Towards Accessible Model-Free Verification

Alexander Krentsel[12], Oliver Ye[12], Anthony Tafoya[12], Xuqian Ma[2], Sylvia Ratnasamy[12], Anees Shaikh[2]

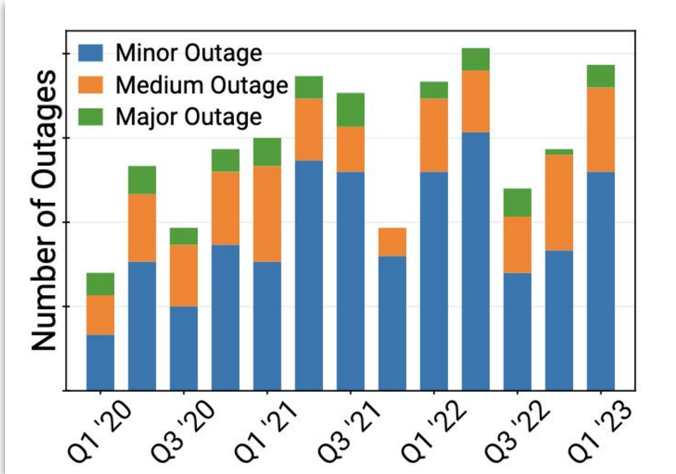
[1]UC Berkeley, [2]Google

(Still) no end in sight for WAN outages

Steady stream of academic verification literature promising safety guarantees…

- Lightyear [SIGCOMM ‘23]
- Hydra [SIGCOMM ‘23]
- Flash [SIGCOMM ‘22]
- Snowcap [SIGCOMM ‘21]
- GRoot [SIGCOMM ‘20]
- …
- BatFish [NSDI ‘15, SIGCOMM ‘23]
- Header Space Analysis [NSDI ‘12]

Yet outages continue at the same rate…

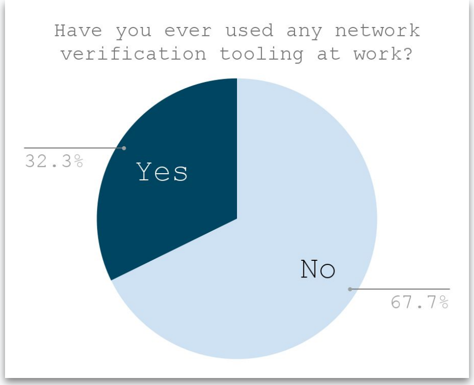


WAN Outage Rate
[Decentralized SDN, SIGCOMM ‘24]

Verification is promising, but not a silver bullet. Why not?

Real-world verification adoption is limited

Surveyed 30 network operators, interviewed 10 more



Key Results:

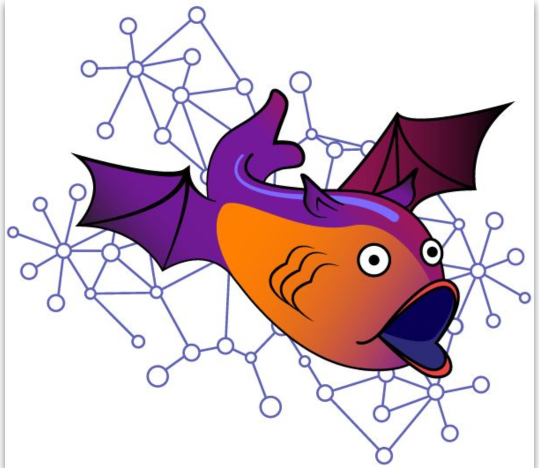
- Majority (70%) familiar with “network verification”
- *Under a third* have used network verification
- 0 / 10 interviewed were currently using verification tooling at work

Large need: 18k config pushes *a day* at Google; *many* opportunities for things to go wrong.

What is hindering verification’s adoption?

