Philip Jonas Franz, Konrad Altenhofen, Jonas Schoenen, Osama Abboud, Tobias Meuser, Björn Scheuermann. *Demo: A Symbolic Execution Framework for Network Analysis and Verification*. Accepted for publication in: Proceedings of the 2026 IEEE 51st Conference on Local Computer Networks (LCN), 2026.

Demo: A Symbolic Execution Framework for Network Analysis and Verification

Philip Jonas Franz*, Konrad Altenhofen*, Jonas Schoenen*, Osama Abboud†,
Tobias Meuser*, Björn Scheuermann*
*Communication Networks Lab, Technical University of Darmstadt, Germany
†Huawei Heisenberg Research Center, Munich, Germany
E-Mail: *{jonas.franz, konrad.altenhofen, jonas.schoenen, tobias.meuser, bjoern.scheuermann}@tu-darmstadt.de,
†osama.abboud@huawei.de

***Abstract*—Modern computer networks are becoming increasingly complex, complicating their management and verification. Existing tools for infrastructure management do not support network semantics-based reachability analysis, which would enable resilience and security analysis. To enable such analysis of network graphs, we propose a versatile symbolic execution framework that implements a stateful graph search. While being domain-agnostic, we demonstrate the usefulness of our approach specifically for computer network path discovery and firewall rule verification.**

***Index Terms*—Symbolic Execution, Network Analysis, Security, Resilience**

I. Introduction

Modern communication networks consist of a multitude of different components that form an increasingly complex system. Managing and analysing these networks has therefore become challenging. To this end, network management systems that serve as a central information base gain increasing importance. The available tools, however, focus on inventory management and offer only limited capabilities for network semantics-based analysis and verification. At the same time, reachability questions, the correctness of firewall rules, resilience planning, and configuration verification all require the underlying ground truth to reflect network semantics accurately.

We present a lightweight graph traversal engine that leverages symbolic execution techniques for path discovery and analysis based on network semantics. The presented engine operates on a graph that is capable of encoding network semantics and metadata. This approach enables the analysis of entire networks and individual network components at varying levels of detail. In addition to packet-based analysis, the engine can reference metadata, *e.g.*, OS and vendor information, during analysis. The presented approach can therefore serve as an analysis layer on top of a ground truth. We contribute the following: (i) propose an encoding schema for graph models with meta information, (ii) utilise symbolic execution for analysis of such graphs, and (iii) demonstrate the capabilities of our approach for network analysis and verification. We further open-source our tool[1] to encourage the integration and adoption into other projects.

This work has been partly funded by the LOEWE initiative (Hesse, Germany) within the emergenCITY center [LOEWE/1/12/519/03/05.001(0016)/72], the German Federal Ministry of Research, Technology and Space (BMFTR) within the Software Campus Project "MoRNAD", and by the Deutsche Forschungsgemeinschaft (DFG, German Research Foundation) under grant 503231190.

II. Related Work

Traditionally, inventory management tools such as *Infrahub* [1] and *Netbox* [2] are used as network information-related ground truth. Since release 1.10.0, *Infrahub* supports tracing relationships of objects (such as devices, interfaces, sites, circuits, and platforms) stored in the underlying graph database. These paths provide insights into transitive dependencies, but do not primarily trace possible packet paths through the network. Mogul et al. [3] present a more complex model. It features multiple abstraction layers and is capable of capturing topology information, as well as operators' intent.

King et al. [4] introduce symbolic execution as a formal verification method for software testing. Symbolic execution, among other formal methods, has been utilised to analyse and verify networks [5]. Most notably, the authors of [6] developed a tool to describe symbolic actions on packet headers. A similar approach is used in header space analysis, where headers are represented as bit-vectors, and transfer functions encode the behaviour of middleboxes [7]. The authors of [8] developed a tool to further consider symbolic state in middleboxes. Zhang et al. [9] present a symbolic SDN router that can also analyse symbolic failure states. These approaches enable the verification of network properties such as reachability, loop detection, change tracking, header visibility, memory safety, isolation, and load-balancer properties.

We present a lightweight symbolic graph traversal engine that employs symbolic path discovery to analyse information from the network's ground truth. In contrast to infrastructure management tools, the presented engine can verify model properties based on network semantics. In contrast to network configuration tools, it encodes network semantics in a graph instead of behavioural or code-like descriptions. We build upon the concept of header space analysis, extending network models with queryable metadata, supporting what-if and failure analysis.

