A Fast and Quantitative
What-If Analysis Tool for MPLS Networks

Peter Gjøl Jensen  
Aalborg University  
Denmark

Dan Kristiansen  
Aalborg University  
Denmark

Stefan Schmid  
University of Vienna  
Austria

Morten Konggaard Schou  
Aalborg University  
Denmark

Bernhard Clemens Schrenk  
University of Vienna  
Austria

Jiří Srba  
Aalborg University  
Denmark

ABSTRACT
We present an automated what-if analysis tool for MPLS networks which allows us to verify both logical properties (e.g., related to the policy compliance) as well as quantitative properties (e.g., concerning the latency) in polynomial time and even under multiple link failures. Our tool relies on weighted pushdown automata, a quantitative extension of classic automata theory, and takes into account the actual dataplane configuration, rendering it especially useful for debugging. In particular, our tool collects the different router configurations and then builds a pushdown system, on which quantitative reachability is performed based on an expressive query language. Our experiments show that our tool outperforms state-of-the-art approaches (which have been until now restricted to logical properties) by several orders of magnitude; furthermore, our quantitative extension only entails a moderate overhead in terms of runtime. The tool comes with a platform-independent user interface and will be released open-source, together with all other experimental artefacts.

1 INTRODUCTION
While communication networks are a critical infrastructure of our digital society, their correct configuration and operation is complex, requiring operators to become "masters of complexity" [17]. As many recent network outages were caused by human errors, e.g., [7, 12, 13], we currently witness major research efforts toward more automated and formal approaches to operate and verify networks [4, 6, 14, 15, 19, 20, 24, 28, 32].

A particularly critical but challenging task for human operators is to reason about failures. In order to meet their stringent dependability requirements, most modern communication networks come with fast recovery mechanisms which revert traffic to alternative paths [10, 11, 23, 26]. While this is attractive, already a single link failure can lead to unintended network behaviors which are easily overlooked and may violate the network policy [7]. Especially multiple link failures, which are more likely to occur in large networks and can be caused e.g. due to shared risk link groups [5, 16, 25], may threaten network dependability.

It is often insufficient to only ensure the logical correctness (e.g., policy compliance) of the network behavior under failures. A dependable network also needs to satisfy quantitative properties and meet certain performance guarantees. For example, traffic should be rerouted along short paths, e.g., regarding link latency (offering a low latency) or number of hops (reducing load), even under a certain number of link failures.

We are particularly interested in networks based on Multiprotocol Label Switching (MPLS) [2]. MPLS networks are widely deployed in the Internet today, especially in IP networks and for traffic engineering purposes. The study of MPLS networks is also interesting from a theoretical perspective, as the header size in these networks is not fixed; rather, additional labels may be pushed on the header while rerouting packets around failed links, creating "tunnels". This makes the employment of formal methods particularly challenging as we must deal with a possibly unbounded set of packet headers.

Our Contributions. We present a what-if analysis tool for MPLS networks, Tool (name changed for double-blind submission), which supports a fully automated and fast verification of the network behavior under failures. In particular, Tool relies on an expressive query language and allows us to test both logical properties (such as the policy compliance) as well as quantitative properties (such as the latency, number of hops, required label stack size resp. tunneling depth, or number of failed links), in polynomial-time and under an arbitrary number of link failures. Tool operates directly on the dataplane configuration, allowing to debug issues not visible in the control plane.
At the heart of Tool lies a weighted pushdown automaton, a quantitative generalization of classical automata: based on the router configurations and a query (the input to the tool), we build a weighted pushdown system and then perform a quantitative reachability analysis. To improve performance, Tool uses novel algorithms tailored to this use case.

We offer an optimized C++ implementation of Tool and report on its platform-independent user interface. Considering a case study in cooperation with a major network operator, we show that our tool outperforms state-of-the-art approaches (only applicable to verification of logical properties) by several orders of magnitude in terms of runtime. We also demonstrate that our quantitative extension only entails a reasonable overhead. As a contribution to the research community and in order to ensure reproducibility, we release our tool as open source together with this paper, and we also share all our experimental artefacts. We invite the reviewer to try out our (anonymized) tool interactively at https://nettool.z000.org/.

