Concise Paper: In-Band Update for Network Routing Policy Migration

Shuyuan Zhang, Sharad Malik*, Sanjai Narain†, Laurent Vanbever**
* Princeton University, {shuyuanz, vanbever, sharad@princeton.edu}
† Applied Communication Sciences, {snarain@appcomsci.com}

***Abstract*—Network operators often need to change their routing policy in response to network failures, new load balancing strategies, or stricter security requirements. While several recent works have aimed at solving this problem, they all assume that a fast and conveniently dimensioned out-of-band network is available to communicate with any device. Unfortunately, such a parallel network is often not practical. This paper presents a technique for performing such updates *in-band*: it enables reconfiguration control messages to be sent directly within the fast production network. Performing such updates is hard because intermediate configurations can lock out the controller from devices before they are updated. Thus, updates have to be carefully sequenced. Our technique also minimizes the total update time by updating the network in parallel, whenever possible. Our technique takes into account *in-band* middleboxes, such as firewalls. We have implemented our framework using Integer Linear Programming, and experimentally validated it on problems of realistic scale.**[1]

***Keywords*-Software-Defined Networks; Routing Policy Migration; Network Update; In Band Update; Configuration**

I. Introduction

Network policy migration occurs frequently. For instance, network operators may want to adapt their routing policy because of new traffic characteristics (*e.g.*, upon a surge in popularity of a hosted content). They may also want to reconfigure their traffic engineering policy to optimize the network bandwidth. Alternatively, newer security requirements may force them to redirect all ingress traffic to middleboxes such as firewalls or intrusion detection agents. Due to the frequency of these events—more than one per day in large networks [1]—it is crucial for routing policies to be migrated *(i)* without bringing down the entire network and *(ii)* without impacting service availability.

Recently, there has been a lot of work on network updates in the context of both traditional (*e.g*, [1], [2]) and Software-Defined Networks (SDN) (*e.g.*, [3], [4], [5], [6]). However, all the current solutions have assumed that a fast and dedicated out-of-band (OOB) network was available. Performing updates using an OOB network is easy because bi-directional connectivity between the controller (or the network management system) and the forwarding equipment is guaranteed at all time. This assumption makes sense in the context of traditional networks where control messages are usually small, consisting of few configuration lines. However, this assumption does not hold in the context of SDN where the content of the forwarding tables themselves, each of them containing potentially thousands of forwarding rules, has to be exchanged over the OOB network. In-band networks are usually much faster than OOB (for obvious cost reasons) and the management of the OOB network is often difficult because of the extra wiring. Therefore, there is significant incentive to be able to perform updates in-band in the context of SDN. While performing OOB updates correctly is known to be hard [2], performing in-band updates correctly is even more challenging for at least two reasons. First, in-band updates require to maintain network-wide consistency *in addition to* bi-directional communication between the controller and any forwarding equipment at all time. Second, as control messages are sent on the production network directly, in-band updates also need to take into consideration in-path middleboxes that could drop them unexpectedly.

In this work, we present a general in-band reconfiguration framework that addresses both concerns. We make the following contributions:

1) We introduce and formally define the in-band network update problem. Our approach applies to both traditional and SDN networks, but focuses on the latter.
2) We present an Integer Linear Programming (ILP) based symbolic modeling of the network that can handle general forwarding devices such as routers, switches, and access control devices such as firewalls.
3) We show how we can compute a valid update ordering of the switches that can successfully migrate the network; or determine that no such solution exists.
4) We show how to maintain full bi-directional connectivity between the controller and the forwarding devices.
5) We show how we minimize the time it takes to perform the update by simultaneously updating multiple devices whenever possible.

The rest of the paper is organized as follows. §II demonstrates the challenges of in-band updates with a simple example. §III defines and formulates the in-band update problem. §IV describes our reconfiguration approach. §V demonstrates the scalability and effectiveness of the approach via experimental results. Finally, §VI provides a comparison with related work and §VII, concluding remarks.

[1]This work has been funded by Air Force Research Laboratory under contract FA8750-13-C-0030. Approved for Public Release; Distribution Unlimited: 88ABW-2014-3771 20140814. This work was supported in part by C-FAR, one of the six SRC STARnet Centers, sponsored by MARCO and DARPA.

