Polynomial-Time What-If Analysis for Prefix-Manipulating MPLS Networks*

Stefan Schmid[1,2] Jiří Srba[2]
[1] University of Vienna, Austria [2] Aalborg University, Denmark

***Abstract*—While automated network verification is emerging as a critical enabler to manage large complex networks, current approaches come with a high computational complexity. This paper initiates the study of communication networks whose configurations can be verified *fast*, namely in polynomial time. In particular, we show that in communication networks based on prefix rewriting, which include MPLS networks, important network properties such as reachability, loop-freedom, and transparency, can be verified efficiently, even in the presence of failures. This enables a fast *what-if analysis*, addressing a major concern of network administrators: while configuring and testing network policies for a fully functional network is challenging, ensuring policy compliance *in the face of (possibly multiple) failures*, is almost impossible for human administrators. At the heart of our approach lies an interesting connection to the theory of prefix rewriting systems, a subfield of language and automata theory.**

I. Introduction

The operation of traditional computer networks is known to be a difficult manual and error-prone task, forcing administrators to become "masters of complexity". Over the last years, even tech-savvy companies such as GitHub, Amazon, GoDaddy, etc. have reported major issues with their network, due to misconfigurations and including loops [1], leading to disruptive downtimes [2]. A recent misconfiguration led to an hour-long, nation-wide outage for Time Warners backbone network [3], and BGP-related incidents make news every few months [3]. Ensuring a correct network operation is particularly challenging in the face of (one or even multiple) link failures, introducing combinatorial complexity [4], [5]. To make things worse, our methodologies and techniques to manage and debug networks have hardly evolved over the last twenty years, and tools such as ping and traceroute are limited to test end-to-end connectivity [6].

As a response to the difficulty of maintaining policy compliance, and given the critical role that computer networks (including the Internet, datacenter networks, enterprise networks) play today, researchers have started developing more principled approaches to networking and specification, in particular in the context of software-defined networking. Indeed, over the last years, we have witnessed great advances in the development of mathematical foundations for computer networks and the emergence of high-level network programming languages [7], [8].

While powerful, however, existing formal frameworks often come with potentially high (super-polynomial) running times in the worst-case: the underlying decision procedures can be PSPACE-complete [7] or even undecidable [9], which potentially limits their usability in practice. Indeed, existing solutions tend to trade generality over efficiency, and today we lack a good understanding of the tradeoff between expressiveness and computational tractability [10].

A. Our Contributions

Our work is motivated by the question whether and to which extent it is possible to verify computer network configurations *in polynomial time*. We find that fast verification is indeed possible for a large class of communication networks, namely networks based on prefix-rewriting (not to be confused with IP prefix routing). The key to the fast analysis algorithms for prefix rewriting networks lies in the specific, stack-like structure of packet headers and forwarding rules.

MPLS networks [11], networks based on multiprotocol label switching, are the most prominent and widely-deployed class of prefix rewriting networks. Accordingly, they serve as a case study and main application in this paper, however, our framework is applicable to other prefix-manipulating routing protocols like e.g. segment routing. To the best of our knowledge, our paper is the first to provide a formal model and fast automated analysis of prefix-manipulating networks under (possibly multiple) failures.

In particular, we present a formal framework which allows us to verify, automatically and in polynomial-time, fundamental network properties such as reachability, cycle detection, and transparency, both in fully functional networks as well as *under (permanent and transient) failures*. That is, our framework also allows us to conduct automated and fast *what-if analysis*. This is attractive as link failures, even multiple ones at a time, are common in practice [5] and introduce a combinatorial complexity which can overburden a human operator or system administrator.

Networks based on prefix rewriting rely on a natural (but not well-known) connection to the theory of pushdown automata rewriting systems [12], a subfield in language and automata theory. Unlike much prior work on network verification, this connection also allows us to deal with dynamically changing packet header sizes: a reality in many networks.

More generally, this paper also informs about the impact of the specific structure of forwarding rules on the verification complexity: we prove that arbitrary rewriting rules (as they are for example supported by modern OpenFlow switches) can render many underlying problems computationally intractable.

*Patent pending.

In particular, we prove that for any given maximum number of link failures, checking interface connectivity, transparency, and loop-free routing, are undecidable problems.

B. Related Work

Our work is motivated by the verification complexities introduced by (one or multiple) link failures, and we are particularly interested in fundamental reachability properties. Indeed, recent incidents reported in [4] and [8] demonstrate the potential security threats introduced by link failures and the resulting, policy-violating changes in network reachability. While already small violations of the intended reachability policies in computer networks can compromise availability, security, and performance of the network [8], network operators today often use a pragmatic and manual "fix it when it breaks" approach.

Researchers have started developing network verification tools which enable operators to systematically reason about their networks [13]. Often, network verification is considered from a program verification perspective [8]: the network control plane can be thought of as a program that takes configurations to generate a data plane, and the data plane is a program that takes a packet and its location (i.e., a router port) as input, and outputs a packet on a different location.

While early formal verification in networking revolved around correctness and protocol security [14], [15], recent work [7], [16]–[19] focused on the analysis of forwarding tables. Well-known static analysis tools include HSA [20], Libra [21] and VeriFlow [22]. There also exist efforts on active testing of the data plane, using traffic [23], [24]. Fayaz et al. recently initiated the study of "exhaustive" reachability analysis and verification, in *all* possible environments in which a network can incarnate [8].

Existing network verification frameworks often have a high runtime complexity, and complexity is often seen as the enemy of correctness and security [25]. Already the verification of simple networks based on stateless routers communicating is known to be PSPACE-complete [7], and by introducing a notion of state (e.g., stateful middleboxes but also modern switches [26]–[28]), the problems may even become undecidable [9], [29].

For an interesting study of complexities involved in verifying different types of middleboxes, we refer the reader to a recent paper by Velner et al. [29]: the authors also discuss polynomial time verification algorithms based on dynamic programming (for stateless and increasing middleboxes) and suggest abstracting away channel ordering, enabling safety checking in EXPSPACE for stateful middleboxes. The polynomial time algorithm from [29] relies on the restriction of a finite set of header symbols, and contrary to our work, this does not allow to model networks with unboundedly many headers, like MPLS and segment routing networks. The complexity question is also related to the recent studies on the (sometimes surprising) computational power (and hence complexity) of different networks: e.g., Peresini and Kostic [30] showed how to simulate a Rule 110 cellular automaton in a general network model, and Newport and Zhou [31] highlighted the computational power of SDNs.

Our work on *strictly polynomial-time verification* relies on the theory of prefix rewriting systems [12]. As a side contribution, our approach, unlike much prior work [7], [20], also supports packet headers whose size can change dynamically and be potentially unbounded (e.g., MPLS shim layer headers [11]).

C. Preliminaries

Our approach leverages a connection to language and automata theory. In this section, we introduce the necessary preliminaries. Let Σ be a set. Then Σ^* is a set of all strings over Σ including the empty string ϵ. The length of $w \in \Sigma^*$ is denoted as $|w|$. A (nondeterministic) finite automaton $A = (Q, q_0, G, \Sigma, \rightarrow)$ consists of a finite set of states Q, the initial state $q_0 \in Q$, the set of accepting (goal) states $G \subseteq Q$, a finite alphabet Σ and the transition relation $\rightarrow \subseteq Q \times \Sigma \times Q$, written as $q \xrightarrow{a} q'$ whenever $(q, a, q') \in \rightarrow$. A finite automaton A accepts the language $L(A)$ consisting of all strings $w = a_1 a_2 \dots a_n \in \Sigma^*$ for which there exists a path, from the initial state, $q_0 \xrightarrow{a_1} q_1 \xrightarrow{a_2} q_2 \xrightarrow{a_3} \dots \xrightarrow{a_n} q_n$ to $q_n \in G$. A language $L \subseteq \Sigma^*$ is *regular* if there is a finite automaton A such that $L = L(A)$. We recall that regular languages are closed under the operations of intersection, union, Kleene star, concatenation and complement. It is decidable in a polynomial time whether a given finite automaton A accepts a string w and whether $L(A)$ is empty. For an introduction to finite automata theory, we refer the reader e.g. to [32].

