Fast Lossless Traffic Migration for SDN Updates

Long Luo, Hongfang Yu, Shouxi Luo, and Mingui Zhang*
Key Laboratory of Optical Fiber Sensing and Communications, Ministry of Education
University of Electronic Science and Technology of China, Chengdu, P. R. China
*Huawei, Beijing, P. R. China

***Abstract*—Migration of traffic from one configuration to another is common in SDNs due to node/link failures, network maintenance, policy reconfiguration, intrusion detection, network upgrade, and etc. When network devices are informed by SDN controller to execute the traffic migration, it's difficult even impossible to force all network devices to perform the update action in a strict synchronized way. Thus the network is likely to see transient overlapped traffic from both the new configuration and the old one. This kind of overlap may cause overload to those hot spots. This paper reveals the transient congestion problem during traffic migration in an SDN update. According to the observation, it's feasible for the controller to schedule ingress nodes to perform the migration in an order thus the transient congestion is avoided. This scheduling problem is formulated as a Mixed Integer Linear Program (MIP) model. If feasible orders exist for ingress network nodes to perform the migration, the MIP can always find the order that achieves the minimum steps in all possibilities. A heuristic method (named ATOMIP (ATOmic-MIP)) is proposed to speed-up the solving of this MIP. Evaluation based on network topologies observed from real ISPs shows that the lossless migration happens in seconds.**

I. INTRODUCTION

Software-Defined Networking (SDN) brings a new architecture for network configuration and management [1]. Although SDN is originally designed for data center networks, more and more ISPs are also willing to adopt the new SDN architecture for the management of their networks. With a logically centralized controller, the knowledge of the whole network can be acquired thus a globally optimal decision for the network can be made. Since network state is volatile, it's natural for the controller to update network configuration accordingly from time to time in the day-to-day operation [2, 3].

Migration of traffic from one network configuration to another is required in SDN updates. Although the controller can easily issue the update to all network nodes in a one-shot manner, the time that all network nodes apply the new policies varies greatly. What is worse, the processing speed of the update differs from one network node to another due to their hardware disparity and CPU load variation [2]–[5]. After the network nodes receive the upgrade signal issued from controller, some nodes will forward packets based on the new configuration, while other network devices are still acting according to the old network configuration. It means there is a transient period of time in which the "old" and "new" configurations are used simultaneously. Therefore, some traffic flow may be forwarded along mixed routes or even loops due to the inconsistence. Besides, some of flight-in packets may be already passed along its new path while other may be forwarded along the original one. Thus, even if traffic congestion will not happen at either the old network configuration or the new network configuration, the network is still probably suffering from transient traffic loss due to the fact that "new-configuration" traffic load and "old-configuration" traffic load are racing for bandwidth of some hot links that causes overload to these links. Mixed routes and loop problems can be perfectly avoided by adopting a two-phase update mechanism [2, 3]. However, this update mechanism still cannot prevent the transient congestion of the whole network due to the asynchronous upgrade of ingress network nodes.

In this paper, we propose ATOMIP (ATOmic-MIP) to compute a fast and congestion-free traffic migration plan for an SDN update. Essentially, the computation of the optimal (i.e., fastest and lossless) update sequence that ingress network nodes follow can be formulated as an MIP (Mixed Integer Linear Program). However, solving the MIP problem for a large network may be time-consuming. ATOMIP recursively splits the original MIP into a series of atomic/small MIPs until it finds a feasible solution or exceeds the maximum allowed update rounds. ATOMIP classifies the set of all the to-be-updated flows into a minimum number of atomic sets. In each step, the controller informs the related ingress nodes of those flows in an atomic set to shift to the new configuration. These ingress nodes just take the update by themselves. Whatever the actual update order is, congestions will not happen.

ATOMIP is built on a simple observation: scheduling ingress network nodes to execute flows' updates in a well-designed order can usually avoid transient congestions in most cases. Essentially, ATOMIP can compute the lossless update sequence with the minimized rounds. Besides, when there is actually no feasible congestion-free solution, ATOMIP can output the one that introduces the slightest traffic loss. According to observations, such case rarely happens in practice.