/978-1-4799-6204-4/14$31.00

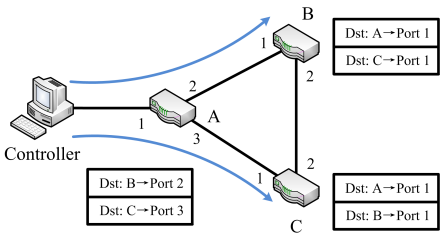

Figure 1. Example: Routing State 1

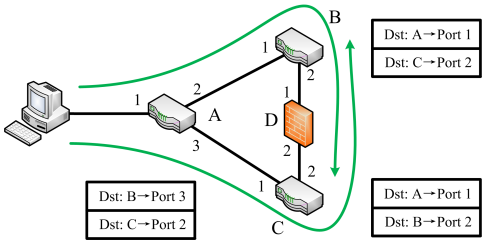

Figure 2. Example: Routing State 2

II. Motivating Example

In this section, we walk through a small example to demonstrate why in-band network update is difficult. Consider the network shown in Figure 1 composed of a management unit, which we call a controller, and three forwarding devices (*i.e.*, SDN switches or IP routers). The controller is in charge of setting up the forwarding entries in each device. It also initializes and manages the entire update process. Initially, the controller reaches B or C in one-hop, via A. Likewise, packets between B and C go through A. Now assume the network operator adds a firewall, D, configured to act as a Deep Packet Inspection (DPI) device, between B and C (Figure 2) and she wants to force all network traffic to traverse the firewall. To make the forwarding action as simple as possible, the operator configures the firewall to forward all packets that arrived at port 1 to port 2 and vice versa. Another solution would be to configure the firewall to forward whatever traffic it receives to the same port it came from. This however would make forwarding decision at B and C more difficult as they would have to consider the ingress port of the packets in addition to the destination to avoid forwarding loops. After the update, device A forwards packets destined to C to port 2 and packets destined to B to port 3. B will have one entry in the routing table to route packets to C to port 2 and similarly for device C. The question is how to update the network from the initial to the final routing state, *using only the in-band network*.

A strawman approach is to update the devices according to their relative distance to the controller. For instance, update the devices that are directly connected to the controller first, then the ones located one-hop away, etc. In this case, the controller would first update the forwarding table on A. Doing so, A will start forwarding packets destined to C to port 2 and packets destined to B to port 3. However, B and C have not been updated yet, and when B (resp. C) receives a packet destined to C (resp. B), it will send it back to A. Hence, packets will get stuck in a forwarding loop. This loop is permanent and renders B and C *completely isolated* from the controller. Restoring controller-to-device connectivity requires the network operators to *manually* reconfigure the forwarding tables on B and C (using a physical interface such as a serial port). Another solution would be to update devices located further first, and work backwards towards the controller. Again, this strategy does not guarantee that the controller will be able to reach all equipment. As an illustration, consider the opposite migration in which the network is updated from the final state (Figure 2) to the initial one (Figure 1). If B is updated first, C is no longer reachable from the controller until A is updated. However, if we update A before B and C, the controller can reach all devices during the entire updating process.

As we can see from this example, an ill-designed updating sequence could lead to permanent unreachability problems. Moreover, in this example the firewall only inspects the packets (for simplicity). In real networks, the presence of firewalls would complicate the update order even further as they could block packets in the middle of the migration.

III. Problem Definition

In this work, we define the network as a controller, a set of switches, and their connecting links. The controller is the management unit in traditional networks that initiates the update or the centralized controller in SDN. In this work, both forwarding devices and access control devices are called switches (denoted as S_i with i being its unique ID) and they are connected by links. The routing behavior of each switch is completely determined by its policy. For forwarding devices, such policy can be extracted from the *Forwarding Information Base* in routers or *IP Routing Table* in Linux kernel-based routing devices. For firewalls, such policy can be the ACL table. The policy is denoted as Q_i^1 or Q_i^2. The superscript is used to indicate the version of the policy. We assume an atomic update from Q^1 to Q^2, as supported by modern routers [7]. The policy can be a priority based routing table as in SDN or an IP prefix based routing table as in traditional networks as long as the policy can determine a specific routing decision based on the incoming packets. Since the update can be implemented round by round, we use P_i to represent in which round switch S_i shall be updated. For example, if $P_i = 3$, the controller should update S_i in the third round. If two devices have the same value of their update indices, they are updated in the same round. A round need not be updated atomically, i.e., the switches in a round can be updated independently.

