Modular Verification and Repair for Networks

Dexin Zhang

Network failure is a big problem

CBC

Human error caused 2022 Rogers outage, system 'deficiencies' made it worse: report

The Rogers outage lasted 26 hours, cut 12M people off from 911 and other network services

Peter Zimonjic · CBC News · Posted: Jul 05, 2024 2:51 PM EDT | Last Updated: July 6, 2024

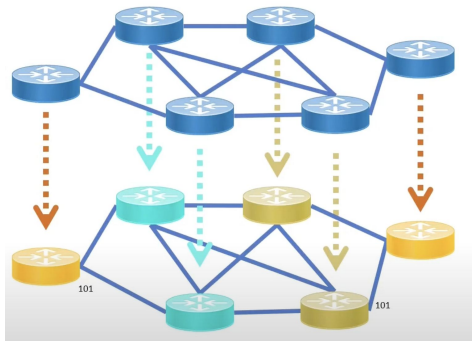
Optus outage causes chaos in Australia before services restored

By **Renju Jose** and **Byron Kaye**

November 8, 2023 1:50 AM EST · Updated a year ago

Formal methods in network analysis

- Data plane verification
  - Tools: Veriflow, RCDC, Atomic Predicates…
- Control plane verification
  - Tools: Bagpipe, Minesweeper, Lightyear, Timepiece…
- Configuration synthesis & repair
  - Tools: Propane/AT, Avenir, CPR, Hoyan…



control plane

data plane

⇦ my work

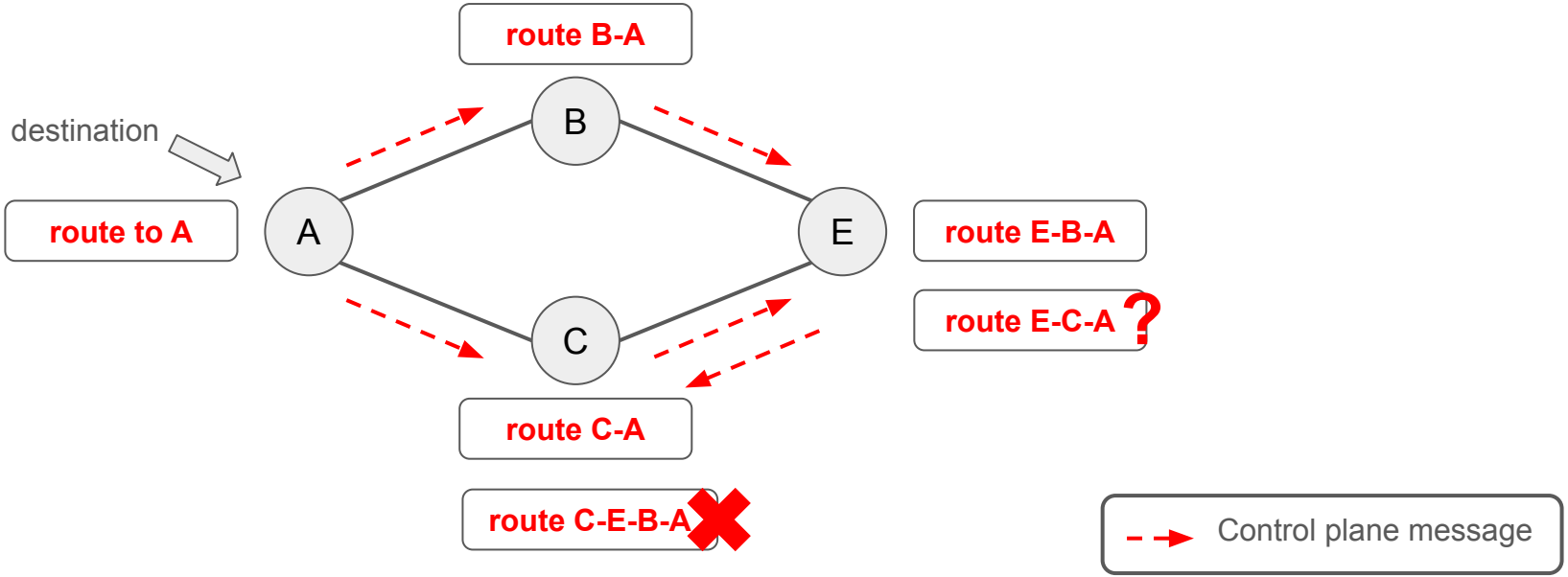
⇦ my work

Overview

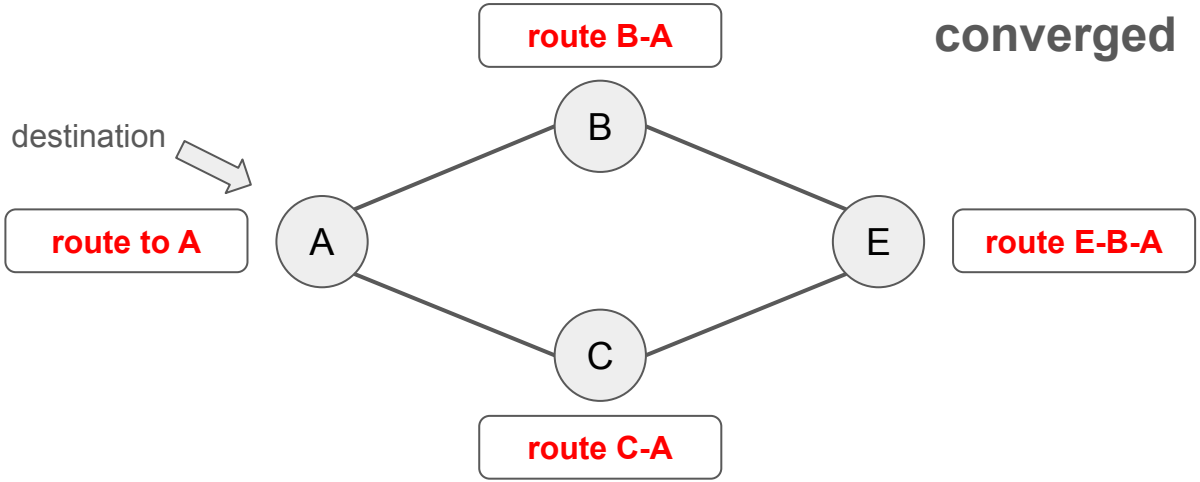
- First, I will present CBGraphs, a new network verification approach that
  - Utilize modularity to achieve scalability
  - Improve the usability
- Second, I will present a approach of network repair, as an application of CBGraphs

