Good Network Updates for Bad Packets

Waypoint Enforcement Beyond Destination-Based Routing Policies

Arne Ludwig[1], Matthias Rost[1], Damien Foucard[1], Stefan Schmid[1,2]
[1] TU Berlin, Berlin, Germany; [2] T-Labs, Berlin, Germany
{aludwig,mrost,dfoucard}@inet.tu-berlin.de, stefan@net.t-labs.tu-berlin.de

ABSTRACT

Networks are critical for the security of many computer systems. However, their complex and asynchronous nature often renders it difficult to formally reason about network behavior. Accordingly, it is challenging to provide correctness guarantees, especially during network updates.

This paper studies how to update networks while maintaining a most basic safety property, *Waypoint Enforcement* (WPE): each packet is required to traverse a certain checkpoint (for instance, a firewall). Waypoint enforcement is particularly relevant in today's increasingly virtualized and software-defined networks, where new in-network functionality is introduced flexibly.

We show that WPE can easily be violated during network updates, even though both the old and the new policy ensure WPE. We then present an algorithm WAYUP that guarantees WPE at any time, while completing updates quickly. We also find that in contrast to other transient consistency properties, WPE cannot always be implemented in a wait-free manner, and that WPE may even conflict with Loop-Freedom (LF). Finally, we present an optimal policy update algorithm OPTROUNDS, which requires a minimum number of communication rounds while ensuring both WPE and LF, whenever this is possible.

Categories and Subject Descriptors

C.2.3 [**Network Operations**]: Network management

General Terms

Algorithms

1. INTRODUCTION

The Software-Defined Networking paradigm enables a logically centralized and programmatic operation of computer networks. The envisioned network operating system has the potential to radically simplify the network management, as well as render the network more flexible: the software controllers can install, update and verify "the paths that packets follow", i.e., the (network) *policies* [2], fast and in a globally consistent manner.

However, today, we do not have a good understanding yet of the limitations of a more dynamic network management in general, and the Software-Defined Network (SDN) paradigm in particular. Over the last years, especially the problem of consistent network updates has received much attention [4, 5, 8, 10]. In a first line of works, initiated by Reitblatt et al. [10], network updates providing strong consistency guarantees have been studied: even during the transition from an old policy π_1 to a new policy π_2, the *Per-Packet Consistency* (PPC) property is ensured, i.e., each packet will either be forwarded according to π_1 (exclusively-)or π_2, but not a combination of both. In a second line of works, initiated by Mahajan and Wattenhofer [8], weaker transient consistency properties have been investigated for *destination-based policies*: during a network update, a packet may be forwarded according to the old policy π_1 at some switches and according to the new policy π_2 at other switches; however, the update still provides more basic transient guarantees, such as *Loop-Freedom* (LF): packets will never be forwarded along a loop. LF can be realized much more efficiently than PPC, and also does not require the tagging of packets.

This paper investigates fast updates for policies which describe *arbitrary* routes and are not destination-based: Indeed, the fact that routing decisions may not only depend on the destination, but also on the source or even the application, constitutes a key advantage of SDN, enabling interesting new opportunities for traffic engineering [14]. Furthermore, arbitrary routes are also attractive because they reduce the dependencies between different paths to the same destination, facilitating even faster network updates.

Moreover, we identify and initiate the study of a new fundamental transient property, namely *Waypoint Enforcement* (WPE). WPE is an important property in

Permission to make digital or hard copies of all or part of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies bear this notice and the full citation on the first page. Copyrights for components of this work owned by others than ACM must be honored. Abstracting with credit is permitted. To copy otherwise, or republish, to post on servers or to redistribute to lists, requires prior specific permission and/or a fee. Request permissions from Permissions@acm.org.
HotNets'14, October 27-28, 2014, Los Angeles, CA, USA.
Copyright is held by the owner/author(s). Publication rights licensed to ACM.
ACM 978-1-4503-3256-9/14/10 ...$15.00.
http://dx.doi.org/10.1145/2670518.2673873 .

