Explaining Network Configurations Under Failures via Localized Subspecifications

Yaxuan Lin
ShanghaiTech University
Shanghai, China
linyx2025@shanghaitech.edu.cn

Yongzheng Zhang
ShanghaiTech University
Shanghai, China
yongzheng@shanghaitech.edu.cn

Haoxian Chen*
ShanghaiTech University
Shanghai, China
hxchen@shanghaitech.edu.cn

Abstract

Network fault-tolerance verification checks whether configurations satisfy properties such as reachability and isolation under a bounded set of failures, but produces only a global pass/fail result. Operators are left without guidance on which configuration fields are responsible for correctness or which local edits are safe under the full failure model. We explore *fault-tolerant subspecifications* as a way to bridge this gap: for each configuration location h, a constraint $\Psi(h)$ that characterizes the replacement values guaranteed to preserve the global property across all failure scenarios within the budget. Our approach combines per-device SMT encodings with a failure-aware symbolic route; guarded routing branches induce branch-local constraints on h, whose intersection yields the fault-tolerant subspecification. We present the approach, illustrate it on a small BGP example, and discuss preliminary results that demonstrate feasibility and interpretability. We conclude with open problems and directions for future work.

CCS Concepts

• **Networks** → **Network manageability**; **Formal specifications**.

Keywords

network verification, fault-tolerance, localized subspecifiation

ACM Reference Format:
Yaxuan Lin, Yongzheng Zhang, and Haoxian Chen. 2026. Explaining Network Configurations Under Failures via Localized Subspecifications. In *3rd Workshop on Formal Methods Aided Network Operation (FMANO '26), August 17–21, 2026, Denver, CO, USA.* ACM, New York, NY, USA, 6 pages. https://doi.org/10.1145/3789240.3822361

1 Introduction

Network configurations control distributed routing protocols that determine route propagation and forwarding behavior. Since routing decisions emerge from interactions among many devices, local configuration changes can have non-local effects on reachability, isolation, and traffic-engineering objectives. This makes it difficult for operators to ensure that configurations correctly implement high-level design intent. Even when automated tools confirm that a configuration is correct, operators often cannot tell why it is correct or what constraints each configuration element must satisfy.

Recent work has proposed *subspecifications* to address this gap [16, 17]. For a configuration location, a subspecification derives the minimal local constraint that suffices to preserve the global property, given the rest of the configuration. This makes explicit how each configuration element contributes to the overall behavior. However, existing subspecifications are defined with respect to a single stable routing state. This limits the approach to the failure-free setting, where the network converges to a unique behavior.

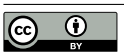
In practice, network correctness is often expected to be robust to failures. Failure-aware verification tools check whether properties such as reachability and isolation continue to hold under bounded link and device failures [13, 14, 19]. Under failures, the network may converge to different routing states. This makes the explainability problem significantly harder. Different failures can activate different backup paths and impose different constraints on the same configuration location. No existing explainability framework addresses this setting.

Explainability for fault-tolerance verification. This paper introduces *fault-tolerant subspecifications*, which explain why a configuration remains correct under a k-failure model. For a configuration location, different failure scenarios may activate different backup paths and impose different local constraints. A fault-tolerant subspecification captures the constraints that must hold across all failure scenarios, so that any configuration satisfying it preserves the property under the entire failure model. This gives operators visibility into which locations are failure-critical, what each location must satisfy to remain correct under all bounded failures, and which failure scenarios make each constraint necessary.

Localized subspecifications via symbolic routes. We extend subspecifications to the k-failure setting using the symbolic route approach [14, 22]. Rather than fixing a single routing state, we propagate routes symbolically, encoding link availability as Boolean variables. A single execution then captures routing behavior across all failure combinations, yielding a *symbolic route* whose entries are conditional on which links are available. From the symbolic route, we derive per-location constraints as the intersection of requirements across failure scenarios, producing fault-tolerant subspecifications without enumerating failures explicitly.

Challenges. Extending subspecifications to the fault-tolerant setting raises three challenges. (1) *Formulation.* Existing subspecifications decompose analysis using a fixed stable state; under failures the stable state varies across scenarios, so the decomposition must be redesigned. (2) *Constraint composition.* Different failure scenarios induce different local constraints at the same location; combining

*Corresponding author



FMANO '26, Denver, CO, USA

ACM ISBN 979-8-4007-2467-1/26/08
https://doi.org/10.1145/3789240.3822361

Symbol	Meaning
C	Network configuration.
φ	Correctness property.
h	Configuration location (*e.g.,* a field or policy entry).
$C[h \leftarrow g]$	Configuration C with location h replaced by g.
$\psi(h)$	Subspecification at h (failure-free).
k	Failure budget: max simultaneous failures.
F	A failure scenario (set of failed components).
$\mathcal{K}$	All failure scenarios within budget k.
$\psi_F(h)$	Per-scenario subspecification under F.
$\Psi(h)$	Fault-tolerant subspec: $\bigwedge_{F\in\mathcal{K}} \psi_F(h)$.
RIB_F	Concrete RIB under failure scenario F.
$SRIB_F$	Symbolic RIB covering failure scenario F.