D. Organization

The remainder of this paper is organized as follows. Section II introduces our network model and its desirable properties we want to test automatically. Section III provides the motivation for our work by showing that the basic network properties studied in this paper are generally undecidable. In Section IV we introduce a formal model for MPLS networks which serves as a case study for prefix-rewriting networks. Section V is the heart of this paper, and describes the polynomial-time verification procedure for prefix-rewriting networks in general and MPLS networks in particular. We conclude in Section VI.

II. Network Model and Properties

This section introduces our formal network model, as well as the properties it should provide in terms of connectivity, reachability, loop-freedom and transparency.

A. Network Model

Definition 1 (Network): A *network* is a tuple $N = (V, E, I_v^{in}, I_v^{out}, \lambda_v, L, \delta_v^F)$ where

- V is a finite nonempty set of *nodes* (the switches or routers),
- $E \subseteq V \times V$ is the set of edges or *links*,
- for each node $v \in V$, the finite set I_v^{in} is the set of *incoming interfaces* of v, and the finite set I_v^{out} is the set of *outgoing interfaces* of v,

- for each node $v \in V$ the function $\lambda_v : I_v^{in} \cup I_v^{out} \to V$ is the *interface function* mapping each incoming interface to the previous-hop node, and each outgoing interface to the next-hop node such that $(\lambda_v(in), v) \in E$ and $(v, \lambda_v(out)) \in E$ for all $v \in V$, all $in \in I_v^{in}$ and all $out \in I_v^{out}$,
- L is a nonempty, finite set of *labels* used in packet headers, and
- for each set of failed links $F \subseteq E$ and each node $v \in V$ we have a commutable *routing function* $\delta_v^F : I_v^{in} \times L^* \to 2^{(I^{out} \times L^*)}$ such that for all incoming interfaces $in \in I_v^{in}$ and all packet headers $h \in L^*$ (sequence of labels), the set $\delta_v^F(in, h)$ is finite and consists of pairs of outgoing interfaces together with modified packet headers.

In our network model we consider only stateless nodes (that have no memory) and we are particularly interested in the routing behavior under failures. We represent a packet routing sequence consisting of forwarding operations as tuples $(v_i, in_i, h_i, out_i, h_{i+1}, F_i)$: node v_i receives, on the incoming interface in_i, a packet with header $h_i \in L^*$; given that links F_i are currently down, the node forwards the packet to the (live) outgoing interface out_i, with a new header h_{i+1}.

Definition 2 (Packet routing with at most k-link failures): A *packet routing* in a network with at most k-link failures, given a packet with the header h_1 received at incoming interface in_1 of v_1, is a finite or infinite sequence of tuples

$$(v_1, in_1, h_1, out_1, h_2, F_1),$$

$$(v_2, in_2, h_2, out_2, h_3, F_2),$$

$$\dots$$

where for all applicable $i \geq 1$, we have $v_i \in V$, $in_i \in I_{v_i}^{in}$, $h_i \in L^*$, $out_i \in I_{v_i}^{out}$, $F_i \subseteq E$ such that $|F_i| \leq k$ and where

- $(out_i, h_{i+1}) \in \delta_{v_i}^{F_i}(in_i, h_i)$, and
- $(v_i, \lambda_v(out_i)) = (\lambda(in_{i+1}), v_{i+1}) \in E \setminus F_i$.

The visited nodes during the packet routing are given by the sequence $v_1, v_2, \dots$. If the packet routing sequence is finite and ends with the tuple $(v_n, in_n, h_n, out_n, h_{n+1}, F_n)$, we say that the packet with the header h_1 at incoming interface in_1 of node v_1 can *arrive* (in a network with at most k link failures) to an outgoing interface out_n of the node v_n with the header h_{n+1}. Note that we allow dynamic changes of link failures during the packet routing, under the assumption that at any moment at most k links can fail. In the rest of the paper we implicitly assume that $k \leq |L|$.

Example 1: Let us consider an example in the context of MPLS. Figure 1 shows an excerpt of a larger MPLS network, inspired by the similar example given in the Cisco MPLS specification [33]. We assume that the link between v_1 and v_2 is down, forcing traffic to be rerouted via v_3 and v_4. We defer a detailed discussion of the failover behavior to later. Identifying the interface names with edges in the network (and where $(-, v_1)$ denotes the only incoming interface of the node v_1 and $(v_2, -)$ the only outgoing interface of the node v_2), we obtain the following packet routing sequence:

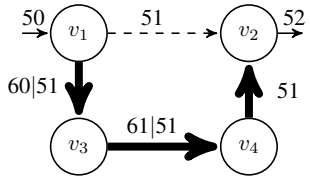

Fig. 1: Example: MPLS Fast-Reroute [33]. The failed link is *dashed* and the *bold arrows* show the current flow allocation. In this case, a backup tunnel (v_1, v_3, v_4) has been established around the "protected" link (v_1, v_2) that failed.

$$(v_1, (-, v_1), 50, (v_1, v_3), 60|51, \{(v_1, v_2)\}),$$
$$(v_3, (v_1, v_3), 60|51, (v_3, v_4), 61|51, \{(v_1, v_2)\}),$$
$$(v_4, (v_3, v_4), 61|51, (v_4, v_2), 51, \emptyset),$$
$$(v_2, (v_4, v_2), 51, (v_2, -), 52, \emptyset)\ .$$

The routing sequence assumes that during the first two packet forwards the link between v_1 and v_2 is down, after which the link was repaired (denoted by the empty set of failed links).

B. Properties and Problems

One of the most fundamental tasks of any computer network, is to provide connectivity between endpoints. At the same time, due to policy and security constraints, connectivity may also be explicitly prohibited: for example, a packet entering from an insecure port or a packet having a certain header should not be forwarded to a certain secure port. We will refer to this property as the *interface connectivity*. Another fundamental property many networks should provide is *transparency*: it is often desirable that from the outside, changes in the internal technology or routing of a network are not visible, facilitating a simple interface to neighboring networks. In other words, a network may be abstracted as a single "big switch". Another fundamental property any useful network should provide is *loop-freedom*. As we will see in the following, loop-freedom comes in different flavors, and indeed, certain loops may be unavoidable or even desirable, e.g., during the traversal of middleboxes or virtualized network functions deployed in the network core, as long as they are finite and "on purpose"'.

Succinctly, we define the desirable properties in a parametrized way, given a fixed number k as an upper bound on the number of tolerable link failures.