Batfish

- Most frequently mentioned verification tool
- SIGCOMM Networking System Award Winner
- 1,200+ GitHub stars
- Active, ongoing development; 12,800 commits



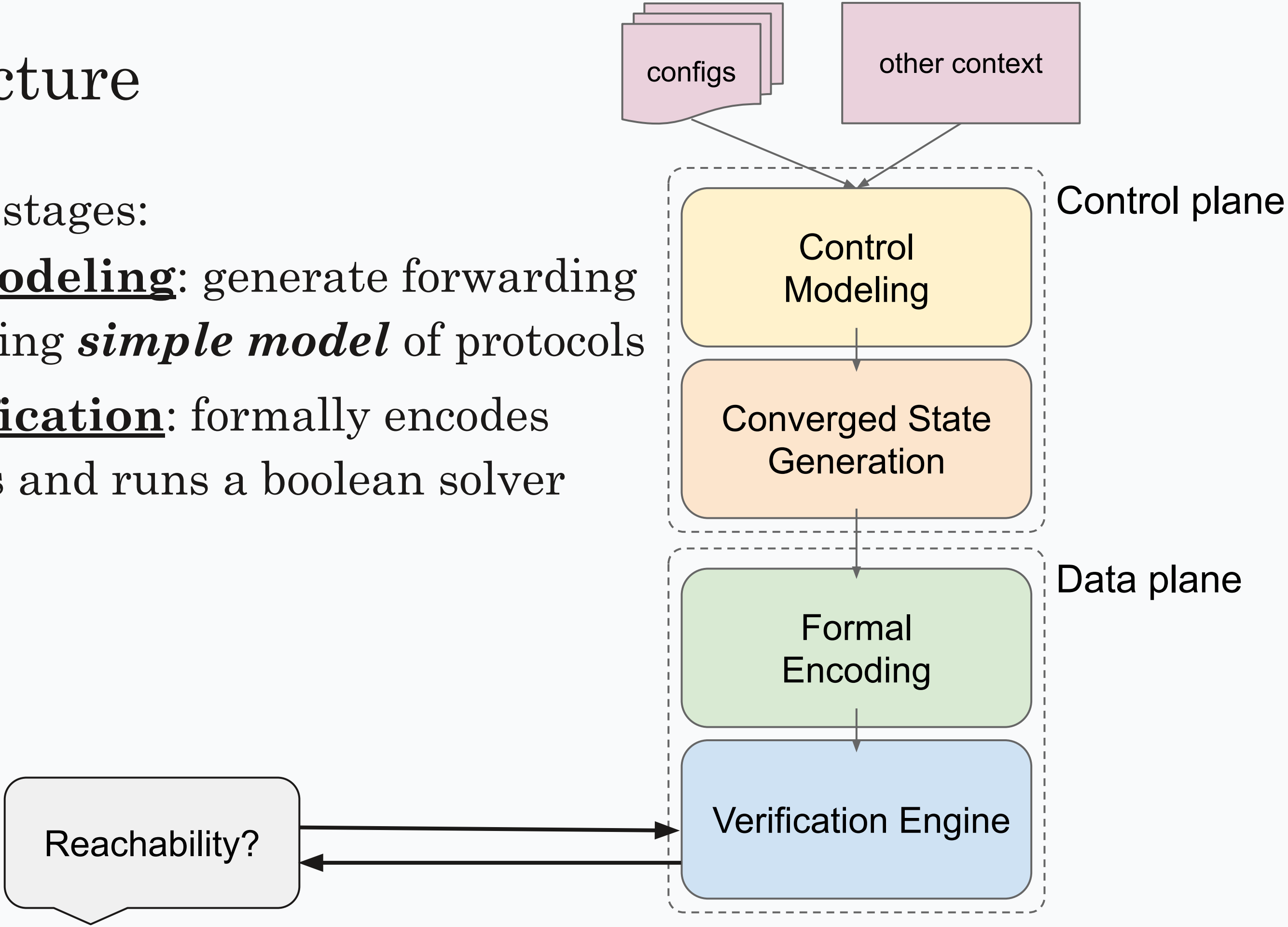
Control plane verification: given set of configuration, provide formal guarantees about post-convergence behavior (*before* deployment).