Related work and novelty. The problem of how to render networks more automated and formally verifiable has recently received much interest, both for specific networks and protocols, such as BGP [31], OpenFlow [4, 20], or MPLS [28] networks, as well as for networks which are protocol agnostic [19]. The different systems rely on different approaches, including e.g., algebraic approaches [4], static verification based on geometric approaches [19], or automata-theoretic approaches [18]. We specifically consider MPLS networks and use an automata-theoretic approach. We directly verify the router configurations, which allows us to also catch errors in the dataplane [30].

Unlike most existing tools, we focus on polynomial-time verification, even under failures: many existing approaches in the literature do not consider failure scenarios explicitly (and still exhibit a super-polynomial runtime, e.g., due to SAT solving [24]), and/or have to iterate through all possible failure scenarios (and dataplane configurations) [20], which introduces a combinatorial complexity. Furthermore, most existing approaches do not support arbitrary header sizes, which however arise in the context of MPLS networks: by representing MPLS networks symbolically as pushdown automata, we hence achieve an exponential speedup.

To the best of our knowledge, our tool is the first to consider what-if analysis of quantitative aspects as well, and we are not aware of any applications of weighted automata theoretical results in this context. The few weighted solutions that exist do not consider failure scenarios and have an exponential runtime [22]. The paper closest to ours is P-Rex [18], a polynomial-time approach to verify logical properties of MPLS networks, also accounting for possible failures. As we demonstrate in our evaluation, our tool not only adds the novel quantitative dimension, but also outperforms P-Rex by several orders of magnitude. We further contribute an interactive user interface and make a leap forward regarding applicability for network operators.

2 MPLS NETWORK MODEL

We first introduce a formal model of an MPLS network and the query language.

2.1 Network definition

An MPLS network consists of a network topology together with the set of forwarding rules.

Definition 1. A network topology is a directed multigraph \((V, E, s, t)\) where \(V\) is a set of routers, \(E\) is a set of links between routers, \(s : E \rightarrow V\) assigns the source router to each link, and \(t : E \rightarrow V\) assigns the target router.

We assume that links in the network can fail. This is modelled by a set \(F \subseteq E\) of failed links. A link is active if it belongs to \(E \setminus F\). We assume asymmetric link failures that can be caused e.g., by congestion in one direction, resulting in packet drops that can also appear as a link failure.

Let \(L\) be a nonempty set of MPLS labels used in packet headers. We define the set of MPLS operations on packet headers as \(Op = \{\text{swap}(\ell) \mid \ell \in L\} \cup \{\text{push}(\ell) \mid \ell \in L\} \cup \{\text{pop}\} \). Given an alphabet \(A\), let \(A^\ast\) denote the set of all finite words over the elements of \(A\), including the empty word \(\epsilon\).

Definition 2. An MPLS network is a tuple \(N = (V, E, s, t, L, \tau)\) where \((V, E, s, t)\) is a network topology, \(L = L_M \sqcup L_{IP} \sqcup L_{IP}\) is a finite set of labels partitioned into (1) the MPLS label set \(L_M\), (2) the set of MPLS labels with the bottom of the stack bit set to true \(L_{IP}\), (3) a set of IP addresses \(L_{IP}\), and \(\tau : E \times L \rightarrow (2^{E \cup Op})^\ast\) is the routing table.

The routing table, for every link \(e \in E\) and a top (leftmost) packet label \(\ell\), returns a sequence of traffic engineering groups \(t(e, \ell) = O_1 O_2 \ldots O_n\) where each traffic engineering group is a set of the form \(\{(e_1, \omega_1), \ldots, (e_m, \omega_m)\}\) where \(e_j\) is the outgoing link such that \(t(e) = s(e_j)\) and \(\omega_j \in Op^\ast\) is a sequence of operations to be performed on the packet header. In a given traffic engineering group, we can nondeterministically select any active link and forward the packet via that link while applying the corresponding sequence of MPLS operations. This allows us to abstract away from various routing policies that facilitate e.g. splitting of a flow along multiple shortest paths. The group \(O_i\) has a higher priority than \(O_{i+1}\) and during the forwarding, we always select the traffic engineering group with the highest priority and at least one active link.
l ∈ L, h ∈ H and e is the empty sequence of operations:

\[ \begin{aligned}
&\mathcal{H}(\ell h, \omega) = \begin{cases}
\ell h & \text{if } \omega = \epsilon \text{ and } l \in L \\
\mathcal{H}(\ell'h\omega') & \text{if } \omega = \text{swap}(\ell') \circ \omega' \text{ and } l'h \in H \\
\mathcal{H}(h\omega, \omega') & \text{if } \omega = \text{push}(\ell') \circ \omega' \text{ and } l'h \in H \\
\mathcal{H}(h, \omega') & \text{if } \omega = \text{pop} \circ \omega' \text{ and } l \in L_M \cup L_M^L \\
\text{undefined} & \text{otherwise .}
\end{cases}
\end{aligned} \]

