $(i = 0, \ldots, g-1)$ where $e_i.p_{in} = p_{in}$, they are in descending order of priority, and they are not matched with the packet. Consider also that Flow Entry e_g in which $e_g.p_{in} = p_{in}$ is matched with the packet. In this case, we can regard e_g as $\left(e_g - \cup_{i=0,\ldots,g-1} e_i\right)$. For every attribute in Flow Entry e_g, we can obtain "$\{\mathrm{v}_g\} - \cup_{i=0,\ldots,g-1}\{v_i\}$". It is called a *differential set value*, which was referred to previously. If $\{\mathrm{v}_g\} - \cup_{i=0,\ldots,g-1}\{v_i\} = \emptyset$, then the network element gives up the transfer of the packet. The value is represented as $\perp$ in this case.

D. Detailed algorithm

We define Function $b_t(e_0, \cdots, e_{n-1}, h_{out})$ as the header space at Incoming Port $e_0.p_{in}$ of the flow leaving Port $e_{n-1}.p_{out}$ with Header Space h_{out} by it being applied with a sequence of Flow Entries $e_0, \cdots, e_{n-1}$. It is defined in Eq. (9).

$$b_t(e_0, \cdots, e_{n-1}, h_{out}) \stackrel{\text{def}}{=} \zeta_{\mathrm{e}_0} \circ \ldots \circ \zeta_{\mathrm{e}_{n-1}}(h_{out}) \quad (9)$$

Then, using $b_t(e_0, \cdots, e_{n-1}, h_{out})$, we define $b_{ts}(p_{in}, p_{out}, h_{out})$, which calculates the header space at Port p_{in} of the flows to Port p_{out} with Header Space h_{out}. This is defined as:

$$b_{ts}(p_{in}, p_{out}, h_{out}) \stackrel{\text{def}}{=} \bigcup \left\{ b_t(e_0, \cdots, e_{n-1}, h_{out}) \;\middle|\; \begin{array}{c} e_0, \cdots, e_{n-1} \\ \text{are matced} \\ \text{entries in a flow} \\ \text{from } p_{in} \text{ to } p_{out}. \end{array} \right\} \quad (10)$$

Equation (10) means the union of the header spaces of $b_t(e_0, \cdots, e_{n-1}, h_{out})$ of all paths from Ports p_{in} to p_{out}.

We define $\zeta_e(h)$ as the composition of two functions: $\mathrm{m}^{\#}$ and $\mathrm{a}^{\#}$:

$$\zeta_e(h) \stackrel{\text{def}}{=} \mathrm{m}^{\#}(e.\mathrm{m}) \circ \mathrm{a}^{\#}(e.\mathrm{a})(h) \quad (11)$$

First, we define $\mathrm{m}^{\#}: M \to (H \to H)$. Function $\mathrm{m}^{\#}(e.\mathrm{m})(h)$ calculates a header space that is applied to Matching Rule $e.\mathrm{m} \in M$ and results in Header Space h.

TABLE II. summarizes attributes of the header space of packets that are the calculated results of $\mathrm{m}^{\#}(e.\mathrm{m})(h)$. Line 1 in the table indicates an attribute of the header space h of a given incoming packet. Column 1 in the table indicates an attribute of the matching rule of Flow Entry e.

Second, we define $\mathrm{a}^{\#}: A \to (H \to H)$. Function $\mathrm{a}^{\#}(e.\mathrm{a})(h)$ calculates the header space that is applied to Action $e.\mathrm{a} \in A$ and results in Header Space h.

TABLE III. summarizes the attributes of the header space of incoming packets that are the calculated results of $\mathrm{a}^{\#}(e.\mathrm{a})(h)$. Column 1 in the table shows an attribute of the action of Flow Entry e. An attribute of the action is a direct value (i.e. v), or **T**, and it does not take a value space, while that of the matching rule in TABLE II. takes a value space as well as a direct value and **T**.