Batfish Architecture

Two logically separate stages:

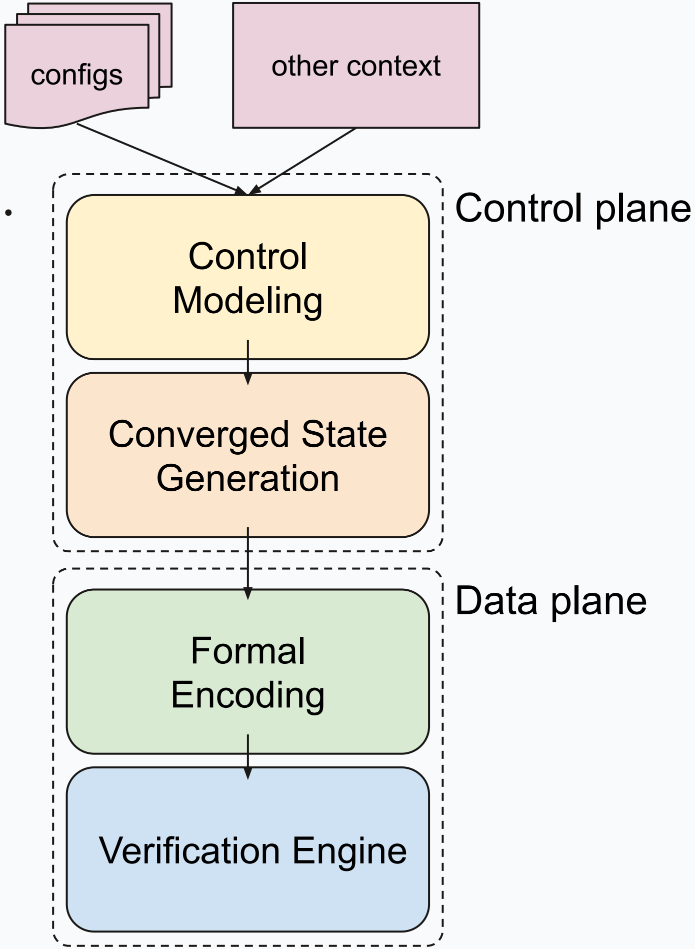
- **Control plane modeling**: generate forwarding entries by simulating ***simple model*** of protocols
- **Data plane verification**: formally encodes forwarding entries and runs a boolean solver

User queries:



