ANK Workbench: An Integrated Platform for Dual-Fidelity Network Design Validation

Anonymous Author(s)

Abstract

Network engineers validating design changes must choose between fast but approximate protocol simulation and slow but realistic container-based emulation. In practice, they use both — running a simulator for rapid iteration, then deploying to a container lab for ground-truth verification — but no existing platform connects these two environments or automates the comparison between them.

We present ANK Workbench, a web-based orchestration platform that integrates the full network design lifecycle: declarative topology modelling, protocol simulation, container-based emulation via ContainerLab, and automated comparison of results across fidelity levels. The platform introduces a *dissonance detection* pipeline that normalises routing state from both environments and surfaces genuine divergences, a content-addressed pipeline that tracks artifact staleness across workflow steps, and a dual-backend assertion framework that runs the same validation suite against both simulator and emulated devices.

ANK Workbench is open-source and has been used to validate topologies ranging from 5-node test beds to 200-node enterprise designs. We describe its architecture, the key mechanisms that enable dual-fidelity workflows, and lessons learned from building and using the platform.

Keywords

network validation, network emulation, digital twin, dual-fidelity testing, configuration drift

1 Introduction

Consider an enterprise network engineer planning a BGP migration: converting a full-mesh iBGP topology to route reflectors across three sites, affecting 47 peering sessions and 200 routers. She writes the target state as a declarative YAML model and runs a lightweight protocol simulation. Twelve seconds later, the simulator reports clean OSPF and BGP convergence. She then deploys the same topology to a ContainerLab environment running real Arista EOS containers — and discovers that two nodes retain stale static routes the simulator could not model. To find this, she had to SSH into each of 200 containers, run `show ip route`, and manually compare the output against the simulator's routing tables.

This scenario illustrates a fundamental tension in network validation. *Simulators* such as Batfish [5] and NetPlumber [8] verify reachability properties against an abstract model but cannot detect real-world state drift, vendor-specific quirks, or data-plane forwarding bugs. *Emulators* such as GNS3 [7], EVE-NG [4], and Cisco Modeling Labs run real device images but provide no automated comparison against design intent — operators must manually inspect device state and mentally diff it against their model. *Network state collectors* such as Suzieq [9] gather operational data but lack the integrated design-validate-compare workflow. In practice, engineers cobble together ad-hoc scripts to bridge these tools, wasting hours on integration plumbing rather than engineering judgment.

ANK Workbench addresses this gap with a single observation: *validate at simulation speed first, then surface only the discrepancies that survive comparison with full emulation.* By treating simulation and emulation as two fidelity levels within one pipeline — rather than separate tools — the platform reduces the operator's attention from N^2 route entries across N devices to the handful of genuine divergences that require action.

ANK Workbench is a web-based orchestration platform that integrates the network design lifecycle under a content-addressed pipeline: topology generation, declarative modelling, protocol simulation, container-based emulation, automated assertion testing, dissonance detection, and one-click remediation. Being domain-specific allows us to exploit network structure in ways a general-purpose tool cannot: canonical route normalisation, graph-theoretic design guardrails, and deterministic VxLAN-based topology distribution across Docker hosts.

We make the following contributions:

1. **Dual-fidelity dissonance detection (§4):** A pipeline that normalises routing state from both simulator and container environments, using a SHA-based stabilisation loop to guard against mid-convergence snapshots, and produces structured diff reports with per-interface granularity.
2. **Content-addressed pipeline orchestration (§5):** A typed state machine coordinating async workflow steps, with SHA-hash-based staleness propagation that eliminates redundant reruns and pushes invalidation events to the frontend in real time.
3. **Dual-backend assertion framework (§6):** A common assertion model (ping, path trace, BGP session, OSPF neighbour count) that executes against both the simulator via Unix domain sockets and ContainerLab via Docker exec, enabling pre-flight validation before committing to full emulation.
4. **Integrated platform and lessons learned (§3, §8):** An open-source, end-to-end platform connecting topology generation, simulation, emulation, validation, chaos injection, and visual analytics, with a discussion of practical lessons from building and deploying it.

2 Background and Motivation

2.1 The Fidelity Spectrum

Network validation tools span a fidelity spectrum from fast-but-approximate to slow-but-realistic:

Static analysis tools like Batfish [5] parse device configurations and compute a forwarding model without executing any protocol code. They answer reachability queries in seconds but cannot model dynamic convergence, vendor-specific timer behaviour, or runtime state.