[1] https://github.com/tuda-kom/symbolic-network-traversal-engine

© LCN, 2026. The documents distributed by this server have been provided by the contributing authors as a means to ensure timely dissemination of scholarly and technical work on a non-commercial basis. Copyright and all rights therein are maintained by the authors or by other copyright holders, not withstanding that they have offered their works here electronically. It is understood that all persons copying this information will adhere to the terms and constraints invoked by each author's copyright. These works may not be reposted without the explicit permission of the copyright holder.

Philip Jonas Franz, Konrad Altenhofen, Jonas Schoenen, Osama Abboud, Tobias Meuser, Björn Scheuermann. *Demo: A Symbolic Execution Framework for Network Analysis and Verification*. Accepted for publication in: Proceedings of the 2026 IEEE 51st Conference on Local Computer Networks (LCN), 2026.

III. Design

In this section, we introduce the design of the stateful graph search. First, we introduce the symbolic state representation and the graph model, and then we explain the search algorithm.

Symbolic State Model. We model the symbolic state of the search as a finite set of variables. The set of all variables $\mathcal{K}$ is given through the graph model. Possible variables could be, *e.g.*, the source IP address or destination port. For string variables, the value domain is finite and constrained to all constant strings in the model. For integer variables, the value domain is defined as $\mathbb{Z}$. A symbolic state S maps each variable $k \in \mathcal{K}$ to a symbolic value, which is a subset of or equal to the domain over which the variable is defined. By default, variables are wildcards, *i.e.*, they contain all values of their domain.

We define the match operation that checks whether two given states overlap. Given states S_1 and S_2, we say S_1 *matches* S_2 iff $\forall k \in \mathcal{K} : S_1[k] \cap S_2[k] \neq \varnothing$. We further define two operations that are applied to a state during edge traversal. First, the narrow operation $\odot$, which narrows a state S_1 based on a second state S_2:

$$S_1 \odot S_2 := S_1 \cap S_2 = (S_1[k] \cap S_2[k])_{k \in \mathcal{K}}$$

Second, the action operation $\oplus$, which, given a state S and a series of key-value pairs, overwrites the variables $k_1, \ldots, k_n \in \mathcal{K}$ of the state with the respective symbolic values $v_1, \ldots, v_n$:

$$S \oplus ((k_1, v_1), \ldots (k_n, v_n)) := S[k_i] \leftarrow v_i : i = 1, \ldots, n$$

Loop Detection. Loops are pruned through subsumption checking, a technique used in previous works [10]. Given two states S_1 and S_2, iff $\forall k \in \mathcal{K} : S_1[k] \supseteq S_2[k]$, we say S_1 *subsumes* S_2 and write $S_1 \succeq S_2$. Using this property, if a search path encounters a node with state S_2, which was previously traversed by the same search path with a state S_1, we conclude that the search encountered a loop if $S_1 \succeq S_2$.

Graph Model. Building on the symbolic state model, we formulate the graph model. Edges are directed and optionally carry a match-guard and actions. The match-guard is a state that the current search state must match in order to continue the search over this edge. Afterwards, the match-guard is applied to the search state using the $\odot$ operation, narrowing the state, and enforcing the constraint. This pattern is crucial to encode network semantics into the model: suppose an edge has a match-guard m[dst_ip: 10.4.0.4|10.2.0.5]. The search arrives with state $S = \{\text{dst_ip: } 10.4.0.0/16\}$. The state matches the match-guard, and is narrowed to $S' = \{\text{dst_ip: } 10.4.0.4\}$. This reflects network semantics: a packet would not be routed over a link if the IP address is not reachable via that link. Actions are defined as a series of key-value pairs, which overwrite the search state with their value using the $\oplus$ operation, after the state was narrowed by the $\odot$ operation. This is required in some cases, *e.g.*, Network Address Translation (NAT). Additionally, nodes and edges in the graph carry metadata labels encoded as a key-value map, *e.g.*, the device manufacturer or the OS family.