Let \( L_M = \{30, 31\}, L_M^L = \{s20, s21\} \) and \( L_P = \{ip_1\} \). We use here and in what follows the convention that all labels that are on the bottom of the MPLS label stack (have the bottom of stack bit set to true) are prefixed with small s. Then \( \mathcal{H}(30 \circ s20 \circ ip_1, \text{pop} \circ \text{swap}(s21) \circ \text{push}(31)) = 31 \circ s21 \circ ip_1 \).

### 2.3 Example network

An example of a simple network topology is given in Figure 1a together with the forwarding table described in Figure 1b. The example defines two label switching paths for IP-packet routing from \( v_0 \) to \( v_3 \), either via the links \( e_1 \) and \( e_4 \), or the links \( e_2 \) and \( e_3 \). The respective path can be selected nondeterministically. Moreover, packets arriving via the link \( e_5 \) with the service label \( s40 \) (agreement with the neighboring network operator) are routed via the links \( e_1, e_5, e_6 \) and leave the network on the link \( e_7 \).

Every forwarding rule for the router \( v \) is represented by a line in the table and depending on the incoming link \( e_m \) (where \( t(e_m) = v \)) and the top of the stack label, it determines the outgoing link \( e_{out} \) (where \( s(e_{out}) = v \)) and a sequence of stack operations that replace the top label. Each such rule has a priority that is depicted by the priority column in the middle of the table. In our example, it is only the router \( v_2 \) that has more than one priority group associated to its forwarding table in order to protect the link \( e_4 \). If a packet arrives via the link \( e_1 \) with label \( s20 \) on top of the stack then it is primarily forwarded via the link \( e_4 \) while the label is swapped with \( s21 \). Only if the link \( e_4 \) fails, a backup rule with priority 2 is used so that it forwards the traffic via the link \( e_5 \), first swapping the top label with \( s21 \) and then pushing a new label \( 30 \) on top of the label stack. The router \( v_4 \) then pops the label and the packet arrives to \( v_3 \) with the same label as if the link \( e_4 \) did not fail.

### 2.4 Network traces

We now define valid traces in an MPLS network \( N = (V, E, s, t, L, tr) \) under the assumption that the links in the set \( F \subseteq E \) failed. For a traffic engineering group \( O = \{ (e_1, \omega_1), (e_2, \omega_2), \ldots, (e_m, \omega_m) \} \) we let \( E(O) = \{ e_1, e_2, \ldots, e_m \} \) denote the set of all links in the group. The group \( O \) is active if it contains at least one active link, i.e. \( E(O) \cap F = \emptyset \). Further, we define \( \mathcal{A}(O_1, O_2, \ldots, O_n) = \{ (e, \omega) \in O_j \mid e \in \text{active link} \} \) where \( j = \text{the lowest index such that } O_j \text{ is an active traffic engineering group, and we let} \)

---

**Figure 1:** A small network example

### 2.2 Valid MPLS headers

The MPLS labels can be nested only in a specific way. For a given network \( N = (V, E, s, t, L, tr) \), we define the set of valid headers \( H \subseteq L^* \) by \( H = L_P \cup \{ \alpha t_0 \mid \alpha \in L_M^L, t_0 \in L_P \} \). MPLS operations now manipulate the label-stack header by modifying the topmost label. The semantics of the operations must ensure that the result of operations performed on a valid header is itself a valid header.

**Definition 3.** The semantics of MPLS operations is a partial header rewrite function \( \mathcal{H} : H \times O^* \rightarrow H \) where \( \omega, \omega' \in O^* \),
We present a powerful query language that allows us to match any router so that if and outgoing link in the network. If this is the case, the query \( \varphi \) is satisfied and we call \( \sigma \) a witness trace.