Table 1: Summary of notation.

Approach	Specification	Failure-aware?
Verification [3, 10]	global	×
Failure verification [14, 22]	global	✓
Subspecification [17]	localized	×
This work	localized	✓

Table 2: Positioning against related approaches.

them requires a principled intersection procedure that remains sound and interpretable. (3) *Scalability.* The number of failure combinations grows combinatorially in k; the approach must avoid explicit enumeration to remain tractable.

Contributions and organization. This paper makes the following contributions.

(1) *Fault-tolerant subspecifications (Section 3).* We introduce fault-tolerant subspecifications, which explain why a configuration remains correct under all failures up to k. They associate each required constraint with its inducing failure scenarios, revealing failure-specific constraints invisible in failure-free explanations.

(2) *Derivation framework (Section 4).* We present an algorithm that derives fault-tolerant subspecifications from symbolic routes, reasoning over all failure scenarios without explicit enumeration.

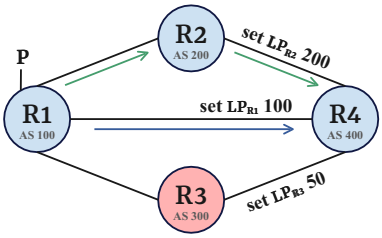
(3) *Preliminary results* (Section 5). We apply a prototype to a small BGP network to assess feasibility and illustrate how fault-tolerant subspecifications expose failure-specific constraints.

We conclude with open problems and future directions.

2 Background

Network verification. Network verifiers check if a router configuration C satisfies a correctness property φ [2, 3, 10]. Properties are typically stated over forwarding paths: reachability requires that traffic between two endpoints always reaches its destination, and isolation requires that traffic from one group never reaches another. Verifiers encode the control plane—BGP route selection, import/export policies, and prefix filters—as logical formulas, and check $C \models \varphi$ via SMT solving or control-plane simulation.

Fault-tolerance verification. Fault-tolerance verification requires φ to hold under all combinations of up to k simultaneous link failures. Enumerating failure scenarios is intractable for large k, so efficient approaches propagate symbolic routes that compactly represent routing behavior across failure scenarios without explicit



Specification: R4 must reach prefix P without traversing R3

No failure subspecs	1-failure subspecs
$LP_{R2} > 100$	$LP_{R2} > 100$
$LP_{R1} < 200$	$LP_{R1} > 50 \wedge LP_{R1} < 200$
$LP_{R3} < 200$	$LP_{R3} < 100$

Figure 1: Topology, spec and subspecs (no failure / 1-failure).

enumeration. Hoyan [22] introduces this symbolic route propagation for reachability on a global WAN, and YU[14] generalizes the approach to traffic load properties under arbitrary k failures.

Network subspecification. Recent work [16] addresses configuration explainability by introducing *network subspecifications*. Given $C \models \varphi$ and a configuration location h (*e.g.,* a local preference value or a route-map entry), the subspecification $\psi(h)$ is a constraint on h such that any replacement $g \models \psi(h)$ preserves φ with the rest of C fixed: $\forall g.\ g \models \psi(h) \Rightarrow C[h \leftarrow g] \models \varphi$. Intuitively, $\psi(h)$ turns a global verification result into a local contract: it tells an operator exactly which values at h are safe, without re-running the verifier.

This prior work derives $\psi(h)$ under a fixed, failure-free topology. This paper extends subspecification to the fault-tolerant setting, where different failure scenarios induce different routing states and may impose different requirements on the same location.