- *Interface connectivity problem*: Given an incoming interface in of a node v, an outgoing interface out of a node v', and a subset $L' \subseteq L$ of label symbols, we ask: is there a header $h \in L'^*$ such that a packet with the header h at interface in of node v can arrive at the interface out of node v' with some modified header $h' \in L^*$?
- *Transparency problem*: Given two nodes v and v', an incoming interface in of v and an outgoing interface out

of v', we ask: for any packet arriving with an empty header at interface in of v and reaching the interface out of v', is its header at the interface out also empty?
- *Cyclic routing problem*: Given an incoming interface in of a node v, we ask: is there a header $h \in L^*$ such that there exists an infinite packet routing sequence for a packet with the header h starting at the interface in of node v?
- *r-Repeated routing problem*: Given an incoming interface in of a node v, we ask: is there a header $h \in L^*$, a node $v' \in V$ and packet routing sequence for a packet with header h, starting at the interface in of the node v, such that the node v' is visited at least r-times?

III. Motivation: Complexity and Undecidability

Our paper is motivated by the observation that checking the properties defined above can be undecidable or computationally infeasible, already in networks based on simple forwarding rules.

Theorem 1: For any given number k of maximum link failures, the interface connectivity, transparency and cyclic routing problems are undecidable.

On the other hand, the r-repeated routing problem is decidable for any r.

Theorem 2: For any given number k of maximum link failures, the r-repeated routing problem is decidable.

Note that while the problem is decidable, the brute force algorithm runs in exponential time even for a fixed number r. We will show later on that for specific networks (like MPLS), we can decide the r-repeated routing problem in polynomial time for any fixed number r, even when considering an arbitrary number k of link failures.

IV. Case Study and Model for MPLS Networks

Prefix-rewriting networks as studied in this paper are not a new concept, but rather, one of the most widely deployed network protocols today is based on prefix rewriting: networks based on Multiprotocol Label Switching (MPLS) [11]. We will hence consider MPLS networks as our main case study. In this section, we first provide some background on MPLS networks and then introduce a formal model for MPLS networks accordingly. We will conclude with an extended example.

A. MPLS Networks

MPLS networks are used to establish tunnels across a transport medium. In a nutshell, forwarding decisions are based on the contents of labels assigned to data packets. MPLS typically operates between Layer 2 and Layer 3 and uses a *label stack*, where each stack entry contains four fields: (1) a label value; (2) a traffic class field for QoS (quality of service) priority and ECN (Explicit Congestion Notification); (3) a 1-bit *bottom of stack* flag; (4) an 8 bit TTL (time to live) field. The Traffic Class field is also known under the name *exp field*, which is "reserved for experimental use"[1].

[1]For example, this field can be set to indicate that a label has been popped already, to remove load from the egress router.

An MPLS node serving as *label switch router* (a.k.a. transit router) uses the label included in the packet header to determine the next hop on the label switched path (LSP). On this occasion, the old label may be replaced with a new label before the packet is routed forward. An MPLS node serving as *label edge router* acts as the entry and exit point for the network: a label edge router pushes an MPLS label onto an incoming packet resp. pops it from an outgoing packet.

Upon receiving a packet, an MPLS node examines the label *at the top of the stack*, and depending on the content, performs a *swap*, *push* or *pop* operation on the packet label stack:

1) In a swap operation the label is swapped with a new label, and the packet is forwarded along the path associated with the new label.
2) In a push operation a new label is pushed on top of the existing label, encapsulating the packet in another layer of MPLS and introducing hierarchical routing.
3) In a pop operation the label is removed from the packet. If the popped label was the last on the stack, the packet leaves the MPLS tunnel.

We are particularly interested in analyzing network behaviors under failure. MPLS includes a local protection mechanism called *fast reroute* resp. *local protection*, allowing to protect a label switched path by a backup path. Let us first recall the example in Figure 1. In this example, in the absence of failures, node v_1 forwards packets arriving with (top-of-stack) label 50 to node v_2, swapping the label from 50 to 51. However, if link (v_1, v_2) is currently unavailable (indicated as *dashed* line), v_1's failover rules for link (v_1, v_2) apply (if any are defined). In this case, to route around link (v_1, v_2), v_1 pushes a label, say 60, on top of the label 51 (note: *not* 50), and forwards the packet to v_3. Using a standard forwarding rule installed at v_3, we will swap the top label to 61, and v_4 will pop the top label. Thus, the rerouted packet arrives at v_2 with the original label 51, and the packet can continue its journey similarly to the non-faulty scenario.

In general, MPLS link protection comes in two flavors, which we will simply call *local* and *global* protection. In a local protection scheme, each failed link (u, v) is protected individually, i.e., traffic is rerouted along an alternative path from u to v. The use of backup routes is recursive, in the sense that if along the backup route from u to v another link e fails, we recursively reroute around e as well, before continuing on our backup route to v. The main advantage of the local protection scheme is that it is relatively simple: essentially, we have to push a new label (of the backup tunnel) for each failed link, recursively routing around the failed link. The main disadvantage however is that local rerouting severely limits rerouting flexibilities, compared to the *global protection scheme* where traffic can be rerouted directly to the destination, by swapping MPLS labels accordingly: especially in case of multiple link failures, insisting on rerouting to the endpoint of each failed link may unnecessarily increase route lengths and network loads. The main drawback of the global rerouting scheme is that it renders what-if analysis even more complex:

computing all possible routes along which traffic may be rerouted under a set of failures is challenging for a human admin. Accordingly, admins are often explicitly discouraged of using global rerouting.

We will provide a more detailed example of both the *local* and *global* protection at the end of this section, after having introduced our formal model.

B. A Formal Model of MPLS Networks

We shall now formally define an MPLS network model as a particular instance of the general network model $N = (V, E, I_v^{in}, I_v^{out}, \lambda_v, L, \delta_v^F)$ introduced in Section II.

First, we define the set Op of MPLS operations on header sequences:

$$Op = \{swap(\ell) \mid \ell \in L\} \cup \{push(\ell) \mid \ell \in L\} \cup \{pop\} \ .$$

An MPLS network is defined via the local *routing table* τ_v for each node $v \in V$:

$$\tau_v : I_v^{in} \times (L \cup \{\bot\}) \hookrightarrow I_v^{out} \times Op$$

is a partial function that for a packet arriving at an incoming interface in and the top-most label symbol ℓ (or $\bot$ that stands for the empty label-stack), either returns $\tau_v(in, \ell) = (out, o)$ where out is an outgoing interface and o is some label operation to be performed; or the output is undefined meaning that the packet is dropped. We require that whenever $\tau_v(in, \bot) = (out, o)$, then o must be of the form $push(\ell)$ (in other words pop and $swap$ operations may not be applied on an empty label-stack).

We also employ a local link protection mechanism (fast reroute): for each node $v \in V$, a partial *link protection* function

$$\pi_v : I_v^{out} \times (L \cup \{\bot\}) \hookrightarrow I_v^{out} \times Op$$

is defined. Here, if $\pi_v(in, \bot) = (out, o)$ then o must be the push operation. The intuition is that if some outgoing interface out is down, the link protection function suggests (in case that out is protected) a backup outgoing interface, together an additional operator on the header. If the link protection function is undefined, the packet is dropped.

As links can have multiple protections, we define the *link protection dependency graph* $(I_v^{out}, \rightarrow)$ such that $out \rightarrow out'$ iff $\pi_v(out, \ell) = (out', o)$ for some label $\ell \in L \cup \{\bot\}$ and some operation o. For sanity reasons (to avoid cyclic link protection), in the rest of the paper we require that every link protection function π_v has an acyclic link protection dependency graph. If needed, this fact can be easily verified in polynomial time.