Examples of queries are provided in Figure 1d. The query \( \varphi_0 \) asks about the existence of a trace that starts and ends with the label-stack containing only the IP header, such that the first link is incoming to the router \( v_0 \), followed by zero or more hops via unspecified links, and ending with link that leaves the router \( v_3 \), all under the assumption of no link failures. The traces \( \sigma_0 \) and \( \sigma_1 \) satisfy the query, and we defer the discussion of the additional examples to Appendix A.1.

3 QUANTITATIVE EXTENSION

After describing the syntax and semantics of our network model and the query language used in our tool, we now present a novel extension of the framework which allows us to account for quantitative aspects, like latency, number of hops, tunnels (label stack size), number of failures, and linear combinations of these measures.

For a given network query, there can be several network traces that satisfy the query and for some queries there exist even infinitely many witness traces. From the user perspective, it is hence essential that when debugging the reasons why a certain query holds, we can impose quantitative constrains on the traces and specify what kind of witness traces we wish to visualize. For traffic engineering purposes we may want to find a trace that has the lowest latency or the smallest number of hops. We may be interested in finding a trace that minimizes tunneling depth or the number of failed links required to execute a given trace, or we may wish to find a trace that balances several such measures simultaneously.

We shall start by defining atomic quantitative properties of network traces. Let \( N = (V, E, s, t, L, \tau) \) be an MPLS network and let \( F \subseteq E \) be the set of failed links. An atomic quantity is a function \( p : ((E \setminus F) \times H)^* \rightarrow \mathbb{N}_0 \) that for a given trace \( \sigma \) evaluates to a non-negative integer \( p(\sigma) \) representing the quantitative measure of the trace. In our tool, we support the following atomic quantities of a network trace \( \sigma = (e_1, h_1) \ldots (e_n, h_n) \):

- **Links**: \( L(\sigma) = n \) is the length of the trace,
- **Hops**: \( H(\sigma) = |\{ e \in \{e_1, \ldots, e_n\} | s(e) \neq t(e) \} | \) is the number of hops, where we avoid counting links that are self-loops,
- **Distance**: \( D(\sigma) = \sum_{i=1}^{n} d(e_i) \) is the distance for any distance function \( d : E \rightarrow \mathbb{N}_0 \), e.g., the geographical distance, latency or e.g. inverse bandwidth capacity,
- **Failures**: \( F(\sigma) = \sum_{i=1}^{n} | \text{failed}(i) | \) where \( \text{failed}(i) = \{ e | (e, \omega) \in O_k, 1 \leq k < j \} \), where \( \tau(e, \text{head}(h_i)) = \).

We assume here a standard syntax for regular expressions and by \( \text{Lang}(a) \), \( \text{Lang}(b) \) and \( \text{Lang}(c) \) we understand the regular language defined by the expressions \( a \), \( b \) and \( c \), respectively. For specifying labels in the regular expressions \( a \) and \( c \) we use the abbreviations:

- \( \text{ip} = [i_{p_0}, \ldots, i_{p_n}] \) where \( L_{\text{ip}} = \{ i_{p_0}, \ldots, i_{p_n} \} \),
- \( \text{mpls} = [\ell_0, \ldots, \ell_n] \) where \( L_{\text{mpls}} = \{ \ell_0, \ldots, \ell_n \} \), and
- \( \text{smpls} = [\ell_0^+, \ldots, \ell_n^+] \) where \( L_{\text{smpls}} = \{ \ell_0^+, \ldots, \ell_n^+ \} \).

We further use the following notation for specifying links in the network. If \( u \) and \( v \) are routers, then \([u\#u]\) matches any link \( e \) from \( v \) to \( u \) such that \( s(e) = v \) and \( t(e) = u \). If \( in_1 \) is an interface on router \( v \) that uniquely identifies the outgoing link \( e \), and \( in_2 \) identifies the incoming interface on router \( u \) for the link \( e \), then \([v, in_1\#u, in_2]\) matches exactly the link \( e \). The dot-syntax is used to denote any link in the network and it is extended to match also any router so that \([v\#u] = \bigcup_{u \in V} [v\#u] \) and \([#u] = \bigcup_{u \in V} [v\#u] \).