The in-band network update problem is defined as follows:

Problem 1. *Given two routing configurations of the network $\{Q_i^1\}$ and $\{Q_i^2\}$, find a partial order "$<$" of the switch update indices such that $\forall S_i$, S_i is reachable from the controller using the in-band network if $\forall S_j : P_j < P_i$ have been updated from Q_j^1 to Q_j^2; or prove that no such partial order exists.*

Optionally, we can define a stronger network update problem by enforcing a two-way connectivity, such as "$\forall S_i$, S_i is reachable from the controller for both directions during the update." This will be useful for TCP messages.

The reason why it is a partial order is that it is possible to have two switches which do not share any switches in their update routes, meaning that their update order does not affect the reachability to each other. Therefore, we can cluster such updates together to achieve simultaneous updates when two switches have no relative ordering constraints. The specific actions needed to update from Q_i^1 to Q_i^2 can be flexible and dependent on the scenario.

IV. Solution

In this section, we discuss how to solve the in-band update problem using an ILP formulation. First, we explain the reason why we formulate the problem as a constraint satisfaction formulation and more specifically an ILP problem. After that, we will provide an overview of our approach and the encoding details.

A. Intuition

Our work is based on the the following observation: in order to update switch S_i, the controller has to establish a route destined to the switch. This route may have a set of switches X that have been updated to Q^2 with the rest of switches Y not updated. In order for this route to be successfully established, we have to update X before updating S_i and update Y after we update S_i, i.e.,

$$\forall S_j \in X : P_j < P_i \text{ and } \forall S_k \in Y : P_i < P_k \qquad (1)$$

This constraint is the foundation of our work to find a valid ordering among all the switches and thus our work is based on the satisfaction of a set of order constraints. These order constraints can easily fit into a constraint satisfaction based formulation. As the P_i are integer variables, and thus it is natural to consider formulating the constraint satisfaction problem as an ILP problem. Further, since X and Y are not known beforehand, our method has to consider all possible cases for X and Y and we use a symbolic encoding for the network to accomplish this. If there are firewalls in the network, our approach has to either bypass the firewalls, i.e. there is no firewall on the route, or make sure that firewalls do not block the packets. We now show how the symbolic encoding of the network and the order constraints can be modeled as linear constraints in an ILP formulation.

Table I
Encoding Variables

$v_{i,j,k}^{in}$	To indicate if ingress link k at S_j is on the route from the controller to S_i
$v_{i,j,k}^{out}$	Similar to $v_{i,j,k}^{in}$ but for the egress link k at S_j
$v_{i,j}^{sw}$	To indicate if switch S_j is on the route from the controller to S_i
$v_{i,j}^{version}$	To represent which version of the routing policy is used to find the route to switch S_j
$H_{i,j}$	The forwarding variable for S_j to update S_i
T_i	The network encoding variable to find the route to S_i

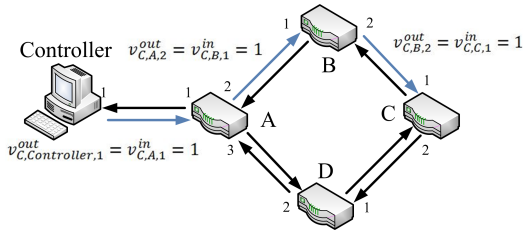

Figure 3. Network Model

B. Overview

As mentioned in the previous subsection, we need to determine if we can successfully establish a valid route from the controller to the switches. In real networks, there may be a large number of such routes. In order to consider all possible routes, we present an ILP-based modeling that symbolically encodes the network. We then add constraints to this symbolic network model so that we can find a valid route that satisfies the ordering constraint. For each S_i that needs to be updated, we have one symbolic representation of the entire network, which is used to find the valid route from the controller to that switch. For example, if there are 3 switches to be updated, we conjunct 3 encodings of the entire network together. The variables in one encoding are disjoint with the variables in the other except the update index variables P_i, which bridge all encodings together by forcing the global ordering constraint described in Equation 1.

The symbolic encoding has two parts, one for network encoding and the other for constraint encoding. The network encoding, represented as T_i, specifies the forwarding function of the network. The constraint encoding captures the ordering constraint 1 and other constraints.