today's increasingly virtualized networks where functionality is introduced also in the network core. For example, in security-critical environments (e.g., in a financial institution), it is required that packets traverse certain checkpoints, for instance, an access control function (implemented by e.g., a middlebox [12], an SDN match-action switch [7], or an NFV function [3]), before entering a protected network domain. In other words, in order to prevent a bad packet from entering a protected domain, not only the old policy π_1 as well as the new policy π_2 must ensure WPE, but also any other transient configuration or policy combination that may arise during the network update. So far, waypoints could only be enforced using PPC, which by definition implies that new links can never be used earlier. [8]

Contribution. This paper initiates the study of network update problems where routing policies do not have to be destination-based but can describe arbitrary paths, and where weak transient consistency properties are ensured. In addition, we introduce an important new transient consistency property, namely Waypoint Enforcement (WPE), and show that at the heart of the WPE property lie a number of interesting fundamental problems. In particular, we show the following results:

1. We demonstrate that WPE may easily be violated if no care is taken. Motivated by this observation, we present an algorithm WayUp that provably updates policies in a consistent manner, while minimizing the number of controller interactions.

2. We show that in contrast to other transient consistency properties, such as LF, WPE cannot always be implemented in a wait-free manner, in the sense that the controller must rely on an upper bound estimation for the maximal packet latency in the network. Moreover, the transient Waypoint Enforcement WPE property may conflict with Loop-Freedom LF, in the sense that both properties may not be implementable simultaneously.

3. We present an optimal policy update algorithm OptRounds, which provably requires the minimum number of communication rounds while ensuring both WPE and LF, whenever this is possible.

In order to measure the "speed" of a network update algorithm, we introduce a new metric called the *round complexity*: the number of sequential controller interactions needed during an update. We believe that optimizing the round complexity is natural given the time it takes to update an individual OpenFlow switch today (see, e.g. [5]). Especially for scaling service chains on large NFV-enabled networks, as, e.g., envisioned by the UNIFY project[1], quickly updating policies while guaranteeing correctness will be of importance [9]. Moreover,

[1]http://www.fp7-unify.eu

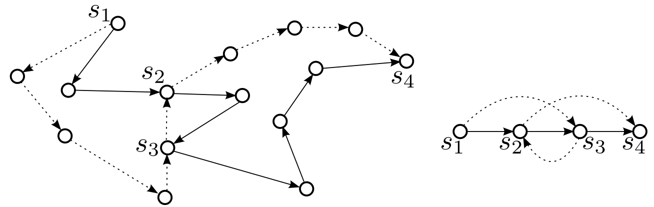

Figure 1: Updating policies describing arbitrary paths: the old policy π_1 is depicted as a *solid line*, the new policy π_2 as a *dashed line*. *Left:* Original situation. *Right:* Collapsed "line representation", where immediately updatable switches have been pruned.

this paper also shows that the optimization of existing metrics, like the number of currently updated links [8], can unnecessarily delay the policy update; even worse, we show that a greedy selection of links may fail to install the policy entirely: a deadlock configuration may occur where the policy installation cannot be completed.

2. EXAMPLE

In order to acquaint ourselves with the problem of fast and consistent network updates fulfilling Waypoint Enforcement (WPE) and Loop-Freedom (LF), in this section, we consider a simple example.

Before presenting the example in detail, we introduce some terminology. In this paper, a *(routing) policy* describes a forwarding path (see also [2, 8, 10]). Routing policies may not only depend on the destination address or port, but also, e.g., on the source or application. Such policies allow us to treat different flows with the same destination independently, which can also reduce the problem complexity. Figure 1 illustrates this point: On the left, two policies, i.e., paths from the network are shown: an old policy π_1 (*solid line*) and a new policy π_2 (*dashed line*). Each switch which is only part of π_2 but not π_1 can be updated immediately, thus reducing the network update problem to the situation presented on the right. In the following, we will make use of this simple line representation, and will depict the old policy, π_1, as a straight line.

With these concepts in mind, we can now introduce our example, see Figure 2 for an illustration. The old policy π_1 connects four switches, from left to right (depicted as a *straight, solid line*, $s_1 \to s_2 \to s_3 \to s_4$); the new policy π_2 is shown as a *dashed line*. The second switch, s_2 (in *black*), represents the waypoint which needs to be enforced.