Notice that the return values of $\zeta_e(h)$ in B-HSA are the same with the return values of $\phi_{\mathrm{e}}^{-1}(h)$ if h is in the domain of Function ϕ_{e}^{-1}. However, the domain of Function ζ_e is wider than that of Function ϕ_{e}^{-1}. For example, consider that the value of an attribute in the matching fields in Flow Entry e is $e.\mathrm{m} = \mathrm{v}$ and the value of the attribute is $e.\mathrm{a} = \mathbf{T}$. Function $\phi_{\mathrm{e}}^{-1}(\mathbf{T}) = \perp$ while $\zeta_e(\mathbf{T}) = \mathrm{m}^{\#}(e.\mathrm{m}) \circ \mathrm{a}^{\#}(e.\mathrm{a})(\mathbf{T}) = \mathrm{v}$. Then, $\phi_{\mathrm{e}}^{-1}(\mathbf{T}) \neq \zeta_e(\mathbf{T})$.

We will explain how isolation and reachability are verified by using these functions.

A network operator assumes that Slice *S* has both Terminal Point $t_{p_{in}@S}$ with Header Space $h^{\mathrm{in}}_{p_{in}@S}$ at Port p_{in} and Terminal Point $t_{p_{out}@S}$ with Header Space $h^{\mathrm{out}}_{p_{out}@S}$ at Port p_{out}.

Isolation is satisfied if $h^{\mathrm{in}}_{p_{in}@S} \supseteq b_{ts}\left(p_{in}, p_{out}, h^{\mathrm{out}}_{p_{out}@S}\right)$. This is because all flows arriving at Terminal Point $\mathrm{t}_{p_{out}@S}$ leaves from Terminal Point $t_{p_{in}@S}$, and all flows from terminal points other than $t_{p_{in}@S}$ do not arrive at Terminal Point $t_{p_{out}@S}$.

Reachability is satisfied if $h^{\mathrm{in}}_{p_{in}@S} \subseteq b_{ts}\left(p_{in}, p_{out}, h_{p_{out}@S}\right)$. This is because all flows from Terminal Point $t_{p_{in}@S}$ arrive at Terminal Point $t_{p_{out}@S}$.

TABLE II. DEFINITION OF MATCHING RULE OF BACK-TRACE FUNCTION: $\mathbf{m}^{\#}$

$\mathbf{m}^{\#}(\boldsymbol{e}.\mathbf{m})(\boldsymbol{h})$	$\boldsymbol{h} = \mathbf{v}'$	$\boldsymbol{h} = \mathbf{p}'$	$\boldsymbol{h} = \mathbf{T}$
$\boldsymbol{e}.\mathbf{m} = \mathbf{v}$	v if v=v', $\perp$ if v $\neq$ v'	v if v $\in$ p', $\perp$ if v $\notin$ p'	V
$\boldsymbol{e}.\mathbf{m} = \mathbf{p}$	v' if v' $\in$ p, $\perp$ if v' $\notin$ p	p' if p' $\subset$ p, p if p $\subset$ p', $\perp$ if p $\cap$ p' $= \emptyset$	P
$\boldsymbol{e}.\mathbf{m} = \mathbf{T}$	v'	p'	**T**

TABLE III. DEFINITION OF ACTION PART OF BACK-TRACE FUNCTION: $\mathbf{a}^{\#}$

$\mathbf{a}^{\#}(\boldsymbol{e}.\mathbf{a})(\boldsymbol{h})$	**h=v'**	**h=p'**	**h=T**
$\boldsymbol{e}.\mathbf{m} = \mathbf{v}$	**T** if v=v', $\perp$ if v $\neq$ v'	**T** if v $\in$ p', $\perp$ if v $\notin$ p'	**T**
$\boldsymbol{e}.\mathbf{m} = \mathbf{p}$	v'	p'	**T**

E. Estimation of Performance

B-HSA can re-use the calculated results that were calculated on the way of other back-trace paths. Therefore, $\zeta_e(h)$ is calculated once for every port. We can estimate that the verification time for the whole network is $\mathrm{O}(pR)$. The number of edge ports p is proportional to the number of flow entries R, and then $\mathrm{O}(pR) = \mathrm{O}(R^2)$. This means that B-HSA is more scalable than HSA, in which the verification time is $\mathrm{O}(dR^3)$ as was explained in Section I. We will confirm this estimation through the evaluation in Section VI.