3 Motivating Example

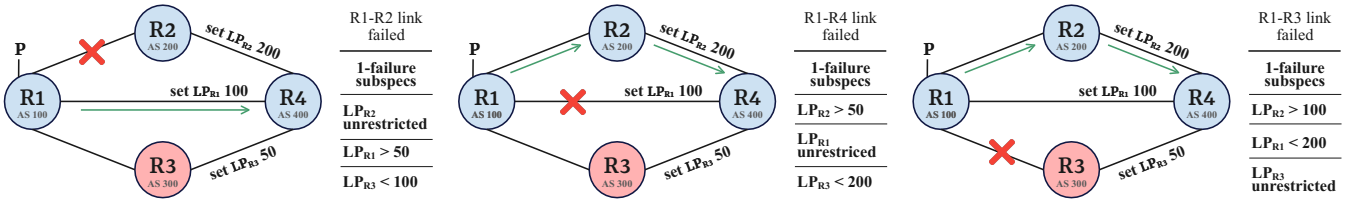
Network setup. Figure 1 depicts a network of four routers, R1–R4, where R1 advertises prefix P. The specification requires that R4 must reach prefix P without traversing R3. At R4, routes to P are ranked by their incoming neighbor: routes via R2 have local preference 200, via R1 have 100, and via R3 have 50. This design makes R1–R2–R4 the primary path and R1–R4 the backup path.

Maintenance challenges. Suppose the operator wants to modify LP_{R3} during routine maintenance, increasing it from 50 to 150. To verify safety under a single-link failure ($k = 1$), they re-run fault-tolerance verification. The check fails, producing a counterexample where the R1–R2 link fails and reachability from R4 is broken. However, the counterexample does not explain what LP_{R3} must satisfy to restore correctness. The operator faces several challenges: (1) they must mentally simulate route convergence across failure scenarios to understand why a particular route is selected over alternatives; (2) each proposed fix requires re-running verification, which is time-consuming; (3) without knowing the root constraint, the operator iterates blindly.

The difficulty stems from how failures interact with routing. The five links in the network yield three classes of relevant single-link failure scenarios (Figure 2), each activating a different path from R4 to R1 and imposing a distinct constraint on R4's configuration.

Fault-tolerant subspecifications. A fault-tolerant subspecification makes these per-scenario constraints explicit. For the field LP_{R3} at R4, each failure scenario induces a different requirement:

Failure of (R1–R2) (Figure 2a). The primary path R1–R2–R4 is unavailable, so R4 must select the backup path R1–R4. This requires



(a) R1-R2 link failed. **(b) R1-R4 link failed.** **(c) R1-R3 link failed.**

Figure 2: Single-link failure scenarios and their corresponding subspecs.

the route from R1 to be preferred over R3: $LP_{R1} > LP_{R3}$, constraining $LP_{R3} < LP_{R1}$.

Failure of (R1–R4) (Figure 2b). The direct backup path is unavailable, so R4 must use the primary path R1–R2–R4. The route learned from R2 must be preferred over R3: $LP_{R2} > LP_{R3}$, constraining $LP_{R3} < LP_{R2}$. The local preference LP_{R1} is unconstrained, as the direct path is down.

Failure of (R1–R3) (Figure 2c). The path via R3 is unavailable, so the no-R3 requirement is automatically satisfied. R4 must choose between the primary and backup paths, preferring R2 over R1: $LP_{R2} > LP_{R1}$. The local preference LP_{R3} is unconstrained, since the path through R3 is unavailable.

The fault-tolerant subspecification for LP_{R3} is the intersection of these per-scenario constraints. It reveals that any assignment above 100 would override the intended primary or secondary path under some failure, violating fault-tolerant correctness. The operator can immediately identify safe values without re-running verification.

The example illustrates three benefits of localized subspecifications. First, many configuration fields are unconstrained under a given failure; the subspecification quickly directs operator attention to the fields that actually matter. Second, the safe edit range for each field is stated directly, making it immediately clear how to maintain correctness during updates. Third, the subspecification compresses the failure scenarios into a single constraint per field, explaining why each configuration line matters and under which failures it becomes binding.

4 Overview

Problem formulation. Building on the insight from the motivating example, we define a fault-tolerant subspecification as follows. Given a network configuration C, a correctness property φ, a failure budget k, and a target configuration field h, the fault-tolerant subspecification $\Psi(h)$ is a constraint on h such that any value satisfying it preserves φ under all failure scenarios within the budget. Formally, $\Psi(h) = \bigwedge_F \psi_F(h)$, where $\psi_F(h)$ is the minimal requirement on h that preserves φ under failure scenario F. Each $\psi_F(h)$ is annotated with its inducing scenarios, giving operators direct traceability into why each constraint is required.

Insights. Our approach rests on two insights.

- *Efficient failure analysis.* Symbolic routes compress failure case analysis: a single symbolic route propagation captures routing behavior across all failure scenarios without enumerating them.
- *Scalability.* Per-device SMT encodings allow the analysis to scale to large networks by reasoning about one device at a time.