How can we update the policy π_1 to π_2? A simple solution is to update all switches *concurrently*. However, as the controller needs to send these commands over the asynchronous network, they may not arrive simultane-

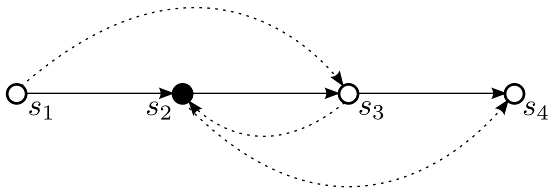

Figure 2: Updating all switches in one round may violate WPE.

ously at the switches, which can result in inconsistent states. For example, if s_1 is updated before s_2 and s_3 are updated, a temporary forwarding path may emerge which violates WPE: packets originating at s_1 will be sent to s_3 and from there to the destination s_4—the waypoint s_2 is *bypassed*.

This illustrates the challenge of updating the SDN, which is an inherently asynchronous and distributed system. One solution to overcome this problem would be to perform the update in *two (communication) rounds*: in the first round, only s_2 and s_3 are updated, and in a second round, once these updates have been performed and acknowledged, the controller also updates s_1. Note that this 2-round strategy indeed maintains the waypoint at any time during the policy update. However, the resulting solution may still be problematic, as it violates another desirable transient consistency property, namely *loop-freedom*: if the update for switch s_3 arrives before the update at switch s_2, packets may be forwarded in a loop, from switch s_2 to s_3 and back.

Both Waypoint Enforcement WPE as well as Loop-Freedom LF can be ensured (for this specific example) in a *three-round* update: in the first round, only s_2 is updated, in the next round s_3, and finally s_1.

We, in this paper, are interested in consistent network updates which are *fast*: (parts of) the new paths should be used as soon as possible during the update. Concretely, we want to minimize the *round complexity* of the policy update: the number of communication rounds where in each round, the controller sends another batch of updates to a subset of switches, and waits for their completion before starting the next round.

3. FAST WAYPOINT ENFORCEMENT

It turns out that the transient enforcement of a waypoint is non-trivial. We first show an interesting negative result: it is not possible to implement WPE in a "wait-free manner", in the following sense: a controller does not only need to wait until the switches have acknowledged the policy updates of round i before sending out the updates of round $i+1$, but the controller also needs some estimate of the maximal packet latency: if a packet can take an arbitrary amount of time to traverse the network, it is never safe to send out a policy update

Algorithm 1: WayUp

1 **Input:** old policy π_1, new policy π_2, threshold θ
2 **update** switches of π_2 which are not in π_1
3 **update** switches of $\pi_1^{>wp}$ with backw. rules in $\pi_2^{<wp}$
4 **update** remaining switches of $\pi_2^{<wp}$
5 **wait** θ
6 **update** switches of $\pi_2^{>wp}$

for certain scenarios. We are not aware of any other transient property for which such a negative result exists. For ease of presentation, we will use the notation $\pi_i^{<wp}$ to refer to the first part of the route given by policy π_i, namely the sub-path from the source to the waypoint, and $\pi_i^{>wp}$ to refer to the second part from the waypoint to the destination.

Theorem 1. *In an asynchronous environment, a new policy can never be installed without risking the violation of WPE, if a switch is part of $\pi_1^{<wp}$ and $\pi_2^{>wp}$.*

Proof. Consider the example in Figure 2 again, but imagine that the waypoint is on switch s_3 and not on switch s_2, and assume the following update strategy: in the first round, s_1 and s_3 are updated, and in the second round, s_2. This strategy clearly ensures WPE, *if (but only if)* the updates of round 2 are sent out after packets forwarded according to the rules before round 1 have left the system. However, if packets can incur arbitrary delays, then there might always be packets left which are still traversing the old (*solid*) path from s_1 to s_2. These packets have not been routed via the waypoint (s_3) so far but will be sent out to s_4 by s_2 in the new path, violating the WPE property. This problem also exists for any other update strategy. □

Fortunately, in practice, packets do not incur arbitrary delays, and Theorem 1 may only be of theoretical interest: it is often safe to provide an update algorithm with some good upper bound θ on the maximal packet latency. The upper bound θ can be seen as a parameter to tune the safety margin: the higher θ, the higher the probability that any packet is actually waypoint enforced.