Our major contributions are summarized as follows:

- The fast and lossless scheduling problem is formulated as an MIP model.
- A heuristic is proposed to speed up the computation of MIP. The key idea is to recursively split the original updates into a series of atomic/small updates, where each sub-update can be easily scheduled by solving a smaller MIP.
- Network topologies observed from real networks are used to

This work is supported in part by the 973 Program under Grant No. 2013CB329103, the 863 Program under Grant No. 2015AA011901, the National Natural Science Foundation of China under Grant No. 61271171, and the Open Foundation of State Key Laboratory of Networking and Switching Technology under Grant No. SKLNST-2014-1-09.

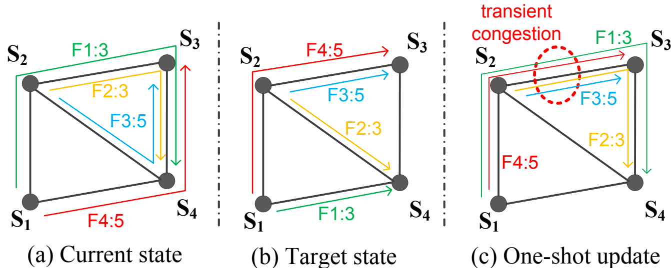

Fig. 1. Example: transient congestion caused by unsynchronized update.

evaluate ATOMIP. Experimental results imply that ATOMIP is able to find the near-optimal solution within sub-second.

The rest of the paper is organized as follows. Section II reveals the problem of transient congestions during traffic migration in SDN update. Observations and challenges of this problem are provided. In Section III , the solution for the transient congestion problem is formulated as an MIP model. In conjunction with the MIP model, a heuristic algorithm 'ATOMIP' is proposed. The performance of ATOMIP is evaluated in Section IV. Section V investigates related works and Section VI concludes the paper.

II. PROBLEM AND CHALLENGE

In this section, we reveal why the one-shot update (e.g, two-phase mechanism [2, 3]) might cause transiently congestions and how to avoid these congestions by letting ingress network nodes perform updates in a well-designed order.

The Problem: Let's take the two-phase update mechanism as an example. In two-phase update [2, 3], once the controller wants to change the flows' forwarding paths, it first installs new paths in the middle of the network and then activates ingresses to migrate flows to their new paths in "one shot". After that, the new coming packets travel along the new paths; it is safe to remove the old paths after all "old" packets drain from the network. However, since updates on devices take effect asynchronously in practice, those hot links that belong to the common parts of some new paths and old paths might be seriously congested when they are injected with packets that go through the network along the new paths before the packets over the old paths drain. For example, the hot link $S2 \rightarrow S3$ will be transiently overloaded with 16 units traffic load if $\{F3, F4\}$ change before $\{F1, F2\}$ when the network updates routing from state(a) to state(b) as Fig.1 shows, where each link has a capacity of 10 units and each flow's size is marked as the figure says.

Observations: Fortunately, we observe that managing flow updates in a well-designed order will avoid such transiently congestion in many cases. For instance, for the case shown in Fig.1, scheduling the updates in accordance with the order of $[F1 \rightarrow F2 \rightarrow F3 \rightarrow F4]$ will never cause any link overloaded. Furthermore, since the updates of $F1$ and $F2$ here can be performed in parallel without inducing congestion, we can aggregate such flows updates into one round to reduce the total update steps, which further shortens the update duration in turn. For example, for the prior mentioned case, we can simply arrange the updates into two rounds: $[(F1, F2) \rightarrow (F3, F4)]$.