Definition 3 (MPLS network): An MPLS network is a network $N = (V, E, I_v^{in}, I_v^{out}, \lambda_v, L, \delta_v^F)$ defining for each node v a local routing table τ_v and a link protection function π_v. The δ_v^F function is defined as follows (here ϵ is the empty string of labels, $\ell \in L$ and $h \in L^*$):

- $\delta_v^F(in, \epsilon) = \mathcal{P}_v^F(out, \ell)$ provided that $\tau_v(in, \bot) = (out, push(\ell))$,
- $\delta_v^F(in, \ell h) = \mathcal{P}_v^F(out, \ell' h)$ provided that $\tau_v(in, \ell) = (out, swap(\ell'))$,
- $\delta_v^F(in, \ell h) = \mathcal{P}_v^F(out, \ell' \ell h)$ provided that $\tau_v(in, \ell) = (out, push(\ell'))$,
- $\delta_v^F(in, \ell h) = \mathcal{P}_v^F(out, h)$ provided that $\tau_v(in, \ell) = (out, pop)$

where $\mathcal{P}_v^F(out, h)$ is the fast reroute function defined via the link protection function π_v as follows:

$$\mathcal{P}_v^F(out, h) =$$

$$\begin{cases} \{(out, h)\} & \text{if } (v, \lambda_v(out)) \notin F \text{ else} \\ \mathcal{P}_v^F(out', \ell) & \text{if } h = \epsilon \text{ and } \pi_v(out, \bot) = (out', push(\ell)) \\ \mathcal{P}_v^F(out', \ell' h') & \text{if } h = \ell h' \text{ and } \pi_v(out, \ell) = (out', swap(\ell')) \\ \mathcal{P}_v^F(out', \ell' h) & \text{if } h = \ell h' \text{ and } \pi_v(out, \ell) = (out', push(\ell')) \\ \mathcal{P}_v^F(out', h') & \text{if } h = \ell h' \text{ and } \pi_v(out, \ell) = (out', pop) \\ \emptyset & \text{otherwise.} \end{cases}$$

Note that because the dependency graph $(I_v^{out}, \rightarrow)$ is acyclic, the recursive definition of $\mathcal{P}_v^F(out, h)$ is well founded.

C. Example

Let us illustrate our model and notation with two examples in Figures 2 (for *local* rerouting) and 3 (for *global* rerouting). We indicate the current flow in *bold*, the failed link *dashed* and for simplicity we identify node interfaces with the corresponding links. Figure 2 *top* shows the current flow before the link failure. If link (v_2, v_3) fails, node v_2 pushes a label, essentially rerouting the flow via v_6 and v_7 to v_3 again (Figure 2 *middle*). If in addition, also link (v_2, v_6) fails, this link is protected by rerouting the traffic via v_5 (Figure 2 *bottom*).

The main disadvantage of this local protection scheme is that the flow from in_2 to out_2 is forced to be routed via v_3, although from v_7, it could be in principle directly forwarded to v_8, reducing the overall traffic.

Figure 3 *top* (single link failure) shows how the flexibilities introduced by a global rerouting scheme can be exploited to overcome this problem. If two links fail (Figure 3 *bottom*), a hybrid approach may be used: it is sufficient to protect the link (v_2, v_6) using a local rerouting.

The examples above show the application of the push rules only in case of a link failure, however, our MPLS network model is general enough to allow for the use of push rules also as a standard router behaviour e.g. for tunneling.

V. Polynomial-Time Verification

The key to the polynomial-time verifiability of the MPLS-like networks introduced in this paper lies in the specific structure of the switch resp. router rules. As we will see, this allows to leverage the theory of *prefix rewriting system* and in particular *pushdown automata rewriting* towards fast network verification. In the following, we first formally introduce prefix rewriting systems. Subsequently, we show how to build MPLS-based prefix rewriting networks, and prove the polynomial-time verifiability of the fundamental properties introduced in Section II.

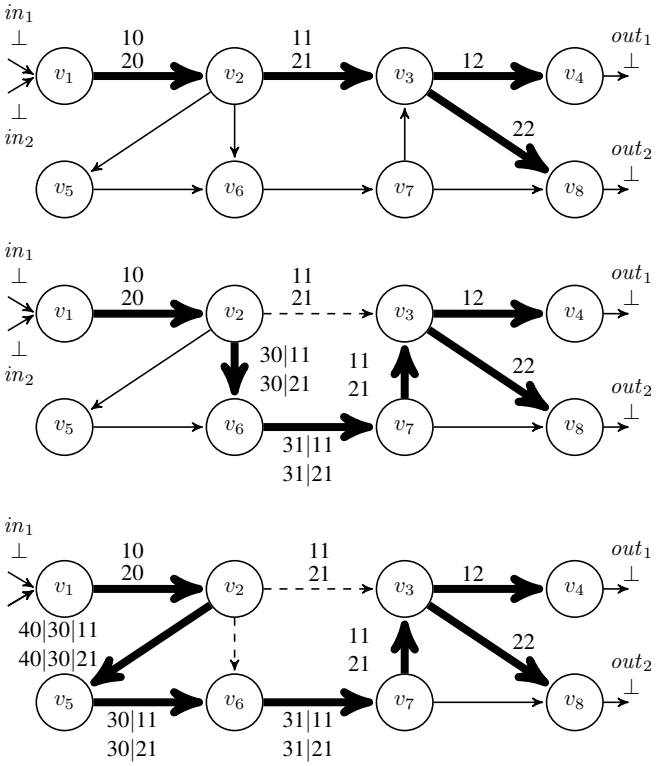

Fig. 2: Example of local protection. *Top:* Initially, traffic from in_1 (entering with empty stack $\perp$) to out_1 is routed along the path $v_1 \rightsquigarrow v_2 \rightsquigarrow v_3 \rightsquigarrow v_4$ and then to out_1. Traffic from in_2 to out_2 is routed along the path $v_1 \rightsquigarrow v_2 \rightsquigarrow v_3 \rightsquigarrow v_8$ and then to out_2. *Middle:* Behavior when link (v_2, v_3) fails. *Bottom:* Behavior when links (v_2, v_3) and (v_2, v_6) fail.

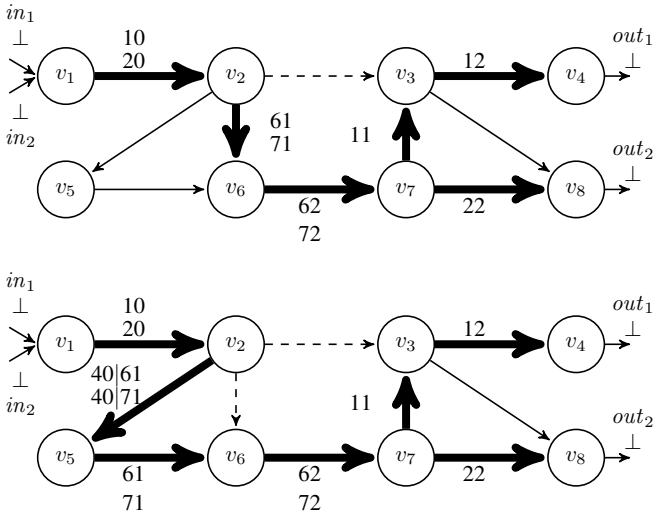

Fig. 3: Example of global protection. Same example as in Figure 2, however, failed links do not have to be masked individually, but traffic can be rerouted more flexibly. *Top:* rerouting around failed link (v_2, v_3): now the flow from in_2 to out_2 travels directly via v_7 and v_8 to the destination. *Bottom:* In addition also link (v_5, v_6) fails, which however is masked with local protection.