V. PROOF OF THE PROPOSED ALGORITHM

A. Propositions to be proved

We will describe the propositions to be proved before we give the proof of the correctness of the verification in B-HSA. We define $f_{ts}(p_{in}, p_{out}, h_{in})$, which returns the header space at Port p_{out} of the flow that leaves Port p_{in} with Header Space

h_{in}. Function $\mathrm{f}_{ts}(p_{in}, p_{out}, h_{in})$ is defined, similarly to Eq. (12).

$$f_{ts}(p_{in}, p_{out}, h_{in}) \stackrel{\text{def}}{\equiv} \bigcup \left\{ f_t(e_0, \cdots, e_{n-1}, h_{in}) \;\middle|\; \begin{array}{c} e_0, \cdots, e_{n-1} \\ \text{are matced} \\ \text{entries in a flow} \\ \text{from } p_{in} \text{ to } p_{out}. \end{array} \right\} \quad (12)$$

where $f_t(e_0, \cdots, e_{n-1}, h_{in}) \stackrel{\text{def}}{\equiv} \phi_{e_{n-1}} \circ \ldots \circ \phi_{e_0}(h_{in})$.

TABLE IV summarizes the attributes of header space of outgoing packets that are the calculation result of $\phi_e(h)$. Line 1 in the table indicates an attribute of header space h of a given incoming packet. Column 1 in the table indicates an attribute of the action of Flow Entry e, and Column 2 indicates an attribute of the matching rule. Variables v, v_0, and v" represent direct values. Variables p and p_0 represent value spaces. Value $\perp$ means that not answer has been derived, and we cannot longer continue calculation if one of the attributes results in $\perp$.

By using $f_{ts}(p_{in}, p_{out}, h_{in})$, we define a proposition for isolation and that for reachability.

- **(Proposition 1: isolation)**: A flow with Header Space $b_{ts}(p_{in}, p_{out}, h_{out})$ at Port p_{in} goes to Port p_{out}, and the header space of the flow at Port p_{out} includes h_{out}. In short, $f_{ts}\big(p_{in}, p_{out}, b_{ts}(p_{in}, p_{out}, h_{out})\big) \supseteq h_{out}$

Isolation is proved with Proposition 1 if $f_{ts}\big(p_{in}, p_{out}, b_{ts}(p_{in}, p_{out}, h_{out})\big) \supseteq h_{out}$. That means that the flow that does not belong to h_{out} does not belong to $b_{ts}(p_{in}, p_{out}, h_{out})$, neither. Therefore, we can prove isolation if we can prove Proposition 1.

- **(Proposition 2: reachability)**: A flow with Header Space $b_{ts}(p_{in}, p_{out}, h_{out})$ at Port p_{in} goes to Port p_{out}, and the header space of the flow at Port p_{out} included in h_{out}. In short, $f_{ts}\big(p_{in}, p_{out}, b_{ts}(p_{in}, p_{out}, h_{out})\big) \subseteq h_{out}$.

Reachability is proved with Proposition 2 if $f_{ts}\big(p_{in}, p_{out}, b_{ts}(p_{in}, p_{out}, h_{out})\big) \subseteq h_{out}$. It means that a flow with Header Space h_{in} at Port p_{in} always arrives at Port p_{out} with h_{out}. We can therefor prove the reachability if we can prove Proposition 2.

To prove both Proposition 1 and 2, we will just prove $f_{ts}\big(p_{in}, p_{out}, b_{ts}(p_{in}, p_{out}, h_{out})\big) = h_{out}$.

Function $\zeta_e(h)$, which is a component of $b_{ts}(p_{in}, p_{out}, h_{out})$, is defined as $\zeta_e(h) = \mathrm{m}^{\#}(e.\mathrm{m}) \circ \mathrm{a}^{\#}(e.\mathrm{a})(h)$. We can therefore calculate $\zeta_e(h)$ by using TABLE II. for Function$\mathrm{m}^{\#}$ and TABLE III for Function $\mathrm{a}^{\#}$. TABLE V. summarizes the results of $\zeta_e(h)$ calculated from TABLE II and TABLE III..

