Predicting Network Futures with Plankton

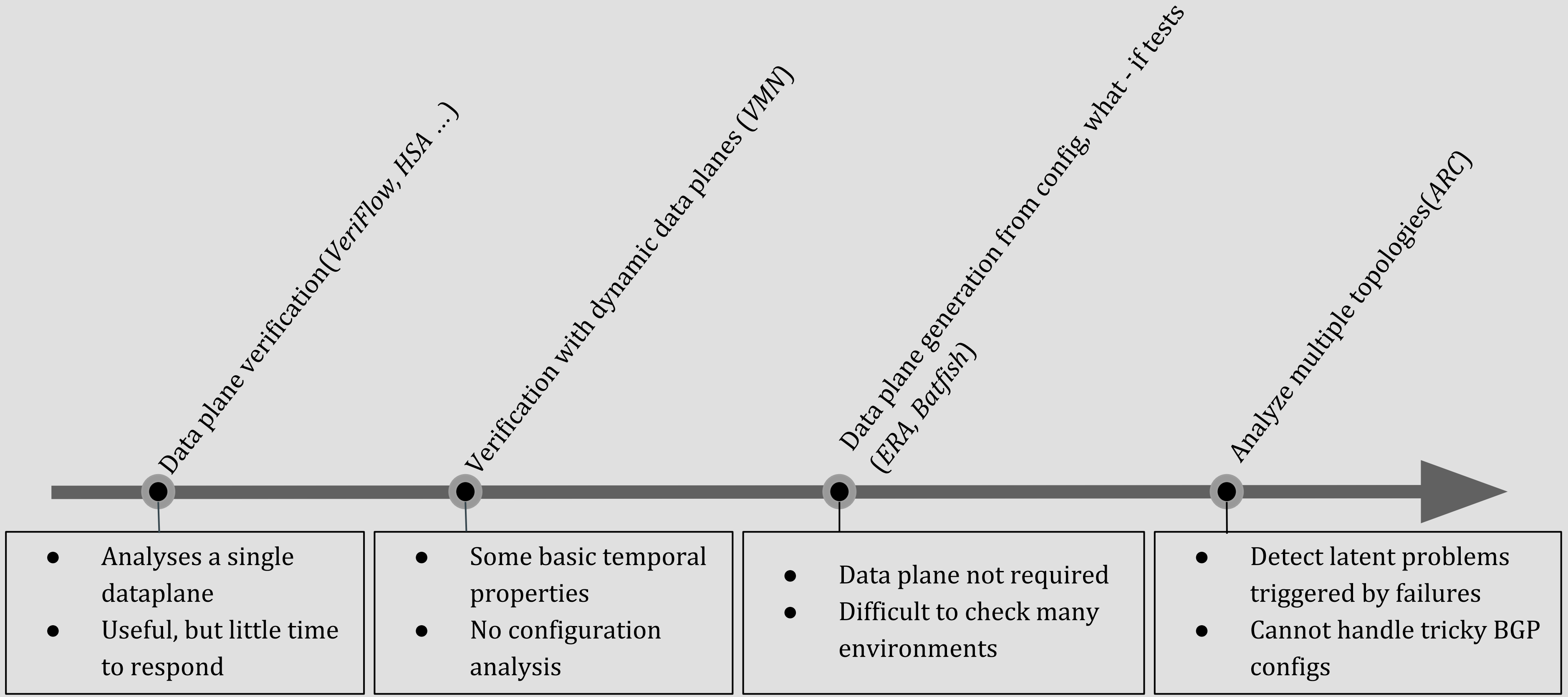
Santhosh Prabhu, Ali Kheradmand, Brighten Godfrey, Matthew Caesar
University of Illinois at Urbana-Champaign

Networks are alive!