With these concepts in mind, we now describe our algorithm WayUp which always ensures correct network updates, i.e., updates which consistently implement WPE if the maximal packet transmission time is bounded by θ. We define $s_1 \prec_{\pi_i} s_2$ to express that a switch s_1 is visited before s_2 on π_i. An update rule (s_2, s_1) with $s_1 \prec_{\pi_1} s_2$ is called a *backward rule* (with respect to the initial direction of the line).

The round complexity of WayUp is *four*: in the first round, all switches are updated which were not part of the old policy π_1, and therefore do not have an impact on current packets (as shown in Figure 1). In the second

round, each switch behind the waypoint (i.e., $\pi_1^{>wp}$) which is part of $\pi_2^{<wp}$ and which has a backward rule, is updated. This allows us to update the remaining switches from $\pi_2^{<wp}$ in the third round, since each packet which is sent "behind" the waypoint will eventually come back, according to the consistency properties of the new policy. After this round, the algorithm will wait θ time to ensure that no packet is on $\pi_1^{<wp}$ anymore. In the fourth round it is possible to update all switches of $\pi_2^{>wp}$ in one round, because the update cannot interfere with $\pi_2^{<wp}$ anymore, and hence it cannot violate WPE.

THEOREM 2. WAYUP *takes four rounds and guarantees the WPE property at any time.*

PROOF. The round complexity follows from the algorithm definition. The transient consistency can be proved line-by-line: Line 2 of Algorithm 1 cannot violate WPE since no packet is crossing any of these switches. Line 3 does not interfere with $\pi_1^{<wp}$ and therefore each packet will still be sent via $\pi_1^{<wp}$ towards the waypoint. As long as π_2 is consistent, any packet that reaches any switch of $\pi_2^{<wp}$ will eventually reach the waypoint during Line 4, since all backward rules are already updated and no rule will bypass them. In Line 6, WPE is already guaranteed, since $\pi_2^{<wp}$ is already in place and θ time has elapsed. □

4. INCORPORATING LF

The update strategy presented in the previous section provably fulfills WPE at any time. However, it may violate LF. Sometimes, this may not be a problem: if a network update is relatively fast, the number of packets ending up in a transient loop may be limited; moreover, Time-To-Live (TTL) mechanisms could be used to remove the trapped packets quickly. Nevertheless, it is sometimes desirable to maintain LF as well.

This section also starts with a negative result: WPE and LF may conflict, i.e., it is sometimes impossible to simultaneously guarantee both properties.

THEOREM 3. *WPE and LF may conflict.*

PROOF. Consider the example depicted in Figure 3. Clearly, the source s_1 can only be updated once s_4 is updated, otherwise packets will be sent to s_5 directly, which violates WPE. An update of s_4 can only be scheduled after an update of s_3 without risking the violation of LF. However, s_3 needs to wait for s_2 to be updated for the same reasons. This leaves an update of s_2 as the last possibility, which however violates WPE again. Hence there is no update schedule which does not violate either WPE or LF. □

Fortunately, in practice, such conflicts can be identified, and if they exist, can be resolved with other mechanisms (e.g., by sacrificing speed and using the

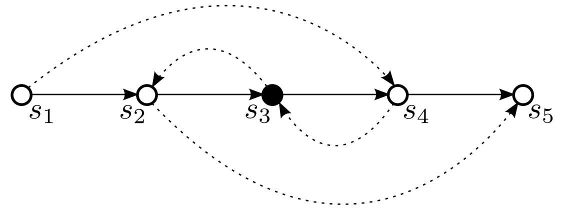

Figure 3: WPE and LF may conflict.