Workflow. Figure 3 shows the workflow in four steps. We first generate a symbolic SMT encoding for each device (Step 1). We then compute symbolic routes for the network (Step 2). We combine the two to construct a seed subspecification per target field (Step 3). Finally, we simplify and aggregate to produce the fault-tolerant subspecification (Step 4).

Step 1: Symbolic encoding. We encode each device's configuration as an SMT formula in which each field is a symbolic variable pinned to its concrete value by an equality constraint. This equality is later released for the target field h, allowing it to range freely while all other fields remain fixed. We then slice the global SMT encoding into per-device encodings, producing formulas specific to each device's configuration.

Step 2: Symbolic route computation. Symbolic routes encode routing behavior across failure scenarios, as developed in [14]. Given such symbolic routes, we apply k-failure pruning: branches whose guard requires more than k simultaneous failures are discarded. The result is a compact, multi-branch symbolic route where each branch represents a feasible routing outcome under the k-failure model.

Specifically, for each prefix, the source router generates initial *messages* using a tuple $(prefix, path, cond)$, where *path* tracks the routers traversed and *cond* is a logical formula composed of binary link aliveness variables (a_n). Upon receiving a message, a router creates a *symbolic route entry* $(device, prefix, path, attributes, guard)$ where the *guard* combines message conditions with policy and attribute constraints to indicate when the route is selected.

Figure 4 illustrates R4's symbolic route construction showing only forward message propagation. Messages m_1, m_3, and m_5 carry candidate routes propagated to R4. Specifically, upon receiving m_1, R4 installs the first symbolic route route_1; When the higher-priority route carried by m_3 is considered, R4 enforces guard mutual exclusion: route_2's guard becomes g_2, and route_1's guard updates to $\neg g_2 \wedge a_2$; After receiving m_5, R4 reevaluates the guards of all symbolic route entries to account for route priorities and path feasibility. This process produces R4's final symbolic routes, where each entry captures path feasibility and the correct selection order under BGP best-path semantics.

The symbolic route also serves as the decomposition reference for Step 3 and Step 4. Each branch defines a failure condition under which the seed subspecification is evaluated.

Step 3: Seed subspecification. For each symbolic route entry whose guard is satisfiable under the k-failure budget, we construct a *seed encoding* by combining the per-device SMT encoding from Step 1 with symbolic-route information from Step 2. As shown in

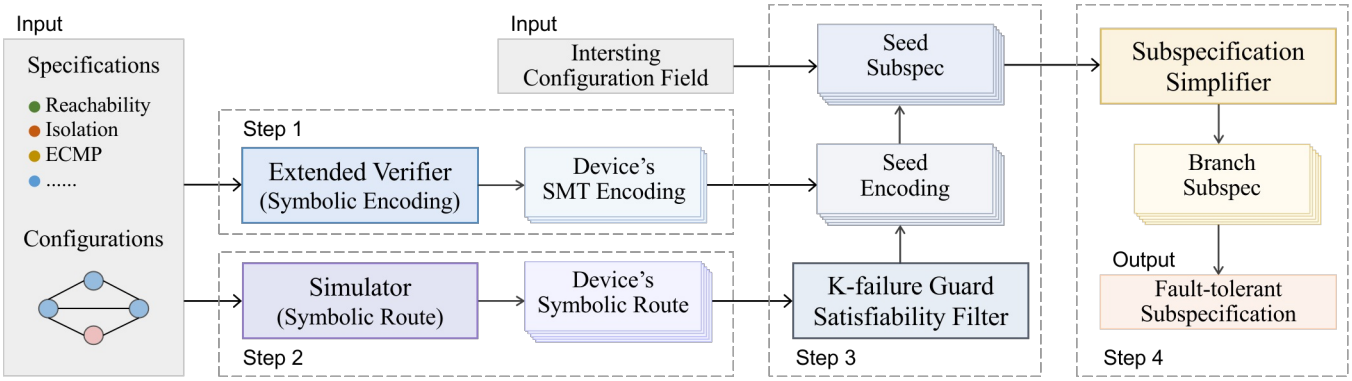


Figure 3: A framework for fault-tolerant subspecifications.

