Efficient SMT-based Network Fault Tolerance Verification

Yu Liu[1], Pavle Subotic[2][0000−0002−6536−3932], Emmanuel Letier[3][0000−0002−8935−343X], Sergey Mechtaev[3][0000−0001−6088−4993], and Abhik Roychoudhury[1][0000−0002−7127−1137]

[1] National University of Singapore, Singapore {liuyu,abhik}@comp.nus.edu.sg
[2] Microsoft, Serbia pavlesubotic@microsoft.com
[3] University College London, United Kingdom {e.letier,s.mechtaev}@ucl.ac.uk

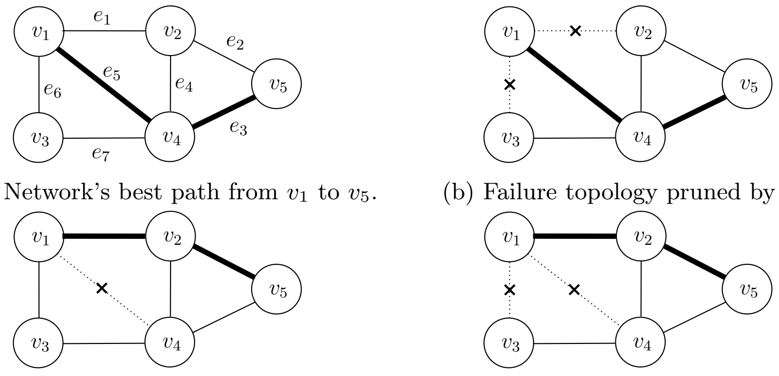
Abstract. Network control planes are highly sophisticated, resulting in networks that are difficult and error-prone to configure. Although several network verification tools have been developed to assist network operators, they are limited and inefficient in handling fault-tolerance policies. In this paper, we propose a novel SMT encoding to speed up control plane fault tolerance verification by pruning failed topologies. This encoding exploits the observation that the verifier has to check failures only for the links lying on a set of best paths which can be computed by a recursive algorithm. We implemented our technique in Minesweeper, a state-of-the-art SMT-based verifier. Our evaluation shows that the new encoding speeds up verification by the factor of 3.1-26.9X.

1 Introduction

Correctly configuring modern computer networks is hard due to their size and complexity. The total lines of the low-level network configuration code may reach millions [6]. To alleviate network operators' burden, automatic control plane verifiers [4,16,1] have been proposed. These tools take network configurations as input and analyze them to gauge the possible routing behaviours. Then, they answer questions such as whether router A computes a route to router B, *i.e.* reachability policy, or whether router A computes a route to router B regardless of any number of failed links, *i.e.* fault-tolerance reachability policy.

SMT-based network verifiers have been successfully employed on large scale industrial networks [8,9]. These tools encode the network verification problem in logic and rely on the SMT solver to check the encoded property. For these verifiers, fault-tolerant policies are particularly challenging. SMT-based verifiers are inefficient in checking fault-tolerance policies, because their algorithms do not scale to the number of possible failed topologies, which combinatorially grows with the number of failed links. For example, a widely-used verifier Minesweeper [4] encodes all possible failed topologies using SMT constraints.

To address the combinatorial explosion of the failed topologies, we propose an SMT encoding that prunes the space of topologies by eliminating those that are



(a) Network's best path from v_1 to v_5. (b) Failure topology pruned by RBP.

(c) New best path after the link e_5 fails. (d) Extra failure topology pruned by RBP.

Fig. 1: Topologies pruned by recursive best path optimization.

guaranteed not to change the verification result. We refer to this encoding as *recursive best path optimization (RBP)*. Our SMT encoding represents information about best paths using logical constraints, and applies cardinality constraints to control the number of network failures. This proposed encoding significantly improves performance of fault tolerance verification in similar tools [4].

We have implemented our proposed approach as the open source tool[4] Trailblazer, a variant of Minesweeper with the RBP encoding. We have evaluated Trailblazer on three benchmarks from the zoo topology [14] and a network from Rocketfuel [17]. The results show that Trailblazer is 3.1-26.9X faster than Minesweeper, while being able to verify the same policies. Our contributions are listed as the following:

- A novel SMT encoding for optimising network fault tolerance verification.
- A comprehensive evaluation of Trailblazer on widely-used benchmarks.