Part I: modular verification of network

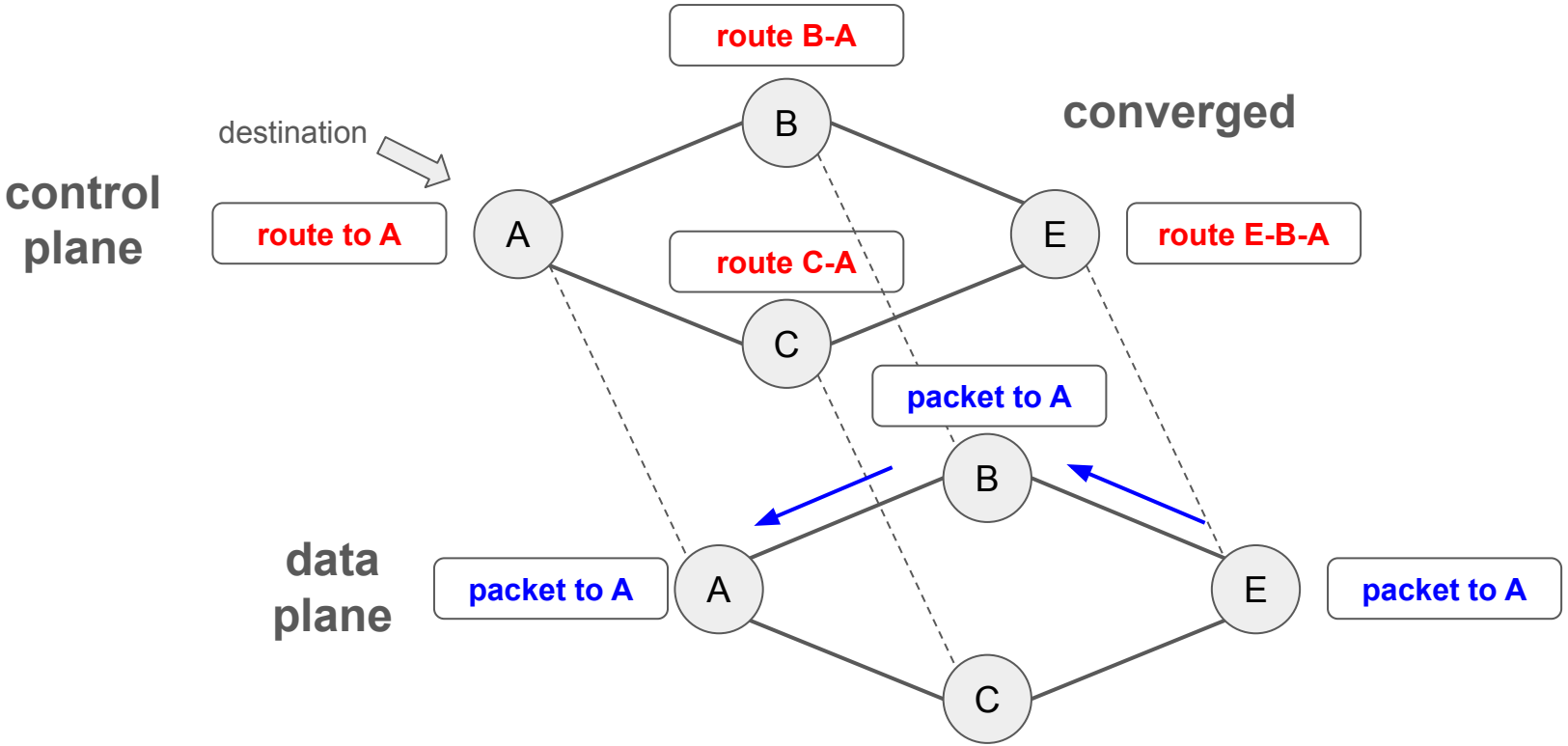
A quick look at routing model



A quick look at routing model



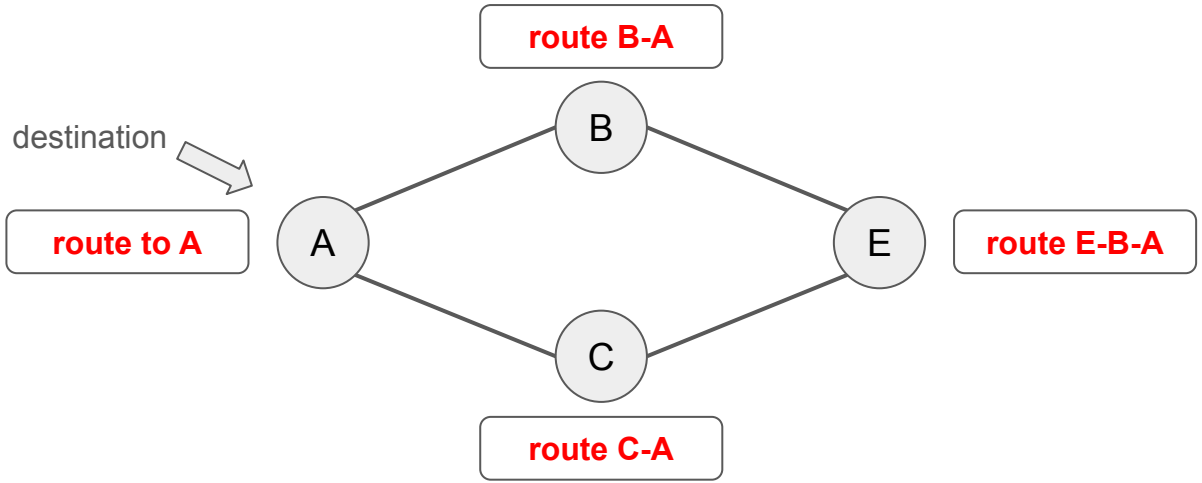
A quick look at routing model



A quick look at routing model

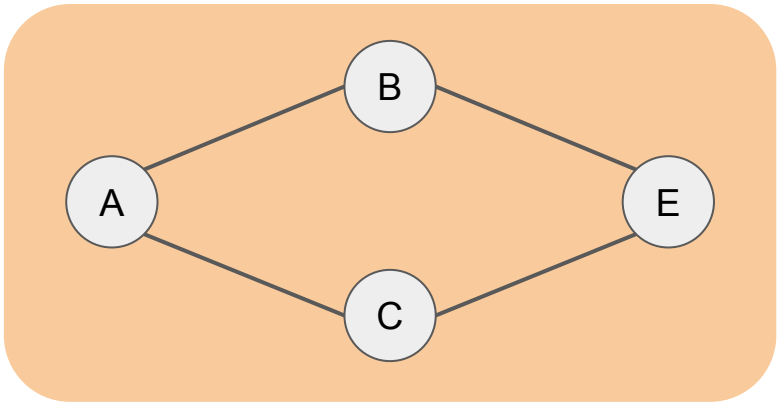
Properties to verify:

- E can reach A (after converged)
- B will select the shortest path (after converged)
- E will prefer E-B against of E-C (after converged)
- …



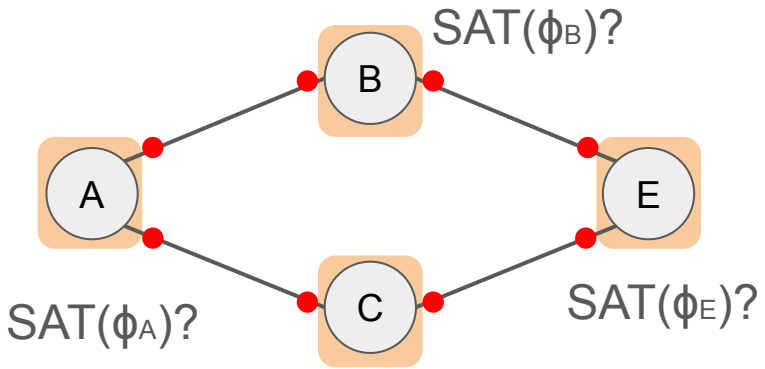
Monolithic vs. modular verification

Monolithic verification

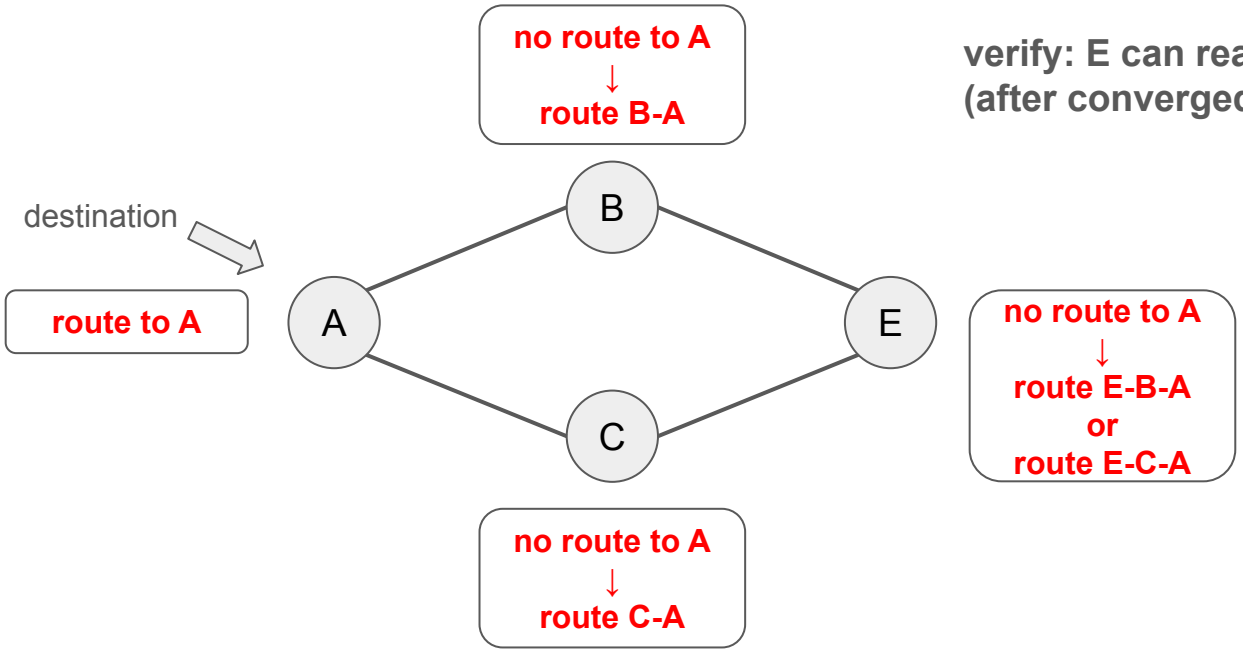


SAT(ϕ)?