Graph Search. Based on the previously introduced concepts, we formulate the stateful search algorithm. The search algorithm, as outlined in Algorithm 1, has four input parameters: the initial symbolic state S_0, the start node n_0, and two key-value maps, which define accept- and reject-labels. The accept-label map A identifies target nodes during the search. If the labels of a node match all of the accept-labels, the node is considered to be a target node and the current search path is added as a result. The reject-label map R is the inverse: if a node or edge contains a label that matches any of the reject labels, the search will not traverse the node or edge. The search algorithm outputs a set of paths, which are a list of connected edges.

Algorithm 1 Symbolic Search Algorithm

```
Initialize queue Q ← {(n_0, S_0, [], {n_0 : {S_0}})}
Initialize result set Π ← ∅
while Q ≠ ∅ do
    (n_i, S, E, V) ← Q.dequeue()
    if ∃r ∈ R : r ∈ n_i.labels then
        continue                                   ▷ Node rejected
    else if ∀k ∈ A : A[k] = n_i.labels[k] then
        Π ← Π ∪ {E}                                ▷ Node accepted, path found
        continue
    end if
    for edge (n_i, e, n_j) where S matches e.match_guard do
        if ∃k ∈ R : R[k] = e.labels[k] then
            continue                               ▷ Edge rejected
        end if
        S' ← (S ⊙ e.match_guard) ⊕ e.actions
        if ∃ S'' ∈ V[n_j] : S'' ⪰ S' then
            continue                               ▷ State subsumed, loop
        end if
        V' ← V ∪ {n_j : {S'}}
        Q.enqueue((n_j, S', E + [e], V'))
    end for
end while
return Π
```

IV. Demonstration

Our graph traversal engine, as described in Section III, loads the graph model from a `JSON` model file. It can be used through a Command Line Interface (CLI). For this demonstration, we additionally visualise the model and the search results in a web-based User Interface (UI). The web-based UI, as shown in Fig. 1, allows (i) to create and edit models, (ii) to visualise the model and query results, and (iii) to query a model using the graph traversal engine. A search query, entered via the CLI or the UI, specifies a starting node and a list of accept labels that determine the search target. Additionally, an initial search state and a set of reject labels can be specified to refine a query. The discovered paths are returned as `JSON` and can be visualised and inspected in the UI.

A. Example Applications

Our approach can be used for a variety of applications in network management, configuration, analysis, and verification. The next part showcases three of them, namely, using the

© LCN, 2026. The documents distributed by this server have been provided by the contributing authors as a means to ensure timely dissemination of scholarly and technical work on a non-commercial basis. Copyright and all rights therein are maintained by the authors or by other copyright holders, not withstanding that they have offered their works here electronically. It is understood that all persons copying this information will adhere to the terms and constraints invoked by each author's copyright. These works may not be reposted without the explicit permission of the copyright holder.

Philip Jonas Franz, Konrad Altenhofen, Jonas Schoenen, Osama Abboud, Tobias Meuser, Björn Scheuermann. *Demo: A Symbolic Execution Framework for Network Analysis and Verification*. Accepted for publication in: Proceedings of the 2026 IEEE 51st Conference on Local Computer Networks (LCN), 2026.