Table 1. Feature comparison of network validation platforms.

	Simulation	*Emulation*	*Auto compare*	*Assertions*	*Remediation*	*Web UI*
Batfish	✓			✓		✓
GNS3		✓				✓
EVE-NG		✓				✓
Cisco CML		✓				✓
ContainerLab		✓				
Suzieq				✓		✓
ANK Workbench	✓	✓	✓	✓	✓	✓

Protocol simulators execute simplified protocol state machines (OSPF/BGP/IS-IS) against a topology model. They model convergence dynamics and produce routing tables, but use idealised link behaviour and omit vendor-specific control-plane implementation details.

Container-based emulators such as ContainerLab [3] run real vendor NOS images (Arista cEOS, Nokia SR Linux) inside containers. They produce ground-truth routing state but require gigabytes of vendor images, minutes to boot, and significant compute resources.

Hardware labs use physical devices. They provide the highest fidelity but are expensive, slow to reconfigure, and inaccessible to most engineers.

In practice, engineers use simulation for rapid iteration and emulation for final validation — but they use these as *separate tools* with no automated bridge between them. ANK Workbench occupies the integration layer, connecting the simulation and emulation tiers under a single pipeline.

2.2 Existing Tools and Their Gaps

Table 1 summarises the capabilities of existing platforms.

The central gap is the *automated comparison* column: no existing platform systematically compares routing state across fidelity levels and presents actionable differences to the operator.

3 System Architecture

ANK Workbench is structured as a layered web application with a Python/FastAPI backend and a React frontend, connected by a WebSocket event bus for real-time updates. Figure 1 shows the system architecture.

3.1 Design Principles

Three principles guided the architecture:

Fidelity as a parameter, not a tool choice. The same topology model drives both the protocol simulator and ContainerLab deployment. Assertions, dissonance checks, and visualisation all accept a backend parameter rather than being hardcoded to one execution environment.

Content-addressed artifacts. Every pipeline step produces a JSON artifact whose SHA-256 hash is recorded. Downstream steps are invalidated only when upstream hashes change, not when timestamps advance. This makes the pipeline idempotent and reproducible across machines.

Real-time feedback. All long-running operations (simulation, deployment, dissonance collection) stream progress events over the WebSocket bus. The frontend renders incremental results as they arrive — per-node dissonance status, per-assertion pass/fail, per-container readiness — rather than waiting for batch completion.

3.2 Workflow Pipeline

The platform orchestrates a linear pipeline:

`generate` → `validate` → `simulate` → `deploy` → `assert` → `compare` → `remediate`

Each step is a node in a dependency DAG managed by a workflow coordinator backed by a typed state machine. Steps can be run individually or as a full pipeline. The coordinator publishes staleness-changed events when upstream artifacts are modified, allowing the UI to show stale-step badges without polling.

4 Dual-Fidelity Dissonance Detection

The dissonance engine is the platform's primary analytical contribution. It answers the question: *where does the simulator's prediction disagree with the emulated reality, and why?*

4.1 Overview

Dissonance detection operates in two independent modes:

- **Routing dissonance** compares per-node routing tables between the simulator (RIB) and the container environment (kernel FIB).
- **Structural dissonance** compares the intended topology model against an uploaded operational topology discovered from a live network.

Both modes produce structured reports consumed by the same frontend components, but they are independently invokable and address orthogonal failure modes.

4.2 Routing Dissonance Pipeline

The routing dissonance pipeline proceeds in four stages:

Stage 1: Collection. The simulator's routing tables are extracted from the most recent simulation artifact stored on disk. Concurrently, the live FIB is collected from each running container by executing `ip -j route show table all` via Docker exec. Each command returns JSON-formatted kernel routes.

Stage 2: Stabilisation. A naïve snapshot of a container's FIB during protocol convergence would produce spurious differences. To guard against this, the collector implements a *stabilisation loop*: it polls each node's FIB repeatedly and computes the SHA-256 hash of the JSON response. Collection is accepted only when two consecutive polls produce identical hashes. If stability is not reached within a configurable timeout (default: 30 seconds), the node is marked `NOT_STEADY` and excluded from the diff.