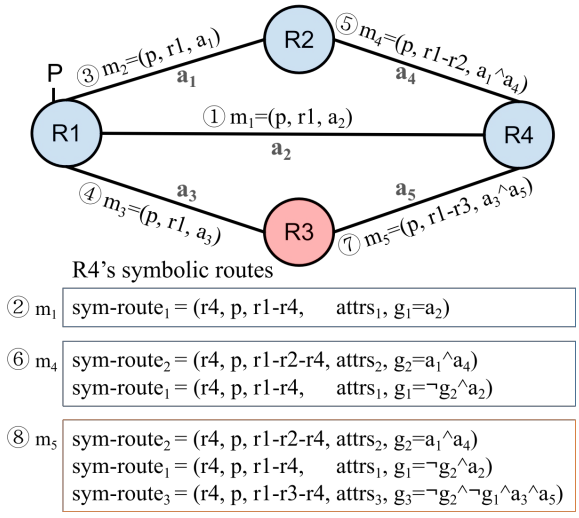


Figure 4: Symbolic route generation.

Device's SMT Encoding	best route, export policies (all connected peers) import policy, route selection, best route (device)
Device's State	control forwarding state
Device's Current Symbolic Route	one current symbolic route' attributes *(local-preference, AS-path, MED, communities, ...)*
Selected peer's Symbolic Route	one corresponding predecessor symbolic route' attributes *(local-preference, AS-path, MED, communities, ...)*
Other Peers' Symbolic Routes	all symbolic routes' attributes, each with its corresponding connected link state *(local-preference, AS-path, MED, communities, ...)* *∧ its corresponding connected link state* . . . . .
Device's Link State	link state to the select peer (via symbolic route condition)

Figure 5: Components of the seed encoding.

Figure 5, the seed encoding comprises: (1) the device-level SMT encoding; (2) the control-forwarding state; (3) the device's selected symbolic route; (4) the predecessor symbolic route at the chosen peer; (5) all candidate symbolic routes at the remaining peers with their connected-link states; and (6) the link state to the selected peer. The seed encoding retains only directly connected link states; effects of remote links are already captured in the predecessor and candidate peer symbolic routes.

We illustrate with sym-route$_1$ at R4 (R1 is the selected peer). Component (2) sets the control-forwarding variable for R1 to true and all others to false. Component (3) carries attrs$_1$; component (4) carries the predecessor attributes from R1. For component (5), R2 has an empty route guarded by $\neg a_1$ and a non-empty route guarded by a_1; under the guard $\neg(a_1 \wedge a_4) \wedge a_2$ of sym-route$_1$, the empty route leaves a_4 unconstrained while the non-empty route requires a_4 = false; both routes at R3 similarly leave a_5 unconstrained. Component (6) requires the R4–R1 link to be active; candidate routes whose next hop is R4 are discarded to prevent routing loops.

Because the device-level SMT encoding and the symbolic-route analysis are produced by separate tools and may use different models of device behavior, we further add a consistency condition to the seed encoding. This condition requires the symbolic route selected at the device to be realizable by the SMT encoding, ensuring that the two representations agree on the routing outcome.

Finally, for each target configuration field h, we construct a seed subspecification by replacing the constraint that fixes h to its original concrete value with a fresh unconstrained symbolic variable, while keeping all other configuration fields fixed. This abstraction is performed separately for each target field, producing one seed subspecification per field.

Step 4: Subspec derivation and aggregation. For each branch of the symbolic route, we conjoin its guard condition with the seed subspecification and simplify by constant propagation. This yields a branch-local subspecification: the constraint on h sufficient to realize the expected routing behavior under that failure scenario.

These branch-local subspecifications are then aggregated across all branches. The result is the fault-tolerant subspecification for h, capturing the constraints the device must satisfy under the k-failure model and guaranteeing that the global specification holds under all considered failure scenarios.

Soundness analysis. Table 3 summarizes the soundness relations under the three settings. In the failure-free setting, subspecification derivation is based on a concrete routing state RIB. Since this routing state satisfies the given specifications, the subspecification derived from it is sound with respect to that state.

A fault-tolerant approach based on failure enumeration extends this reasoning to every admissible failure scenario. For each failure F, the network produces a concrete routing state RIB$_F$. Because

Approach	Condition	Routing State	Soundness Relation
Failure-free	None	Concrete RIB	RIB $\Rightarrow$ Specifications
Fault-tolerance (enumeration)	Failure F	Concrete RIB_F (under F)	$\text{RIB}_F \Rightarrow$ Specifications (under F)
Fault-tolerance (symbolic routes)	Failure F	Symbolic routes SRIB_F (cover F)	$(\text{SRIB}_F \wedge F) \Rightarrow \text{RIB}_F \Rightarrow$ Specifications (under F)

Table 3: Soundness relations for subspecification derivation.

RIB_F satisfies the specifications under F, the corresponding subspecification is sound for that failure scenario. However, explicitly enumerating all failures does not scale.