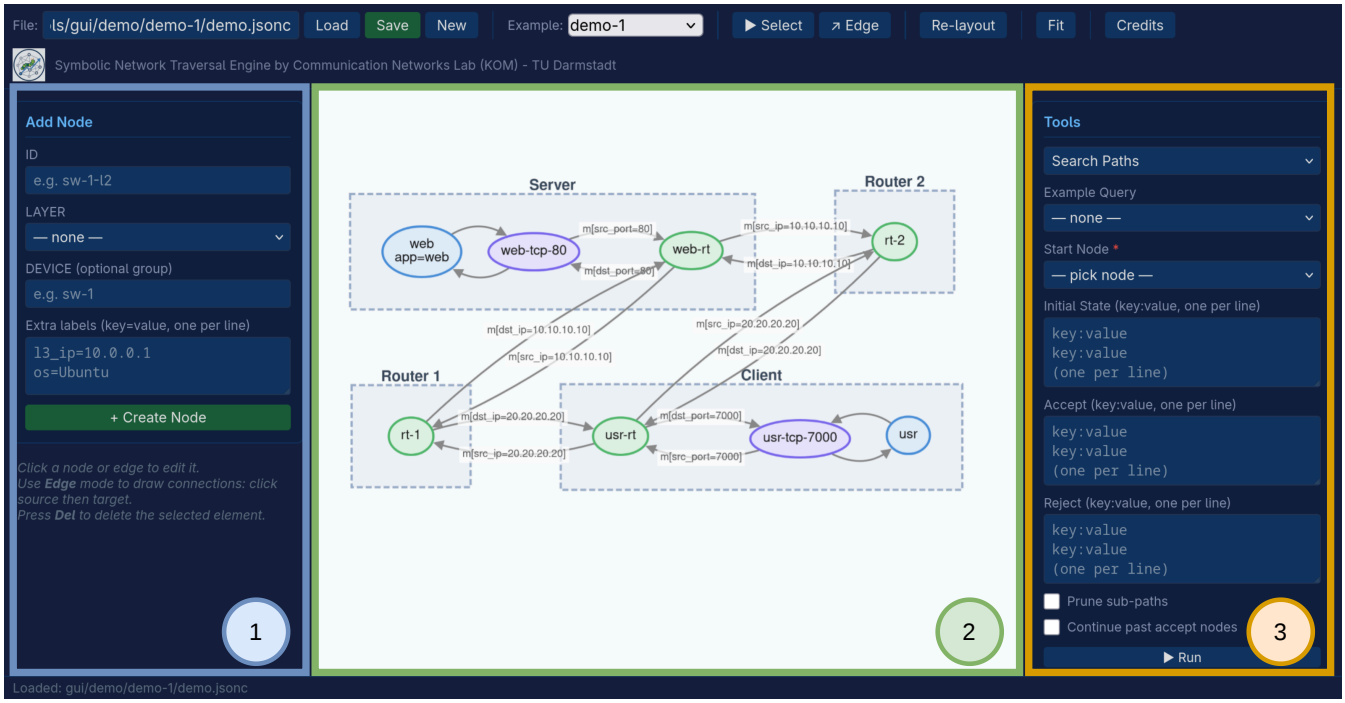

Fig. 1: UI of the graph traversal engine with (1) model editor, (2) model viewer, and (3) query interface.

traversal engine (i) to discover paths in the network model, and (ii) to validate firewall rules (iii) to compute available network capacity.

Path Discovery. As depicted in Fig. 1, a network can be modelled by representing components as groups of nodes that carry meta-information as labels, and edges that describe how the symbolic state changes upon traversal by match-guard m[] and action a[] annotations. The search starts at a given start node and discovers all semantically possible paths to all nodes that match the accept label. Every search starts with a given initial state. This state can be a wildcard or specify initial values for individual state variables. Upon edge traversal, the symbolic state is narrowed by match-guards and potentially overwritten by actions, allowing the final state of every discovered path to show which class of packages could have taken this path. The path's metadata, including the intermediate states and all labels on the path, is tracked, too. Paths that exist physically, but are semantically impossible, are pruned by means of match-guards. The reject field enables what-if analysis and complex path requirements, such as excluding paths traversing routers of specific vendors, enabling resilience analysis, policy-based, and sovereign routing.

Firewall Analysis. Besides analysis of network graphs, our approach can also be used to analyse singular components, *e.g.*, to verify the functionality of firewall rules, which restrict flows based on header fields, such as source and destination IP address or ports. These rules can be mapped to the corresponding match-guards on edges. Conjunctions of rules can be mapped by placing multiple edges in succession, while disjunctions can be implemented by placing multiple outgoing edges from a node. The symbolic path discovery then traverses the graph, and the search state is automatically narrowed to the allowed header fields corresponding to the rules' allowed headers. If the current search state does not match any outgoing edge's match-guards, the search path terminates, which corresponds to a rejected packet. Because the graph model can define arbitrary state variables, the representation can be used to model a wide range of protocols and complex match conditions. Stateful firewalls can only be modelled by encoding connection-tracking information into the graph representation, due to the graph being static while the search is running.