Challenges: Although scheduling flow updates in a carefully-arranged order would realize congestion-free, how to (i) find a valid order for update operations and (ii) aggregate them to a minimized rounds are still hard because of the following two challenges. First, there are too many possible combinations of flow update order; thus, naive enumeration-based approaches are computationally intractable for large network updates. Second, even if the target state is valid, there may do not exist any feasible solution. For instance, recall the example shown in Fig.1, if either $F1$ or $F2$ has a size of 6, we cannot find an order to achieve traffic lossless updates. Actually, the authors of Dionysus have proved that, finding the fastest update order for such a schedule problem is NP-complete [5].

III. SOLUTIONS

In this section, we first formulate the schedule problem as a MIP which pursues to update the network with the minimum traffic loss (towards congestion-free) and the minimum update rounds (towards fast). However, since the MIP is time-consuming for updates in large networks, we further design an efficient heuristic named ATOMIP to find "near optimal" update scheduling in real-time.

A. Basic Formal Model

Generally, we assume that the network G consists of a set of switches S and a set of directed links E, where the capacity of each link $e \in E$ is denoted by c_e. For an update which is to migrate network state from $C = \{c_{f,e}, \forall(f \in F, e \in E)\}$ to $C' = \{c'_{f,e}, \forall(f \in F, e \in E)\}$[1], where F is the set of flows to be updated, $c_{f,e}$ and $c'_{f,e}$ is the traffic of flow f on link e before and after the update respectively, we use F_k to denote the set of flows that are updated in round k ($1 \leq R \leq |F|$). Obviously, all the updates can be done within most $|F|$ rounds. If we let $x_{f,k} \in \{0, 1\}$ to represent whether flow f is updated to the new path(s) in the k-th round and $y_k \in \{0, 1\}$ to represent whether there is any flow update(s) in that round, the optimal object we pursue here is to minimize the total update rounds:

$$\min \sum_{k=1}^{R} y_k \tag{1}$$

Constraints: First, since both $x_{f,k}$ and y_k are 0 or 1 and a flow can be updated only once, there must be:

$$\forall(f, k) : x_{f,k} \in \{0, 1\}, y_k \in \{0, 1\} \tag{2}$$

$$\forall f : \sum_{k=1}^{R} x_{f,k} = 1 \tag{3}$$

$$\forall k : \sum_{f \in F} x_{f,k} \leq y_k \times |F| \tag{4}$$

[1]Note that, we do not consider the flows that will not be updated here, this is easy to achieve by only taking the governable bandwidth into account as calculating the capacity of each link for the update.

Second, for the k-th round update, it is congestion-free, means the summation of both the background traffic and new incoming flows would not overflow any links. Thus,

$$\forall(e,k) : d_{e,k} + \sum_{f \in F} x_{f,k} \times a_{f,e} \leq c_e \tag{5}$$

where $d_{e,k}$ is size of the background traffic in that round and $a_{f,k}$ is the increased traffic size caused by flows migrating in. Notice that, as equation (6) says, the background traffic is the union of both updated (the first item) and un-updated flows (the second item).

$$d_{e,k} = \sum_{f \in F} c'_{f,e} \sum_{t=1}^{k-1} x_{f,t} + \sum_{f \in F} c_{f,e}(1 - \sum_{t=1}^{k-1} x_{f,t}) \tag{6}$$

$$a_{f,e} = \begin{cases} c'_{f,e} - c_{f,e}, & c'_{f,e} > c_{f,e} \\ 0, & c'_{f,e} \leq c_{f,e} \end{cases} \tag{7}$$

MIP Model: Finally, for each update, we can get the optimal update scheduling by solving the ILP of: *object to* equation: (1); *subject to* equations: (2)-(7). Unfortunately, as we show in Section II, for rare update cases, there exists no feasible solution. To make the model solvable for all update cases, we add an overload ratio to each link and modify the original ILP to a Mixed Integer Programming (MIP), which is to strive the target of finding the scheduling causes the minimum traffic loss when there does not exist a congestion-free scheduling as Fig.2 shows. In the MIP, α is a weight factor ($0 < \alpha \ll 1$) and equation (8) is defined as follows:

$$\forall(e,k) : d_{e,k} + \sum_{f \in F} x_{f,k} \times a_{f,e} \leq c_e + l_e, l_e \geq 0 \tag{8}$$

$$\min \sum_{e \in E} l_e + \alpha \sum_{k=1}^{R} y_k \tag{9}$$

$s.t.$ constraints (2)-(4), (6) and (8).