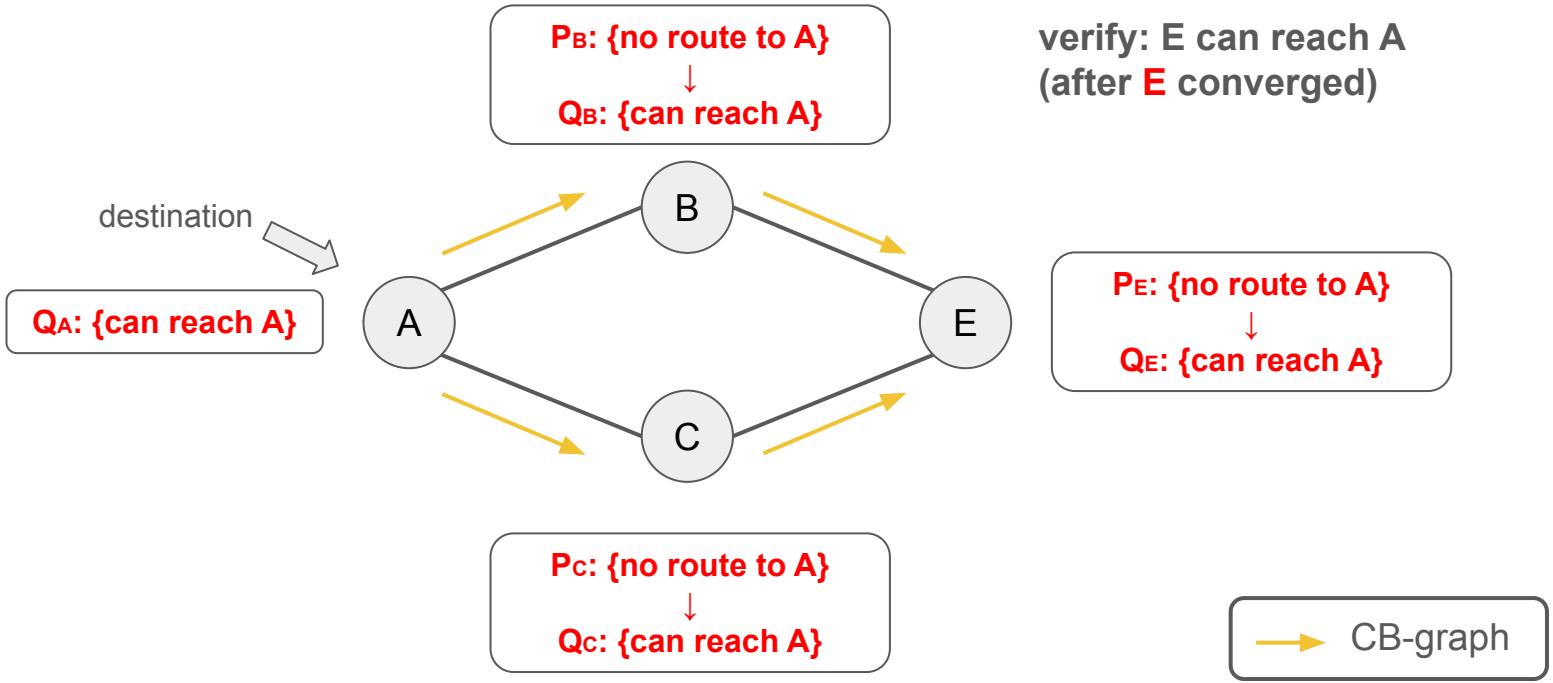
Modular verification



Motivating example 1: verifying reachability

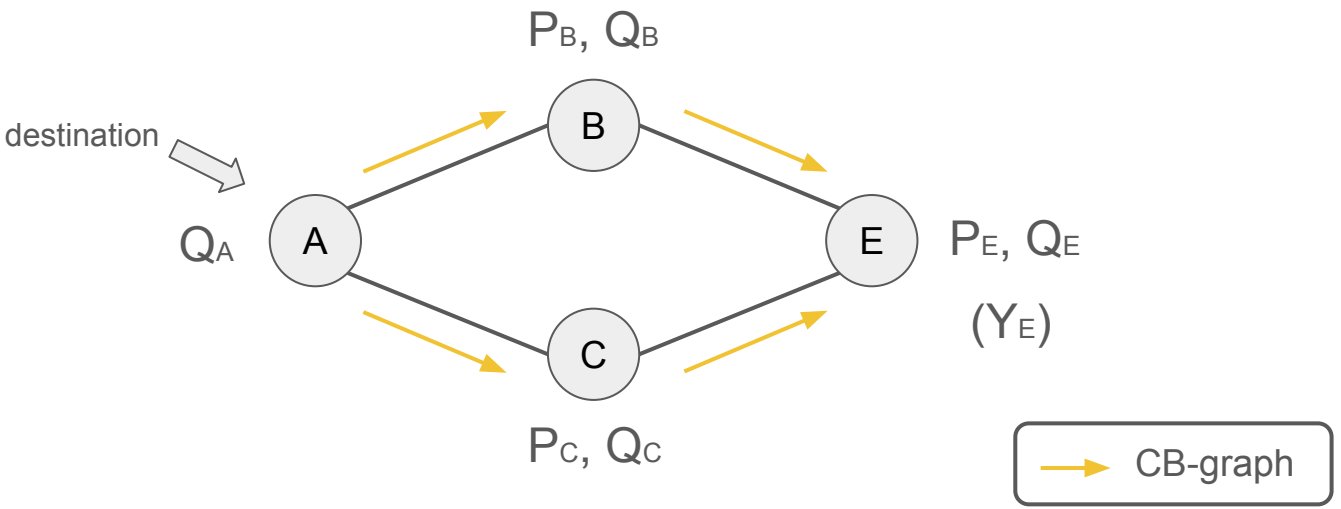


Motivating example 1: verifying reachability

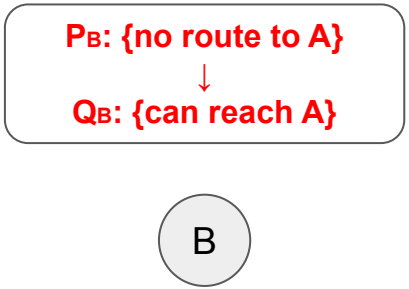


CB-graphs

- A CB-graph (CB for converge before) is an **acyclic** subgraph of the network
- Each node annotated with two interfaces:
  - pre-convergence condition (P), post-convergence condition (Q)
- Additional: property to verify (Y)

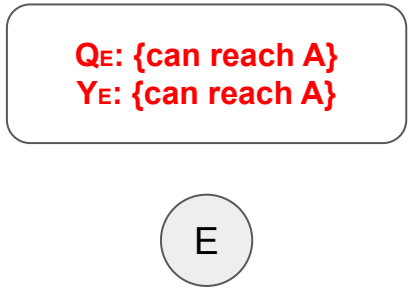


CB-graphs: initial check (1/5)