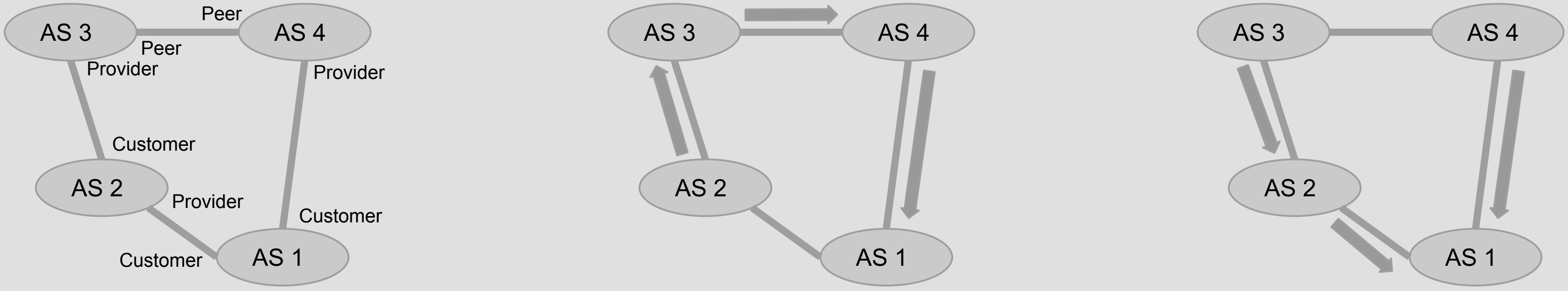
- Responding to external events
- Dynamic data plane elements
- Non-determinism
  - Protocols such as BGP
  - Inter-protocol interactions
  - Environment (failures etc)
- Correctness is more than just reachability
  - Protocol convergence
  - Temporal behavior

    “Traffic can hit any IDS, but always the same one”

Formal Network Verification - The state of the art



BGP Wedgies - A case study



Relationships

Ideal Convergence

Non-Ideal Convergence

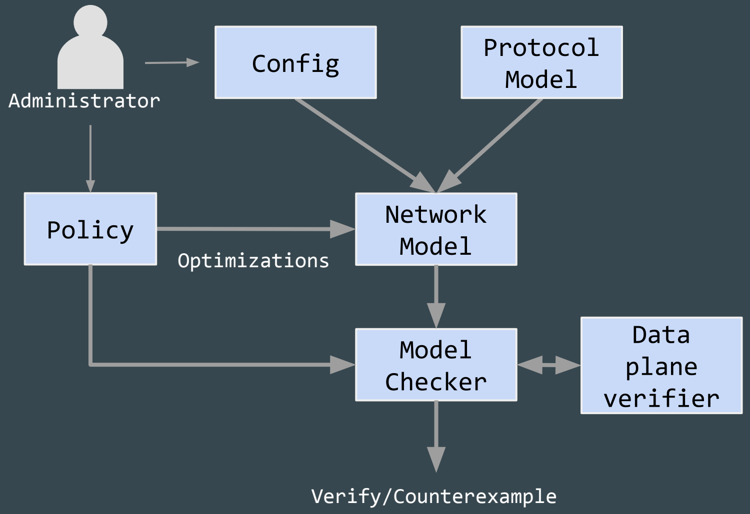
- Data plane analysis can detect the problem only after it occurs
- Topology in both cases identical, so today's configuration analysis tools cannot predict the violation
- Requires the verification platform to model failures, non-determinism etc

Plankton - verify the network system

- First verification platform capable of analysing non-deterministic evolutionary paths of the network.
- Verify not only reachability properties but also temporal properties including protocol convergence.
- Performs exhaustive exploration of the control plane, including external events. Uses a dataplane verifier as an oracle.
- Successfully found BGP wedgies, non-convergence, non-deterministic reachability violations etc.

Design Overview

- Per - Equivalence Class modeling
- Model the control plane and the environment as a non-deterministic finite state program
- Explicit-state model checker to explore the network program
- Data plane verifier to evaluate predicates over the data plane states generated
- Specify temporal properties in the model checker over these predicates



Design

Single Equivalence Class Modeling

Packet headers that have identical behavior

In Plankton: Identical behavior in hypothetical scenarios

Computed from config. Headers in the same EC has identical configuration throughout the network

Single EC model = Limited set of policies. Eg: *"Packets A and B behave identically"* cannot be checked

Explicit State Model Checking

Generates and verifies each state separately.