PPC algorithm described in [10]). In the following, we will focus on algorithms which find efficient policy updates for scenarios where WPE and LF do not conflict. A naive approach to find such a consistent update may be to split the update into two distinct parts: the part *before* and the part *after* the waypoint, i.e., $\pi_2^{<wp}$ and $\pi_2^{>wp}$, and use a LF update algorithm on both parts (e.g., the strategy in [8]). Unfortunately, this approach can fail: only if $\pi_2^{>wp}$ has no overlaps with $\pi_1^{<wp}$ and vice versa, and if $\pi_2^{<wp}$ has no overlaps with $\pi_1^{>wp}$, it is safe to update both paths in parallel. This result even holds for a consecutive update of $\pi_2^{<wp}$ and $\pi_2^{>wp}$. The new policy shown in Figure 2 cannot be updated without inducing loops on $\pi_1^{>wp}$ (i.e., $s_2 \to s_3 \to s_2$) if $\pi_2^{<wp}$ is updated first; and a similar example can be constructed for a schedule where $\pi_2^{>wp}$ is updated first.

In general, an update of a switch violates LF whenever the update leads to a loop. However this loop might not be part of the current path from the source to the destination, and not forward policy-relevant packets. To ensure that there are no packets left which were forwarded by the old policy, we use the concept of a maximal latency θ introduced in Section 3. Assume that the example shown in Figure 3 is only a part of a policy update and that s_3 is not the waypoint. If s_1 and s_2 have been updated in a previous round, then an update of s_3 does not violate LF after time θ.

Comparing Objectives.

The speed of a network update is measured in terms of communication rounds in this paper: the number of times a controller needs to send updates to a subset of network elements. As we show in the following, previous objectives [8] which greedily maximize the number of updated links in a *single* round may (1) unnecessarily delay the policy update, namely by a factor up to $\Omega(n)$ and (2) fail to find a valid update schedule if there exists one. Let us refer to the max link objective by O_1, and to the min round objective by O_2. Figure 4 and Figure 5 show the construction of a worst-case scenario: a greedy update of all possible switches in the first round (hence optimizing O_1), leads to a situation where from the third round onward, only one switch can be updated per round. The first round would include every link which is a forward link. The second round includes

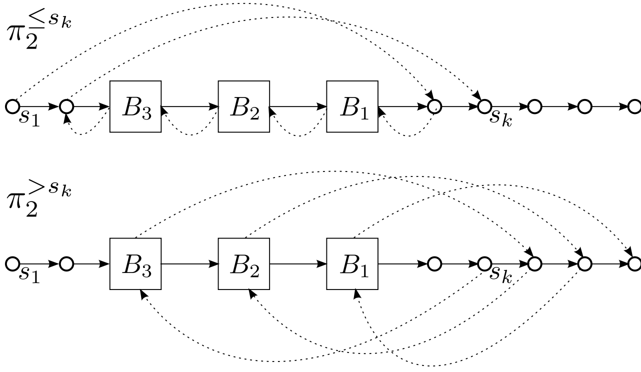

Figure 4: Pattern (*here:* consisting of three blocks B) for updates which result in a high number of rounds, if updated according to O_1. The blocks B_i are shown in detail in Figure 5. The notation $\pi^{\leq s_k}$ and $\pi^{>s_k}$ is used to refer to the switches left (including s_k) resp. right from s_k.

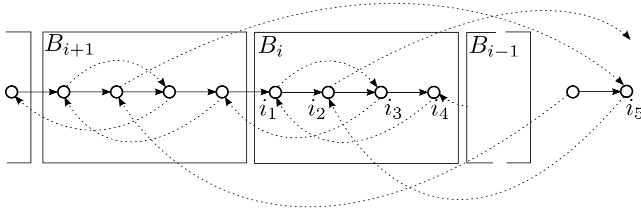

Figure 5: Pattern of a block used for the example in Figure 4.

all backward links shown in the lower part of Figure 4, and the first backward link from B_3. In the upcoming rounds only one switch at a time can be updated. The upper part of Figure 4 shows the first part of π_2 which is basically a chain of *backward* links between blocks B_i. Each block B_i includes two of these interleaved backward rules. Each of these switches is dependent on an updated successor, before it can be updated itself without violating LF.

The addition of every block (four switches) and the switch pointing to it, increases the number of rounds by two. This leads to $2n/5$ rounds. An update minimizing the number of rounds (objective O_2) would not update the switches characterized by switch i_1 in Figure 5 in the first round, even though this would be possible. This breaks the long dependency chain in each block and leads to a fixed number of rounds (namely four), independent of n. Therefore, maximizing the number of updates per round can take up to $n/10$ more rounds.