Similarly with $b_{ts}(p_{in}, p_{out}, h_{out})$, Function $f_{ts}(p_{in}, p_{out}, \mathrm{h}_{in})$ consists of ϕ_e. Function ϕ_e is defined in TABLE IV. We can prove $f_{ts}\big(p_{in}, p_{out}, b_{ts}(p_{in}, p_{out}, h_{out})\big) = h_{out}$ if the results from applying TABLE IV to TABLE V are identical to the input header spaces in TABLE IV.

TABLE IV. PACKET MATCHING RULE WITH FLOW ENTRY: ϕ_e

entry / Action	packet / Matching	$\mathbf{v_0}$	$\mathbf{p_0}$	**T**
***e*.a=v"**	***e*.m=v**	v" if $\mathrm{v} = \mathrm{v}_0$, $\perp$ if $\mathrm{v} \neq \mathrm{v}_0$	v" if $\mathrm{v}'' \in \mathrm{p}_0$, $\perp$ if $\mathrm{v}'' \notin \mathrm{p}_0$	v"
	***e*.m=p**	v" if $\mathrm{v}_0 \in \mathrm{p}$, $\perp$ if $\mathrm{v}_0 \notin \mathrm{p}$	v" if $\mathrm{p}_0 \in \mathrm{p}$ $\perp$ if $\mathrm{p}_0 \notin \mathrm{p}$	v"
	***e*.m=T**	v''	v''	v"
***e*.a=T**	***e*.m=v**	v if $\mathrm{v} = \mathrm{v}_0$, $\perp$ if $\mathrm{v} \neq \mathrm{v}_0$	v if $\mathrm{v} \in \mathrm{p}_0$, $\perp$ if $\mathrm{v} \notin \mathrm{p}_0$	v
	***e*.m=p**	v_0 if $\mathrm{v}_0 \in \mathrm{p}$, $\perp$ if $\mathrm{v} \neq \mathrm{v}_0$	p_0 if $\mathrm{p}_0 \subset \mathrm{p}$, p if $\mathrm{p} \subset \mathrm{p}_0$, $\perp$ if $\mathrm{p} \cap \mathrm{p}' = \emptyset$	p
	***e*.m=T**	v_0	p_0	**T**

TABLE V. DEFINITION OF BACK-TRACE FUNCTION ζ_e

Entry / Action	Packet / Matching	$\mathbf{v}'$	$\mathbf{p}'$	**T**
***e*.a=v"**	***e*.m=v**	v if v''=v', $\perp$ if $\mathrm{v}'' \neq \mathrm{v}'$	v if $\mathrm{v}'' \in \mathrm{p}'$, $\perp$ if $\mathrm{v}'' \notin \mathrm{p}'$	**T**
	***e*.m=p**	p if v''=v', $\perp$ if $\mathrm{v}'' \neq \mathrm{v}'$	p if $\mathrm{v}'' \in \mathrm{p}'$, $\perp$ if $\mathrm{v}'' \notin \mathrm{p}'$	**T**
	***e*.m=T**	v'' if v''=v', $\perp$ if $\mathrm{v}'' \neq \mathrm{v}'$	v'' if $\mathrm{v}'' \in \mathrm{p}'$, $\perp$ if $\mathrm{v}'' \notin \mathrm{p}'$	**T**
***e*.a=T**	***e*.m=v**	v if v=v', $\perp$ if $\mathrm{v} \neq \mathrm{v}'$	v if $\mathrm{v} \in \mathrm{p}'$, $\perp$ if $\mathrm{v} \notin \mathrm{p}'$	v
	***e*.m=p**	v' if $\mathrm{v}' \in \mathrm{p}$, $\perp$ if $\mathrm{v}' \notin \mathrm{p}$	p' if $\mathrm{p}' \subset \mathrm{p}$, p if $\mathrm{p} \subset \mathrm{p}'$, $\perp$ if $\mathrm{p} \cap \mathrm{p}' = \emptyset$	p
	***e*.m=T**	v'	p'	**T**