Stage 3: Normalisation. Raw routes from both sources are normalised into a canonical `RouteEntry` structure: destination prefix in CIDR notation, sorted next-hop list, metric, and protocol. User-configurable `ToleranceKnobs` control which fields participate in comparison — for example, `ignore_next_hop_order` collapses

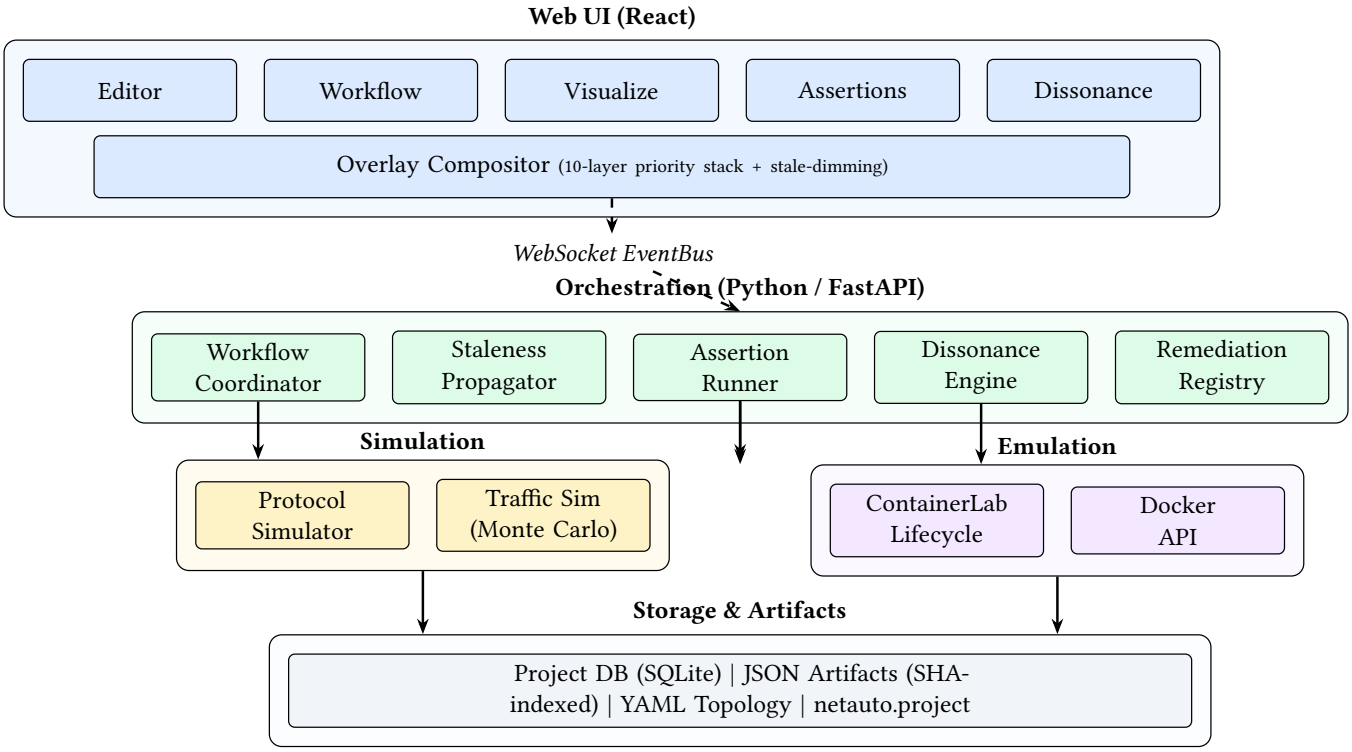

Figure 1. ANK Workbench system architecture. Solid arrows indicate data flow; dashed arrows indicate WebSocket event channels. The orchestration layer dispatches work to simulation and emulation backends, storing results as content-addressed artifacts.

ECMP path ordering differences, and `ignore_protocol` suppresses differences between "connected" and "direct" route types that are semantically equivalent across vendors.

Stage 4: Diff. The normalised route sets are serialised to deterministic text lines keyed on destination prefix and compared as string sets. The diff produces three categories per node: *added* (routes present in the container but not the simulator), *removed* (the reverse), and *changed* (same destination but different attributes). Nodes are classified as `same`, `different`, or `unknown`.