Even worse, an algorithm optimizing the objective O_1 may not only increase the number of rounds, but it may fail to find a valid solution entirely. The only possible updates for the scenario in Figure 6 are the updates s_1 and s_2. Updating both of them leads to a

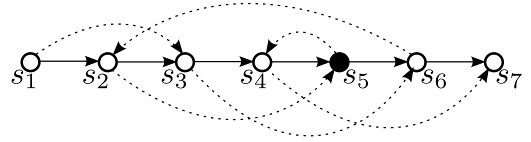

Figure 6: Pattern of a block used for the example in Figure 4.

situation where no more switches can be updated, since they are either violating WPE (s_3 and s_4) or LF (s_5 and s_6). The only possible update schedule delays s_1 and only updates s_2.

5. EXACT ALGORITHM

In the following, we present the Mixed-Integer Program (MIP) OptRounds, which generates an update scheme requiring the minimal number of rounds. According to the line representation presented in Section 2, policies π_1 and π_2 are described as (simple) paths E_{π_1} and E_{π_2} on the common set of switches V. Both E_{π_1} and E_{π_2} connect the start switch $s \in V$ to the target switch $t \in V$.

As the task of the MIP 1 is to find the minimal number of rounds, we generally allow for $|V| - 1$ many rounds, denoted as $\mathcal{R} = \{1, \ldots, |V|-1\}$. We use binary variables $x_v^r \in \{0, 1\}$ to indicate whether the forwarding policy of switch $v \in V$ is updated in round $r \in \mathcal{R}$ or not. Constraint 2 enforces the switching policy of each switch to be changed in exactly one of the rounds. The objective to minimize the number of rounds is realized by minimizing $R \geq 0$ which is lower bounded by all the rounds in which an update is performed (see Constraint 1).

Given the assignment of switch updates to rounds, the Constraints 3 and 4 set variables $y_e^r \in [0, 1]$ accordingly to indicate whether the edge $e \in E_{\pi_1} \cup E_{\pi_2}$ is contained after the successful execution of updates in round $r \in \mathcal{R}$. In the following we show how to enforce both the LF and the WPE properties.

Enforcing LF.

To enforce the LF property, we need to guarantee transient states between rounds to be loop-free. To this end, we first define variables $a_v^r \in \{0, 1\}$ to indicate whether a switch $v \in V$ may be reachable or *accessible* from the start $s \in V$ under any order of updates between rounds $r - 1$ and r. The variables are set to 1 if, and only if, there exists a (simple) path from s towards $v \in V$ using edges of either the previous round or the current round (see Constraints 5 - 7). Similarly, and based on this reachability information, the variables $y_{u,v}^{r-1 \vee r} \in \{0, 1\}$ are set to 1 if the edge $(u, v) \in E$ may be used in transient states, namely if the edge existed in round $r-1$ or r and u could be reached (see Constraints 8 and 9). Lastly, to ensure that a flow cannot be forced onto loops, we employ well-known Miller-Tucker-Zemlin

Mixed-Integer Program 1: Optimal Rounds

$$\min R \qquad \text{(Obj)}$$
$$R \geq r \cdot x^r_v \qquad r \in \mathcal{R}, v \in V \quad (1)$$
$$1 = \sum_{r \in \mathcal{R}} x^r_v \qquad v \in V \quad (2)$$
$$y^r_{u,v} = 1 - \sum_{r' \leq r} x^r_u \qquad r \in \mathcal{R}, (u,v) \in E_{\pi_1} \quad (3)$$
$$y^r_{u,v} = \sum_{r' \leq r} x^r_u \qquad r \in \mathcal{R}, (u,v) \in E_{\pi_2} \quad (4)$$
$$a^r_s = 1 \qquad r \in \mathcal{R} \quad (5)$$
$$a^r_v \geq a^r_u + y^{r-1}_{u,v} - 1 \qquad r \in \mathcal{R}, (u,v) \in E \quad (6)$$
$$a^r_v \geq a^r_u + y^r_{u,v} - 1 \qquad r \in \mathcal{R}, (u,v) \in E \quad (7)$$
$$y^{r-1 \vee r}_{u,v} \geq a^r_u + y^{r-1}_{u,v} - 1 \qquad r \in \mathcal{R}, (u,v) \in E \quad (8)$$
$$y^{r-1 \vee r}_{u,v} \geq a^r_u + y^r_{u,v} - 1 \qquad r \in \mathcal{R}, (u,v) \in E \quad (9)$$
$$y^{r-1 \vee r}_{u,v} \leq \frac{l^r_v - l^r_u - 1}{|V| - 1} + 1 \qquad r \in \mathcal{R}, (u,v) \in E \quad (10)$$
$$\overline{a}^r_s = 1 \qquad r \in \mathcal{R} \quad (11)$$
$$\overline{a}^r_v \geq \overline{a}^r_u + y^{r-1}_{u,v} - 1 \qquad r \in \mathcal{R}, (u,v) \in E_{\overline{\mathsf{WP}}} \quad (12)$$
$$\overline{a}^r_v \geq \overline{a}^r_u + y^r_{u,v} - 1 \qquad r \in \mathcal{R}, (u,v) \in E_{\overline{\mathsf{WP}}} \quad (13)$$
$$\overline{a}^r_t = 0 \qquad r \in \mathcal{R} \quad (14)$$