Next, we explain the specific details of the encoding. Many logical operations ($\wedge, \vee, \Rightarrow$) are used in the formulation and these can be converted to linear constraints using known transformations [8].

C. Network Encoding

The set of variables used in the encoding is listed in Table I and the set of constraints is listed in Table II. Although switches are connected by bidirectional links, for the convenience of symbolic encoding, each bi-directional link is replaced by a pair of unidirectional links as shown in

Table II
NETWORK ENCODING

R^1	$v^{sw}_{i,j} = \bigvee_k v^{in}_{i,j,k}$
R^2	$H_{i,j} = v^{sw}_{i,j} \Rightarrow (v^{out}_{i,j,l} \wedge \bigwedge_{k \neq l} \neg v^{out}_{i,j,k})$
R^3	$H_{i,j} = \bigwedge_k \neg v^{out}_{i,j,k}$
R^4	$H_{i,j} = \Big((v^{sw}_{i,j} \wedge \neg v^{version}_{i,j}) \Rightarrow (v^{out}_{i,j,l^1} \wedge \bigwedge_{k \neq l^1} \neg v^{out}_{i,j,k})\Big)$ $\wedge \Big((v^{sw}_{i,j} \wedge v^{version}_{i,j}) \Rightarrow (v^{out}_{i,j,l^2} \wedge \bigwedge_{k \neq l^2} \neg v^{out}_{i,j,k})\Big)$

Figure 3. Each unidirectional ingress/egress link is encoded as one binary variable ($v^{in}_{i,j,k}$ or $v^{out}_{i,j,k}$ as shown in Table I) to indicate if this link is on the route from the controller to the destination switch. As shown in Figure 3, the highlighted links are those from the variables assigned to 1. These form a route to the switch that we need to update, which in this case is C. These variables are not shared among different encodings of the networks. For example, $v^{in}_{1,j,k}$, which is used to find a route to S_1, is a different variable from $v^{in}_{2,j,k}$, which is used for finding the route to S_2. Since the ingress link of a switch must be an egress link of the adjacent switch, $v^{in}_{i,j,k} = v^{out}_{i,l,m}$ if the ingress link k of S_j is connected to the egress link m of S_l. $v^{sw}_{i,j}$ is used to indicate if switch S_j is on the route to S_i. A switch is on the route if and only if at least one of its ingress links is on the route (R^1 in Table II).

The switch function is to express which egress ports shall be on the route based on the routing decision. For example, let us consider the symbolic encoding to update S_i. Assume the controller uses packet pkt to communicate with S_i and the routing policy at $S_j (S_j \neq S_i)$ forwards pkt to port l. In the encoding to find a route to S_i, the routing function for S_j is shown in R^2 of Table II, meaning that if S_j is on the route, the egress port l also has to be on the route. If the switch happens to be a firewall and it may block the packets, the function used is R^3 of Table II. If the firewall permits the packet, the firewall is the same as a switch. As we can see, firewalls are not treated differently from normal switches and hence can be easily included in the framework. In many cases, the switch may have two versions of routing decisions based on either Q^1_j or Q^2_j. We use another symbolic Boolean variable $v^{version}_{i,j}$ to encode which action to use. Assume Q^1_j forwards pkt to port l^1 while Q^2_j forwards to l^2. If $v^{version}_{i,j}$ is 0, S_j forwards to l^1; otherwise, to l^2. Then, the function used for S_j is R^4 in Table II. The final network encoding variable is $T_i = \bigwedge_j H_{i,j}$ and it represents the complete forwarding behavior of the network to update S_i. Finally, we constrain $\bigwedge_i T_i$ to be 1 all the time.

D. Constraint Encoding

The network encoding provides a generic encoding for the network forwarding functionality. However, without further constraints, just satisfying the encoding constraints will not produce meaningful results. In this subsection, we explain how to use constraints to further restrict the solution space.