Progress events are published per-node to the WebSocket bus, allowing the frontend to render streaming results as each container is processed.

4.3 Structural Dissonance

The structural diff engine operates on graph topology rather than routing state. It compares the ANK `TopologyModel` (the design intent) against an `OperationalTopology` (discovered from a live network via tools like ank-parse or NetBox export).

Node sets are compared by name; link sets are compared using undirected frozensets of `(node_name, interface_name)` pairs to handle directionality. For matched nodes, per-interface fields (IP address normalised to CIDR, operational status) are compared to produce `InterfaceDiscrepancy` records.

4.4 Remediation Generation

Dissonance findings can be acted upon. The `RemediationRegistry` maps `(platform, issue_type)` tuples to generator functions via a decorator:

Listing 1. Remediation generator registration.

```
@registry.register("arista_eos", "interface_ip_mismatch")
def fix_ip(iface, expected_ip, actual_ip):
    return [
        "configure terminal",
        f"interface {iface}",
        f"ip address {expected_ip}",
        "exit",
    ]
```

The `RemediationService` iterates a node's discrepancies, looks up each `(platform, issue)` pair, and emits an ordered CLI command sequence. The frontend exposes a "Push Fix" button that streams these commands to the device via Docker exec.

5 Pipeline Orchestration

5.1 State Machine

The workflow is governed by a typed state machine with states CREATED, VALIDATING, SIMULATING, VISUALIZING, COMPLETE, and FAILED. Guard hooks enforce preconditions before each transition — for example, simulation cannot start until validation has completed. The state machine supports idempotent self-transitions and rerun transitions from COMPLETE, enabling incremental reruns without resetting the full pipeline.

5.2 Content-Addressed Staleness

Rather than using timestamps, the staleness propagator tracks the SHA-256 hash of each step's input artifacts. When a step completes, the current hashes are snapshotted. Before a step runs, its input hashes are compared against the snapshot: if any differ, the step and all its DAG descendants (computed via `networkx.descendants()`) are marked stale.

This design has two advantages over timestamp-based approaches:

1. It is correct across time zones, filesystem clock skew, and NFS-mounted project directories.
2. It avoids redundant reruns when a user edits and then reverts a topology — the hash returns to its original value, clearing the stale flag.

Staleness changes are pushed to the frontend via WebSocket, where stale pipeline steps are badged and all visualisation overlays are automatically dimmed to $\alpha = 0.55$ to warn that results may not reflect the current topology.

6 Dual-Backend Assertions

The assertion framework provides repeatable, automated verification of network properties. An assertion suite is a list of typed assertions, each specifying a *kind* (ping, path trace, BGP session state, OSPF neighbour count), source and target nodes, and expected outcome.

6.1 Backend Abstraction

The same assertion suite runs against two backends:

Simulator backend. Connects to per-node Unix domain socket consoles exposed by the simulator process. Sends a command string (e.g., `ping 10.0.0.2 count 3`) and classifies stdout by keyword matching: "0% packet loss" maps to pass, "100% packet loss" maps to fail. Gracefully degrades to `unsupported` if the socket is unavailable.

ContainerLab backend. Resolves the container ID by node name via the deployment status API. Checks tool availability inside the container (`command -v ping`) with per-container caching to avoid repeated probes. Executes the assertion command via `docker exec`. A fallback heuristic handles cross-distribution differences in `ping` flag syntax (`-W` vs `-w` for timeout) by detecting usage-error patterns in stderr.

6.2 Execution Model

Assertions execute sequentially within a suite, each wrapped in `asyncio.wait_for` with a per-assertion timeout. Per-assertion `started`/`completed` events stream to the WebSocket bus. The runner handles cancellation cleanly and caps output snippets to 2,000 characters to prevent oversized artifacts.

The key value of the dual-backend design is *pre-flight validation*: an engineer can run the assertion suite against the simulator in seconds to catch obvious errors, then re-run the same suite against ContainerLab to verify against real device behaviour. Failures in the simulator run save the cost of deploying containers unnecessarily.

7 Additional Platform Capabilities

Beyond the core dual-fidelity pipeline, ANK Workbench integrates several capabilities that individually are not novel but collectively provide a unified workflow.