Our symbolic-route approach avoids this enumeration by representing multiple concrete routing states with guarded symbolic routes. For a particular failure F, we collect all seed encodings whose branch guards are satisfied by F. Together, they form the guarded symbolic routing state SRIB_F corresponding to F. Under F, this symbolic state represents the corresponding concrete state: $(\text{SRIB}_F \wedge F) \Rightarrow \text{RIB}_F$. Since RIB_F satisfies the specifications under F, the subspecification derived from SRIB_F is also sound for that failure. Therefore, provided that the symbolic routes soundly represent the concrete routing states covered by their guards, deriving subspecifications from symbolic routes preserves the soundness argument of failure enumeration without explicitly constructing every concrete RIB.

Consistency and scalability. Our framework decouples symbolic route generation from device-level SMT encoding. This separation improves scalability but introduces a consistency requirement: the symbolic route used in a seed encoding must be realizable under the device-level SMT semantics. Importantly, the seed encoding is constructed first and the target field h is then released as a symbolic variable. This allows the SMT encoding to recompute route selection under any value of h. As a result, changes to h, including those affecting routing preferences, are handled within SMT and do not invalidate the symbolic-route decomposition.

This guarantees correctness under the full k-failure model, but may exclude values that are behaviorally feasible under specific routing interpretations, yielding a conservative characterization of safe replacements. The consistency check introduces SMT constraints and solver overhead. Its cost depends on the number of symbolic branches and SMT encoding complexity. Evaluating this overhead at scale remains future work.

5 Preliminary Results

To validate the feasibility of our approach, we implemented a small prototype of the derivation pipeline and applied it to the motivating example from Section 3: a four-router network with three candidate paths from R4 to prefix P, checking that P remains reachable from R4 without traversing R3 under all single-link failures ($k = 1$).

For the configuration location LP_{R1}, R4's import preference for routes from R1, the prototype generates a fault-tolerant subspecification matching the manually derived result in the motivating example and provides an initial sanity check on correctness.

Each constraint in the subspecification is annotated with the guard of the symbolic route branch that induced it, identifying the specific failure condition under which the bound becomes necessary. The symbolic route of R4 contained 3 branches and the final subspecification consisted of 2 inequality constraints. Symbolic route generation took 0.1 s and subspecification derivation took 6 s.

This is a preliminary case study; scaling to larger networks is not yet validated. The per-device decomposition of our approach, grounded in the given data plane, provides a natural path toward scalability. Future work includes evaluating subspecification quality across larger networks and a broader range of properties and failure budgets, and conducting user studies to assess whether the output helps operators reason about safe configuration changes.

6 Related Work

Network verification. Minesweeper [3], Tiramisu [2], and Batfish [10] check properties such as reachability and isolation by encoding the control plane symbolically or via simulation. These tools answer whether a configuration is correct; our work explains which local constraints keep a verified configuration correct under failures.

Fault-tolerance verification. Prior work checks network properties under failures via counterexample-guided refinement and SMT encodings [11, 15], while Hoyan [22] and Li et al. [14] handle reachability and traffic load properties at scale using symbolic routes. None of these approaches explain how individual failure scenarios constrain configuration locations. Fault-tolerant subspecifications instead turn failure-aware correctness into per-location contracts.

Network subspecification. The most relevant line of work introduced localized subspecifications as an explanation mechanism for network configurations [8, 17]. A subspecification constrains the admissible values at a configuration location that preserve a global property, assuming the rest of the configuration is fixed. Our work extends this idea from a single, failure-free stable state to a failure-budgeted family of stable states, deriving branch-local requirements from failure-aware routing branches and intersecting them into fault-tolerant subspecifications.

Network synthesis and repair. Synthesis and repair tools generate configurations from high-level intents [1, 4, 5, 9, 18]. These systems return one or a few candidate configurations. Fault-tolerant subspecifications instead characterize the full space of safe local replacements under a failure budget for maintenance and audit.

Differential analysis and intent inference. Differential analysis compares behavior across network snapshots [20, 21, 23], and intent inference recovers high-level specifications from configurations [6, 12]. Both explain or infer global behavior. Our work assumes a given property and derives local, failure-aware constraints that candidate edits must satisfy before an update.

Debugging and provenance. Network provenance systems explain realized behaviors by tracing routing and forwarding derivations [7, 24, 25]. Our explanations are configuration-local and counterfactual: they state what each location must satisfy to preserve correctness across all feasible failures, retaining branch guards as provenance.

7 Future Directions

This work opens up several interesting research directions.