1) Switch Constraint:

- First, we have to ensure that exactly one egress port of the controller has to be on the route because we want the route to have only one source port. This can be expressed as $\sum_k v^{out}_{i,controller,k} = 1$.
- Further, the controller's ingress link shall not be on the path; otherwise, there will be a loop in the route with the controller in it. We use $\sum_k v^{in}_{i,controller,k} = 0$ to express that.
- At least one ingress port of the destination switch has to be on the route and this guarantees a route with valid destination: $\sum_k v^{in}_{i,i,k} > 0$. This constraint guarantees that if there are firewalls on the route, they do not block the control packets.
- In the end, we have to ensure that no switch has more than two ingresses on the route. This constraint will guarantee a simple loopless route [9]. This can be expressed as $\bigwedge_j (\sum_k v^{in}_{i,j,k} < 2))$.

2) Update Order Constraint: Equation 1 has to be captured in the constraint formula and it can be easily expressed using $v^{version}$ and v^{sw}. If $v^{version}_{i,j} = 0$ and $v^{sw}_{i,j} = 1$, S_j is on the route to update S_i and it uses Q^1_j as the forwarding policy. This means that S_j is updated later than S_i and thus $P_i < P_j$. Similarly, if both $v^{version}_{i,j}$ and $v^{sw}_{i,j}$ are 1, S_j is on the route and it uses Q^2_j as the forwarding policy and thus $P_i > P_j$. Overall, the formula to capture the update order constraint is $\big(v^{sw}_{i,j} \wedge \neg v^{version}_{i,j} \Rightarrow (P_i < P_j)\big) \wedge \big(v^{sw}_{i,j} \wedge v^{version}_{i,j} \Rightarrow (P_i > P_j)\big)$.

E. Optimization

Using the constraints expressed in the network encoding and the constraint encoding, we are able to find a sequence of P_i values if one exists. These values provide a valid ordering of the switch update. As mentioned in Section III, since the update order is a partial order, there may exist some update order indices that have the same value, meaning we could update those switches simultaneously. Usually, it is preferable to update the network as fast as possible i.e., using as few update rounds as possible. However, the solution returned by the ILP solver does not reflect this. One solution is to let the ILP solver minimize $max\{P_i\}$. We set an upper bound for each P_i to P^{upper} ($\forall i : P_i < P^{upper}$) and let the solver minimize the upper bound P^{upper}. Depending on how P^{upper} is constrained, it may be possible that the solver does not return any results in a reasonable amount of time. This will likely happen when P^{upper} is not too tight or too loose a constraint. The former may be easy to determine as being infeasible and the later may be easy to satisfy. To deal with this, we can set a time limit for each run and we can regard time-outs as "Infeasible" cases. Thus, we get a result that may not be the minimum number of rounds, but possibly very close to it. We refer to this solution as the "minimal" solution to distinguish it from the minimum one.

F. Two-Way Communication

In the formulation mentioned in the previous subsections, we only considered the communication from the controller to the switch. In many cases, after the switch updates to its new routing policy, it may send reply messages back to the controller to mark the end of the update; otherwise, the controller will not know when the update is completed. In order to include such "two-way" communication, we can add another version of the symbolic encoding for each update switch and this encoding tries to find a valid route from the update switches to the controller. Almost everything is the same as the previous network encoding except that we need to reverse the constraint on the ingress and egress links of the controller and the update switch. We specify that at least one of the egress links of the update switch and exactly one of the controller's ingress links are on the route.

V. Experimental Results

We implemented our ILP based approach using IBM CPLEX as the underlying ILP solver [10] with at most 7 threads enabled. We ran a variety of experiments to test the effectiveness of our approach. We tested our tool on a set of synthetic benchmarks by using the Waxman topology [11] as the network topology and we randomly connect the controller to two switches. We used the shortest path algorithm to generate different versions of the routing policies. The machines used in these experiments have Intel Xeon 3.20 GHz processors running Linux with kernel version 3.2.0. Out tool successfully finds valid orderings for all the experiments we ran.

We increased the size of the networks from 10 switches to 100 switches with an increment of 10 switches with about 50% of the switches needing to be updated. We ran the experiments twice without optimizing for P^{upper}. One enables the two-way communication from the update switch to the controller and the other without it. The execution time is shown in Figure 4 and the total number of indices required for such an update to finish is shown in Figure 5. The total number of indices represents the number of rounds needed for the update. As we can see from Figure 4, the execution time increases as the total number of switches grows. This is expected as the size of the formula is linear in the size of the network and the total number of switches that need to be updated. The requirement of two-way communication also increases the execution time because the search space is more constrained. On average, the two-way communication increases the execution time by about 2.8 times. In Figure 5, "Opt" presents the minimal P^{upper} achieved by setting the time limit to be 200s. The cases without the "Opt" flag represents the total number of update rounds returned. Since we only constrain the relative relationship between P_i, there are no constraints on the specific values of P_i. In this case, we count how many unique P_i there are. As we can see, the total number of update rounds increases with the size of

