Distributed Consistent Network Updates in SDNs: Local Verification for Global Guarantees

Klaus-Tycho Foerster · Faculty of Computer Science, University of Vienna, Austria

Stefan Schmid · Faculty of Computer Science, University of Vienna, Austria

Abstract. While SDNs enable more flexible and adaptive network operations, (logically) centralized reconfigurations introduce overheads and delays, which can limit network reactivity. This paper initiates the study of a more distributed approach, in which the consistent network updates are implemented by the switches and routers directly in the data plane. In particular, our approach leverages concepts from local proof labeling systems, which allows the data plane elements to locally check network properties, and we show that this is sufficient to obtain global network guarantees. We demonstrate our approach considering three fundamental use cases, and analyze its benefits in terms of performance and fault-tolerance.

1 Introduction

Given the increasingly stringent requirements on the dependability and performance of communication networks, it becomes important that networks be able to flexibly adapt to their context, e.g., react to failures or to changes in the demand, in an *automated* manner. Software-Defined Networks (SDNs) provide such flexibilities by allowing to update network configurations programmatically, disburdening human operators from their most complex tasks and significantly improving reaction times. Indeed, over the last years, the algorithmic problem of how to update networks consistently has received much attention [1].

However, while outsourcing and consolidating the control over switches and routers provides great flexibilities, indirection via (remote) controllers comes with overheads in terms of communication and computation costs, and can hence lead to delays. In fact, it is known that updating routes in a network while providing even simple transient properties such as loop-freedom, requires many interactions with the SDN controller in the worst case [2, 3], unless one resorts to packet header rewriting. Given that the control plane can operate orders of magnitude slower than the data plane [4], this is problematic.

This paper investigates opportunities to overcome these overheads and hence further improve network reactivity. To this end, we explore a more *distributed* approach to updating routes in networks, reducing interactions with the control plane *without* sacrificing flexibility and consistency. This is challenging, as without a (logically) centralized network view, switches and routers need to be able to check certain network properties *locally*.

We propose and investigate the use of distributed mechanisms based on local proof labeling systems [5], to propagate and implement network updates *entirely in the data plane*. In particular, we present a solution which allows switches and routers to check *locally* if a certain network property is fulfilled and whether a rule update can be safely applied. Consequently, a controller (or multiple controllers, in case of distributed SDN control planes) can simply submit update requests to the network, which are then propagated and implemented by the data plane autonomously. To demonstrate our approach, we consider two fundamental properties, both related to connectivity.

- *Blackhole freedom:* There is always a matching rule forwarding a packet to the next hop switch or router.
- *Loop freedom:* The forwarding rules never contain a loop.

We also evaluate the benefits of our approach analytically and investigate potential speed ups and fault-tolerance.

Contributions. This paper presents a distributed approach to operate and consistently update software-defined networks, by relying on local proof labeling systems. We show the feasibility and benefits of our approach on two case studies, demonstrating that using our approach, simple local verification is sufficient to provide global correctness guarantees. We also show that our approach can lead to faster and fault-tolerant network updates.

Overview. The remainder of this paper is organized as follows. After introducing our model and preliminaries in Sections 2, we present our main approach in Section 3. We discuss our two case studies in Section 4 (considering efficient updates limited to the affected routes) and Section 5 (removing the need for packet tagging), and then examine potential speed up and fault-tolerance aspects (Section 6). After reviewing related work in Section 7, we conclude in Section 8.

2 Model and Preliminaries

We follow standard assumptions [5, 6, 7] in our work, both regarding the network and the local verification model.

Network model. The considered networks are modeled as connected graphs $G = (V, E)$ with n nodes (switches, routers) with unique identifiers and m full-duplex links.

The network is equipped with a logically centralized controller that can collect the network state and send out conditional network updates to the nodes, *e.g.,* changing a forwarding rule once a certain local condition is met [7].

Local Verification. We will also leverage a connection [5] between proof labeling schemes [8, 9] and the SDN model [1], see Section 3. A proof labeling scheme can be characterized by a prover-verifier-pair $(\mathcal{P}, \mathcal{V})$ as follows: Given some property $\mathcal{S}$ that the network state could uphold after updates (*e.g.,* loop freedom), the prover $\mathcal{P}$ sends new labels to the nodes. The verifier $\mathcal{V}$ is a distributed algorithm, running on each node v, that can collect the labels from all neighbors $\mathcal{N}(v)$. It outputs YES if property S holds and the labels are from $\mathcal{P}$, but at least one node must output NO, if the property S is violated.