7.1 Topology Generation and Validation

The platform includes a pattern-based topology generator supporting spine-leaf, mesh, hub-spoke, ring, and campus-tier patterns. An IP allocator partitions management (`10.0.0.0/16`) and point-to-point (`192.168.0.0/16`) address spaces to prevent collisions. Generated YAML is deterministically key-sorted for Git-diffable output.

A validation engine runs graph-theoretic checks on topology drafts in real time: `networkx.articulation_points()` identifies single points of failure, `nx.bridges()` flags critical links, and IP/protocol rules catch addressing errors and configuration inconsistencies. Findings carry node and interface references, enabling the frontend to spatially annotate the topology canvas.

7.2 Multi-Host Topology Distribution

For topologies that exceed a single machine's capacity, the topology generator partitions nodes by their `execution_host` attribute and generates one ContainerLab YAML per host. Cross-host links are rendered as VxLAN tunnel endpoints with deterministic VNI assignment (`10000 + link_index`), requiring no user reasoning about underlay networking. Deployment proceeds in parallel across hosts.

7.3 Chaos Injection

A chaos engineering panel allows operators to inject faults — link cuts, latency, node reboots — into running ContainerLab topologies and observe convergence behaviour on the topology canvas in real time. Chaos state is rendered as a highest-priority overlay layer that visually supersedes all analytical overlays, ensuring impaired elements are always visible.

7.4 Visual Analytics

The topology canvas supports 10 composable overlay layers: base style, link utilisation, protocol state, routing tables, dissonance status, graph centrality, resilience metrics, live telemetry, time-machine highlights, and chaos state. A single `useMemo` pass computes the merged style map with deterministic priority ordering. When the topology hash or artifact hash diverges from the current state, all overlays are automatically dimmed to $\alpha = 0.55$ ("stale-dimming"), preventing operators from acting on outdated analysis without adding a blocking modal.

7.5 Simulation Time Machine

A temporal scrubber allows operators to navigate protocol convergence events at microsecond granularity. A two-phase fetch strategy — bounds-only query followed by a windowed event fetch around the cursor position — avoids loading the entire trace into the browser while enabling smooth interactive scrubbing. When the simulator does not emit native microsecond traces, the system

synthesises a coarse trace from protocol-level events and displays a visual indicator of reduced precision.

8 Lessons Learned

We reflect on lessons from building and using ANK Workbench over several months of active development.

Stabilisation is essential, not optional. Our first implementation of dissonance detection produced alarming false-positive rates. Routes would appear as "missing" simply because a container's OSPF process had not yet converged when we snapshotted its FIB. The SHA-based stabilisation loop, while simple, eliminated the vast majority of these spurious differences. Any dual-fidelity comparison system must account for the fact that emulated devices boot and converge on different timescales.

Tolerance knobs are a user-interface problem. The normalisation pipeline's tolerance parameters (ignore next-hop ordering, ignore protocol labels, ignore metrics) significantly affect which differences surface. We found that the "right" settings depend heavily on the topology and the question being asked. Exposing these as UI controls, rather than hardcoding them, was critical for adoption.

Content-addressed staleness prevents a class of user confusion. In early versions, users would edit a topology, re-run simulation, but forget to re-run dissonance detection. The stale-dimming mechanism — automatically reducing overlay opacity when artifacts are out of date — proved more effective than modal warnings, which users dismissed. Subtle visual degradation communicates staleness without interrupting the workflow.

Container readiness is harder than deployment. Deploying containers via ContainerLab is reliable; detecting when a container's NOS has fully booted is not. Vendor images have widely varying boot times (10 seconds for Alpine Linux hosts, 3 minutes for Arista cEOS) and no standardised readiness signal. Per-kind readiness strategies (process list checks, SSH port probes) are necessary but fragile.

Cross-vendor CLI differences compound. Even simple commands like `ping` differ across container base images (`-W` vs `-w` for timeout). Remediation command generation must be strictly per-platform. We initially underestimated the surface area of vendor differences, even within Linux-based containers.

9 Related Work

Network verification and simulation. Batfish [5] models device configurations as a vendor-independent data plane and answers reachability queries without executing protocol code. Minesweeper [2] and ARC [6] apply formal verification to network configurations. These tools provide strong guarantees within their model's scope but cannot detect divergence from real-device behaviour. ANK Workbench complements verification: simulation results serve as the baseline that dissonance detection compares against ground truth.