A. Prefix Rewriting Systems

Prefix-manipulating networks can be verified by translating them into pushdown automata and applying the automata-theoretic techniques that give us polynomial time decision procedures.

Let Γ be a nonempty and finite alphabet. A *prefix rewrite system* is a finite collection of rewrite rules $R \subseteq \Gamma^* \times \Gamma^*$. We write $v \to w$ of an element $(v, w) \in R$ and require that $v \neq \epsilon$ where ϵ is the empty string. A prefix rewrite system R generates an infinite transition system $G_R = (\Gamma^*, \to_R)$ such that $vt \to_R wt$ if and only if $(v, w) \in R$ and $t \in \Gamma^*$. A *prefix rewrite system* is called a *pushdown system* if for every $(v, w) \in R$ we have $|v| = 2$ and $1 \leq |w| \leq 3$. Hence the first symbol in v and w is interpreted as the control state of the pushdown automaton and the second symbol in v is the current top of the stack. If $|w| = 1$ then the rule pops the top of the stack. If $|w| = 2$ then it swaps the top with another symbol and if $|w| = 3$ then it swaps the current top of the stack and pushes a new symbol on the top of the stack.

Let $S \subseteq \Gamma^*$ be a set of strings. We define the sets $post^*_R(S) = \{u' \mid u \to^*_R u', u \in S\}$ and $post^+_R(S) = \{u' \mid u \to^+_R u', u \in S\}$ where $\to^+_R$ is the transitive closure of $\to_R$ and $\to^*_R$ is the reflexive and transitive closure of the relation $\to_R$.

Theorem 3 ([34], [35]): Let R be a pushdown system over the alphabet Γ. Let $S \subseteq \Gamma^*$ be a regular set of pushdown configurations given by a finite automaton A such that $L(A) = S$. Then we can in polynomial time (w.r.t. to the size of A and R) construct finite automata A^{post*} and A^{post+} such that $L(A^{post*}) = post^*_R(S)$ and $L(A^{post+}) = post^+_R(S)$.

B. Translating MPLS Networks to Prefix Rewrite Systems

We now show how to encode entire network models, in particular MPLS networks, as a pushdown prefix rewriting system. We use the following intuition for encoding of packet routing as a pushdown system:

- control states are of the form (v, in) or (v, out, i) where v is a node, in incoming interface of v and out an outgoing interface of v, and i is the number of times a link failed in the given node, and
- labels will be stack symbols (including a special symbol $\perp$ for the bottom of the stack).

Hence a packet with the header $h \in L^*$ arriving at incoming interface in of a node v will be represented by the pushdown configuration $(v, in)h\perp$. A packet with header h that should be forwarded at node v to the outgoing interface out is represented by the configuration $(v, out, i)h\perp$, where i represents the number of times the fast reroute mechanism was employed at the node v. We have $0 \leq i \leq k$ where k is the maximum number of link failures. Note that since π_v is only a partial function, it is possible that a given outgoing interface out is

protected fewer times than k (or it is unprotected and then the control state $(v, out, 0)$ appears only with the number 0).

Assume a given MPLS network $N = (V, E, I_v^{in}, I_v^{out}, \lambda_v, L, \delta_v^F)$ together with the routing table τ_v and link protection π_v for each $v \in V$. Let k be a fixed number of maximum link failures such that $k \leq |E|$.

We define the corresponding pushdown system $R(N)$ over $\Gamma = L \cup \{\bot\} \cup \{(v, in) \mid v \in V,\ in \in I_v^{in}\} \cup \{(v, out, i) \mid v \in V,\ out \in I_v^{out},\ 0 \leq i \leq k\}$ such that $R(N)$ contains the following rules:

a) $(v, out, i)\ell \to (v', in)\ell$ for every $\ell \in L \cup \{\bot\}$ and every i, $0 \leq i \leq k$, such that $v' = \lambda_v(out)$ and $v = \lambda_{v'}(in)$,
b1) $(v, in)\bot \to (v, out, 0)\ell\bot$ if $\tau_v(in, \bot) = (out, push(\ell))$,
b2) $(v, in)\ell \to (v, out, 0)\ell'$ if $\tau_v(in, \ell) = (out, swap(\ell'))$,
b3) $(v, in)\ell \to (v, out, 0)\ell'\ell$ if $\tau_v(in, \ell) = (out, push(\ell'))$,
b4) $(v, in)\ell \to (v, out, 0)$ if $\tau_v(in, \ell) = (out, pop)$,
c1) $(v, out, i)\bot \to (v, out', i+1)\ell\bot$ for every i, $0 \leq i < k$, where $\pi_v(out, \bot) = (out', push(\ell))$,
c2) $(v, out, i)\ell \to (v, out', i+1)\ell'$ for every i, $0 \leq i < k$, where $\pi_v(out, \ell) = (out', swap(\ell'))$,
c3) $(v, out, i)\ell \to (v, out', i+1)\ell'\ell$ for every i, $0 \leq i < k$, where $\pi_v(out, \ell) = (out', push(\ell'))$, and
c4) $(v, out, i)\ell \to (v, out', i+1)$ for every i, $0 \leq i < k$, where $\pi_v(out, \ell) = (out', pop)$.

The rule a) makes the connection between the outgoing interface of a node to its next hop and the packet is forwarded there (and hence the top most label ℓ does not change). Rules b1) to b4) implement the node forwarding operation on the top most label ℓ of the forwarded packet. Note that an empty header $\bot$, arriving for example from another network using a different routing protocol, is handled separately by the rule b1). Finally, the rules c1) to c4) enumerate all possible reroute options for up to k link failures. As $0 \leq i \leq k$ and k is w.l.o.g. assumed to be smaller than or equal to the number of edges in N, the system $R(N)$ can be constructed in polynomial time.

C. Fast Verification

The following theorem is the key to all our decidability results on prefix rewriting networks.

Theorem 4: Consider an MPLS network with at most k link failures. Then a packet with a header h_1 at an incoming interface in_1 of the node v_1 can reach an outgoing interface out_2 of a node v_2 with the header h_2 if and only if $(v_1, in_1)h_1\bot \to^* (v_2, out_2, i)h_2\bot$ for some i, $0 \leq i \leq k$.

Sketch: Let us first assume a routing sequence as in Definition 2 for a network N with at most k link failures. We shall argue that the first packet forwarding in the routing sequence

$$(v_1, in_1, h_1, out_1, h_2, F_1)$$

where $\lambda_{v_1}(out_1) = v_2$ and $(v_1, v_2) \in E \setminus F_1$, can be captured by a series of rewriting rules in the pushdown system. This scheme can be then extended to the whole routing sequence by a simple inductive argument. By the semantics of the MPLS networks, and by a single application of one of the rules b1)–b4) and up to $|F_1|$ applications of the rules c1)–c4), we get the following rewriting sequence in the pushdown system $(v_1, in_1)h_1\bot \to (v_1, out, 0)h\bot \to (v_1, out', 1)h'\bot \to (v_1, out'', 2)h''\bot \to \ldots \to (v_1, out_1, i)h_2\bot$ where $i \leq |F_1| \leq k$. This sequence can be extended by the application of the rule a) such that $(v_1, out_1, i)h_2\bot \to (v_2, in_2)h_2\bot$ to the next packet forward and so on.