Fig. 2. Mixed Integer Programming (MIP) model for the scheduling problem.

Thus, by solving the MIP, we can find the congestion-free schedule with the minimum update rounds if it exists; otherwise, we get the schedule that causes the least traffic loss.

B. The Design of ATOMIP

Although, the original MIP can find the optimal scheduling for updates, it is time-consuming for updates in large networks because there are overmuch variables and constraints involved in the MIP of those cases. For a network update, the MIP has about $\mathcal{O}(R|F|)$ variables and $\mathcal{O}(MR)$ constraints[2], where F is the set of flows to be updated, R is number of possible update rounds and M is the amount of involved links. Since R is set to $|F|$ and M is set to $|E|$ (E is the set of links) in the original model, solving the MIP is computationally intractable for updates in a large network. Fortunately, as the model size only depends on the scales of possible update rounds, to-be updated flows and involved links, we can greatly simplify the model by limiting the update rounds and reducing the flow number and removing unconcerned links. Based on these basic observations, we design a heuristic to cut the original update into a series of sub-updates and find the "optimal" scheduling of each sub-update by solving a small MIP in the following.

[2]Similarly to the analysis of algorithms, although there are about $R(|F|+1)$ variables and $|F|+R+MR$ constraints in the model, we only take the orders of growth into account for simplicity.

Remove Non-critical Flows & Links: First, we remove both the flows that will definitely not cause any congestion and the links that surely never get overloaded from the MIP for simplification. Obviously, for a link (denoted as e) that will never be overloaded in the update, its must have more than $\sum_{\forall f} a_{f,e}$ free capacity. Thus, we can easily figure out all these congestion-risky links using FINDCRL as Pseudocode 1 shows. Note that, as we will see, the update of F here may be only a sub-update of the entire update and Line 2 in FINDCRL is to figure out the available bandwidth for this round/step of update using equation 6. After all congestion-risky links are determined, we can easily find out all the congestion-risky flows using FINDCRF shown in Pseudocode 1. Finally, to reduce the model size, we only use these congestion-risky links and flows to build the MIP.

Pseudocode 1 Find Congestion-Risky Flows

```
1:  function FINDCRL(F)   ▷ Calc. Congestion-Risky Links.
2:      Update {c_e, ∀e ∈ E} for this calculation;
3:      Get {a_{f,e}, ∀(f ∈ F, e)};
4:      E_+ ← ∅;                  ▷ Store Congestion-risky Links.
5:      for all e ∈ E do
6:          if c_e < Σ_{∀f} a_{f,e} then
7:              E_+ ← E_+ ⋃{e}
8:          end if
9:      end for
10:     return E_+;
11: end function

12: function FINDCRF(F, O)
13:     E ← GETCRL(F);   ▷ Store Congestion-risky Links.
14:     F_+ ← ∅;              ▷ Store Congestion-risky Flows.
15:     F_- ← ∅;               ▷ Store Congestion-free Flows.
16:     for all f ∈ F do
17:         if the new path(s) of f goes though E then
18:             F_+ ← F_+ ⋃{f}
19:         else
20:             F_- ← F_- ⋃{f}
21:         end if
22:     end for
23:     return (F_+, F_-, E);
24: end function
```

Limiting Update Rounds: Second, we restrict the number of possible update rounds to reduce the searching space further.

However, the choice of the round limiting number here is crucial and varies with cases: if the limiting is too low, the MIP will miss the feasible solutions and if the limiting is too high, the MIP will be too slow. The naive way is to enumerate the round limiting number R from 1 to $|F|$, but such a method is still inefficient and we design a more efficient method to pursue the target of achieving the fast (namely using the minimum update rounds) and congestion-free (namely causing the minimum traffic loss) updates.