Throughput Analysis. The path discovery combined with attached metadata can also support throughput analysis and capacity planning, *e.g.*, to compute the aggregate available bandwidth between two sets of hosts. Given a network graph that includes link-speed information and two sets of nodes A and B with $A \cap B = \varnothing$, the maximum throughput can be calculated as follows. Firstly, the set of paths from all nodes $a \in A$ to all nodes $b \in B$ is computed. This requires $|A|$ symbolic searches, assuming a label set exists that uniquely identifies all nodes in B. A set union operation over the nodes and edges in the resulting path set then constructs a subgraph of the original network graph. Finally, a super-source node S and super-sink node T are introduced, such that S is connected to all nodes in A, and T to all nodes in B. The maximum flow per edge is given by the link speed metadata, with a default value of ∞. The maximum blocking throughput on the subgraph can now be computed by a Max-Flow algorithm, such as Dinic's algorithm [11]. To simulate an

© LCN, 2026. The documents distributed by this server have been provided by the contributing authors as a means to ensure timely dissemination of scholarly and technical work on a non-commercial basis. Copyright and all rights therein are maintained by the authors or by other copyright holders, not withstanding that they have offered their works here electronically. It is understood that all persons copying this information will adhere to the terms and constraints invoked by each author's copyright. These works may not be reposted without the explicit permission of the copyright holder.

Philip Jonas Franz, Konrad Altenhofen, Jonas Schoenen, Osama Abboud, Tobias Meuser, Björn Scheuermann. *Demo: A Symbolic Execution Framework for Network Analysis and Verification*. Accepted for publication in: Proceedings of the 2026 IEEE 51st Conference on Local Computer Networks (LCN), 2026.

outage, the symbolic search is run with a sensible reject set R to exclude the desired components. For example, to simulate a switch outage, one could set $R = \{\text{name} : \text{core_sw_1}\}$. This excludes all paths through the switch from the subgraph, thereby excluding the available capacity from the Max-Flow computation, demonstrating the throughput reduction in the event of such a failure.

V. Conclusion & Future Work

In this paper, we introduced a lightweight, stateful graph-traversal engine and a method to model symbolically executable graph models. The demonstrated use cases, path discovery in computer networks, firewall rule verification, and throughput analysis, highlight the versatility of the presented approach and the usefulness under varying degrees of detail. Future work will focus on refining the design of the network model to represent more complex networks and on deriving network graphs from real-world deployments and data sources.

Acknowledgments

The implementation of the UI was assisted by generative AI. The authors have tested the code extensively and take full responsibility.

References

[1] Infrahub Website. [Online]. Available: https://opsmill.com/
[2] Netbox website. [Online]. Available: https://netboxlabs.com/
[3] J. C. Mogul, D. Goricanec *et al.*, "Experiences with Modeling Network Topologies at Multiple Levels of Abstraction," in *USENIX NSDI*, 2020, pp. 403–418.
[4] J. C. King, "Symbolic Execution and Program Testing," *Commun. ACM*, vol. 19, no. 7, pp. 385–394, Jul. 1976.
[5] Y. Li, X. Yin *et al.*, "A Survey on Network Verification and Testing With Formal Methods: Approaches and Challenges," *IEEE Communications Surveys & Tutorials*, vol. 21, no. 1, pp. 940–969, 2019.
[6] R. Stoenescu, M. Popovici *et al.*, "SymNet: Scalable symbolic execution for modern networks," in *SIGCOMM*, 2016, pp. 314–327.
[7] P. Kazemian, G. Varghese, and N. McKeown, "Header Space Analysis: Static Checking for Networks," in *USENIX NSDI*, 2012, pp. 113–126.
[8] Y. Yuan, S.-J. Moon *et al.*, "NetSMC: A Custom Symbolic Model Checker for Stateful Network Verification," in *USENIX NSDI*, 2020, pp. 181–200.
[9] P. Zhang, D. Wang, and A. Gember-Jacobson, "Symbolic router execution," in *SIGCOMM*, 2022, pp. 336–349.
[10] S. Anand, C. S. Păsăreanu, and W. Visser, "Symbolic Execution with Abstraction," *Int. Journal on Software Tools for Technology Transfer*, vol. 11, no. 1, pp. 53–67, 2009.
[11] Y. Dinitz, "Algorithm for Solution of a Problem of Maximum Flow in Networks with Power Estimation," *Soviet Math. Dokl.*, vol. 11, pp. 1277–1280, 01 1970.

© LCN, 2026. The documents distributed by this server have been provided by the contributing authors as a means to ensure timely dissemination of scholarly and technical work on a non-commercial basis. Copyright and all rights therein are maintained by the authors or by other copyright holders, not withstanding that they have offered their works here electronically. It is understood that all persons copying this information will adhere to the terms and constraints invoked by each author's copyright. These works may not be reposted without the explicit permission of the copyright holder.