constraints (see 10) with corresponding leveling variables $l^r_v \in [0, |V|-1]$: if traffic may be sent along edge $(u,v) \in E$, i.e., if $y^{r-1 \vee r}_{u,v} = 1$ holds, $l^r_v \geq l^r_u + 1$ is enforced, thereby not allowing for cyclic dependencies.

Enforcing WPE.

For enforcing WPE a similar reachability construction is employed (cf. Constraints 5 - 7). We define variables $\overline{a}^r_v \in \{0,1\}$ indicating whether switch $v \in V$ can be reached from the start without passing the waypoint. To this end, we introduce the set of all edges $E_{\overline{\mathsf{WP}}} \subset E$ not incident to the waypoint and propagate reachability information only along these edges (see Constraints 11-13). Lastly, Constraint 14 ensures that no packet can arrive at t without passing the waypoint.

6. INITIAL COMPUTATIONAL RESULTS

In our computational evaluation[2] we are interested in the number of scenarios in which no solution for an update schedule can be found. This is either due to conflicting WPE and LF (see Theorem 3) or due to the chosen objective (as shown in Section 4). Figure 7 shows the percentage of solvable scenarios as a function of the problem size in terms of switches. Every scenario which could be solved for objective O_1 (max link) could also be solved for objective O_2 (min rounds) via OptRounds. OptRounds also finds approximately 10% additional solutions for problem sizes of $18-30$ switches. For smaller instances the number of additional solutions is decreasing due to a smaller probability for deadlocks.

[2]See http://www.net.t-labs.tu-berlin.de/~stefan/netup.tar.gz for scenarios and results

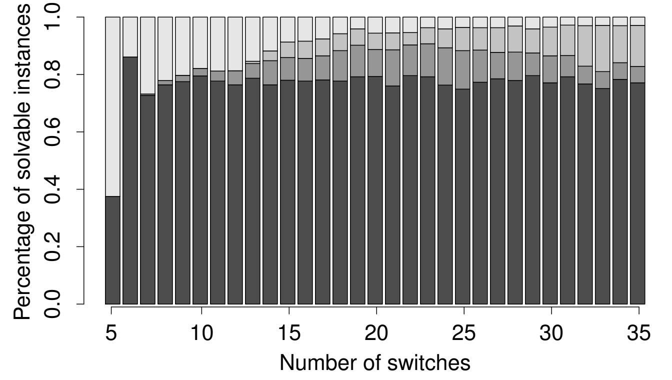

Figure 7: Percentage of solvable scenarios per number of switches (1000 scenarios per size). The bars indicate from dark to bright: (1) Solvable regarding O_1; (2) additionally solvable with O_2 within 600 seconds; (3) not classified after 600 seconds; (4) not solvable.

The decrease for larger instances is a consequence of the capped runtime and the increased problem size which leads to a larger fraction of unclassified instances.

A similar trend can be observed for the total amount of solvable instances which is slightly increasing with the scenario size in the beginning. The percentage of solvable instances is always above 70% (except for the scenario with 5 nodes), and increases towards $85-90\%$ for scenarios with $14-31$ switches.