Heuristic Framework: Generally speaking, making an update fast and congestion-free are two opposite-targets. Specifically, the fastest update schedule is to execute updates in one-shot, however, to ensure congestion-free or congestion-least for updates, the planner has to take more rounds to perform updates.

In this paper, we assume that each update in the network has a pre-known maximum update delay, which is defined by its maximum update round number N. Thus, scheduling update in most N rounds while minimizing the link overload ratios during updates is the target we pursue here.

To this end, we iteratively pick out the update round that causes the most link overloads when it updates in one-shot (Line 4) and split it into multi-rounds (Line 5-15) until the scheduling reaches the maximum update rounds (Line 3), or this round can not be split further (Line 17) (In such a case, we find a congestion-free scheduling or there does not exist a feasible one). In the splitting of a congestion-risky round, we first extract both the congestion-free and congestion-risky flows and congestion-risky links (denoted as F_-, F_+ and E_+ resp., see Line 4), then split the update of congestion-risky flows into multi-rounds using a small MIP (Line 14). Since the update round of MIP is limited to a small number R' and both the number of congestion-risky flows and links are not huge, such a small MIP can be solved nearly in real-time using optimization softwares like CPLEX and MOSEK.

Pseudocode 2 ATOMIP: Schedule Updates Using Tiny MIPs

```
1: function SCHED(FlowSet F, MaxRound N, ILPStep R)
2:     O ← [F];                ▷ The list of scheduled flow sets.
3:     while len(O) < N do
4:         F' ← GETHFS(O); ▷ Choose a flowset to expand.
5:         Remove F' from O;
6:         F+, F−, E+ ← FINDCRF(F', O);
7:         if F− ≠ ∅ then
8:             Append F− to O at the appropriate position;
9:         end if
10:        R' ← N − len(O);
11:        if R' > R then
12:            R' ← R;
13:        end if
14:        O' ← SOLVEMIP(F+, E+, O, R');
15:        Append all F_i ∈ O' to O at appropriate positions;
16:        if len(O') = 1 then
17:            break out the while-loop;
18:        end if
19:    end while
20:    return O;
21: end function

22: function GETHFS(O)
23:     F ← nil; l_max ← −1;
24:     for F_i in O do
25:         Update {c_e, ∀e ∈ E} for F_i's calculation;
26:         Get {a_{f,e}, ∀(f ∈ F_i, e)};
27:         l_i ← Σ_{∀e} max(0, Σ_{∀f:e∈E_f} a_{f,e} − c_e);
                        ▷ E_f is the set of links in f's path(s).
28:         if l_max < l_i then
29:             l_max ← l_i; F ← F_i;
30:         end if
31:     end for
32:     return F;
33: end function

34: function SOLVEMIP(F+, E+, O, R')
35:     Update {c_e, ∀e ∈ E+} for this calculation using O;
36:     Build and Solve the MIP for (F+, E+, R') (See Fig.2);
37:     return the list of to-be-update flow in each round;
38: end function
```

IV. Experimental Results

In this section, we evaluate the effectiveness and efficiency of ATOMIP using topologies observed from real networks and synthetic traffics.

A. Experimental Setup

We use Mosek (http://mosek.com/) as our MIP solver and implement ATOMIP in C++. All experiments are carried out on a laptop running 64-bit Ubuntu 12.04 desktop with 4G memory and a single Intel i5-2450M CPU.

1) Topologies: We use 6 topologies for our tests as Table I shows, where {AS1755, AS3257, AS1239} are 3 inferred ISP topologies get from the Rockefuel dataset [6], and {Waxman1, Waxman2, Waxman3} are 3 synthetic topologies generated by Waxman's random network model [7].