3 Approach and Main Idea

This section presents how to leverage proof labeling schemes in the context of consistent updates for SDNs, both from a methodological and an implementation point of view.

Methodology. Many consistency properties are inherently global, *e.g.,* long loops cannot be detected by considering the forwarding rules in the local neighborhood. Even locally detectable problems can have an impact on nodes far away, such as, *e.g.,* a blackhole downstream from the packet source.

We thus utilize the power of proof labeling schemes to allow for *local* verification of consistency properties, also supporting *distributed* consistent network updates. In our approach, the controller acts as the prover $\mathcal{P}$. Nodes which are aware of the current label state of their neighbors, can now check them in the time intervals deemed necessary. In the simplest case, a node informs all its neighbors once its label state changes.

Once being informed about label state changes, nodes can run the verifier $\mathcal{V}$ to check if the (global) property S is still correct, respectively ring an alarm (*e.g.,* to the controller) if not. The main idea of our approach is that a node will not immediately apply a new label received from the controller, but rather first check if the property S still holds from its point of view after applying said label to itself. As such, we do not need the large overhead of constantly communicating with the centralized controller regarding the updated network state, but can decide completely locally when to update.

The challenge we undertake in this paper is to actually develop approaches that fulfill these criteria for common consistency properties, *i.e.,* generating distributed consistent network updates that can be verified locally.

Implementation. Our approach is timely and can be implemented in OpenFlow and P4-based programmable networks. The implementation of the controller is simple as it only pre-computes the information needed by the switches later, during the network update (reducing communication and computation overheads). Furthermore, our approach does not rely on tight clock synchronization protocols while providing the same benefits [10]. In the dataplane, we can use the approach by ez-Segway [7], leveraging per-switch local controllers to manipulate dataplane state (via OpenFlow).

4 Efficient Certification Limited to Involved Routes

We first present a solution for efficient certification which only involves the nodes along routes that are actually updated (rather than *all* nodes in the network).

We start with a case study on the *blackhole freedom* property. A so-called blackhole occurs when a node has no matching rule for a packet, *i.e.,* the packet is dropped (into a blackhole). A simple scheme to avoid blackholes for a specific network flow is to ensure that new labels for a node v always contain a matching

rule for the flow destination d, where an update is rejected otherwise. However, whereas this scheme is easy to verify and apply, it suffers from the downside that every node in the network must have a forwarding rule for said flow, even if its packets only traverse a small subset of the nodes.

A more efficient solution would supply forwarding rules only to those nodes actually en route, as performed *e.g.,* in [7] for network flows. The authors propose a distributed version of the 2-phase update scheme by *e.g.,* Reitblatt *et al.* [11][1]: the routing path for flow F is updated in reverse, where the destination informs its predecessor on the path to update its rules for F', which in turn informs its predecessor, and so on. Eventually, the packet source will be reached, which then knows it is safe to send packets out tagged with F'.

Providing verifiable blackhole freedom can be directly achieved in this setting if every node v with a new rule for F' only updates if its successor w on the path has been updated. Notwithstanding, what cannot be verified so far is the problem of reachability, *i.e.,* will the packets in F' actually reach their target? In the prover-verifier framework, if each node is informed about its successor, a node w could be successor of two nodes u, v, which in turn can lead to a forwarding loop.

We can resolve this problem with a construction borrowed from reachability in the context of proof labeling schemes [9], by specifying both predecessors and successors of all nodes (besides source and destination). Then, by a connectivity argument, the packets of F' cannot loop and will reach the destination when starting from the source.

While we now have verifiable blackhole freedom for the nodes en route, we cannot use the above scheme to actually deploy a new path for F'. Assume that the path has at least two nodes besides the source and the destination, then no further node en route can actually deploy the rules for F' under common asynchrony [1] assumptions—both a successor and predecessor along the route is needed.

Moreover, from a structural point of view, such a predecessor-successor construction does not remove unnecessary forwarding loops in the network, *e.g.,* a loop disconnected from source/destination cannot be locally detected. While such disconnected loops might not seem as harmful from a routing point of view, they can hinder future updates and also highlight another downside of the above scheme, namely that it is not suitable for purely destination-based schemes, where routing is performed along a forwarding tree. We investigate such scenarios in the next section, but first show how to fix our proposed scheme.