2 Motivation

Let V be the nodes of a network, and $E \subseteq V \times V$ be its edges. The SMT-based verifier Minesweeper encodes the network fault-tolerance verification problem under k link failures into the formula $\psi \wedge \xi_k \wedge \neg\pi$, where ψ encodes network behaviour, π is the property of interest, and ξ_k encodes that the number of failed links, controlled by the pseudo-boolean variables failed_e, is bound by k:

$$\xi_k \triangleq \Sigma_{e \in E} \mathit{failed}_e \leq k \tag{1}$$

If an SMT solver determines that the formula is unsatisfiable, the policy holds on all possible network topologies with at most k link failures.

[4] https://github.com/rainLiuplus/trailblazer

Consider an example network in Figure 1, in which each router forwards traffic via the shortest path. For example, the router v_1 reaches the router v_5 via the path e_5e_3 (Figure 1a). Assume that our goal is to verify the fault tolerance property that v_1 reaches v_5 under two link failures. Since the network consists of seven edges, there are $C(7{,}2) = 21$ possible failure topologies. Thus, an SMT solver implicitly checks the property for the 21 failed topologies, and this number quickly grows with the size of the network.

Our key observation is that a more efficient encoding that takes into account the best path semantics of the network protocol can assist SMT solver in checking the fault tolerance property. Observe that since e_5e_3 is the shortest path, if the failures occur in any edges except e_5 and e_3, the reachability from v_1 to v_5 is not affected. Therefore, all the failed topologies in which both e_5 are e_3 are up can be pruned. An example of such a topology is given Figure 1b. Moreover, significantly more topologies can be pruned if we analyze the new best paths that appear after an edge on the previous best paths fails. Assume that e_5 fails. Then, e_5e_3 no longer exists, and the network computes a new best path from v_1 to v_5, *e.g.* e_1e_2 (Figure 1c). Then, we need to only consider all single-edge failures for this failed topology. Similar to the previous case, the topologies whose dropped links are not in $\{e_1, e_2\}$ can be pruned. Thus, the topologies whose two dropped links are in $\{(e_5, e_6), (e_5, e_7), (e_5, e_3), (e_5, e_4)\}$ can be pruned. An example is given in Figure 1d. The link e_2 can be handled similarly. Totally, this optimisation prunes 17 of 21 topologies. We refer to this optimisation as RBP.

We apply RBP by adding stronger constraints ξ_k. Instead of the formula 1, we encode network semantics under link failures it as

$$\xi_k \triangleq \mathit{failed}_{e_5} \rightarrow \mathit{AtLeast}(1, \{\mathit{failed}_{e_1}, \mathit{failed}_{e_2}\}) \wedge ...$$

where $\mathit{failed}_{e_5} \rightarrow \mathit{AtLeast}(1, \{\mathit{failed}_{e_1}, \mathit{failed}_{e_2}\})$ states that if e_5 fails, the second failed link should be on the new best path (Figure 1d). This is repeated recursively for all edges on the best paths up to a configurable depth d. This encoding significantly optimises constraint solving as shown in Section 4.

3 Trailblazer

In this section, we first provide background on network verification, then discuss the details of Trailblazer.

3.1 Background

For a node v and a destination node dst, a network protocol computes the *forwarding path(s)* FP_v, a set of sequences of edges for forwarding traffic to dst. We denote the set of forwarding paths of all nodes to dst as $\mathcal{FP} \triangleq \{FP_v | v \in V\}$. A network policy is a predicate ϕ over $\mathcal{FP}$. For example, $\mathit{Reachability}_v(\mathcal{FP})$ states that FP_v is not empty. To verify a policy is to check if $\phi(\mathcal{FP})$ holds. A failed link in the network topology changes the forwarding paths. So, to verify a policy ϕ under k failed links, we need to check ϕ on $C(|E|, k)$ failed topologies.

Predicate (ϕ)	Policy	Description
$Reachability_v$	$FP_v \neq \emptyset$	Traffic $v \rightarrow dst$ can reach dst.
$Waypoint_{(v,w)}$	$w \in nodes(FP_v)$	Traffic $v \rightarrow dst$ traverses node w.
$Isolation_v$	$FP_v = \emptyset$	Traffic $v \rightarrow dst$ cannot reach dst.
$Balance_v$	$\lvert FP_v \rvert \geq 2$	Mutiple paths for traffic $v \rightarrow dst$.

Table 1: Common Network Policies

Four common network policies are reachability, waypoint, isolation and balance shown in Table 1. Note that the balance policy is the only one that deals with sets of best paths.

Minesweeper is an SMT-based network verifier, which reduces the verification problem to an SMT problem. Minesweeper's encoding given in Section 2 is described in more details in sections 3-5 of their paper [4].