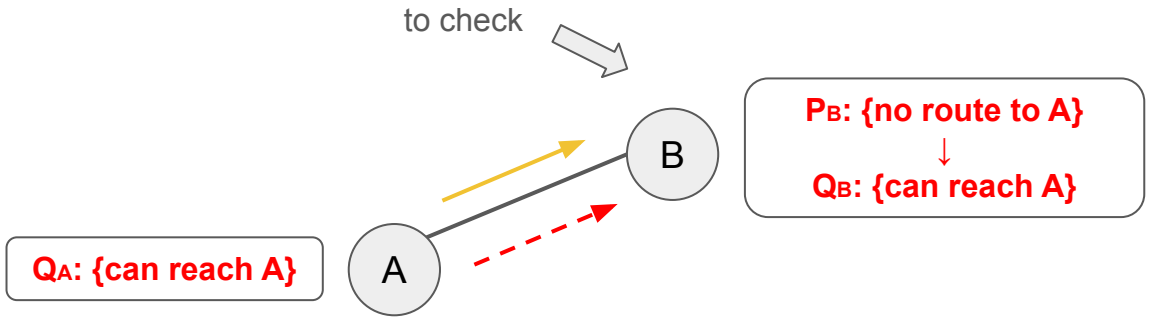
“no route to A” ∈ P_B = {no route to A}

CB-graphs: property check (2/5)



Q_E = {can reach A} $\subseteq$ Y_E = {can reach A}

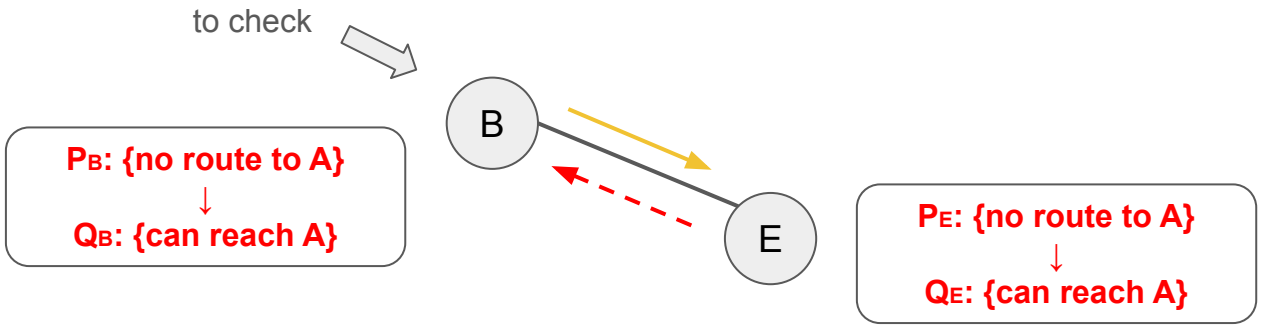
CB-graphs: convergence check (3/5)



$r_A \in Q_A$ = {can reach A} $\wedge$ $r_B \in P_B$ = {no route to A}
$\Rightarrow r_{Bnext} \in Q_B$ = {can reach A}

CB-graph
Control plane message

CB-graphs: post-convergence check (4/5)



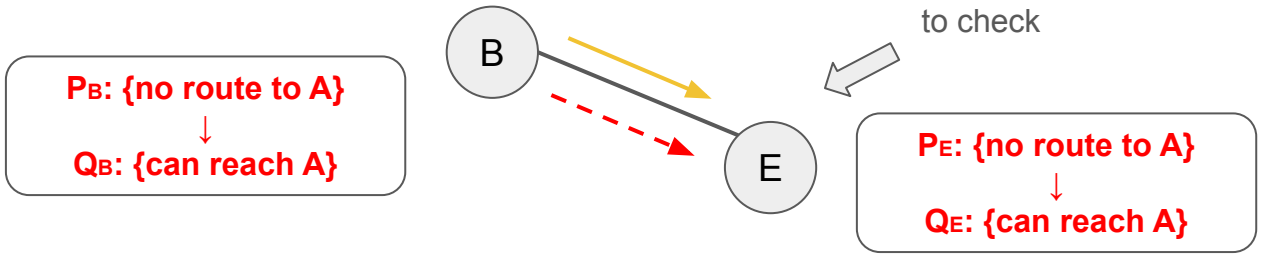
$$r_E \in P_E \cup Q_E \wedge r_B \in Q_B$$
$$\Rightarrow r_{Bnext} \in Q_B$$

CB-graph

Control plane message

CB-graphs: pre-convergence check (5/5)

Case 1: Along CB-edge



$r_B \in P_B \land r_E \in P_E$

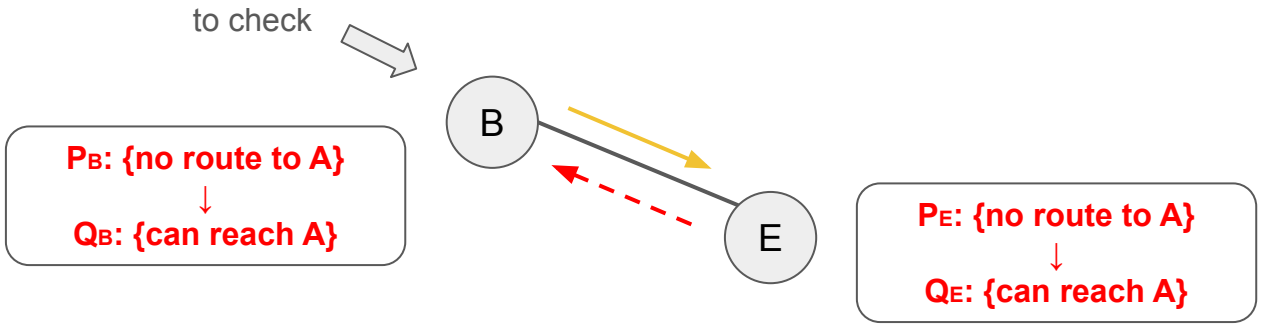
$\Rightarrow r_{Enext} \in P_E$

CB-graph

Control plane message

CB-graphs: pre-convergence check (5/5)

Case 2: Along non-CB-edge



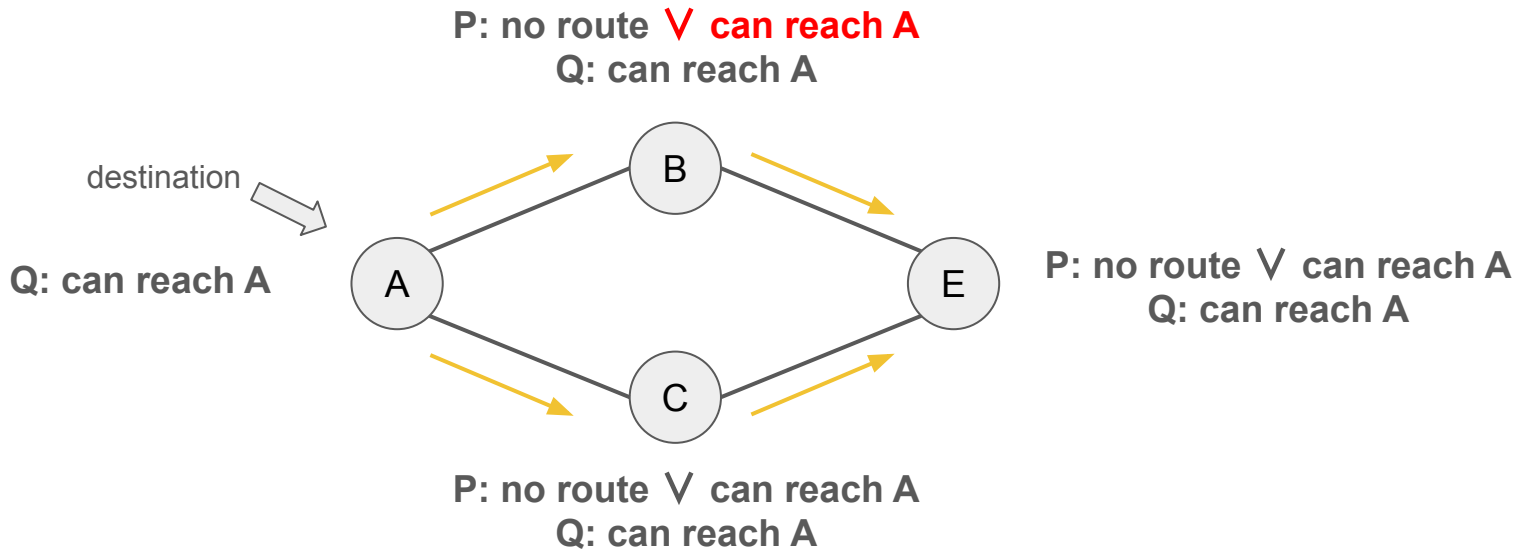
$r_E \in P_E \cup Q_E \land r_B \in P_B$?