Rigorous guarantees and evaluation. The current approach is preliminary. Future work includes formalizing the soundness guarantee so that the subspecification precisely characterizes safe replacements, developing a scalable implementation, and conducting user studies to assess whether fault-tolerant subspecifications help operators understand and safely modify configurations in practice.

Broader property support. The current work focuses on reachability and isolation. Fault-tolerant subspecifications can in principle be extended to a wider class of correctness properties, including waypointing, loop-freedom, and traffic-engineering objectives such as load balancing and bandwidth guarantees. Each extension requires identifying the right per-scenario constraint form and an appropriate intersection semantics for that property class.

Diagnosing verification failures. Fault-tolerant subspecifications currently explain why a correct configuration is correct. A natural extension is to apply the same analysis to configurations that fail verification, using per-scenario constraints to pinpoint which configuration locations violate which failure conditions. This would turn subspecifications into a diagnostic tool for debugging fault-tolerance violations.

8 Conclusion

We present fault-tolerant subspecifications as a practical way to explain network configurations under failures, by exposing the precise conditions each configuration element must satisfy to preserve global properties across a failure budget. Enumeration provides a simple reference formulation: derive a scenario-specific subspecification for each considered failure scenario and intersect the results, so any replacement satisfying the intersection satisfies the local requirements of every scenario. To avoid explicit enumeration, we use a failure-aware symbolic route: guarded selected-route branches induce branch-local subspecifications, whose intersection yields the fault-tolerant subspecification for each configuration location. This focus preserves the soundness intuition of enumeration while enabling compact failure-aware explanations that are difficult to obtain from verification results or counterexamples alone.

Acknowledgments

We sincerely thank the anonymous reviewers of FMANO '26 for their valuable feedback on this paper. This work is supported by the ShanghaiTech Startup Fund.

References

[1] Anubhavnidhi Abhashkumar, Aaron Gember-Jacobson, and Aditya Akella. 2020. Aed: Incrementally synthesizing policy-compliant and manageable configurations. In *Proceedings of the 16th International Conference on emerging Networking EXperiments and Technologies*. 482–495.

[2] Anubhavnidhi Abhashkumar, Aaron Gember-Jacobson, and Aditya Akella. 2020. Tiramisu: Fast multilayer network verification. In *17th USENIX Symposium on Networked Systems Design and Implementation (NSDI 20)*. 201–219.

[3] Ryan Beckett, Aarti Gupta, Ratul Mahajan, and David Walker. 2017. A general approach to network configuration verification. In *Proceedings of the Conference of the ACM Special Interest Group on Data Communication*. 155–168.

[4] Ryan Beckett, Ratul Mahajan, Todd Millstein, Jitendra Padhye, and David Walker. 2016. Don't mind the gap: Bridging network-wide objectives and device-level configurations. In *Proceedings of the 2016 ACM SIGCOMM Conference*. 328–341.

[5] Ryan Beckett, Ratul Mahajan, Todd Millstein, Jitendra Padhye, and David Walker. 2017. Network configuration synthesis with abstract topologies. In *Proceedings of the 38th ACM SIGPLAN conference on programming language design and implementation*. 437–451.

[6] Rüdiger Birkner, Dana Drachsler-Cohen, Laurent Vanbever, and Martin Vechev. 2020. {Config2Spec}: Mining network specifications from network configurations. In *17th USENIX Symposium on Networked Systems Design and Implementation (NSDI 20)*. 969–984.

[7] Ang Chen, Yang Wu, Andreas Haeberlen, Wenchao Zhou, and Boon Thau Loo. 2016. The good, the bad, and the differences: Better network diagnostics with differential provenance. In *Proceedings of the 2016 ACM SIGCOMM Conference*. 115–128.

[8] Haoxian Chen. 2024. Interpretable Network Synthesis via Localized Specifications. In *Proc. SIGCOMM Workshop on FMANO*. 51–53.

[9] Ahmed El-Hassany, Petar Tsankov, Laurent Vanbever, and Martin Vechev. 2018. NetComplete: Practical Network-Wide configuration synthesis with autocompletion. In *15th USENIX Symposium on Networked Systems Design and Implementation (NSDI 18)*. 579–594.

[10] Ari Fogel, Stanley Fung, Luis Pedrosa, Meg Walraed-Sullivan, Ramesh Govindan, Ratul Mahajan, and Todd Millstein. 2015. A general approach to network configuration analysis. In *12th USENIX Symposium on Networked Systems Design and Implementation (NSDI 15)*. 469–483.