B. Proof

We present the calculated results, in TABLE VI, from applying TABLE IV to TABLE V, which mean $\mathrm{f}_{ts}\big(p_{in}, p_{out}, b_{ts}(p_{in}, p_{out}, h_{out})\big)$. Column 1 in TABLE IV, indicates an attribute of the action of given Flow Entry e. Column 2 indicates that of the matching rule. Column 3 indicates the header space at Outgoing Port $e.\mathrm{p}_{out}$. Column 4 is the condition under which the calculation of each line in TABLE IV is executed. Column 5 has the calculated results, which are the header spaces at the incoming port, for Back-trace Function $\zeta_e(h)$. The last column is the header space at the outgoing port. This is calculated with Transfer function ϕ_e applied to the header space at the incoming port. The header spaces in Column 6 is $\perp$, or is equals to the header space in Column 3. We can ignore the case of $\perp$, because there are no flows in this case. Therefore, $\phi_e\big(\zeta_e(h)\big) = h$ holds. It is obvious that $\mathrm{f}_{ts}\big(p_{in}, p_{out}, b_{ts}(p_{in}, p_{out}, h_{out})\big) = h_{out}$ holds because $\phi_e\big(\zeta_e(h)\big) = h$ and the definitions of f_{ts} and b_{ts}. ■

TABLE VI. PROOF OF CORRECTNESS

Entry Action	packet Matching	c_{out}	Cond.	$\zeta_e(h)$	$\varphi_e(\zeta_e(h))$
***e*.a=v"**	***e*.m=v**	v′	if $v' = v"$	v	v′
			if $v' \neq v"$	$\perp$	$\perp$
	***e*.m=p**	v′	if $v' = v"$	p	v′
			if $v' \neq v"$	$\perp$	$\perp$
	***e*.m=T**	v′	if $v' = v"$	v"	v"
			if $v' \neq v"$	$\perp$	$\perp$
***e*.a=T**	***e*.m=v**	v′	if $v' = v$	v	v′
			if $v' \neq v$	$\perp$	$\perp$
	***e*.m=p**	v′	if $v' \in p$	v′	v′
			if $v' \notin p$	$\perp$	$\perp$
	***e*.m=T**	v′		v′	v′

VI. EVALUATIONS WITH PROTOTYPE

We implemented a prototype of the B-HSA algorithm, which calculates all the pairs of the header space at incoming ports and that at outgoing ports. The B-HSA algorithm is written in Java, and peripheral modules to network elements are written in Ruby. We used the SMT solver Simple Theorem Prover (STP) [10], which checks whether the differential set value explained in Section IV is empty or not. We also implemented the HSA algorithm discussed in Section III in Java, to compare these two algorithms. The Hassel [1] is an implementation of the HSA algorithm and it is optimized with the implementation techniques mentioned in [1]. Hassel is, therefore, expected to be faster than our HSA implementation. We implemented the HSA algorithm by modifying the B-HSA prototype instead of Hassel because we only wanted to compare purely the algorithms: We exclude the differences caused in the implementation techniques: We just replaced the back-trace function ζ_e in our B-HSA prototype with the transfer function ϕ and its inverse function ϕ^{-1}, and implemented a prototype of the HSA algorithm.

A. Feasibility test

We intentionally created an erroneous configuration in the sample network and checked whether the prototype tool could detect it. Fig. 6 outlines the sample network we developed with OpenFlow controller Trema [12] and Open vSwitches. It is a two-layered network having a root L3 switch and two L2 switches under the L3 switch. We created a configuration of two VLANs, which were separated from each other. Each edge port of the L2 switches was assigned a VLAN ID, and, in short, each port belonged to a VLAN.

First, we set flow entries in the L2 switches such that these two VLANs are separated. The B-HSA prototype in this normal case calculated the header spaces at incoming ports for the given outgoing ports. We used Weka [11] to display the calculation results in Fig. 6. The x-axis shows VLAN IDs and the y-axis shows the outgoing ports. Fig. 7 shows that the flows with different VLAN IDs are clearly separated.