$\Rightarrow r_{Bnext} \in P_B$

CB-graph

Control plane message

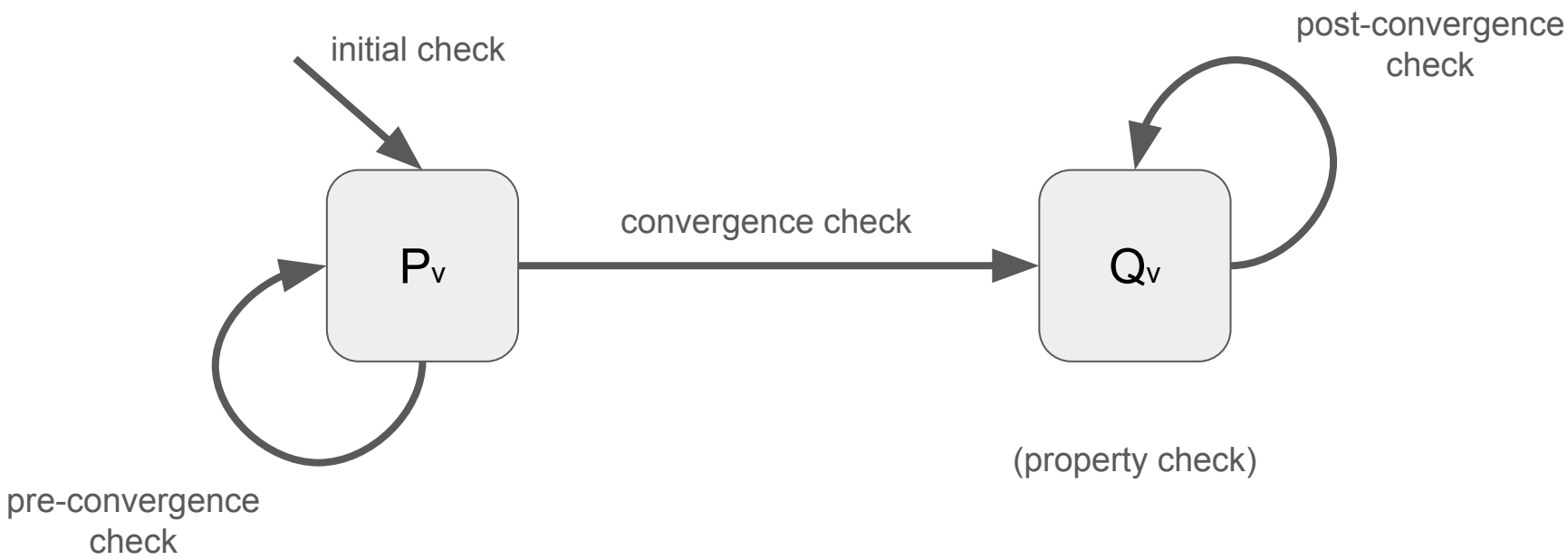
Motivating example 1: verifying reachability



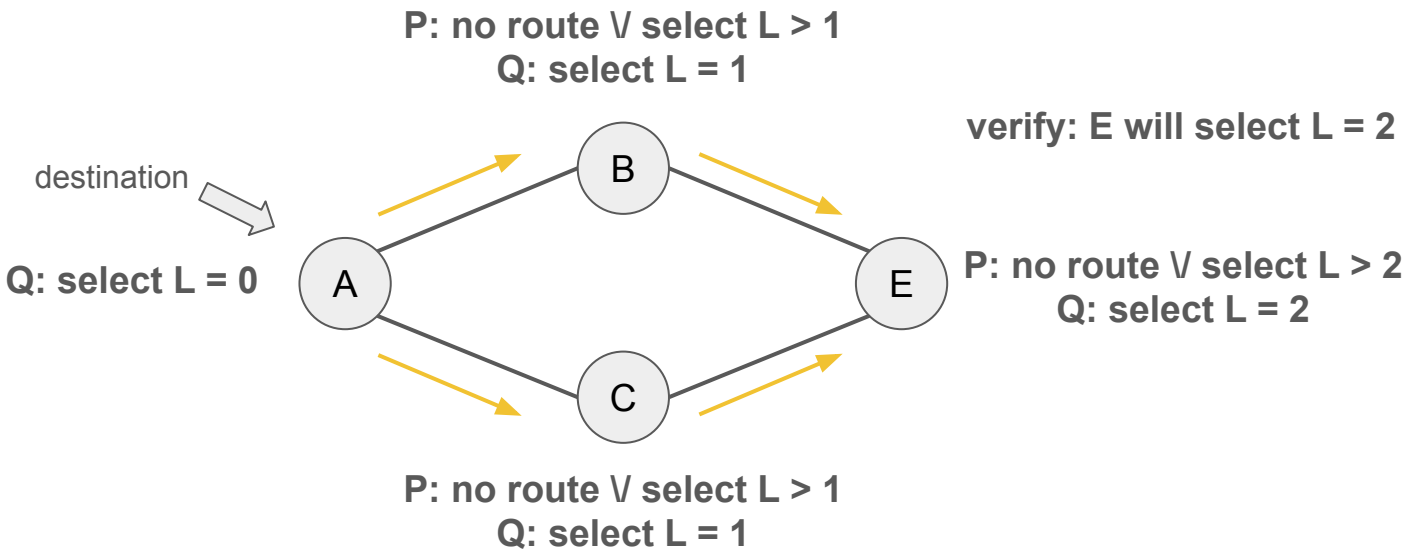
With refined P,Q interfaces, all checks pass.

CB-graph

CB-graphs: summary



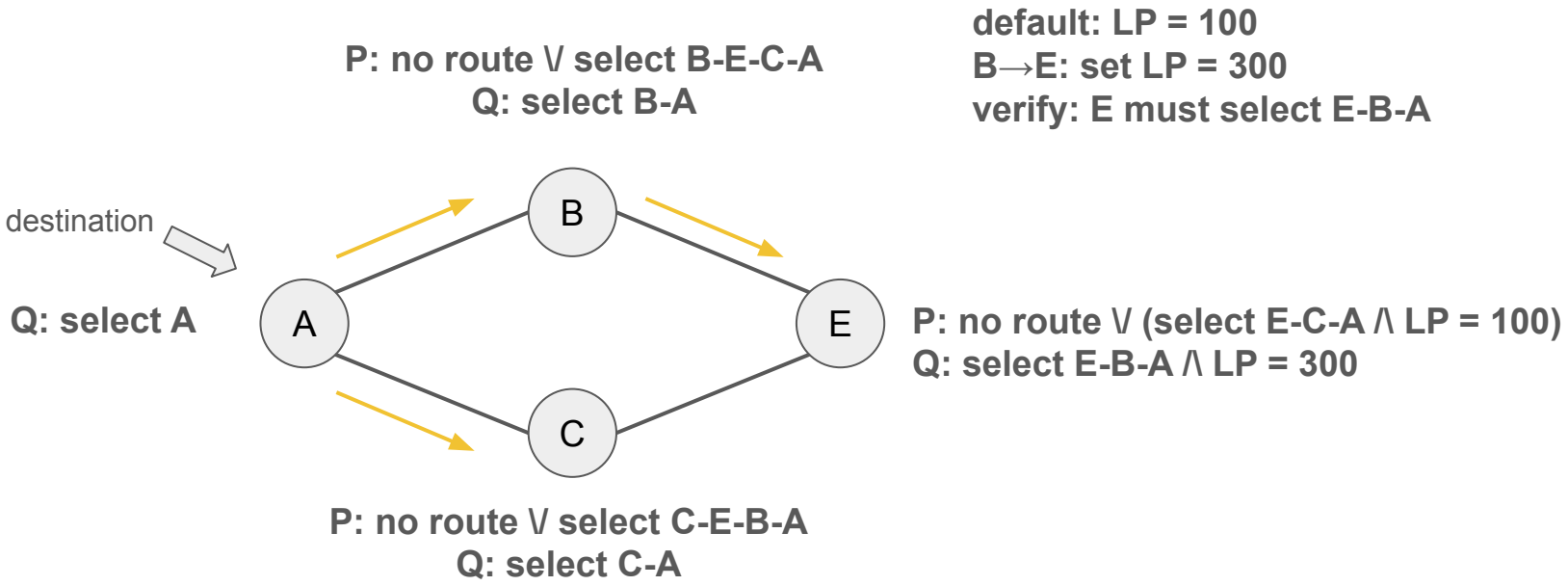
Motivating example 2: verifying shortest path