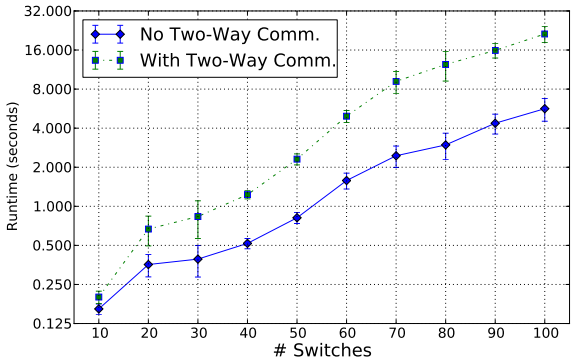

Figure 4. # Switches vs. Time

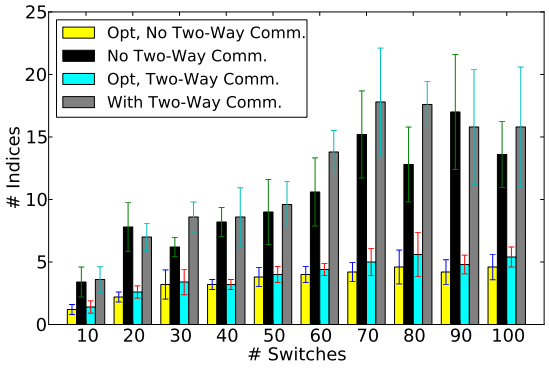

Figure 5. # Switches vs. # Indices

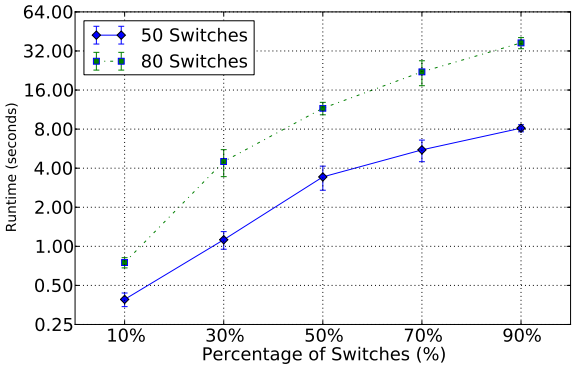

Figure 6. Percent of Updated Switches vs. Time

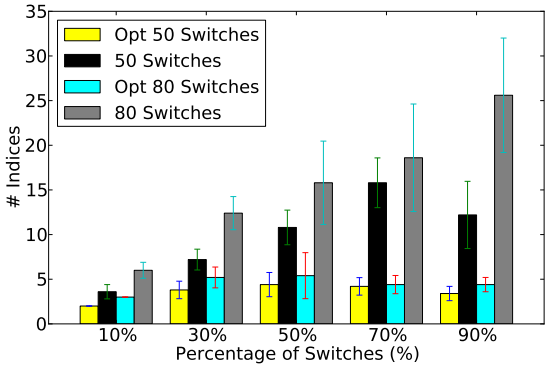

Figure 7. Percent of Updated Switches vs. # Indices

the network and the optimization on P^{upper} is effective in reducing the total update rounds. On average, it reduces the number of update rounds by 62% (compare the 1st bar with the second and the third with the fourth in Figure 5).

In the second set of experiments, we increase the percentage of the switches in the network needing to be updated on two networks, one with 50 switches and the other with 80 switches. The results are shown in Figures 6 and 7. Both execution time and the total number of update indices increase with the percentage of the updated switches. We set the time limit for the execution to be 200s. The optimization on P^{upper} saves an average of 57% of update rounds for the network with 50 switches and 65% for the case with 80 switches.

