论文研讨 - Lightyear: Using Modularity to Scale BGP Control Plane Verification

Alan Tang [1], Ryan Beckett [2], Steven Benaloh [2], Karthick Jayaraman [2], Tejas Patil [2], Todd Millstein [1], George Varghese [1],

[1] UCLA, [2] Microsoft

Pre-Background...

Properties, repersented by **propositions and predicates**, are syntatical objects.

$$\forall r, s \in \mathbb{Q}, r = s \vee \exists t \in \mathbb{Q}, r < t < s$$

Pre-Background...

Properties, repersented by **propositions and predicates**, are syntatical objects.

$$\text{Precondition (Assumptions)} \Rightarrow \text{Postconditions (Assetions)}$$

Pre-Background...

Properties, repersented by **propositions and predicates**, are syntatical objects.

$$\text{Precondition (Assumptions)} \Rightarrow \text{Postconditions (Assetions)}$$

Two way of reasoning about their correctness:

- Formal deduction
-

Pre-Background...

Properties, repersented by **propositions and predicates**, are syntatical objects.

$$\text{Precondition (Assumptions)} \Rightarrow \text{Postconditions (Assetions)}$$

Two way of reasoning about their correctness:

- Formal deduction
- **Model checking**

Background

A lot of desirable properties are **global**:

- When **all nodes** works, no incorrect routes are advertised.
- When **all edges** works, **all nodes** acknowledge correct routes.

Background

A lot of desirable properties are **global**:

- When **all nodes** works, no incorrect routes are advertised.
- When **all edges** works, **all nodes** acknowledge correct routes.

Checking global properties **globally** is HARD.

- Symbolically: Amount of states grows exponentially, amount of checks grows (at least) quadratically.
- Runtime: Synchronization is costly / hard to do right.

It’s much easier to do **local checks**.

It’s much easier to do **local checks**.

LIGHTYEAR

It’s much easier to do **local checks**.

LIGHTYEAR

Given:

- Configuration (Topology and policies)
- Local invariants
- Desirable global properties

Generates:

- Local checks

Paper proves that local constraints imply global properties

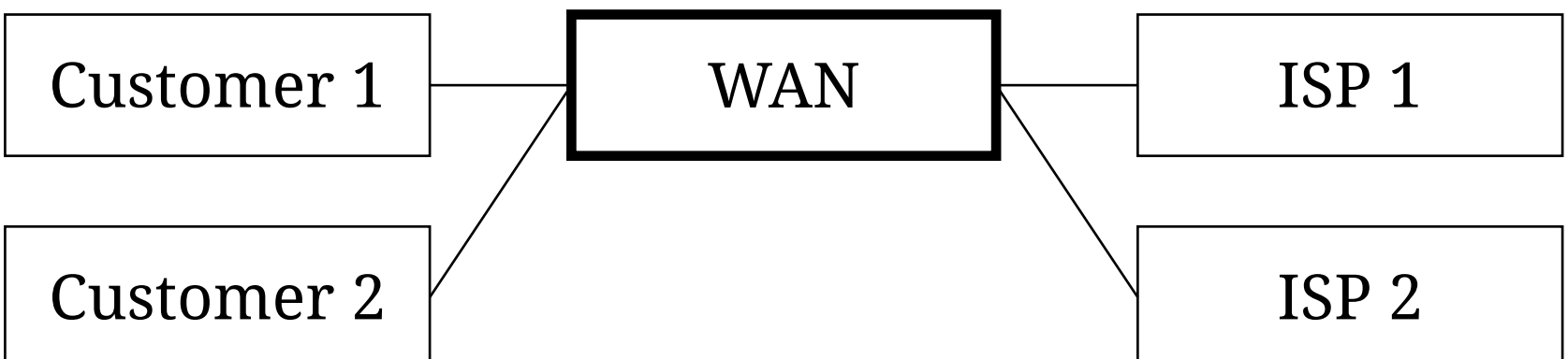
Scope of LIGHTYEAR

Internal WAN (with BGP support).

Scope of LIGHTYEAR

Internal WAN (with BGP support).

Cloud provider: Vultr, Azure, ...



The language of LIGHTYEAR

Basically the quantifier-free fragment of FOL, specifing behaviors at **locations** (nodes or edges).

The language of LIGHTYEAR

Basically the quantifier-free fragment of FOL, specifing behaviors at **locations** (nodes or edges).

E.g. At inner node N:

$$\mathrm{addr}(r) \in \mathrm{dead{:}beef{::}/64}$$

The language of LIGHTYEAR

Basically the quantifier-free fragment of FOL, specifing behaviors at **locations** (nodes or edges).