With the interfaces above, all checks pass.

CB-graph

Motivating example 3



With the interfaces above, all checks pass.

CB-graph

Soundness theorem

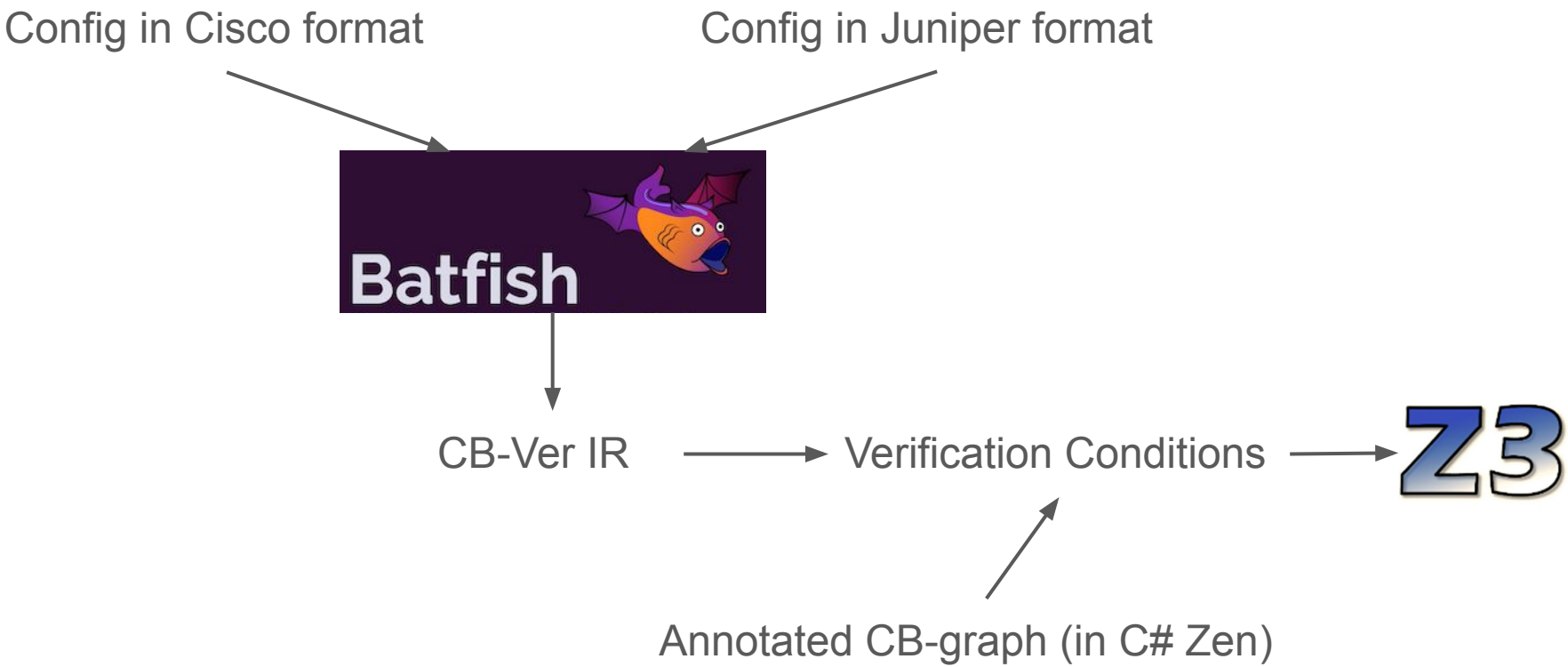
- Asynchronous network semantics: node → (logical) time → selected route
- Soundness theorem: if the verification condition for (CB, P, Q, Y) holds, then the property Y will hold eventually (∃ tau) and stably (∀ t >= tau)

- Formalization in Lean

```
def sem {N : Network} (S : Schedule N) : N.Node → Nat → N.Route
```

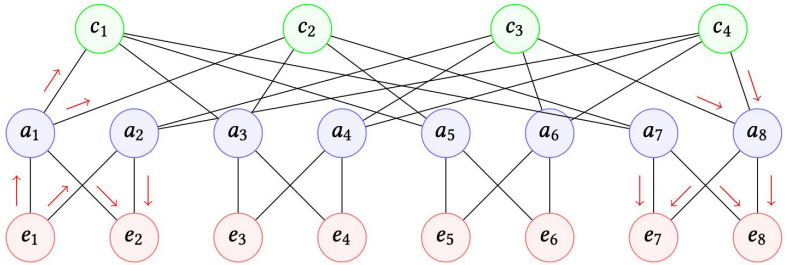
```
theorem soundness {S : Schedule N} {T : Triple N} :
  T.VC → ∀ v, ∃ τ, ∀ t ≥ τ, T.Property v (sem S v t)
```

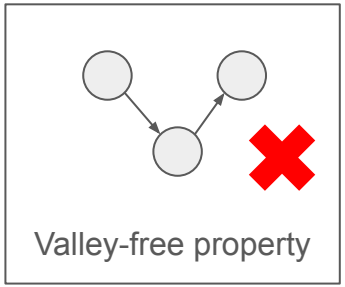
Implementation of CB-Ver

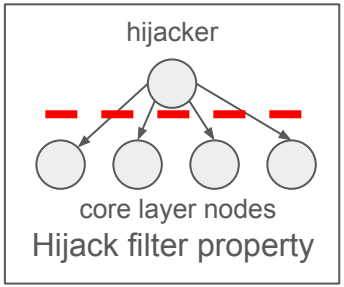


Evaluation

- Synthetic fattrees
  - 20 to 2,000 nodes
  - Four properties
    - reachability
    - shortest path
    - valley-free
    - hijack filter