Problem 1 (Query Satisfiability Problem). Given an MPLS network \( N \) and a query \( \varphi = \langle a > b < c > k \rangle \), decide if there exists a trace \( \sigma = (e_1, h_1) \ldots (e_n, h_n) \) in the network \( N \) for some set of failed links \( F \) such that \( |F| \leq k \) where \( h_1 \in \text{Lang}(a), e_1 \ldots e_n \in \text{Lang}(b), \) and \( h_n \in \text{Lang}(c) \).
Consider again the traces for our running example from Figure 1c. We have $\text{Hops}(\sigma_6) = \text{Links}(\sigma_6) = 4$ and $\text{Hops}(\sigma_5) = \text{Links}(\sigma_5) = 5$. We also observe that $\text{Failures}(\sigma_2) = 1$ while $\text{Failures}(\sigma_3) = 0$. Finally, we can see that e.g. $\text{Tunnels}(\sigma_1) = 1$, $\text{Tunnels}(\sigma_2) = 2$ and $\text{Tunnels}(\sigma_3) = 0$.

We can now combine the atomic quantities in order to define composed criteria for trace weight specification, by constructing linear expressions of the form

$$expr ::= p \mid a \cdot expr \mid expr_1 + expr_2$$

where $p$ is an atomic quantity and $a \in \mathbb{N}$. A vector of linear expressions $(expr_1, expr_2, \ldots, expr_n)$ allows us to specify trace properties by priorities, so that $expr_1$ has a higher priority than $expr_2$ etc. For a trace $\sigma$, there is a natural evaluation of linear expressions to nonnegative integers and for a vector of linear expressions, we assume the lexicographical ordering $\leq$ on vectors of nonnegative integers, by abuse of notation extended to traces.

**Problem 2 (Minimum Witness Problem).** For a network, a query that is satisfied in the network and a vector of linear expressions $(expr_1, expr_2, \ldots, expr_n)$, we want to find a witness trace $\sigma$ such that $\sigma \leq \sigma'$ for any other witness trace $\sigma'$.

Consider the query $\phi_4$ in our running example from Figure 1 where we want to find witness traces that will minimize the vector $(\text{Hops}, \text{Failures} + 3 \cdot \text{Tunnels})$. The query has two witness traces $\sigma_2$ and $\sigma_3$ and when we evaluate them on the minimization vector, we get the pair $(5, 1 + 3 \cdot 2) = (5, 7)$ for $\sigma_2$ and $(5, 0 + 3 \cdot 0) = (5, 0)$ for $\sigma_3$. As lexicographically $(5, 0) \leq (5, 7)$, the answer to the minimum witness problem is the trace $\sigma_3$. In general, there can be several minimum witness traces, and we may return any of those, or add another minimization criterion to the vector of linear expressions.
After this, we perform a series of reductions (based on static analysis) on the constructed (weighted) pushdown automaton by removing redundant rules in order to decrease its size. The reduced pushdown is then sent either to the Moped engine (possible only if the weight requirements are not specified) or to our solver that accepts both weighted and unweighted pushdown automata. If the verification result says that the query is not satisfied, we achieve a conclusive answer and report it to the GUI. Otherwise, the produced network trace must be verified for its feasibility and the fact that it does not exceed in total \(k\) link failures (for a fixed trace this can be done in polynomial time). If the trace reconstruction succeeds, we have a witness trace (possibly one of the minimal ones in case the weight objective is given) and we can report that a query is satisfied. Otherwise, our tool constructs an under-approximating pushdown automaton where we add a global failed link counter and use this counter to guarantee that the total number of failed links is not exceeded. This produces only an under-approximation as in case of traces with loops, we may count the same failed link twice. If a valid trace is generated by the under-approximation, we can return it as a witness trace. Otherwise the answer is inconclusive. In our experiments on a real operator network, the answer was inconclusive for only 8 out of 6,000 queries (0.13% of the total)—a more expensive analysis is then needed.

5 PERFORMANCE EVALUATION

We evaluate the performance of our tool on a real-world operator network (anonymized for the double-blind peer-review phase), with 31 routers and more than 250,000 forwarding rules. The operator uses an advanced MPLS routing in its network, including numerous service labels by which it communicates with neighboring networks. In order to increase the variety of different types of networks, we create several variants of networks from Internet Topology Zoo [1] with queries of the type like in Table 1 and in our running example. The experiments were run on our cluster with AMD EPYC 7551 processors running at 2.55 GHz with boost disabled, using 16GB memory limit and 10 minutes timeout.