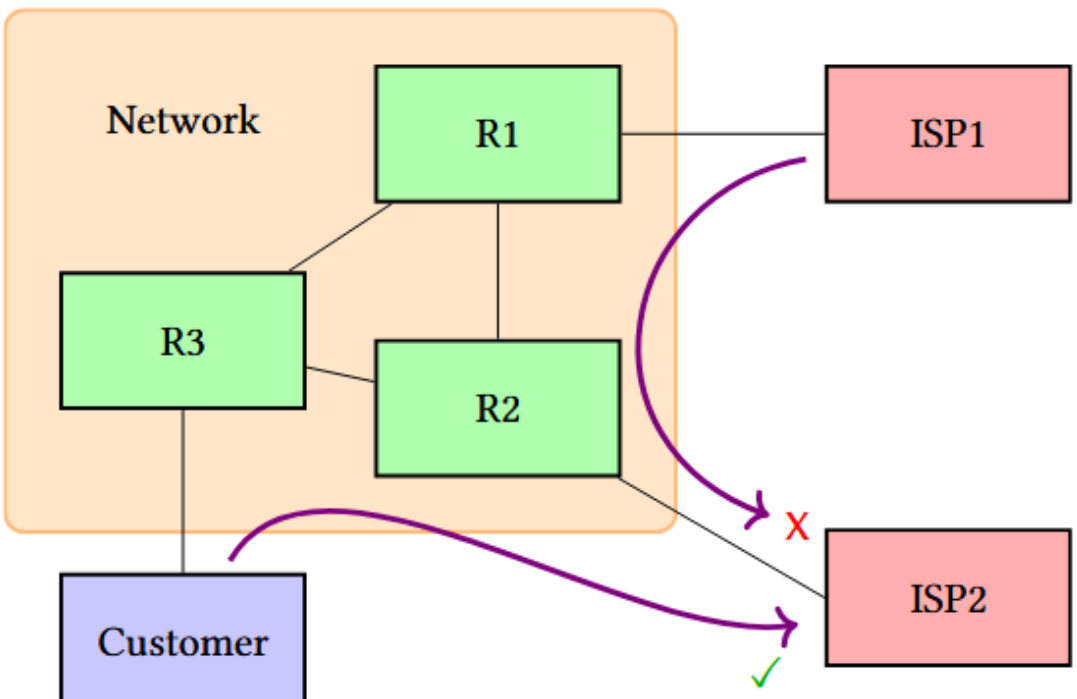
E.g. At inner node N:

$$\mathrm{addr}(r) \in \mathrm{dead{:}beef{::}/64}$$

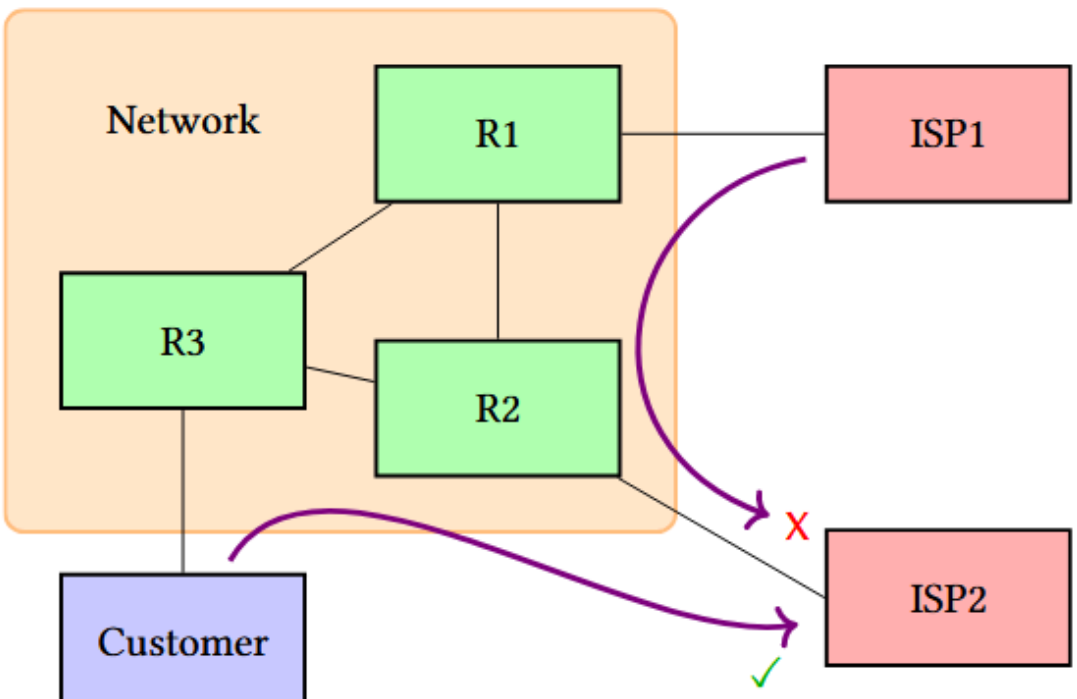
Generated functions: `Import, Export, Originates`

- `Import:` $\texttt{Edge} \times \texttt{Route} \to \texttt{Route} \cup \{\texttt{Reject}\}$

An example...