Let us on the other hand assume that $(v_1, in_1)h_1\bot \to^* (v_2, out_2, i)h_2\bot$ for some i, $0 \leq i \leq k$. We shall argue that there is a corresponding routing sequence by analyzing the rules a), b1)–b4) and c1)–c4). First we observe, that the application of the rule a) is compatible with the definition of packet routing sequence, as it simply forwards the packet from the outgoing interface out to the incoming interface in; it is ensured that they are connected by an edge. The header is not altered by the application of the rule a). Rules b1)–b4) perform the prefix manipulation of the packet arriving at the incoming interface in exactly according to the forwarding table τ_v, following the semantics from Definition 3. In a similar fashion, the rules c1)–c4) perform, in a series of steps, the link protection mechanism given by the table π_v. By every application of the rule, the counter in the third component increases by one, simulating the function $\mathcal{P}_v^F(out, h)$ from Definition 3 that applies the link protection recursively for up to $|F|$ steps where $|F| \leq k$. Hence every computation in the pushdown system corresponds to a real packet routing sequence in a network with up to k link failures. ■

1) Application to Interface Connectivity Problem: Assume that we are given an MPLS network $N = (V, E, I_v^{in}, I_v^{out}, \lambda_v, L, \delta_v^F)$ with at most k link failures and let $R(N)$ be the pushdown system for N constructed above. Let in be a given incoming interface of a node v, let out be a given outgoing interface of a node v' and let L' be a subset of label symbols.

We want to provide an answer to the question whether there is a header $h \in L'^*$ such that a packet with this header at incoming interface in of v can arrive to the interface out of v'. We can decide this question by first constructing a finite automaton A accepting the regular language $\{(v, in)h\bot \mid h \in L'^*\}$. Clearly such A can be constructed by using only three states and $O(|L'|)$ transitions. Then, in polynomial time, we construct the automaton A^{post*} that recognizes $post^*_{R(N)}(L(A))$. By a direct application of Theorem 4, the answer to the interface connectivity problem is positive if and only if there is an i, $0 \leq i \leq k$, such that the string $(v', out, i)h'\bot$ is accepted by A^{post*} for some $h' \in L^*$. This can be verified in polynomial time e.g., by running DFS from the initial state of A^{post*}: the first symbol that we read must be of the form (v', out, i); afterwards, we perform a search to check whether some accepting state is reachable (ignoring the symbols on the transitions). Hence we can conclude with the following theorem.

Theorem 5: The interface connectivity problem is decidable in polynomial time for MPLS networks.

2) Application to Cyclic Routing Problem: Assume a given MPLS network $N = (V, E, I_v^{in}, I_v^{out}, \lambda_v, L, \delta_v^F)$ in which

there are at most k link failures. For a given incoming interface in of a node v, we want to find out if there is a header $h \in L^*$ such that there is an infinite routing sequence for the packet with the header h starting at the interface in of v. We shall again use the pushdown system $R(N)$ constructed above and first we realize the following fact.

Lemma 1: There is an infinite routing sequence in N for a packet with the header $h \in L^*$ starting at the incoming interface in of v iff there is an infinite computation of the pushdown system $R(N)$ starting at the configuration $(v, in)h\bot$.

Proof: Follows from Theorem 4 and the fact that one step in the routing sequence is simulated by at most $k+2$ rewritings in the pushdown system—one for changing from the output interface to the input one by rule a), one for forwarding the packet by one of the rules b1) to b4), followed by at most k applications of the rules c1) to c4). ∎

Let us now consider some infinite computation of $R(N)$ of the form $q_1h_1\bot$, $q_2h_2\bot$, ... where each q_j is of the type (v_j, in_j) or (v_j, out_j, i_j). We call a configuration $q_jh_j\bot$ in this sequence a *nondecreasing configuration* if $|h_j| \leq |h_m|$ for all $m > j$. Surely the infinite sequence must contain an infinite number of nondecreasing configurations. Moreover, because we have finitely many control states in $R(N)$ and finitely many labels that are on the top of the label stack, there must be some q and some ℓ such that there are infinitely many nondecreasing configurations of the form $q\ell h'_j\bot$ for some h'_j where $1 \leq j$. In fact, during this computation the h'_j part of the stack is never popped, so we necessarily get that

$$q\ell \rightarrow^*_{R(N)} q\ell h' \text{ for some } h' \in L^* \ . \tag{1}$$

On the other hand, if there is some q and ℓ such that Equation 1 holds, this implies the existence of an infinite computation sequence from any configuration of the form $q\ell h$, for any $h \in L^*$. We call any pair (q, ℓ) that satisfies Equation 1 a *repeating head*. Clearly, all repeating heads (there are only polynomially many of those) can be enumerated in polynomial time by constructing a finite automaton A^{post+} for $post^+_{R(N)}(\{q\ell\})$ and checking if from some state of A^{post^+} that is reachable from its initial state under the sequence q, ℓ, an accepting state of A^{post+} is reachable. We can now conclude with the following theorem.

Theorem 6: The cyclic routing problem in MPLS networks is decidable in polynomial time.

Proof: For each repeating head (q, ℓ) check if $q\ell h\bot \in post^*(\{(v, in)h'\bot \mid h' \in L^*\})$ for some $h \in L^*$. This can be done in polynomial time. If this succeeds at least for one repeating head (q, ℓ), we can claim that the answer to the cyclic routing problem is positive; if it fails for all repeating heads, then the answer is negative. As there are only polynomially many repeating heads, the test can be performed in polynomial time. ∎

3) Application to r-Repeated Routing Problem: As before, assume a given MPLS network N with at most k link failures, together with the corresponding pushdown system $R(N)$ constructed above. Let in be a given incoming interface of a node v. We want to check whether there is some header and routing sequence of packet starting at the interface in of v where some node v' repeats at least r times.

In order to do so, we shall create a modified set of rules $R'(N)$ such that whenever $q\ell \rightarrow q'\alpha'$ where $\alpha' \in L^*$ is a rule in $R(N)$ then

- $(q, -, 0)\ell \rightarrow (q', -, 0)\alpha'$ is a rule in $R'(N)$, and
- $(q, -, 0)\ell \rightarrow (q', v', 1)\alpha'$ is a rule in $R'(N)$ whenever $q = (v', in')$ for some incoming interface in' of v', and
- $(q, v', j)\ell \rightarrow (q', v', j+1)\alpha'$ is a rule in $R'(N)$ whenever $q = (v', in')$ for some incoming interface in' of v' and $1 \leq j < r$.

The modified pushdown system behaves identically to the $R(N)$, but additionally, it can at any moment nondeterministically guess a node v' that should repeat at least r times. It starts counting in the last coordinate of the control state, how many times the node appeared so far. To check whether the network N can perform r-repeated routing from an incoming interface in of a node v, we construct in polynomial time the finite automaton for the regular language

$$post^*_{R'(N)}(\{((v, in), -, 0)h\bot \mid h \in L^*\})$$

and intersect it with the regular language

$$\{((v', out', 0), v', r)h'\bot \mid v' \in V,\ out' \in I^{out}_{v'},\ h' \in L^*\} \ .$$

If the intersection is nonempty, the answer to the r-repeated routing problem is positive; otherwise the answer is negative. This can be all done in polynomial time, assuming that the number r is fixed (not part of the input). Hence we can conclude with the following theorem.

Theorem 7: For MPLS networks, the r-repeated routing problem is decidable in polynomial time, for any fixed r.