Capable of verifying temporal policies, including liveness

Allows use of a dataplane verifier in the loop (Generates one dataplane at a time)

Uses efficient branching, and optimizations like Partial Order Reduction, Bitstate Hashing etc

Design

Network Model

Protocols defined based on RFC standards

Environment: Failure, Reconnection, Packet Arrival…

State transition: Environment action or the processing of one routing update at any device

Written in a language that captures non-determinism

Data plane verifier

Checks predicates over data plane states generated by the Protocol Model

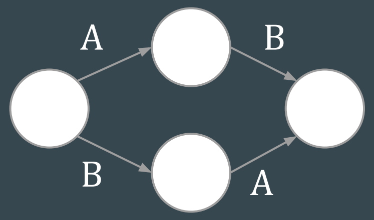
Real-time verification enables this use-case

Currently only one data plane snapshot in an invocation

Maintains its own state, including equivalence classes etc.

Optimizations

Partial Order Reduction



Need to verify only
A → B!

Plankton does PoR separate from the Model Checker

Computed based on a dependency graph that captures the influence of events on each other

Currently uses ad-hoc, per-policy reductions. More comprehensive mechanism being worked on

Cone-of-Influence Reduction

Explore only those events that can cause policy violation

Eg: In OSPF, explore only failures on the shortest path

Defined based on the protocols and the policy being checked

Conservative reductions for each policy

Prototype Implementation

- BGP and OSPF
- Promela Modeling Language
- SPIN Model Checker
- VeriFlow Dataplane Verifier

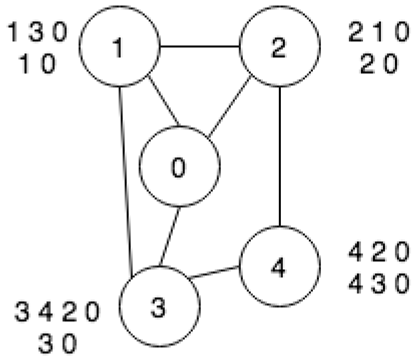
```
inline runProtocols()
{
    d_step {
        needsExecution[PT_BGP]=true;
        needsExecution[PT_OSPF]=true;
    }
    do
        :: needsExecution[PT_BGP] ->
           bgp();


        :: needsExecution[PT_OSPF] ->
           ospf();

        :: else->break;
    od

progress:

    c_code {
        Pinit->assertion=assertionCheck();
    }

    assert(assertion);

}
```

Evaluation

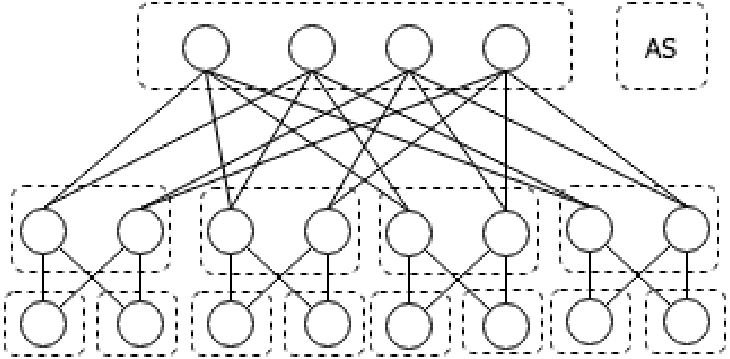
Correctness

- BGP convergence in known networks
- Wedgies - Violations due to failures/race conditions
- Device sequencing in data centers

Correct results every time, but not always as expected!