Second, we intentionally created an erroneous configuration in the sample network. We removed the matching rule for VLAN IDs from some flow tables. Fig. 8 presents the results obtained from verifying the network with the erroneous configuration. It demonstrates that the flow with any VLAN ID can enter any network slices. This means that isolation is violated. We found through this evaluation that the prototype tool could correctly detect violations.

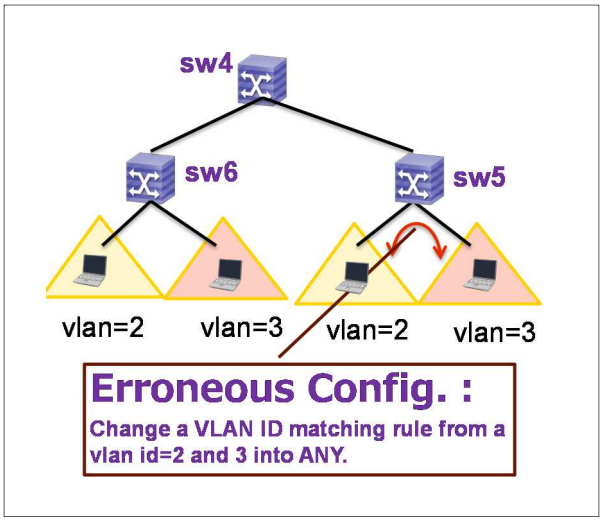

Fig. 6. Feasibility evaluation environment

B. Evaluation of Performance

We evaluated the performances of the B-HSA and HSA algorithms on the machine (Xeon X5690 3.46GHz 2CPUs, 90GB). Fig. 10 outlines the evaluation environment. We have N L3 switches, in the top level, which make a loop path. We have M L2 switches in the second level under each L3 switch. We have P L2 switches in the third level under each L2 switch in the second level. Each port of the L2 switch in the third level is assigned to a VLAN and it is connected to a host. We have L VLANs in the network, and, therefore, each L2 switch in the third level has $(KL + 1)$ ports including an uplink port.

Network slices are defined in the L2 switches in the second level in this environment. In other words, we have (NM) L2 switches in the second level, where each port of the L2 switches in the second level is assigned to L VLANs, and, therefore, we have totally (NML) VLANs (i.e. (NML) network slices) in this network.

We measured the time to verify isolation under the environment while changing the values of $N, M, K, L,$ and P. For the verification, the prototype tool reads the flow tables stored in the L3 switches in the first level and those in the L2 switches in the second level.

Fig. 11 plots the results obtained from the evaluations. The x-axis plots the total number of flow entries in the network, and the y-axis plots the verification time in seconds. B-HSA is much faster than HSA. There is an $O(R^3)$ approximate curve of HSA. The R squared value (R^2 in the graph) is about 0.95. That is very similar to the estimation of HSA in Subsection IV.E.

B-HSA can verify the network with more than 10,000 flow entries. Fig. 11 plots the verification time of B-HSA including the area more than 10,000 flow entries. It plots an $O(R^2)$

approximate curve. The R square value is about 0.97. That is very similar to the estimation of B-HSA in Subsection IV.E.

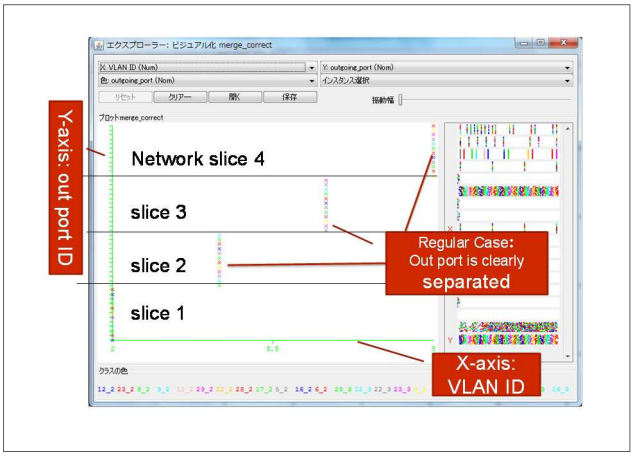

Fig. 7. Verification Result (normal case)