3.2 Our Approach

Trailblazer improves the SMT encoding of Minesweeper using additional constraints that capture network semantics under failures, and optimises the encoding using cardinality constraints. The key intuition of our optimisation is that failed links outside of the best path do not affect the network semantics. To capture this intuition, we add extra constraint λ_L to restrict the failure model:

$$\lambda_L \triangleq \Sigma_{e \in L} failed_e \geq 1 \tag{2}$$

where L is the set of links in the best path. The encoding of the failure model now becomes:

$$\xi_k \triangleq \Sigma_{e \in E} failed_e \leq k \wedge \lambda_L \tag{3}$$

We first acquire the best paths corresponding to all failure topologies by leveraging Batfish [9]. After obtaining this mapping, we encode them to SMT formulas. RBP works by recursively dropping a link in the best path, then treating the current network with one failed link as new network and compute the new best path on it. We encode these recursive best paths as formulas. For example, if the dropped link is e' and the corresponding best path is L', then they should be encoded as $failed_{e'} \rightarrow \lambda_{L'}$. Similarly, if another edge e'' fails in L' and the new best path is L'', the encoding should be $failed_{e'} \rightarrow failed_{e''} \rightarrow \lambda_{L''}$. The encoding could be interpreted as: if e' is broken and e'' is broken, the remained broken links can only be in L''. Let E_f be the set of failed links and L be the corresponding best path of the topology with E_f dropped. The constraint for restricting the failure model under E_f and L is:

$$\bigwedge_{e \in E_f} failed_e \rightarrow \lambda_L \tag{4}$$

This constraint states that if all links in E_f are down, then there is at least one link should be dropped in its corresponding best path L.

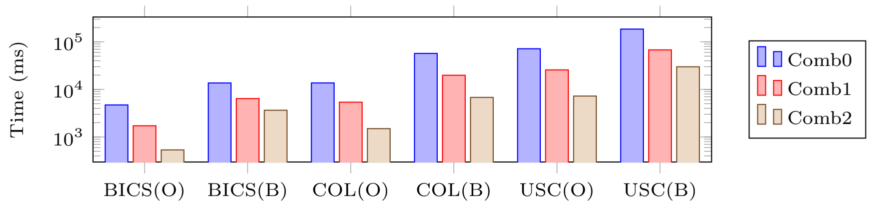
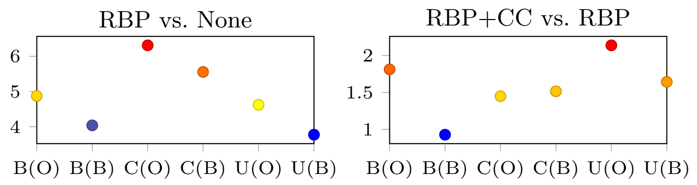


Fig. 2: Verification time of Minesweeper and Trailblazer for k=2, d=1.



(a) Recursive best path optimization. (b) Cardinality constraints.

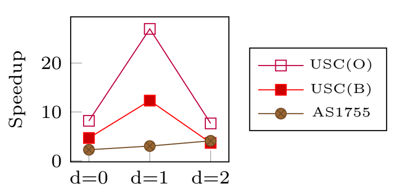
Fig. 3: Speedup for the two optimisations in isolation.

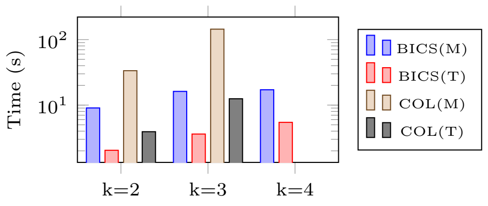
To efficiently encode the failure model, we applied cardinality constraints [2] instead of the integer inequalities used by Minesweeper. Cardinality constraints efficiently place a bound on the number of literals within a given set that can be assigned True. So, given the constraints in Equation (1) and Equation (2) where $failed_e$ is replaced with a boolean binary variable, they are naturally meet the semantics of AtMost and AtLeast constraints, respectively.

4 Evaluation

We conducted experiments on three networks selected from topology zoo[14]: Bics, Columbus and USCarrier, and the network AS1755 from Rocketfuel [17]. Bics consists of 33 routers and 48 links; Columbus has 70 routers and 85 links; USCarrier includes 158 routers and 189 links; AS1755 has 87 routers and 322 links. Each network from topology zoo has two types of network configuration, OSPF and BGP. We collect test policies by using Config2Spec's sampler [7], which infers policies from a data plane. We inferred four kinds of policies: reachability, waypoint, isolation and balance (see Section 3.1 for details). The policies that never hold were filtered out using the trimmer of Config2Spec. We considered four failure models where $k \in [1..4]$, representing there are at most 1, 2, 3 or 4 failed links respectively. We ran experiments on a machine with 64GB RAM and 56 virtual cores with 2.00 GHz. We cross-checked the verification outputs of Trailblazer and Minesweeper; their verification results were all consistent.