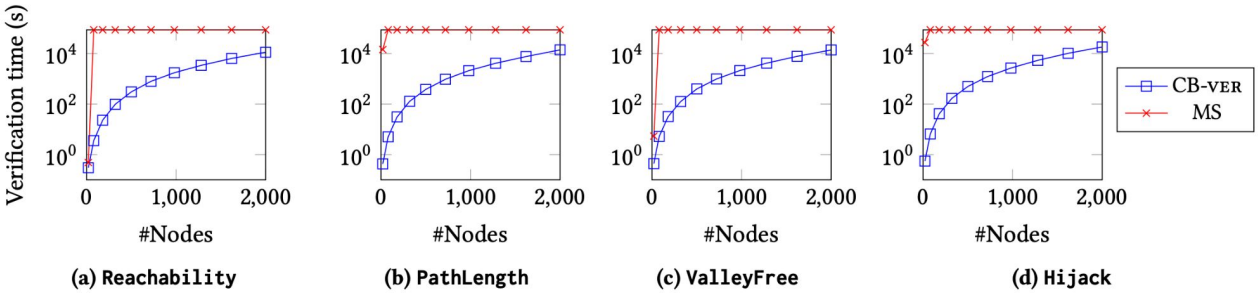






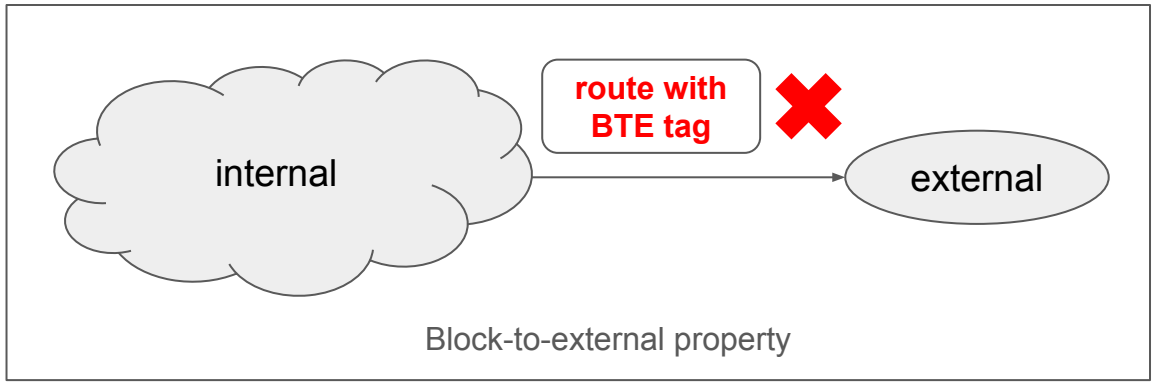
Evaluation

- Synthetic fattrees
  - CB-Ver
    - 22~52 lines of specification, including 10 lines of CB-graphs
    - verifies 2,000 nodes in 7 hours
  - Monolithic approach
    - 10~27 lines of specification
    - can’t verify 80 nodes in 24 hours



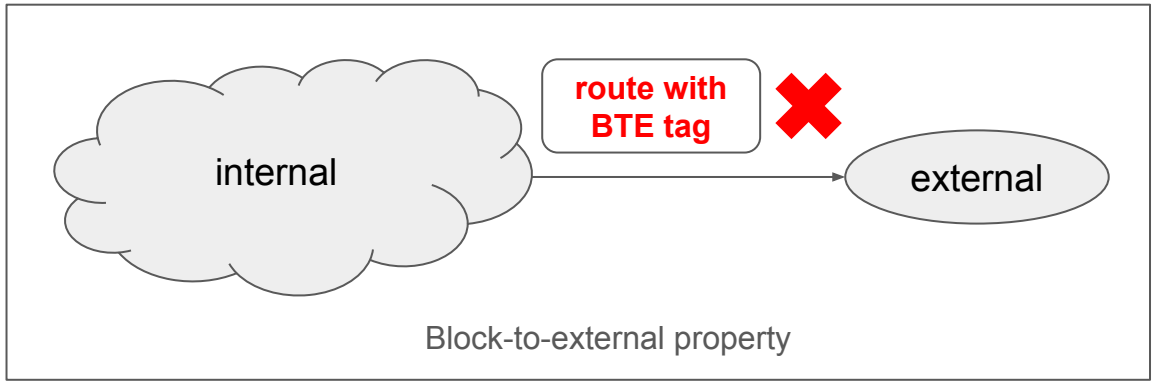
Evaluation on verification

- Internet2
  - 433 nodes (10 internal nodes)
  - 100K lines of configuration in total
  - Property: block to external



Evaluation on verification

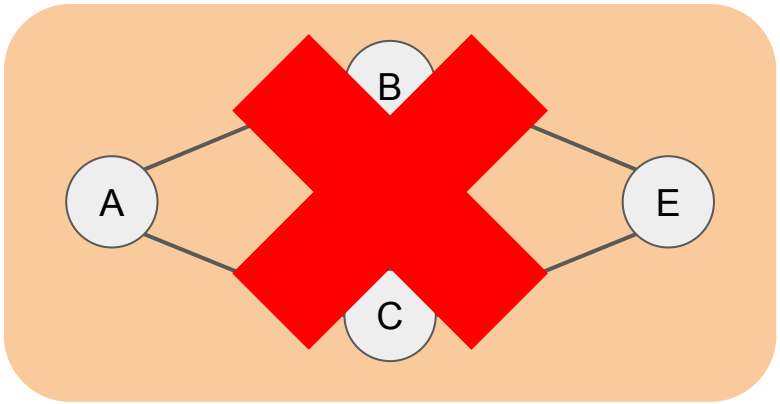
- Internet2
  - CB-Ver
    - 36 lines of specifications
    - takes 2 seconds, finds 3 bugs (reported by others)
  - Monolithic approach
    - 23 lines of interfaces
    - takes ½ hour and finds 1 bug each time



Part II: local repair of network

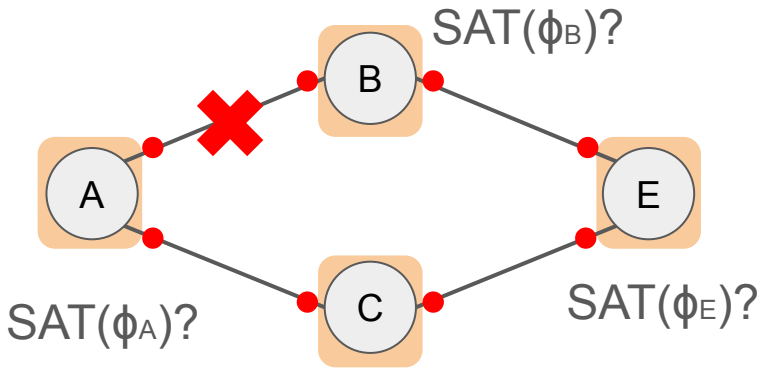
Motivation on modularity

Monolithic verification



SAT(ϕ)?

Modular verification



SAT(ϕ_B)?

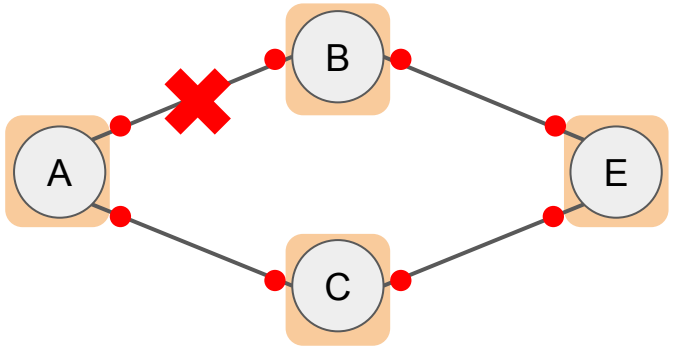
SAT(ϕ_A)?

SAT(ϕ_E)?

SAT(ϕ_C)?

Problem description

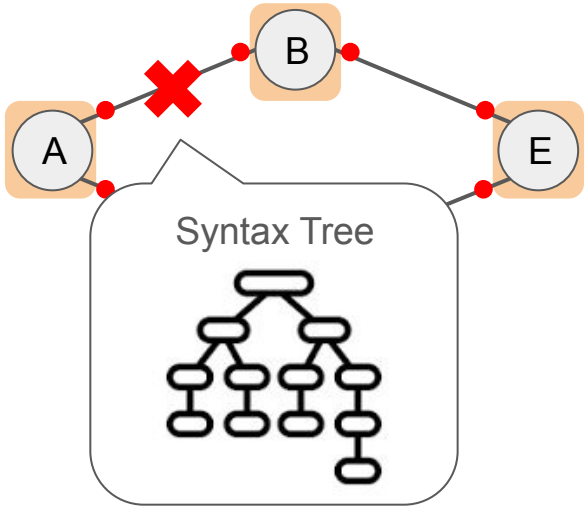
- Motivation: fix errors in configuration
- CB-Ver tells us VC(u,v) failed (error location)