Control modeling is *difficult*

Control plane model created by Batfish engineers…

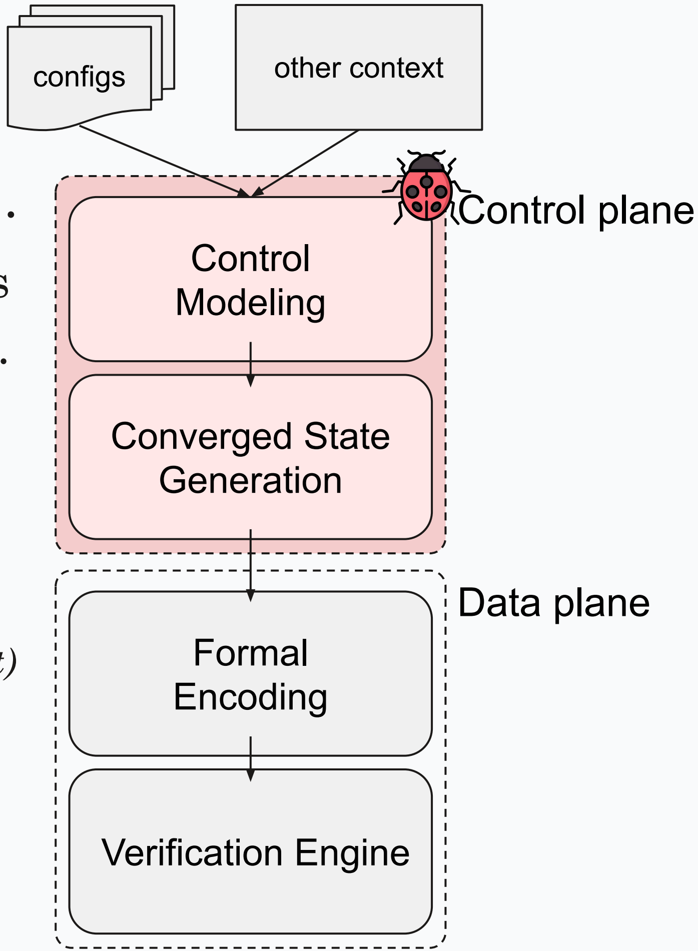


Control modeling is *difficult*

Control plane model created by Batfish engineers…

…but staying on top of all features and protocols used in practice is impossible.

- **Correctness**
  - *Custom model → bugs in model implementation/design*
  - *Misses vendor-specific customizations/behaviors*
- **Coverage**
  - *Completeness: model lags reality (e.g.: no RSVP-TE support)*
  - *Maintainability: requires continuous regression/testing*
    - *…as vendor implementation evolve*
- **Limited Fidelity**
  - *Convergence and protocols are modeled as a partial subset of behaviors*



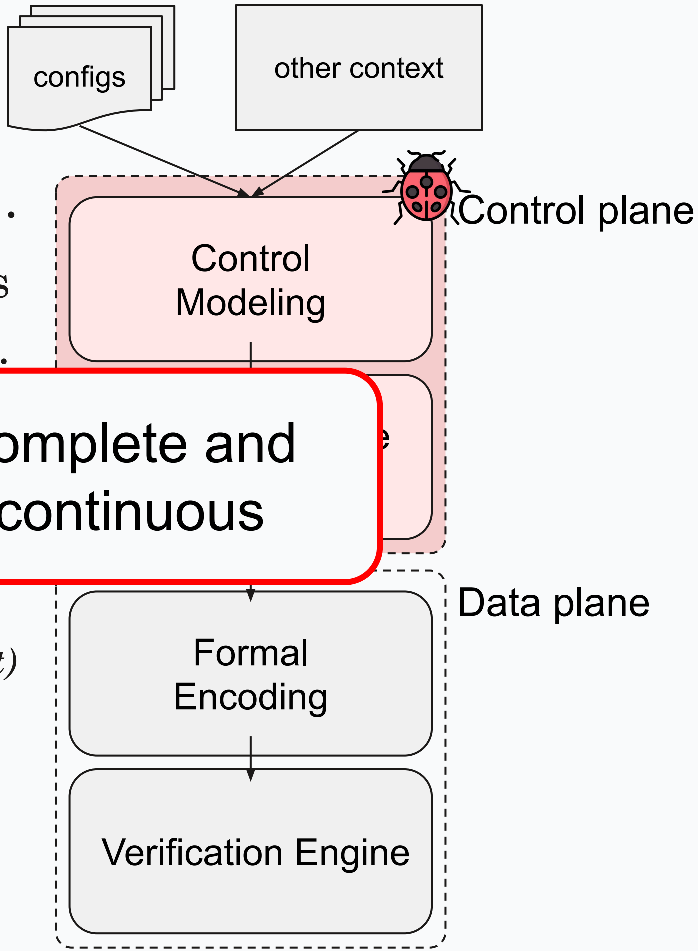
Control modeling is *difficult*

Control plane model created by Batfish engineers…

…but staying on top of all features and protocols used in practice is impossible.