To this end, we replace the predecessor-successor relationship with a distance labeling scheme, as described in, *e.g.,* [8, 9]. Each node along the path of F' also obtains its distance to the destination as part of the label, measured in hops along F'. Then, a node will only update if its successor has already updated and its distance is exactly one less. A counting-to-zero argument can be used to show the correctness of this scheme w.r.t. blackhole and loop freedom, as $a)$ only the destination may have a distance of zero and $b)$ the source only starts to utilize F' once the path has been established.

Theorem 1. *The reverse update scheme in [7] for flows can be made locally verifiable for both blackhole and loop freedom by enhancing it with distance labelings.*

5 Removing the Need for Packet Tagging

It is sometimes possible to remove the need for packet tagging (as required by the approach above), and hence also reduce the number of rules to be stored by the nodes (as they are often per-tag), by slightly relaxing the notion of consistency. Observe that in the previous section, our approach moreover guaranteed so-called per-packet consistency [11], where a packet will either take the old F or the new F' path, but never a mix of both. However, such stronger guarantees are not needed in order to guarantee blackhole and loop freedom.

[1]The 2-phase commit scheme in [11] updates the forwarding for a flow F to F' as follows: The new flow rules for F' are distributed in the network, and once ack'ed to the controller, the controller informs the packet source to from now on tag all flow packets with F', instead of the previous tag of F.

We assume as such that routing is to be performed destination-based along forwarding trees, which in turn have to be blackhole/loop-free. It was already observed in [6] that consistency in this setting can be verified and consistently updated by including the depth of the node v in the forwarding tree in its label. As such, specifying the parent and the depth suffices. In a nutshell, a node v waits until its parent w updates, and then only updates if $\text{DEPTH}(v)=\text{DEPTH}(w)+1$ is satisfied.

A downside of the above scheme is that it only specifies a single transition from old to new forwarding rules. In order for a second and further updates to be performed, the controller needs to again collect acknowledgments that all nodes have switched, inducing unnecessary overhead. In the previous section and in 2-phase commit schemes in general, one can just create a new tag to avoid such issues, *e.g.,* transitioning from F to F' to F'' and so on. Even if F' is never fully implemented, the packet source can transition to F'' once its path is fully provisioned.

It seems at first as if the trick of adding increasing version numbers cannot be directly applied to forwarding trees. In network flows, there is a single node (the source) from which the traffic along the new path originates, whereas in forwarding trees, all nodes can act as sources, potentially sending across combinations of different forwarding trees (in [6]: just 2 trees).

However, instead of waiting for the last update to be completed, we can actually mix different subsequent updates, as long as in each intermediate possible time-step the forwarding is performed along a forwarding tree.[2] As such, we add version numbers to each label and observe that we only need to obey a larger-than relationship: as long as any of v's neighbors w is a parent in some larger version number x, v may switch to its label (tree) with version x if $\text{DEPTH}_x(v)=\text{DEPTH}_x(w)+1$. Observe that a node can also skip intermediate labels.

Correctness is guaranteed by the invariant that a node will never switch to a smaller version number.[3] Nodes using the largest version number form a correct forwarding tree, as they will not forward to nodes in other trees and in each step reduce the distance to the destination. Next, observe that for all other forwarding trees (version numbers), the next routing hop will decrease the distance in the label of the parent, respectively switch to a higher version number. Hence, the packet will reach the destination eventually and loops in the current forwarding state can be locally detected as well: Assume for the sake of contradiction that the forwarding graph contains some loop with no node ringing an alarm (outputting NO). As every node outputs YES, we can follow the routing loop starting from some node u, where in each step, we increase the version number or reduce the distance. However, when we reach u again[4], u must either have a smaller depth or a higher version number than itself, a contradiction.

Theorem 2. *By augmenting the update scheme from [6] with version numbers, s.t. a node v may update to larger version numbers x, if its respective parent w in x is also in version number x and* $\text{DEPTH}_x(v)=\text{DEPTH}_x(w)+1$*, we obtain a locally verifiable scheme which preserves blackhole and loop-freedom.*

6 Discussion: Speed Up and Fault-Tolerance

Potential speed up gains. Nguyen *et al.* [7] showed in their evaluations that decentralized consistent updates can speed up updates by up to 45% at the median, in realistic scenarios.

We briefly analyze what sort of theoretical speed up is possible in extreme cases, from the viewpoint of message propagation delay, where we assume one hop to take unit time.

Consider the scenario analogously to [12, Fig. 2], shown in Figure 1. The task is to update from the old (solid) to the new (dashed) forwarding rules for the destination d in a loop-free fashion. In a centralized

[2] Note that loop freedom is a structural property of the forwarding graph.