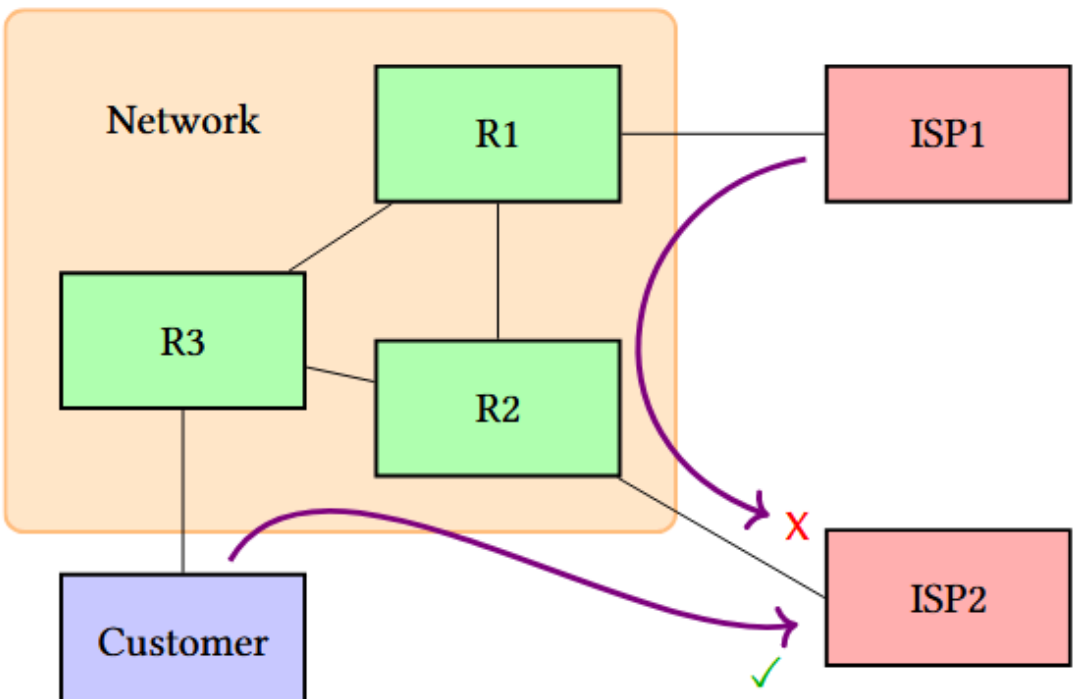
An example...



Safety:

No routes from ISP1 reaches ISP2

An example...



Safety:

No routes from ISP1 reaches ISP2

Liveness

Routes from customer reaches ISP2

Limits of LIGHTYEAR

Traditionally, predicates are **indexed by time**, or reasoning is done in **temporal logic**.

Limits of LIGHTYEAR

Traditionally, predicates are **indexed by time**, or reasoning is done in **temporal logic**.

LIGHTYEAR does not use temporal information. We only care about:

- Safety: No bogus routes
- Liveness: Valid routes will be broadcasted

$$\Diamond\Diamond A \rightarrow \Diamond A$$

Limits of LIGHTYEAR

Traditionally, predicates are **indexed by time**, or reasoning is done in **temporal logic**.

LIGHTYEAR does not use temporal information. We only care about:

- Safety: No bogus routes
- Liveness: Valid routes will be broadcasted

$$\Diamond\Diamond A \rightarrow \Diamond A$$

Topology and policies are fixed, and for liveness checks, the path needs to be provided.

Proof? How?

Proof? How?

SMT

Proof? How?

SMT

For all location l, generates:

$$\left(\bigwedge \mathrm{inv}(l)\right) \wedge (r' = \mathrm{Import}(l, r)) \rightarrow \mathrm{inv}(l)[r/r']$$

Real world deployments

Hundres of routers, tens of thousands of peering sessions.

Implemented in C#, using Zen as SMT library.

Real world deployments

Hundres of routers, tens of thousands of peering sessions.

Implemented in C#, using Zen as SMT library.

11 configuration bugs. Each run takes at most 15 mins.

Conclusion

LIGHTYEAR is a **modular** way to verify global properties of BGP control plane.

- Guaranteed correctness for safety and liveness.
- Constraints are simple to write and understand.
- Implemented in C# which can be directly integrated into existing systems.

Thank you!