Network emulation platforms. GNS3 [7], EVE-NG [4], and Cisco Modeling Labs provide GUI-driven management of emulated network topologies. ContainerLab [3] pioneered lightweight, declarative container-based labs using standard Docker infrastructure. Unlike these tools, ANK Workbench does not implement its own emulation substrate — it orchestrates ContainerLab as a backend — but adds the validation, comparison, and remediation layers above.

Network state collection and testing. Suzieq [9] provides a multi-vendor network observability platform with an assertion framework. NAPALM [10] and Nornir provide programmatic device access for automation tasks. These tools focus on operational state collection; ANK Workbench focuses on the comparison between *intended* state (from the design model) and *actual* state (from the running environment), which requires an integrated pipeline spanning both simulation and emulation.

Network digital twins. The concept of a network digital twin — a virtual replica of a production network for testing and validation — has gained attention in both industry [1] and standards bodies. ANK Workbench can be viewed as a practical dual-fidelity implementation of this concept, where the simulator provides a fast-feedback twin and ContainerLab provides a high-fidelity twin, with dissonance detection bridging the two.

10 Conclusion

ANK Workbench is an open-source platform that integrates network simulation and container-based emulation under a single declarative workflow. Its dissonance detection pipeline automatically surfaces routing and structural divergences between fidelity levels, reducing the manual effort required to validate network designs. The content-addressed pipeline ensures reproducibility, while the dual-backend assertion framework enables pre-flight validation at simulation speed.

The platform has been used to validate topologies ranging from simple 5-node test beds to 200-node enterprise designs. Key lessons include the importance of stabilisation guards for FIB collection, user-configurable tolerance parameters for normalisation, and subtle visual cues (stale-dimming) over disruptive modals for communicating artifact freshness.

Future work includes expanding remediation coverage beyond Arista EOS, formal evaluation with network operators, integration with CI/CD pipelines for continuous validation, and exploration of incremental dissonance detection that avoids full FIB re-collection when only a subset of nodes are affected by a change.

References

[1] Paul Almasan, Jose Suarez-Varela, Krzysztof Rusek, Pere Barlet-Ros, and Albert Cabellos-Aparicio. 2022. Network Digital Twin: Context, Enabling Technologies, and Opportunities. *IEEE Communications Surveys & Tutorials* 24, 4 (2022), 2191–2216.

[2] 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 (SIGCOMM)*. 155–168.

[3] Roman Dodin, Stefan Betz, and Riccardo Scandariato. 2023. ContainerLab: A Modern Approach to Network Emulation. In *Proceedings of the IEEE Conference on Network Function Virtualization and Software Defined Networks (NFV-SDN)*.

[4] EVE-NG Ltd. 2024. EVE-NG: Emulated Virtual Environment — Next Generation. https://www.eve-ng.net/

[5] Ari Fogel, Stanley Fung, Luis Pedrosa, Meg Walber, Ramesh Govindan, Ratul Mahajan, and Todd Millstein. 2015. A General Approach to Network Configuration Analysis. In *Proceedings of the 12th USENIX Symposium on Networked Systems Design and Implementation (NSDI)*. 469–483.

[6] Aaron Gember-Jacobson, Raajay Viswanathan, Chaithan Prakash, Robert Grandl, Junaid Khalber, Sourav Das, and Aditya Akella. 2016. Fast Control Plane Analysis Using an Abstract Representation. In *Proceedings of the Conference of the ACM Special Interest Group on Data Communication (SIGCOMM)*. 300–313.

[7] GNS3 Project. 2024. GNS3: The software that empowers network professionals. https://www.gns3.com/

[8] Peyman Kazemian, Michael Chang, Hongyi Zeng, George Varghese, Nick McKeown, and Scott Whyte. 2013. Real Time Network Policy Checking Using Header Space Analysis. In *Proceedings of the 10th USENIX Symposium on Networked Systems Design and Implementation (NSDI)*. 99–111.

[9] Dinesh Saha and Justin Dutt. 2024. Suzieq: Software for Network Observability. https://www.stardustsystems.net/suzieq/

[10] Mircea Ulinic, David Betz, and NAPALM contributors. 2024. NAPALM: Network Automation and Programmability Abstraction Layer with Multivendor Support. https://napalm.readthedocs.io/