[3] For practical purposes, an appropriate circular ordering could be defined.

[4] As we study a structural property, we assume no updates in the meantime.

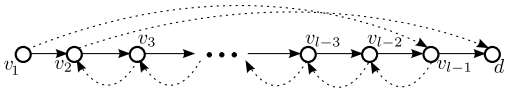

Figure 1: Network with old (solid) and new (forwarding) rules which requires $l - 2 \in \Omega(n)$ rounds to update consistently when enforcing loop freedom. For example, v_3 cannot update before v_2, and so on.

setting, we need $\Omega(n)$ rounds to complete the migration, as only one rule (once: two) can be updated per round [12]. Else, asynchrony could lead to transient loops in the forwarding graph.

While it is impossible to break the $\Omega(n)$ different updates lower bound, distributed updates can drastically improve the message propagation delay overhead. Assume that the controller is connected to or placed on any arbitrary node. In a distributed setting, the controller can pipeline the distribution of the update labels, reaching all nodes in $O(n)$ time. Next, the update messages propagate one hop, each along the new forwarding rules, again taking $O(n)$ time. In contrast, in a centralized setting, the controller needs to obtain an acknowledgement of each update, in turn sending out the next update command. In total, this requires a message propagation delay of $\Omega(n^2)$.

Observation 1. *Distributed updates can speed up the update process by a factor of $O(n)$, w.r.t. message propagation delay.*

Fault-tolerance. Fault-tolerance is largely unexplored w.r.t. proof labeling schemes, the only work that we are aware of relies on a global (unspecified) notification that an error occurred [13], investigating a single link failure. On the other hand, there is also work that studies so-called *local fixing*, where nodes/links can *e.g.,* leave last wills behind in order to restore properties [14]. However, such fixing is not studied from the aspect of verification, to the best of our knowledge.

Interestingly, we can create a heuristic that directly extends our constructions from the last section to fault-tolerance. For destination-based routing, observe that we do not need to forward to a node with a depth exactly one smaller, but any smaller depth (or higher version) would suffice. In this context, fault-tolerance could benefit benefit from link-disjoint forwarding trees [15], which can be computed efficiently [16], along with appropriate optimization for route lengths [17, 18].

7 Related Work

Proof labeling schemes have been widely studied in the context of distributed computing. We take inspiration from, *e.g.,* [8, 9], and also refer to both articles for an introduction to the topic. Similarly, the topic of consistent network updates in SDNs has received much attention in the networking community, see the recent survey in [1]. The idea to leverage proof labeling schemes for verification purposes in SDNs was first investigated in [5], joined with consistent updates for destination-based routing in [6]. We extend the ideas in [6] by handling multiple subsequent updates and also covering flow-based routing, along with speed ups and fault-tolerance.

Nguyen *et al.* [7] lay the practical groundwork for our paper, by showing how to efficiently implement consistent SDN updates in the data plane. We build upon their work by adding local verification to blackhole and loop-free consistent updates, leveraging the concepts of proof labeling schemes.

Lastly, the idea of fault-tolerance in proof labeling schemes was considered in [13], but in contrast required an explicit (unspecified) global failure notification. Related in this context is also the idea of local fixing [14] or preprocessing in distributed control planes in general [5, 19, 20].

8 Conclusion

Given the constantly changing demands and requirements on communication networks, e.g., due to security policy changes, traffic engineering requirements, planned maintenance work or unplanned link failures, among many more, future communication networks are expected to be changed and reconfigured more frequently. This paper presented a distributed approach, based on proof labeling systems, which allows to offload the responsibility for network reconfigurations to the data plane and hence support and speed up such reconfigurations.

We understand our work as a first step, and believe that it opens several interesting avenues for future research. In particular, it will also be interesting to consider the use of randomized [21] and approximate [22] solutions to improve our approach, provide extensions to further consistency properties such as waypoints [23] and congestion [24], as well as seamless updates [25], but also the inherent connections to self-stabilization [26]. More generally, we believe that our approach can provide interesting new perspectives on emerging self-driving networks [27], which center around fine-grained and fast adaptions of networks reacting to their environment, and may hence benefit from our distributed approaches. Furthermore, it will be interesting to investigate opportunities coming from emerging programmable dataplanes, to speed up our approach further, as well as to generalize it to additional use cases.