Whenever OptRounds terminated within 600 seconds, its median runtime was less than 1 second for networks with less than 15 switches. The median runtime for solvable scenarios increases roughly linearly in the number of switches (90 seconds for 35 switches), and stayed below 1 second for not solvable scenarios.

The simulations show that there is a significant fraction of unsolvable scenarios. A possibility to solve these scenarios without violating WPE and LF is by introducing additional edges, so-called *helper rules* [8]. However these can only be utilized when an independent path exists, i.e., a path which does not interleave with the new and the old path from the source to the waypoint.

7. CONCLUSION

We believe that our paper opens a rich and interesting area of research, and the presented fast and independent network update algorithms may also be of interest for the design of more distributed control planes [1, 6, 11, 13]. Accordingly, we understand our work as a first step towards a better understanding of the consistent updates of more complex network policies, which include various network functionality.

Acknowledgments. This research was supported by the BMBF (01IS12056) and the EU FP7 UNIFY project, which is partially funded by the Commission of the European Union.

8. REFERENCES

[1] M. Canini, P. Kuznetsov, D. Levin, and S. Schmid. Software transactional networking: Concurrent and consistent policy composition. In *Proc. ACM SIGCOMM Workshop on Hot Topics in Software Defined Networking (HotSDN)*, August 2013.

[2] M. Casado, M. J. Freedman, J. Pettit, J. Luo, N. McKeown, and S. Shenker. Ethane: Taking control of the enterprise. In *Proc. ACM SIGCOMM*, 2007.

[3] ETSI. Network Functions Virtualisation – Introductory White Paper. 2012.

[4] C.-Y. Hong, S. Kandula, R. Mahajan, M. Zhang, V. Gill, M. Nanduri, and R. Wattenhofer. Achieving high utilization with software-driven wan. In *Proc. ACM SIGCOMM*, 2013.

[5] X. Jin, H. Liu, R. Gandhi, S. Kandula, R. Mahajan, J. Rexford, R. Wattenhofer, and M. Zhang. Dionysus: Dynamic scheduling of network updates. In *Proc. ACM SIGCOMM*, 2014.

[6] T. Koponen, M. Casado, N. Gude, J. Stribling, L. Poutievski, M. Zhu, R. Ramanathan, Y. Iwata, H. Inoue, T. Hama, and S. Shenker. Onix: A distributed control platform for large-scale production networks. In *Proc. USENIX OSDI*, 2010.

[7] D. Levin, M. Canini, S. Schmid, F. Schaffert, and A. Feldmann. Panopticon: Reaping the benefits of incremental sdn deployment in enterprise networks. In *Proc. USENIX Annual Technical Conference (ATC)*, 2014.

[8] R. Mahajan and R. Wattenhofer. On Consistent Updates in Software Defined Networks. In *Proc. 12th ACM Workshop on Hot Topics in Networks (HotNets)*, 2013.

[9] P. Skoldstrom et al. Towards unified programmability of cloud and carrier infrastructure. In *Proc. European Workshop on Software Defined Networking (EWSDN)*, 2014.

[10] M. Reitblatt, N. Foster, J. Rexford, C. Schlesinger, and D. Walker. Abstractions for network update. In *Proc. ACM SIGCOMM*, pages 323–334, 2012.

[11] S. Schmid and J. Suomela. Exploiting locality in distributed sdn control. In *Proc. ACM SIGCOMM Workshop on Hot Topics in Software Defined Networking (HotSDN)*, 2013.

[12] V. Sekar, S. Ratnasamy, M. K. Reiter, N. Egi, and G. Shi. The middlebox manifesto: Enabling innovation in middlebox deployment. In *Proc. 10th ACM Workshop on Hot Topics in Networks (HotNets)*, 2011.

[13] A. Tootoonchian, S. Gorbunov, Y. Ganjali, M. Casado, and R. Sherwood. On controller performance in software-defined networks. In *Proc. USENIX Hot-ICE*, 2012.

[14] P. Vicente and L. Rodrigues. SDX: A Software Defined Internet Exchange. In *Proc. USENIX Open Networking Summit (ONS)*, 2013.