4) Application to Transparency Problem: Given a prefix rewriting network N with at most k link failures, an incoming interface in of a node v and an outgoing interface out of a node v'. We want to make sure that if a packet with empty header arriving at in of the node v ever reaches the interface out of v' then its header must be empty too. In order to verify this property, we first construct a simple three state finite automaton A accepting the language $\{(v, in)\bot\}$ and then in polynomial time construct the automaton A^{post*} recognizing the regular language $post^*_{R(N)}(L(A))$. Finally, we intersect $post^*_{R(N)}(L(A))$ with the regular language $\{(v', out, i)h\bot \mid 0 \leq i \leq k,\ h \in L^*, |h| > 0\}$ and check (again in polynomial time) whether the resulting regular language is empty or not. If it is empty, the answer to the transparency problem is positive; otherwise it is negative.

Theorem 8: The transparency problem for MPLS networks is decidable in polynomial time.

VI. Conclusion

Against the backdrop that already simple forwarding rules can render basic reachability, transparency, and loop-freedom problems computationally hard or even undecidable, as shown in the beginning of this paper, we set off to investigate whether there exist computer networks which support *strictly*

polynomial-time verification. We find that networks based on prefix rewriting offer such an environment, allowing us to exploit a connection to prefix rewriting systems. Prefix rewriting networks are practically relevant, and include, among others, MPLS routing and segment routing. We believe that they can hence be an attractive framework to design and operate computer networks supporting fast verification.

We understand our work as a first step, and hope that it can inspire further research which networks and properties can (and cannot) be computed or verified in polynomial time. For example, our polynomial-time result can be generalized further, e.g., it can be extended to model networks including a number of nodes performing *stateful* processing by encoding them into the control state of the pushdown automaton. As long as the number of such nodes is constant (not part of the input problem), we still preserve polynomial time decidability. We are currently developping a software tool based on our concepts and the initial prototype shows promissing results.

Acknowledgments. The authors would like to thank Bingtian Xue for several discussions about the network model, and Henrik T. Jensen and Jesper B. Rosenkilde from NORDUnet for their technical insights into the MPLS implementation. The first author is partially affiliated with TU Berlin, the second author with FI MU in Brno.

REFERENCES

[1] GitHub, "https://github.com/blog/1346networkproblemslastfriday," in *Website*, 2016.

[2] N. Feamster and H. Balakrishnan, "Detecting bgp configuration faults with static analysis," in *Proc. USENIX Symposium on Networked Systems Design and Implementation (NSDI)*, 2005, pp. 43–56.

[3] M. Anderson, "Time warner cable says outages largely resolved," in *http://www.seattletimes.com/business/time-warner-cable-says-outages-largely-resolved*, 2014.

[4] R. Beckett, R. Mahajan, T. Millstein, J. Padhye, and D. Walker, "Don't mind the gap: Bridging network-wide objectives and device-level configurations," in *Proc. ACM SIGCOMM*, 2016, pp. 328–341.

[5] P. Gill, N. Jain, and N. Nagappan, "Understanding network failures in data centers: measurement, analysis, and implications," in *ACM SIGCOMM Computer Communication Review*, vol. 41 (4), 2011, pp. 350–361.

[6] Barefoot, "The world's fastest and most programmable networks," in *White Paper*, 2016.

[7] C. J. Anderson, N. Foster, A. Guha, J.-B. Jeannin, D. Kozen, C. Schlesinger, and D. Walker, "Netkat: Semantic foundations for networks," *SIGPLAN Not.*, vol. 49, no. 1, Jan. 2014.

[8] S. K. Fayaz, T. Sharma, A. Fogel, R. Mahajan, T. Millstein, V. Sekar, and G. Varghese, "Efficient network reachability analysis using a succinct control plane representation," in *Proc. 12th USENIX Symposium on Operating Systems Design and Implementation (OSDI)*, 2016, pp. 217–232.

[9] K. G. Larsen, S. Schmid, and B. Xue, "Wnetkat: A weighted sdn programming and verification language," in *Proc. 20th International Conference on Principles of Distributed Systems (OPODIS)*, 2016.

[10] M. Dobrescu and K. Argyraki, "Software dataplane verification," *Communications of the ACM*, vol. 58, no. 11, pp. 113–121, 2015.

[11] B. S. Davie and Y. Rekhter, *MPLS: technology and applications*. Morgan Kaufmann Publishers Inc., 2000.

[12] P. Jančar and J. Srba, "Undecidability of bisimilarity by defender's forcing," *Journal of the ACM*, vol. 55, no. 1, pp. 1–26, 2008.

[13] G. Varghese, "Technical perspective: Treating networks like programs," *Communications of the ACM*, vol. 58, no. 11, pp. 112–112, 2015.

[14] E. M. Clarke, S. Jha, and W. Marrero, "Using state space exploration and a natural deduction style message derivation engine to verify security protocols," in *Programming Concepts and Methods*. Springer, 1998, pp. 87–106.

[15] R. W. Ritchey and P. Ammann, "Using model checking to analyze network vulnerabilities," in *Proc. IEEE Symposium on Security and Privacy (S&P)*, 2000, pp. 156–165.

[16] M. Canini, D. Venzano, P. Perešíni, D. Kostić, and J. Rexford, "A nice way to test openflow applications," in *Proc. 9th USENIX Symposium on Networked Systems Design and Implementation (NSDI)*, 2012, pp. 127–140.

[17] N. Foster, D. Kozen, M. Milano, A. Silva, and L. Thompson, "A coalgebraic decision procedure for netkat," in *ACM SIGPLAN Notices*, vol. 50 (1), 2015, pp. 343–355.

[18] H. Mai, A. Khurshid, R. Agarwal, M. Caesar, P. Godfrey, and S. T. King, "Debugging the data plane with anteater," in *ACM SIGCOMM Computer Communication Review*, vol. 41 (4), 2011, pp. 290–301.

[19] R. Beckett, A. Gupta, R. Mahajan, and D. Walker, "A general approach to network configuration verification," in *Proceedings of the Conference of the ACM Special Interest Group on Data Communication*, ser. SIGCOMM '17. ACM, 2017, pp. 155–168.

[20] P. Kazemian, G. Varghese, and N. McKeown, "Header space analysis: Static checking for networks," in *Proc. 9th USENIX Conference on Networked Systems Design and Implementation (NSDI)*, 2012.

[21] H. Zeng, S. Zhang, F. Ye, V. Jeyakumar, M. Ju, J. Liu, N. McKeown, and A. Vahdat, "Libra: Divide and conquer to verify forwarding tables in huge networks," in *Proc. 11th USENIX Symposium on Networked Systems Design and Implementation (NSDI)*, 2014, pp. 87–99.

[22] A. Khurshid, X. Zou, W. Zhou, M. Caesar, and P. B. Godfrey, "Veriflow: verifying network-wide invariants in real time," in *Proc. 10th USENIX Symposium on Networked Systems Design and Implementation (NSDI)*, 2013, pp. 15–27.

[23] C. Guo, L. Yuan, D. Xiang, Y. Dang, R. Huang, D. Maltz, Z. Liu, V. Wang, B. Pang, H. Chen *et al.*, "Pingmesh: A large-scale system for data center network latency measurement and analysis," *ACM SIGCOMM Computer Communication Review*, vol. 45, no. 4, pp. 139–152, 2015.

[24] H. Zeng, P. Kazemian, G. Varghese, and N. McKeown, "Automatic test packet generation," in *Proc. ACM CoNEXT*, 2012, pp. 241–252.