References

[1] K.-T. Foerster, S. Schmid, and S. Vissicchio, "Survey of consistent software-defined network updates," *IEEE Communications Surveys Tutorials*, vol. 21, no. 2, pp. 1435–1461, 2019.

[2] R. Mahajan and R. Wattenhofer, "On consistent updates in software defined networks," in *HotNets*, 2013.

[3] K.-T. Foerster, R. Mahajan, and R. Wattenhofer, "Consistent updates in software defined networks: On dependencies, loop freedom, and blackholes," in *Networking*, 2016.

[4] J. Feigenbaum *et al.*, "Ba: On the resilience of routing tables," in *PODC*, 2012.

[5] S. Schmid and J. Suomela, "Exploiting locality in distributed SDN control," in *HotSDN*, 2013.

[6] K.-T. Foerster, T. Luedi, J. Seidel, and R. Wattenhofer, "Local checkability, no strings attached: (a)cyclicity, reachability, loop free updates in sdns," *Theor. Comput. Sci.*, vol. 709, pp. 48–63, Jan. 2018.

[7] T. D. Nguyen, M. Chiesa, and M. Canini, "Decentralized consistent updates in SDN," in *SOSR*, 2017.

[8] A. Korman, S. Kutten, and D. Peleg, "Proof labeling schemes," *Distributed Computing*, vol. 22, no. 4, pp. 215–233, 2010.

[9] M. Göös and J. Suomela, "Locally checkable proofs in distributed computing," *Theory of Computing*, vol. 12, no. 1, pp. 1–33, 2016.

[10] T. Mizrahi, E. Saat, and Y. Moses, "Timed consistent network updates," in *SOSR*, 2015.

[11] M. Reitblatt *et al.*, "Abstractions for network update," in *SIGCOMM*, 2012.

[12] K.-T. Foerster, A. Ludwig, J. Marcinkowski, and S. Schmid, "Loop-free route updates for software-defined networks," *IEEE/ACM Trans. Netw.*, vol. 26, no. 1, pp. 328–341, 2018.

[13] K.-T. Foerster, O. Richter, J. Seidel, and R. Wattenhofer, "Local checkability in dynamic networks," in *ICDCN*, 2017.

[14] M. König and R. Wattenhofer, "On local fixing," in *OPODIS*, 2013.

[15] M. Chiesa *et al.*, "On the resiliency of static forwarding tables," *IEEE/ACM Trans. Netw.*, vol. 25, no. 2, pp. 1133–1146, 2017.

[16] A. Bhalgat *et al.*, "Fast edge splitting and edmonds' arborescence construction for unweighted graphs," in *SODA*, 2008.

[17] K.-T. Foerster, A. Kamisinski, Y. A. Pignolet, S. Schmid, and G. Trédan, "Bonsai: Efficient fast failover routing," in *DSN*, 2019.

[18] ——, "Improved fast rerouting using postprocessing," in *SRDS*, 2019.

[19] K.-T. Foerster *et al.*, "On the power of preprocessing in decentralized network optimization," in *INFOCOM*, 2019.

[20] K.-T. Foerster, J. H. Korhonen, J. Rybicki, and S. Schmid, "Ba: Does preprocessing help under congestion?" in *PODC*, 2019.

[21] P. Fraigniaud, B. Patt-Shamir, and M. Perry, "Randomized proof-labeling schemes," *Distributed Computing*, vol. 32, no. 3, pp. 217–234, 2019.

[22] K. Censor-Hillel, A. Paz, and M. Perry, "Approximate proof-labeling schemes," *Theoretical Computer Science*, 2018.

[23] A. Ludwig, S. Dudycz, M. Rost, and S. Schmid, "Transiently policy-compliant network updates," *IEEE/ACM Trans. Netw.*, vol. 26, no. 6, pp. 2569–2582, 2018.

[24] S. Brandt, K.-T. Foerster, and R. Wattenhofer, "On consistent migration of flows in sdns," in *INFOCOM*, 2016.

[25] S. Delaët, S. Dolev, D. Khankin, and S. Tzur-David, "Make&activate-before-break for seamless SDN route updates," *Computer Networks*, vol. 147, pp. 81–97, 2018.

[26] S. Dolev and N. Tzachar, "Empire of colonies: Self-stabilizing and self-organizing distributed algorithm," *Theor. Comput. Sci.*, vol. 410, no. 6-7, pp. 514–532, 2009.

[27] N. Feamster and J. Rexford, "Why (and how) networks should run themselves," *arXiv preprint arXiv:1710.11583*, 2017.