The operator asked us to verify a number of specific queries, including those in Table 1. The table shows the verification time for using Moped as the backend engine, our own engine (called Dual as it combines the over- and under-approximation approach as described in the previous section) and our weighted verification engine that uses the \(\text{Failures}\) atomic quantity. The results show that our weighted engine is marginally faster than Moped but our unweighted engine is almost 8 times faster than Moped. The overhead of performing quantitative analysis is hence reasonable as it can still outperform the state-of-the-art unweighted tool. We also note that [18] reports that the unweighted verification of similar queries on a network of comparable size took between 28 minutes (for the simpler queries) and up to 109 minutes (for the more complex ones). This shows an improvement of several orders of magnitude and makes it possible to perform MPLS verification interactively for human operators, in particular in combination with our graphical user interface (the tool prototype from [18] is a command-line tool).

Finally, the cactus plot in Figure 3 shows a comparison (note the logarithmic scale) of the verification times (in seconds) between Moped, our Dual unweighted approach...
and our weighted engine with the quantity Failures. The plot includes over 1800 experiments on different queries on the networks from the Internet Topology Zoo database, ordered by their verification times. Again, we outperform Moped by an order of magnitude by using our unweighted engine. An interesting phenomenon can be observed for our weighted engine that behaves similarly as Moped on the smaller instances; however, on the difficult instances it outperforms our unweighted implementation. This is due to the fact that the guided search for shortest traces, that minimize the number of failures, allows us to find witness traces that are otherwise not discovered by the unweighted search; this highlights the benefits of quantitative analysis.

6 CONCLUSION
We presented an MPLS what-if analysis tool that not only provides an unprecedented performance in theory but also in practice, as demonstrated in our case study with a major network operator. We regard our contribution concerning the automated analysis of quantitative aspects as a first step, and believe that our paper opens interesting avenues for future research in this area. In collaboration with network operators, we are extending the tool to further improve the expressiveness of the query language.

ACKNOWLEDGMENTS
This research was supported by the WWTF project WHATIF under grant number ICT19-045.

REFERENCES
with the exception that we allow for up to 2 link failures and $e$.

The traces $\sigma$ and $\hat{\sigma}$ on top of the IP header. Indeed, the trace $\sigma_3$ has this property and it is a valid trace even in case of no link failures. The next query $\phi_3$ checks the transparency of the routing from $v_0$ to $v_3$ by asking whether a packet with the service label $s40$ can leave our network with at least one additional MPLS label on top of the service label. Should this be the case then our network leaks internal MPLS labels to the neighboring networks, which is not desirable. Even in case of one link failure, the query is not satisfied. Finally, the query $\phi_4$ asks whether in case of one link failure there is an IP routing, with an optional MPLS label on the top of the IP label, with three or more hops between the incoming and outgoing links, and this is indeed the case as documented by the witness traces $\sigma_2$ and $\sigma_3$. In case of no link failures, the query is satisfied only by the trace $\sigma_3$.

A.2 Graphical user interface

The front end of our tool provides a web-browser based visualization. The graphical interface allows us to load a number of predefined networks from the Internet Topology Zoo [1], our operator network used in the experiments as well as the running example used in this tool paper. In the interface we can specify the query, including an online help for router names with interfaces as well as the sets of labels tested at each router. In options, we can set different parameters for the tool and graphically create the vector of linear expressions for the minimum trace specification. If a witness trace is discovered, the GUI visualizes the trace including the operations performed at each router. A screenshot in Figure 7 shows how to specify the minimization vector ($Hops, Failures, +3 · Tunnels$) and the corresponding witness trace.

The GUI is written in JavaScript and the source code is available under the GPL3 license. The backend verification engine is running on a web server but there is also a packaged version of the visualization tool available that can be run locally without the use of a web server (and allows to input additional MPLS networks created by the user). In order to comply with the double-blind submission policy, we created an anonymized URL hosting our tool: https://nettool.z000.org/. The tool has also a homepage with a video tutorial explaining how to use it, however, the links cannot be revealed in the peer-review stage.
Figure 4: Running example loaded in the GUI