VI. Related Work & Discussion

[2] proposes a seamless network update strategy using a different ILP based formulation, which enumerates all the forwarding loops in the network and breaks loops by forcing certain update orders. However, they can only handle forwarding loop based reachability problems and cannot handle packet blocking or parallel updates. Recently, there have been many other works on updating the network using SDN. [4] proposed per-packet consistency which requires that every packet in the network is processed by either the pre-update configuration or the post-update configuration, and never a mixture of the two. Based on per-packet consistency, they proposed a two-phase update that includes a version number in each rule and packet. Whenever an update is scheduled, they first install rules with a newer version number in the middle of the network and then update the new rules at the border of the network. Similarly, [6] proposes an OpenFlow safe update protocol to reduce the rule space requirement of two-phase update by temporarily forwarding packets to the controller in the middle of the update. [5] improves two-phase commit by increasing the number of rounds to complete an entire update. They partition the network into different slices and each slice observes similar network behavior. They update the network one slice at a time. zUpdate [12] is used to update the network without transient violation of the bandwidth requirements on each link. This approach computes how to update one network traffic pattern to another pattern using a sequence of updates. Unlike our approach, these works do not target in-band updates and cannot deal with in-band middleboxes. Our framework is also able to leverage partial ordering constraint to update the network in parallel whenever possible.

VII. Conclusions

In this work, we proposed a general network routing policy update mechanism using in-band networks. In-band networks do not require dedicated links from the controller to the forwarding devices. Our ILP-based symbolic representation of the network model can model the network routing behavior and can handle both normal forwarding devices and access control devices. By constraining the symbolic model, we can find an update order that guarantees valid routes from the controller to all the switches that need to be updated. Based on the route found, we can generate a correct ordering of the switches in which an update can be successfully implemented. We demonstrated that our approach is scalable for real-world networks as it can find a correct ordering within 10 seconds for networks with less than 70 switches (Figure 4). Moreover, our approach can minimize the update rounds by enabling simultaneous updates for switches and it can reduce the total number of the update rounds by around 60%. In the future, we plan to handle networks with additional kinds of middleboxes such as Network Address Translators. We will also extend our framework to maintain not only the connectivity from the controller to the updated switches, but also from end-points to end-points to improve service availability.

References

[1] S. Vissicchio, L. Vanbever, C. Pelsser, L. Cittadini, P. Francois, and O. Bonaventure, "Improving Network Agility With Seamless BGP Reconfigurations," *IEEE/ACM Transactions on Networking*, June 2013.

[2] L. Vanbever, S. Vissicchio, C. Pelsser, P. Francois, and O. Bonaventure, "Seamless Network-Wide IGP Migrations," in *Proceedings of the 2011 ACM SIGCOMM Conference*. Toronto, Canada: ACM, Aug. 2011.

[3] N. McKeown, T. Anderson, H. Balakrishnan, G. Parulkar, L. Peterson, J. Rexford, S. Shenker, and J. Turner, "OpenFlow: enabling innovation in campus networks," *SIGCOMM Comput. Commun. Rev.*, Mar. 2008.

[4] M. Reitblatt, N. Foster, J. Rexford, C. Schlesinger, and D. Walker, "Abstractions for network update," in *ACM SIGCOMM*, 2012.

[5] N. P. Katta, J. Rexford, and D. Walker, "Incremental consistent updates," in *ACM HotSDN*, 2013.

[6] R. McGeer, "A safe, efficient update protocol for OpenFlow networks," in *ACM HotSDN*, 2012.

[7] C. Filsfils, P. Mohapatra, J. Bettink, P. Dharwadkar, P. De Vriendt, Y. Tsier, V. Van Den Schrieck, O. Bonaventure, and P. Francois, "BGP prefix independent convergence (PIC) technical report," Cisco, Tech. Rep, Tech. Rep., 2011.

[8] S. Bradley, A. Hax, and T. Magnanti, "Applied mathematical programming," 1977.

[9] S. Zhang and S. Malik, "SAT based verification of network data planes," in *Automated Technology for Verification and Analysis*, ser. Lecture Notes in Computer Science. Springer International Publishing, 2013.

[10] *CPLEX*, Dec. 2013. [Online]. Available: http://www-01.ibm.com/software/commerce/optimization/cplex-optimizer/

[11] B. Waxman, "Routing of multipoint connections," *Selected Areas in Communications, IEEE Journal on*, dec 1988.

[12] H. H. Liu, X. Wu, M. Zhang, L. Yuan, R. Wattenhofer, and D. Maltz, "zUpdate: Updating data center networks with zero loss," in *ACM SIGCOMM*, 2013.