[11] Nick Giannarakis, Ryan Beckett, Ratul Mahajan, and David Walker. 2019. Efficient verification of network fault tolerance via counterexample-guided refinement. In *International Conference on Computer Aided Verification*. Springer, 305–323.

[12] Ali Kheradmand. 2020. Automatic inference of high-level network intents by mining forwarding patterns. In *Proceedings of the Symposium on SDN Research*. 27–33.

[13] Ruihan Li, Fangdan Ye, Yifei Yuan, Ruizhen Yang, Bingchuan Tian, Tianchen Guo, Hao Wu, Xiaobo Zhu, Zhongyu Guan, Qing Ma, et al. 2024. Reasoning about network traffic load property at production scale. In *21st USENIX Symposium on Networked Systems Design and Implementation (NSDI 24)*. 1063–1082.

[14] Ruihan Li, Yifei Yuan, Fangdan Ye, Mengqi Liu, Ruizhen Yang, Yang Yu, Tianchen Guo, Qing Ma, Xianlong Zeng, Chenren Xu, et al. 2024. A general and efficient approach to verifying traffic load properties under arbitrary k failures. In *Proceedings of the ACM SIGCOMM 2024 Conference*. 228–243.

[15] Yu Liu, Pavle Subotic, Emmanuel Letier, Sergey Mechtaev, and Abhik Roychoudhury. 2023. Efficient SMT-Based Network Fault Tolerance Verification. In *International Symposium on Formal Methods*. Springer, 92–100.

[16] Amirmohammad Nazari, Yifei Huang, Roopsha Samanta, Arjun Radhakrishna, and Mukund Raghothaman. 2023. Explainable Program Synthesis by Localizing Specifications. *Proceedings of the ACM on Programming Languages* 7, OOPSLA2 (2023), 2171–2195.

[17] Amirmohammad Nazari, Yongzheng Zhang, Mukund Raghothaman, and Haoxian Chen. 2024. Localized Explanations for Automatically Synthesized Network Configurations. In *Proceedings of the 23rd ACM Workshop on Hot Topics in Networks*. 52–59.

[18] Sivaramakrishnan Ramanathan, Ying Zhang, Mohab Gawish, Yogesh Mundada, Zhaodong Wang, Sangki Yun, Eric Lippert, Walid Taha, Minlan Yu, and Jelena Mirkovic. 2023. Practical intent-driven routing configuration synthesis. In *20th USENIX Symposium on Networked Systems Design and Implementation (NSDI 23)*. 629–644.

[19] Tibor Schneider, Stefano Vissicchio, and Laurent Vanbever. 2025. Verifying maximum link loads in a changing world. In *22nd USENIX Symposium on Networked Systems Design and Implementation (NSDI 25)*. 1269–1287.

[20] Alan Tang, Siva Kesava Reddy Kakarla, Ryan Beckett, Ennan Zhai, Matt Brown, Todd Millstein, Yuval Tamir, and George Varghese. 2021. Campion: debugging router configuration differences. In *Proceedings of the 2021 ACM SIGCOMM 2021 Conference*. 748–761.

[21] Xieyang Xu, Yifei Yuan, Zachary Kincaid, Arvind Krishnamurthy, Ratul Mahajan, David Walker, and Ennan Zhai. 2024. Relational Network Verification. In *Proceedings of the ACM SIGCOMM 2024 Conference*. 213–227.

[22] Fangdan Ye, Da Yu, Ennan Zhai, Hongqiang Harry Liu, Bingchuan Tian, Qiaobo Ye, Chunsheng Wang, Xin Wu, Tianchen Guo, Cheng Jin, et al. 2020. Accuracy, scalability, coverage: A practical configuration verifier on a global wan. 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*. 599–614.

[23] Peng Zhang, Aaron Gember-Jacobson, Yueshang Zuo, Yuhao Huang, Xu Liu, and Hao Li. 2022. Differential network analysis. In *19th USENIX Symposium on Networked Systems Design and Implementation (NSDI 22)*. 601–615.

[24] Wenchao Zhou, Qiong Fei, Arjun Narayan, Andreas Haeberlen, Boon Thau Loo, and Micah Sherr. 2011. Secure network provenance. In *Proceedings of the twenty-third ACM symposium on operating systems principles*. 295–310.

[25] Wenchao Zhou, Micah Sherr, Tao Tao, Xiaozhou Li, Boon Thau Loo, and Yun Mao. 2010. Efficient Querying and Maintenance of Network Provenance at Internet-Scale. In *Proceedings of the 2010 ACM SIGMOD International Conference on Management of Data*. 615–626.