- **Correc**
  - *Custo*
  - *Misses*
- **Coverage**
  - *Completeness: model lags reality (e.g.: no RSVP-TE support)*
  - *Maintainability: requires continuous regression/testing*
    - *…as vendor implementation evolve*
- **Limited Fidelity**
  - *Convergence and protocols are modeled as a partial subset of behaviors*

Developing and maintaining a complete and accurate model is *difficult* and continuous



Control model accuracy hinders adoption

Survey results match our experience…

- **74%** reported one of main barriers to adoption of verification tooling is verification tooling *not supporting features or protocols* that they use
- **94%** reported managing *multi-vendor networks*, which platform-independent control plane models do not capture

How do we keep powerful guarantees of verification without our models hindering adoption?

Model-free verification; opportunity of emulation

High level idea: replace model with emulation to be "*model-free*."

Why now?

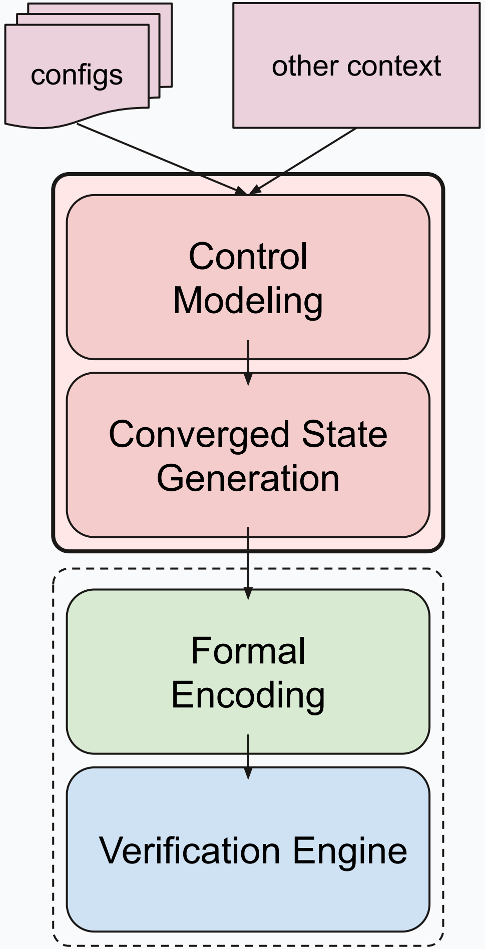
1) Vendors now all provide *containerized* router OS
   a) Resource efficient: as low as 0.5 vCPUs and 1GB of RAM per router.
   b) Can fully emulate 60+ router network on single e2-standard Google Cloud machine.
2) Availability of emulation frameworks to orchestrate virtual device networks
   a) Kubernetes Network Emulator (KNE), Containerlab establish "connections" b/w interfaces
   b) Highly scalable: we successfully run 1,000 router emulated network to convergence

Enables *near perfect-fidelity* control plane emulation at scale.

Model-free verification

We built a prototype using open source components:

- *Batfish* for verification engine and interface
- *Kubernetes Network Emulator* for emulation