[25] F. Momot, S. Bratus, S. Hallberg, and M. Patterson, "The seven turrets of babel: A taxonomy of langsec errors and how to expunge them," in *Proc. IEEE Conference on Secure Development*, 2016.

[26] T. Ball, N. Bjørner, A. Gember, S. Itzhaky, A. Karbyshev, M. Sagiv, M. Schapira, and A. Valadarsky, "Vericon: Towards verifying controller programs in software-defined networks," in *ACM SIGPLAN Notices*, vol. 49 (6), 2014, pp. 282–293.

[27] P. Bosshart, D. Daly, G. Gibb, M. Izzard, N. McKeown, J. Rexford, C. Schlesinger, D. Talayco, A. Vahdat, G. Varghese, and D. Walker, "P4: Programming protocol-independent packet processors," *SIGCOMM Comput. Commun. Rev.*, vol. 44, no. 3, pp. 87–95, Jul. 2014.

[28] T. Nelson, A. D. Ferguson, M. J. Scheer, and S. Krishnamurthi, "Tierless programming and reasoning for software-defined networks," in *Proc. 11th USENIX Symposium on Networked Systems Design and Implementation (NSDI)*, 2014, pp. 519–531.

[29] Y. Velner, K. Alpernas, A. Panda, A. Rabinovich, M. Sagiv, S. Shenker, and S. Shoham, "Some complexity results for stateful network verification," in *Proc. International Conference on Tools and Algorithms for the Construction and Analysis of Systems (TACAS)*, 2016, pp. 811–830.

[30] P. Peresíni and D. Kostic, "Is the network capable of computation?" in *Proc. 21st IEEE International Conference on Network Protocols (ICNP)*, 2013.

[31] C. Newport and W. Zhou, "The (surprising) computational power of the SDN data plane," in *Proc. IEEE INFOCOM*, 2015, pp. 496–504.

[32] M. Sipser, *Introduction to the Theory of Computation*, 2nd ed. Thomson Course Technology, 2006.

[33] Cisco, "MPLS traffic engineering fast reroute–link protection," in *website http://www.cisco.com/c/en/us/td/docs/ios/12_0s/feature/guide/fslinkpt.html*, 2017.

[34] J. Büchi, "Regular canonical systems," *Arch. Math. Logik u. Grundlagenforschung*, vol. 6, pp. 91–111, 1964.

[35] J. Esparza, D. Hansel, P. Rossmanith, and S. Schwoon, "Efficient algorithms for model checking pushdown systems," in *Proc. 12th International Conference on Computer Aided Verification (CAV)*, 2000.

APPENDIX

TABLE I: Forwarding Tables for Figures 2 and 3. At the *top*, we list the standard Forwarding Table (FT), followed by two versions of the failover rules at the *bottom*. In-I and Out-I stands for incoming resp. outgoing interface.

FT	In-I	In-Label	Out-I	op
τ_{v_1}	in_1	$\bot$	(v_1, v_2)	$push(10)$
	in_2	$\bot$	(v_1, v_2)	$push(20)$
τ_{v_2}	(v_1, v_2)	10	(v_2, v_3)	$swap(11)$
	(v_1, v_2)	20	(v_2, v_3)	$swap(21)$
τ_{v_3}	(v_2, v_3)	11	(v_3, v_4)	$swap(12)$
	(v_2, v_3)	21	(v_3, v_8)	$swap(22)$
	(v_7, v_3)	11	(v_3, v_4)	$swap(12)$
	(v_7, v_3)	21	(v_3, v_8)	$swap(22)$
τ_{v_4}	(v_3, v_4)	12	out_1	pop
τ_{v_5}	(v_2, v_5)	40	(v_5, v_6)	pop
τ_{v_6}	(v_2, v_6)	30	(v_6, v_7)	$swap(31)$
	(v_5, v_6)	30	(v_6, v_7)	$swap(31)$
	(v_5, v_6)	61	(v_6, v_7)	$swap(62)$
	(v_5, v_6)	71	(v_6, v_7)	$swap(72)$
τ_{v_7}	(v_6, v_7)	31	(v_7, v_3)	pop
	(v_6, v_7)	62	(v_7, v_3)	$swap(11)$
	(v_6, v_7)	72	(v_7, v_8)	$swap(22)$
τ_{v_8}	(v_3, v_8)	22	out_2	pop
	(v_7, v_8)	22	out_2	pop

local FFT	Out-I	In-Label	Out-I	op
π_{v_2}	(v_2, v_3)	11	(v_2, v_6)	$push(30)$
	(v_2, v_3)	21	(v_2, v_6)	$push(30)$
	(v_2, v_6)	30	(v_2, v_5)	$push(40)$

global FFT	Out-I	In-Label	Out-I	op
π'_{v_2}	(v_2, v_3)	11	(v_2, v_6)	$swap(61)$
	(v_2, v_3)	21	(v_2, v_6)	$swap(71)$
	(v_2, v_6)	61	(v_2, v_5)	$push(40)$
	(v_2, v_6)	71	(v_2, v_5)	$push(40)$

A. Proof of Theorem 1

We shall briefly argue that the problems are undecidable already for $k = 0$ (and for any higher number the δ_v^F functions can be defined as in the case of $F = \emptyset$). We will show that we can encode the computation of a Turing machine as a packet routing sequence in the simple network depicted in Figure 4 (the network contains a self-loop that can be easily removed by adding another node that is simply forwarding all packets back to v without any modification). The idea is to encode the Turing machine configuration (a finite sequence of tape symbols that contains exactly one control state so that the head points to the next tape symbol after the control state) as a header of a packet that is repeatedly routed through interface out_1 and in_1. Initially, a packet arrives at the input interface in and the node v replaces the header with the initial configuration of the Turing machine and forwards it to the outgoing interface out_1. Now, every time a packet arrives at the incoming interface in_1 with the header h, the node v will update the header h to h' so that it encodes a one step computation of the Turing machine (this is surely computable) and forwards the packet with the header h' again at the outgoing port out_1, except for the situation when the header contains an accepting or rejecting control state of the Turing machine. In this situation, the header does not change (and is nonempty as it contains at least the control state) and the packet is forwarded to the outgoing interface out. As the halting problem of a Turing machine is undecidable, this implies that also the interface connectivity problem (can the packet reach the outgoing interface out?) as well as the cyclic routing problem (is the packet forwarding sequence finite or infinite?) are undecidable. By the same arguments the transparency problem for the interfaces in and out is also undecidable: if the machine halts then the packet arriving to in reaches the interface out with a nonempty header and the network is not transparent; if the machine loops, the packet never arrives at out, and the network is transparent.

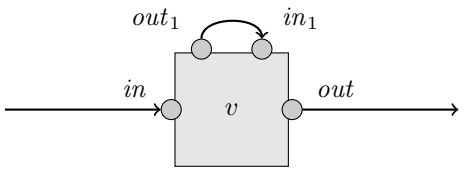

Fig. 4: Simulation of Turing machine computation

B. Proof of Theorem 2

We can enumerate all routing sequences of length at most $r|V| + 1$ (there are finitely many of those). If in any of such routings there is a node that repeats more than r times, then we know that the answer to the r-repeated routing problem is positive: if there is a sequence longer than $r|V| + 1$, then it must for sure contain some node that is repeated more than r-times in its initial prefix of length $r|V| + 1$. If none of the routing sequences of length at most $r|V| + 1$ contains any node repeated more than r times, the answer to the r-repeated routing problem is negative.