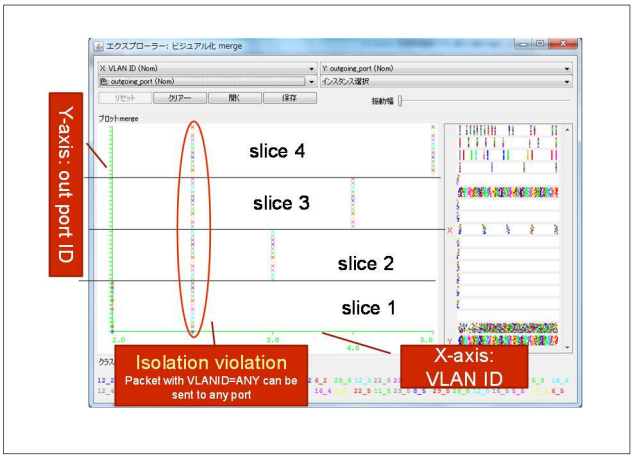

Fig. 8. Verification Result (isolation violation)

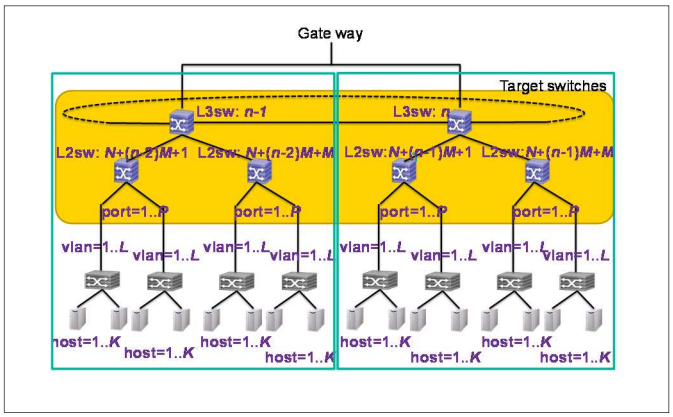

Fig. 9. Performance evaluation environment

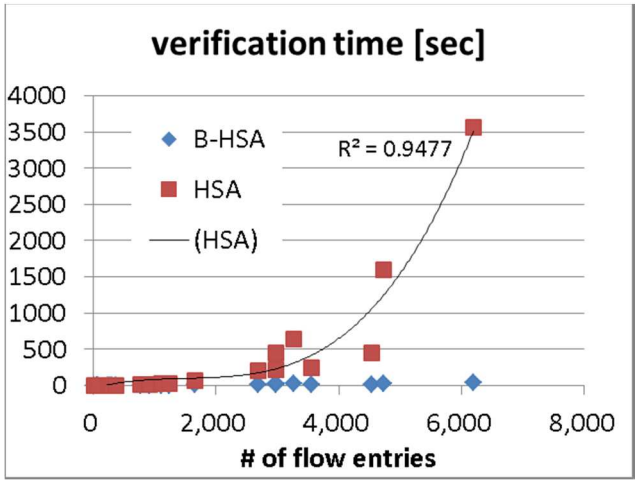

Fig. 10. Performance of verifications of B-HSA and HSA

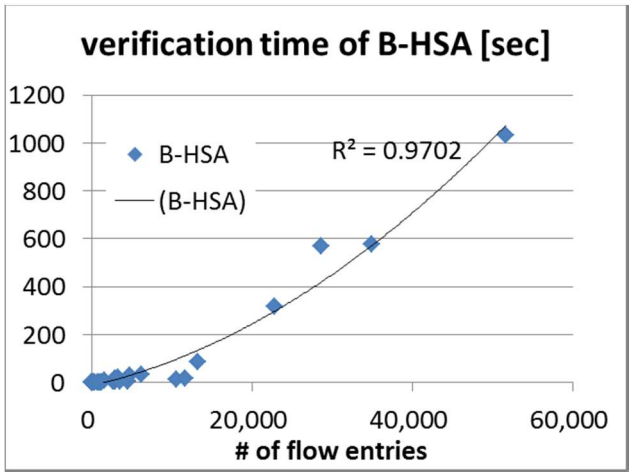

Fig. 11. Performance of verifications of B-HSA

VII. CONCLUSION

We proposed a new verification algorithm that was an extension of well-known HSA. It only focused on the isolation and reachability properties of network, but it was much faster than HSA. We provided a proof of the correctness of the proposed method, and we confirmed its performance through the prototypes.