TABLE I
Experimental Topologies

#Topo	AS1755	AS3257	AS1239	Waxman1	Waxman2	Waxman3
Nodes	18	27	30	50	100	150
Links	66	128	138	200	300	600

2) Traffic Matrixes and Loads: Similar to [8], the capacity for each link is set to 1000 units in each direction and the traffic demand from node (or VM) i to j is given in equation (10), where $c_{m,n}$ is the capacity of link $e_{m,n}$ and σ is a weight factor ranging from 0 to 1. Besides, we assume the network uses the shortest path routing (calculated using Dijkstra) by default.

$$d_{i,j} = \sigma \sum_{t:e_{i,t}\in E} c_{i,t} \frac{\sum_{t:e_{t,j}\in E} c_{t,j}}{\sum_{e_{m,n}\in E} c_{m,n} - \sum_{t:e_{t,j}\in E} c_{t,i}}, i \neq j \quad (10)$$

In all of our tests, we change σ to adjust traffic loads to

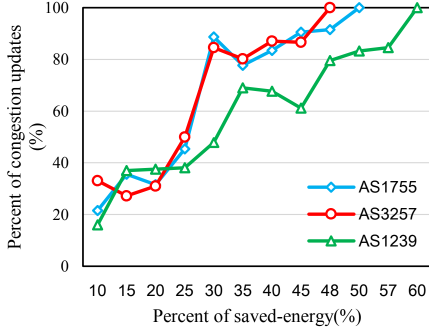

Fig. 3. The possibility of congestion during the one-shot update increases with the percent of saved-energy growing.

ensure: (1) both the initial and target states are congestion-free; (2) the average link utilization ratio is about 35%.

3) Update Scenarios: For each topology, we consider energy-saving driven updates, where the network computes an energy-efficient routing configuration and updates to it periodically. In the tests, we use GreenTE [9] to find energy efficient routings for small topologies: AS1755, AS3257, AS1239, and use SWDA [10] to find energy efficient routings for large topologies: Waxman1, Waxman2, Waxman3. For each topology, we change the percent of saved-energy, a parameter that drives the new routing policies, to simulate the scenes of different traffic loads. We also guarantee the maximum link utilization is less than 90% in the new configurations.

Besides, we set the ILP Step R of ATOMIP to 6 and the MaxRound N to infinite (see Pseudocode 2) in tests.

4) Performance Metrics: We first verify ubiquity of congestion during unplanned updates and test whether ATOMIP's can find the congestion-free solution, then evaluate the update speed of ATOMIP's results using the percent of migrated traffic in each round, and next compare the one-shot and ATOMIP in traffic loss and other aspects, and finally measure its time cost.

B. Results

1) The ubiquity of temporary congestions: Since the congestion in one-shot update that we concern is caused by unregulated update orders, we synthesize a set of update cases that have feasible congestion-free plans and enumerate all possible update permutations to verify the ubiquity of congestions in one-shot update. Fig.3 is the statistics of results. It implies that the possibility of congestion in one-shot update increases with the percent of saved-energy growing in all topologies. In contrast, ATOMIP find congestion-free updates for all these cases.

2) Update speeds: To evaluate the update speeds of ATOMIP's results, we set the percent of saved-energy to 50% and repeat the tests on each topology 20 times. First, we count the update rounds of ATOMIP's results and find that the number of update rounds is determined by the network topology and size as Fig.4 shows. Roughly speaking, the update round number grows with network size when the percent of saved-energy is fixed. In all the cases, ATOMIP can schedule the update in a small update rounds. Then, we count the percent of traffic loads that migrated in each step. The results imply

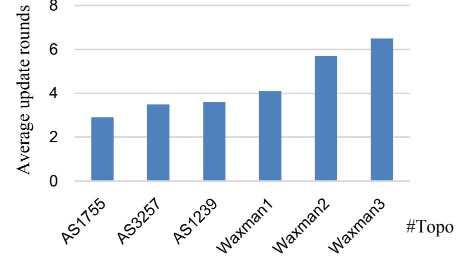

Fig. 4. Average update round grows with network size when the percent of saved-energy is fixed for energy-saving driven updates.