To compare Trailblazer with the vanilla Minesweeper and identify how the proposed techniques contribute to its results, we compared three configurations of Trailblazer: **Comb0** with no optimization (original Minesweeper), **Comb1** with recursive best path optimization, and **Comb2** with recursive best path





(a) Trailblazer's speedup with $d \in [0..2]$; k=3 for USC; k=4 for AS1755.

(b) Time of Minesweeper (M) and Trailblazer (T) with k=2,3,4 and d=1.

Fig. 4: Trailblazer's performance depending on failed links and recursion depth.

optimization + cardinality constraints (Section 3.2). We run the three combinations on BICS, Columbus and USCarrier under the failure model k=2 with the recursion depth d=1. Comb0 and Comb2 in Figure 2 shows the verification time of Minesweeper and Trailblazer. The time is averaged over all policies. The 'O' and 'B' inside brackets denote OSPF and BGP respectively. The average verification time of Minesweeper for OSPF networks is ranged from 4.71s to 71.63s, for BGP networks is 13.65s-185.53s. Trailblazer takes 0.53s-7.25s for OSPF networks and 3.64s-29.91s for BGP networks. Overall, Trailblazer is faster than Minesweeper by the factor of 3.75–9.88X.

To evaluate the effectiveness of RBP and cardinality constraints in isolation, we compared Comb0 with Comb1 and Comb1 with Comb2. The results are shown in Figure 3, where B,C,U represents BICS, Columbus, USCarrier respectively; O and B denote OSPF and BGP. RBP results in speedup from 3.78X to 6.31X. By using cardinality constraints, the verifier is accelerated by 0.92X-2.14X as shown in Figure 3(b).

To investigate how Trailblazer's performance vary with number of failed links and recursion depth, we conducted two sets of experiments. First, we set the tested failure models as k=2,3,4 and recursion depth d=1. The results of an experiment with Columbus and Bics are shown in Figure 4b. All sampled Columbus' policies are pre-pruned when k=4. The speedup ranges from 3.14X to 15.92X. Verification is accelerated the most when k=3, while it is improved the least when k=4. We speculate that when k is large, the policy tends not to hold, and thus it is easier for the solver to find a solution, thus it will benefit less from the pruned search space. Second, we set the failure model to k=3,4 and recursion depths to d=0,1,2. We use USCarrier and AS1755 as the experimental networks. The results shown in Figure 4a demonstrate that d=1 is the optimal recursion depth in USCarrier networks, with the highest speedup reaching 26.94x, while AS1755 has the best performance at d=2. We speculate that the performance will not keep increasing as d increases, since increasing the depth also increases the overhead for computing the best paths.

5 Related Work

Minesweeper [4] is an SMT-based network verifier, which reduces the verification problem to an SMT problem. Trailblazer significantly improves the performance of Minesweeper without sacrificing its generality. Batfish [9] is a control plane simulator, which simulates control plane execution, generates a data plane and analyses the data plane. In respect to fault-tolerance policies, Batfish enumerates every failed topology, an approach that doesn't scale in practice. Plankton [16] models different routing protocols by simple path vector protocol [12]. It uses an explicit model checker, Spin [13], to thoroughly explore the possible network states to check whether there exists a state that violates the queried policy. For fault-tolerance, it checks the policies by naive enumeration, which negatively affects its performance. Plankton implementation is not publicly available. ARC [10] models the network as a graph. It performs verification by standard graph algorithms which are ensured to be polynomial. As a result scales for fault-tolerance policies. However, it is incapable of representing some common network features, including BGP local preferences and communities. Tiramisu [1] is an improved version of ARC. It can support richer network features. However, it trades the scope of its application for verification efficiency e.g. it does not verify properties such as the loop freedom and cannot compute counter-examples. Netdice [18] is a probabilistic verifier which can prune the failed topologies when verifying iBGP networks. Our technique performs similar pruning decisions via an SMT encoding, and it is more general e.g., supports eBGP networks. Surgeries [15] and Bonsai [5] both exploits structural symmetry of network to accelerate verification for network. They achieve good performance when networks are highly symmetrical, but none of them can verify properties pertaining to when network failures may occur. Origami [11] targets on fault-tolerance verification, but it focuses on symmetrical networks and only reasons about reachability policies. MonoSAT [3] is an SMT solver that is efficient at solving problems involving monotonic theories. The reachability policy is monotonic with respect to links since removing a link can decrease the network's reachability but cannot increase it. The same applies to the isolation policy. However, other policies like waypoint and balance are not monotonic, so MonoSAT is unable to verify them efficiently.