BAD GADGET: Non-converging BGP config

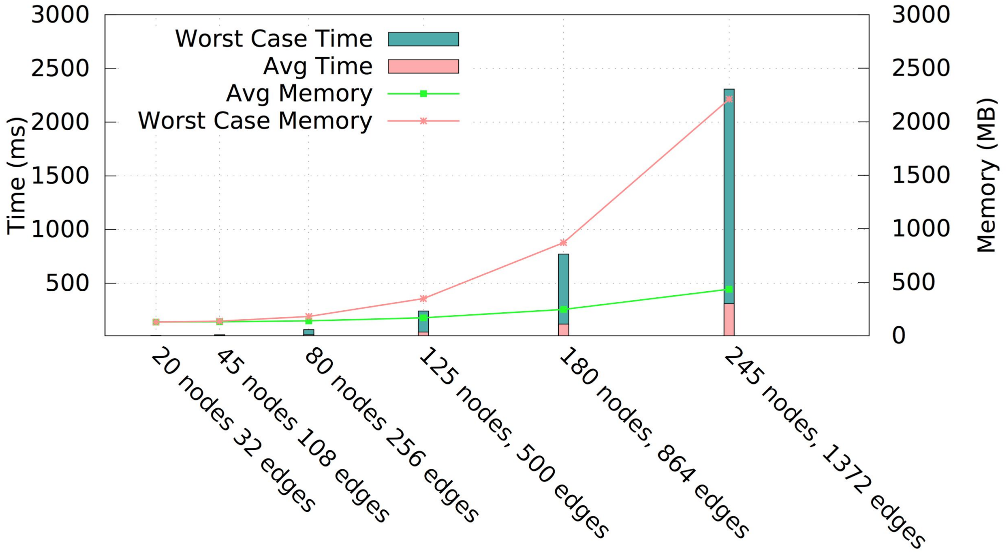


BGP on a Fat Tree

Evaluation

Scalability

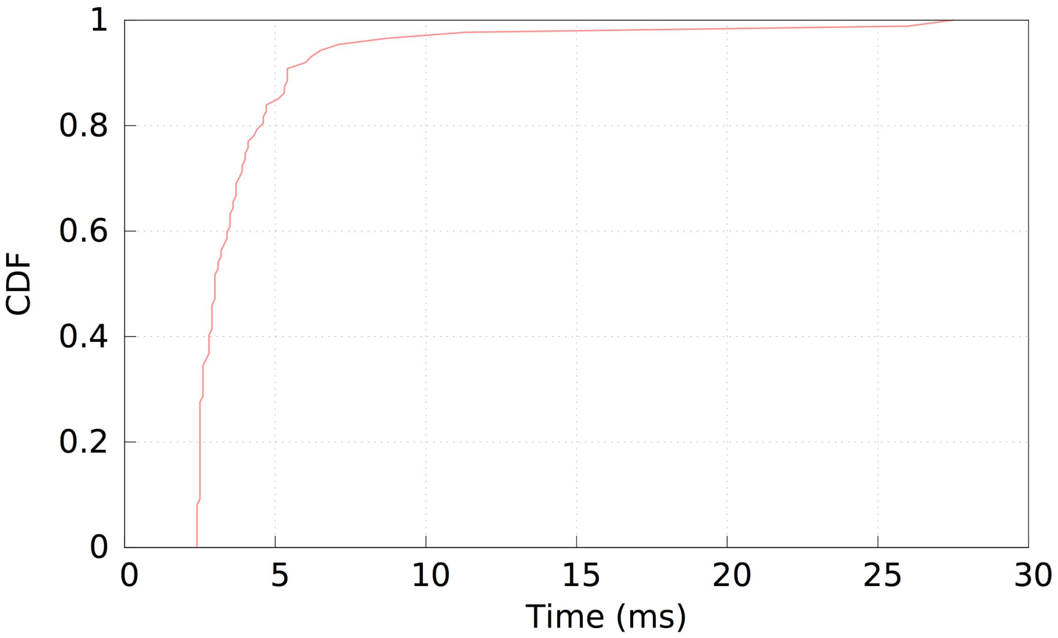
- Data centers running BGP
- Device sequencing policy
- Time/memory taken by the search to find a violation



Evaluation

Scalability

- Real-world BGP relationships (CAIDA)
- Time to check wedgies for one AS



Bitstate Hashing

Use a bloom filter to track explored states
(0.99 ≤ coverage ≤ 1.0)

Experiment	Without bitstate hashing	With bitstate hashing
125 Node DC (Worst Case)	347.5 MB	35.4 MB
180 Node DC (Worst Case)	870.3 MB	69 MB
245 Node DC (Worst Case)	2211.2 MB	121.1 MB
CAIDA Wedgie (Avg Case)	135.6 MB	23.6 MB

Effect of Bitstate Hashing on Memory Overhead

Summary and Future Work

1. Explicit state exploration with real-time data plane verification to verify temporal and reachability policies
2. Captures violations due to evolution of the network
3. Scalable to networks the size of real-world data centers
4. Ongoing work on better methods for Partial Order Reduction, Cone of Influence Reduction etc
5. Switch to symbolic exploration - Need dataplane verifiers that operate on multiple dataplane states simultaneously
6. Other techniques to improve scalability - heuristic search, iterative deepening etc

Thank you!

Questions?