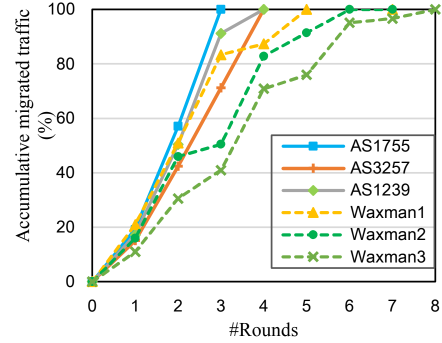

Fig. 5. The volume of migrated traffic in each round is nearly uniform.

that the volume of migrated traffic in each round is nearly uniform, similarly to Fig.5 shows, is the result of one case.

In fact, as our ATOMIP always finds the optimal scheduling uses the minimum update rounds, ATOMIP achieves fast updates using the minimized update rounds and balance the to-be-updated traffic in each round.

3) Performance comparison: To compare the performance of one-shot update with that of ATOMIP , the percent of saved-energy for each network is set to 35%. We adopt the method of one-shot and ATOMIP to schedule the updates of the same network, respectively. First, we use the "lossSum" to indicate the total traffic loss introduced by different update methods. The "lossSum" equals to the sum of all links' overload ratio (1 is representative of a full load link). As Table II shows, the traffic loss of ATOMIP is always much less than that of one-shot. Similarly, we use the "uMax" to indicate the maximum link utilization. As Table II shows, the maximum link utilization of the ATOMIP is less than that of one-shot. In addition, the number of overloaded links and the update rounds are counted, which is indicated by "Overloaded Links" and "Rounds", respectively. The overloaded links of one-shot ranges from 3 to 15, while that of ATOMIP is up to 3. The results represented in Table II say that ATOMIP performs better than one-shot with much fewer overload links, even though it take a bit more update rounds.

4) Complexity and time cost: At last, we collect both the total number of variables and constraints, and the total solver time for ATOMIP's MIPs. Roughly, the results indicate that, via splitting the original fat MIP to several atomic/small MIPs and removing all non-critical flows and links from the model, ATOMIP not only reduces the model size greatly but also makes the computing able to be handled in nearly real-time. For example, in a case that scheduling the update of 870 flows

TABLE II
THE PERFORMANCE COMPARISON OF ONE-SHOT AND ATOMIP

#Topo	lossSum		uMax		Overloaded Links		Rounds	
	one-shot	ATOMIP	one-shot	ATOMIP	one-shot	ATOMIP	one-shot	ATOMIP
AS1755	2.2264	0	1.8117	0.94	4	0	1	3
AS3257	0.8956	0	1.4416	0.937	3	0	1	3
AS1239	1.8652	0.367	1.5991	1.367	9	1	1	2
Waxman1	2.6210	0	1.43699	0.9612	8	0	1	4
Waxman2	9.2435	1.1267	2.028	1.452	15	3	1	6
Waxman3	5.1142	0.7217	1.918	1.373	9	2	1	5

TABLE III
THE EFFECTIVENESS AND EFFICIENCY OF ATOMIP

#Topo	Updated Flows	Variables		Constraints		_solve_time(s)	
		MIP	R_MIP	MIP	R_MIP	MIP	R_MIP
AS1755	306	984	97	504	35	264.691	0.556
AS3257	702	2153	513	1056	179	27.098	0.412
AS1239	870	2568	359	1224	125	56.795	0.517
Waxman1	2450	6950	617	2850	212	89.345	0.59
Waxman2	5000	15300	794	5900	262	241.312	0.539
Waxman3	7760	22950	1234	8010	435	422.452	1.219

in AS1239 topology shown in Table III, ATOMIP reduced the variables and constraints from 2568 to 359 and from 1224 to 125, respectively, which reduces the total computing time from almost 1 minute to half 1 second. Although the computing the time does not stay linearity with the number of variables and constraints (See the cases of AS1755 and AS3257), ATOMIP gains good performance in most cases in practice.