Future work is as follows:

- We assumed that the routing configuration of most network elements could be modeled with a flow table, but some of these had very complex structure, such as the flow table pipeline in OpenFlow v1.1. As some work [5] has been able to handle the pipeline, we have to consider it.
- HSA has some optimization techniques [1], and some of them can also be used in B-HSA. For example, we expect that we can use the “IP-able Compression [3]”, “Lookup based Search”, in B-HSA. We intended to try them and will evaluate the performance.

REFERENCES

[1] Peyman Kazemian, George Varghese, and Nick McKeown. "Header space analysis: static checking for networks". In Proceedings of the 9th USENIX conference on Networked Systems Design and Implementation (NSDI'12). USENIX Association,

[2] E. Al-Shaer , S. Al-Haj, "FlowChecker: configuration analysis and verification of federated openflow infrastructures," *Proceedings of the ACM Workshop on Assurable and Usable Security configuration*, 2010.

[3] Richard P. Draves, Christopher King, Srinivasan Venkatachary and Brian N. Zill, "Constructing Optimal IP Routing Tables", Microsoft Technical Report (MSR-TR-98-59), 1998

[4] Haohui Mai, Ahmed Khurshid, Rachit Agarwal, Matthew Caesar, P. Brighten Godfrey, Samuel Talmadge King, "Debugging the Data Plane with Anteater". ACM SIGCOMM Computer Communication Review, 41(4), 290-301., Aug. 2011

[5] Sooel Son, Seungwon Shin, Vinod Yegneswaran, Porras Phillip, Gu Guofei. "Model checking invariant security properties in OpenFlow". IEEE International Conference on Communications (ICC), 2013, (pp.1974-1979).

[6] N. Lopes, N. Bjorner, P. Godefroid, K. Jayaraman , G. Varghese, "DNA Pairing: Using Differential Network Analysis to find Reachability Bugs," 2014.

[7] A. Khurshid, W. Zhou, M. Caesar, P. B. Godfrey, " VeriFlow: Verifying Network-Wide Invariants in Real Time," ACM SIGCOMM Computer Communication Review, Vol. 42, No. 4, pp. 467-472, Sep., 2012.

[8] Peyman Kazemian, Michael Chang, Hongyi Zeng, George Varghese, Nick McKeown, and Scott Whyte,. "Real time network policy checking using header space analysis". In Proceedings of the USENIX Symposium on Networked Systems Design and Implementation (NSDI), pages 99-111, 2013.

[9] N. Lopes, N. Bjorner, P. Godefroid , G. Varghese, " Network Verification in the Light of Program Verification," 2013.

[10] Vijay Ganesh, et al., "STP Constraint Solver", https://sites.google.com/site/stpfastprover/STP-Fast-Prover

[11] G. Holmes; A. Donkin and I.H. Witten. "Weka: A machine learning workbench" . Proc Second Australia and New Zealand Conference on Intelligent Information Systems, Brisbane, Australia, 1994

[12] "Trema: Full-Stack OpenFlow Framework in Ruby and C", http://trema.github.io/trema/

[13] Ehab Al-Shaer, Will Marrero, Adel El-Atawy and Khalid ElBadawi, "Network Configuration in A Box: Towards End-to-End Verification of Network Reachability and Security".

[14] M. Canini, D. Venzano, P. Peresini, D. Kostic, and J. Rexford, "A nice way to test openflow applications," Proceedings of the 9th USENIX conference on Networked Systems Design and Implementation, NSDI'12, Berkeley, CA, USA, pp.127-140, USENIX Association, 2012.