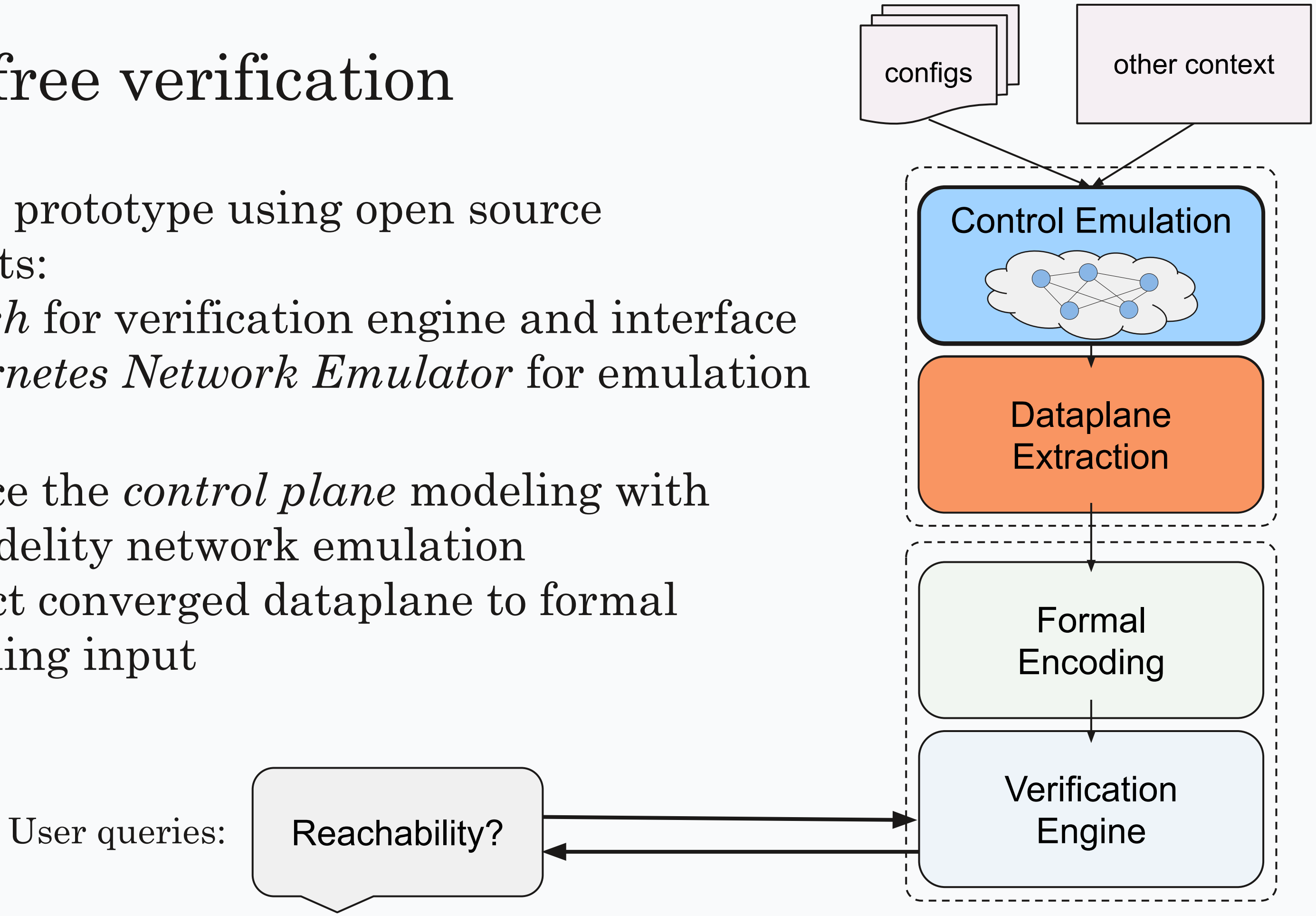
Model-free verification

We built a prototype using open source components:

- *Batfish* for verification engine and interface
- *Kubernetes Network Emulator* for emulation

(1) replace the *control plane* modeling with full-fidelity network emulation

(2) extract converged dataplane to formal modeling input



Early results

Feature testing: verification on a set of simplified configs with production features enabled.

- Ran verification on configs previously unparsable by Batfish
- Successfully detected reachability
- Uncovered bug in Batfish model

Scale testing: testing convergence of large or realistic emulated networks.

- Successfully emulated 1000 router network to convergence.
- ~30 routers with full prod configs, injecting prod-recorded routes (million from each BGP peer)
    - Convergence time including route injection approximately 3 minutes, faster with fewer routes.

See paper for more details.

Many open questions remain…

- Search across scenario space (e.g. up to k cuts) is a challenge
- Exhaustive search of non-deterministic behavior too costly

Key takeaways:

(1) For verification research to be practical, it must have production coverage of features

(2) Open source emulation technology is key to enabling this, and has matured to allow for scale

Alexander Krentsel – akrentsel@google.com

Paper Link