6 Conclusion

In this paper, we proposed an SMT encoding that significantly accelerates the verification of network fault tolerance properties by pruning the space of topologies that are guaranteed not to change the verification result. We implemented this new encoding in the open source tool Trailblazer, a variant of the state-of-the-art verifier Minesweeper with the optimised encoding. Our evaluation shows that Trailblazer significantly improves verification performance compared to Minesweeper, with the highest speedups reaching 26.9X.

References

1. Abhashkumar, A., Gember-Jacobson, A., Akella, A.: Tiramisu: Fast multilayer network verification. In: 17th {USENIX} Symposium on Networked Systems Design and Implementation ({NSDI} 20). pp. 201–219 (2020)
2. Abio, I., Nieuwenhuis, R., Oliveras, A., Rodriguez-Carbonell, E.: A parametric approach for smaller and better encodings of cardinality constraints. In: CP. pp. 80–96. Springer (2013)
3. Bayless, S., Bayless, N., Hoos, H., Hu, A.: Sat modulo monotonic theories. In: Proceedings of the AAAI Conference on Artificial Intelligence. vol. 29 (2015)
4. Beckett, R., Gupta, A., Mahajan, R., Walker, D.: A general approach to network configuration verification. In: Proceedings of the Conference of the ACM Special Interest Group on Data Communication. pp. 155–168 (2017)
5. Beckett, R., Gupta, A., Mahajan, R., Walker, D.: Control plane compression. In: Proceedings of the 2018 Conference of the ACM Special Interest Group on Data Communication. pp. 476–489 (2018)
6. Beckett, R., Gupta, A., Mahajan, R., Walker, D.: Abstract interpretation of distributed network control planes. Proceedings of the ACM on Programming Languages **4**(POPL), 1–27 (2019)
7. Birkner, R., Drachsler-Cohen, D., Vanbever, L., Vechev, M.: Config2spec: Mining network specifications from network configurations. In: 17th {USENIX} Symposium on Networked Systems Design and Implementation ({NSDI} 20). pp. 969–984 (2020)
8. Cook, J.: Reachability analysis for aws-based networks. In: 31ST INTERNATIONAL CONFERENCE ON COMPUTER AIDED VERIFICATION. Computer Aided Verification (2019)
9. Fogel, A., Fung, S., Pedrosa, L., Walraed-Sullivan, M., Govindan, R., Mahajan, R., Millstein, T.: A general approach to network configuration analysis. In: 12th {USENIX} Symposium on Networked Systems Design and Implementation ({NSDI} 15). pp. 469–483 (2015)
10. Gember-Jacobson, A., Viswanathan, R., Akella, A., Mahajan, R.: Fast control plane analysis using an abstract representation. In: Proceedings of the 2016 ACM SIGCOMM Conference. pp. 300–313 (2016)
11. Giannarakis, N., Beckett, R., Mahajan, R., Walker, D.: Efficient verification of network fault tolerance via counterexample-guided refinement. In: International Conference on Computer Aided Verification. pp. 305–323. Springer (2019)
12. Griffin, T.G., Shepherd, F.B., Wilfong, G.: The stable paths problem and interdomain routing. IEEE/ACM Transactions On Networking **10**(2), 232–243 (2002)
13. Holzmann, G.J.: The model checker spin. IEEE Transactions on software engineering **23**(5), 279–295 (1997)
14. Knight, S., Nguyen, H.X., Falkner, N., Bowden, R., Roughan, M.: The internet topology zoo. IEEE Journal on Selected Areas in Communications **29**(9), 1765–1775 (2011)
15. Plotkin, G.D., Bjørner, N., Lopes, N.P., Rybalchenko, A., Varghese, G.: Scaling network verification using symmetry and surgery. ACM SIGPLAN Notices **51**(1), 69–83 (2016)
16. Prabhu, S., Chou, K.Y., Kheradmand, A., Godfrey, B., Caesar, M.: Plankton: Scalable network configuration verification through model checking. In: 17th {USENIX} Symposium on Networked Systems Design and Implementation ({NSDI} 20). pp. 953–967 (2020)

17. Spring, N., Mahajan, R., Wetherall, D.: Measuring isp topologies with rocketfuel. ACM SIGCOMM Computer Communication Review **32**(4), 133–145 (2002)
18. Steffen, S., Gehr, T., Tsankov, P., Vanbever, L., Vechev, M.: Probabilistic verification of network configurations. In: Proceedings of the Annual conference of the ACM Special Interest Group on Data Communication on the applications, technologies, architectures, and protocols for computer communication. pp. 750–764 (2020)