In a word, the results imply that, ATOMIP is able to find the *near-optimal* scheduling for updates in nearly real-time for practical cases.

V. RELATED WORK

Update in SDN networks occur very frequently while the time when all network devices take the update action cannot be easily synchronized. The unsynchronized update may cause transient loops, packet loss, etc. Lots of work has proposed solutions for scheduling updates in SDN [2]–[5]. The two-phase update mechanism and its improver are proposed by Reitblatt et al in [2] and Luo et al [3], respectively. Theoretically, the path inconsistency problem during the update can be solved by coloring each packet with a version number indicating either the old configuration or the new configuration. The controller is required to install all new rules onto middle switches silently, and then it informs ingress switches to change the version number of packets in order to trigger the upgrade. The transient congestion problem as observed early in this paper is remained to be solved.

In order to smooth the update from the old configuration to the new one, zUpdate [4] proposes a method to compute a sequence of intermediate network configurations in between the initial and final configurations. It's proved that congestion-free operations are always feasible. This method assumes all network devices support flow splitting at any proportion. It introduces a series of intermediate configurations, which may affect those not-to-be-updated flows as well. Dionysus [5] focuses on dynamically scheduling network updates based on the update speeds of switches in order to achieve a fast update process. The schedule mechanism proposed in this paper is orthogonal to all these related work. It honors the old and new configurations given in an SDN update. The traffic migration is achieved by just manipulating the order that ingress network nodes take the update action. No additional configurations are involved.

VI. CONCLUSION

This paper studies how to schedule traffic migration to minimize transient congestions. The scheduling problem is formulated as a Mixed Integer Linear Program (MIP), which would find the congestion-free schedule that needs the minimum rounds if there exists; or find the schedule order that causes the minimum traffic loss otherwise. Since solving the original MIP is time-consuming for a large network, a heuristic named ATOMIP is proposed to recursively split the original update into a series of atomic/small updates, where each subupdate is easy and efficient to schedule by solving an MIP. Evaluation based on real topologies implies that ATOMIP is able to find the near-optimal solution within seconds.

REFERENCES

[1] N. McKeown *et al.*, “Openflow: enabling innovation in campus networks,” *SIGCOMM Comput. Commun. Rev.*, vol. 38, no. 2, pp. 69–74, Mar. 2008.

[2] M. Reitblatt *at al.*, “Abstractions for network update,” in *Proc. ACM SIGCOMM*, 2012, pp. 323–334.

[3] S. Luo *at al.*, “Consistency is not easy: How to use two-phase update for wildcard rules?” *IEEE Commun. Lett.*, vol. PP, no. 99, pp. 1–1, 2015.

[4] H. H. Liu *at al.*, “zupdate: Updating data center networks with zero loss,” in *Proc. ACM SIGCOMM*, 2013, pp. 411–422.

[5] L. Jin *at al.*, “Dynamic scheduling of network updates,” in *Proc. ACM SIGCOMM*, 2014, pp. 539–550.

[6] N. Spring, R. Mahajan, and D. Wetherall, “Measuring isp topologies with rocketfuel,” in *Proc. ACM SIGCOMM*, 2002, pp. 133–145.

[7] B. Waxman, “Routing of multipoint connections,” *IEEE J. Sel. Areas Commun.*, vol. 6, no. 9, pp. 1617–1622, Dec 1988.

[8] X. Wang and M. Ran, “Link weights migration without congestion in ip networks,” in *Proc. 23rd ICCCN*, Aug 2014, pp. 1–8.

[9] M. Zhang, C. Yi, B. Liu, and B. Zhang, “Greente: Power-aware traffic engineering,” in *Proc. IEEE ICNP*, 2010, pp. 21–30.

[10] J. Wu, Y. Kai, H. Yu, and D. Liao, “Link weight design for green ip networks,” in *Asia Communications and Photonics Conference 2013*. Optical Society of America, 2013, p. AF2G.12.