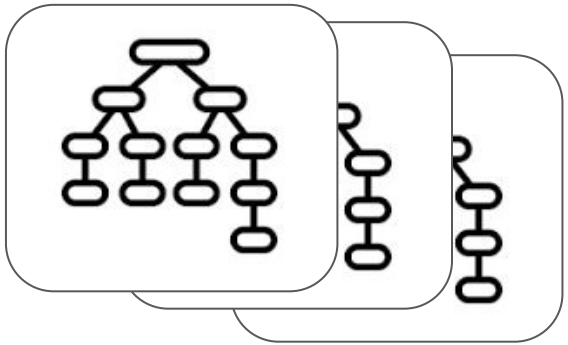
- Approach:
  - Assume interfaces and CB-graph are **correct** and **precise** enough
  - Errors are only in the configuration, i.e., export policy of u and import policy of v
- Goal: repair the configuration at u and v, such that VC(u,v) holds

Overview

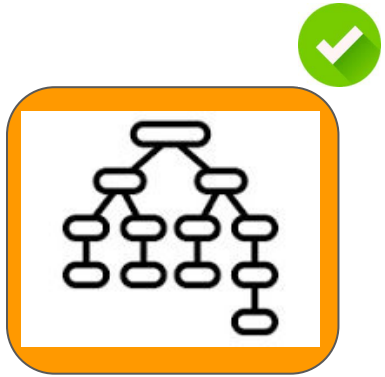
1. Error localization



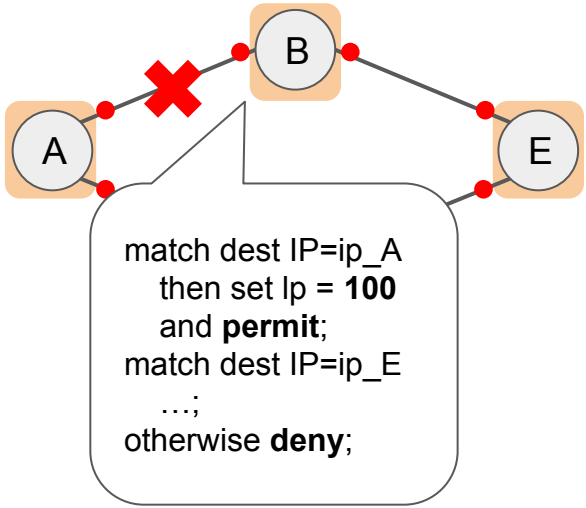
2. Candidate generation

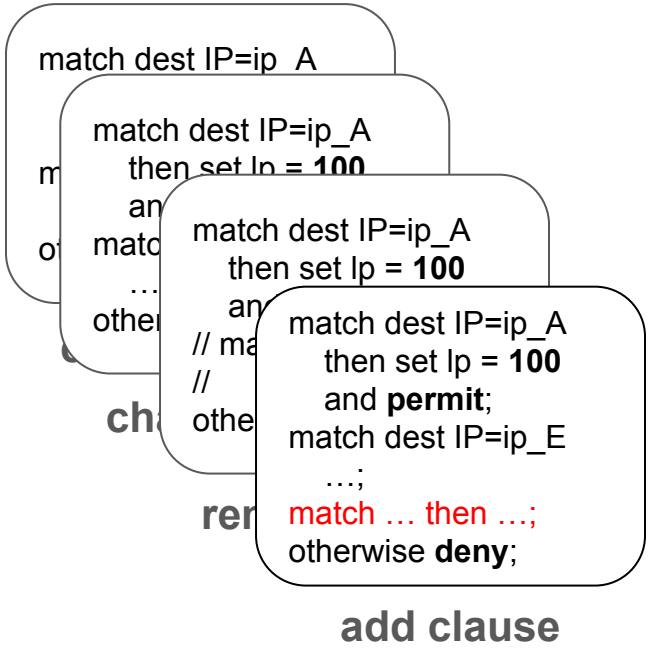


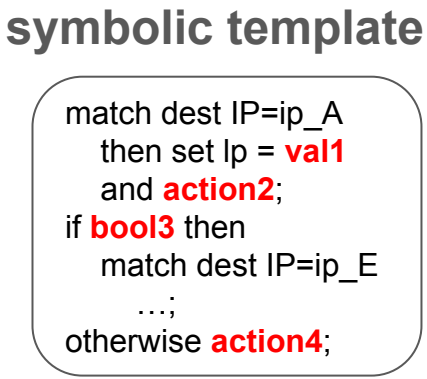
3. Solution selection



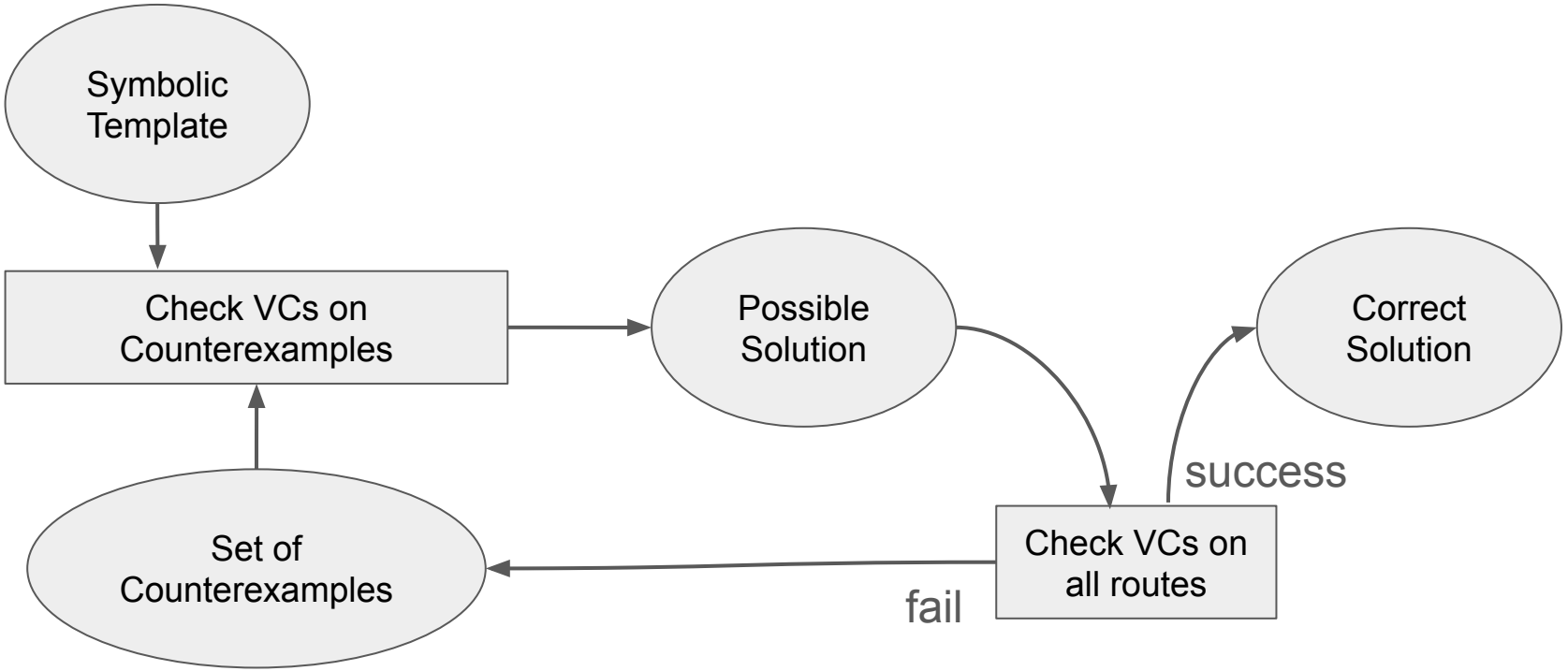
Step 2: repair candidates generation







Step 3: repair solution selection



Cost functions in solution selection

```
match dest IP=ip_A
    then set lp = 300
    and permit;
match dest IP=ip_E
    …;
otherwise deny;
```

cost = 1

```
match dest IP=ip_A
    then set lp = 100
    and deny;
match dest IP=ip_E
    …;
otherwise deny;
```

cost = 1

```
match dest IP=ip_A
    then set lp = 100
    and permit;
// match dest IP=ip_E
//     …;
otherwise deny;
```

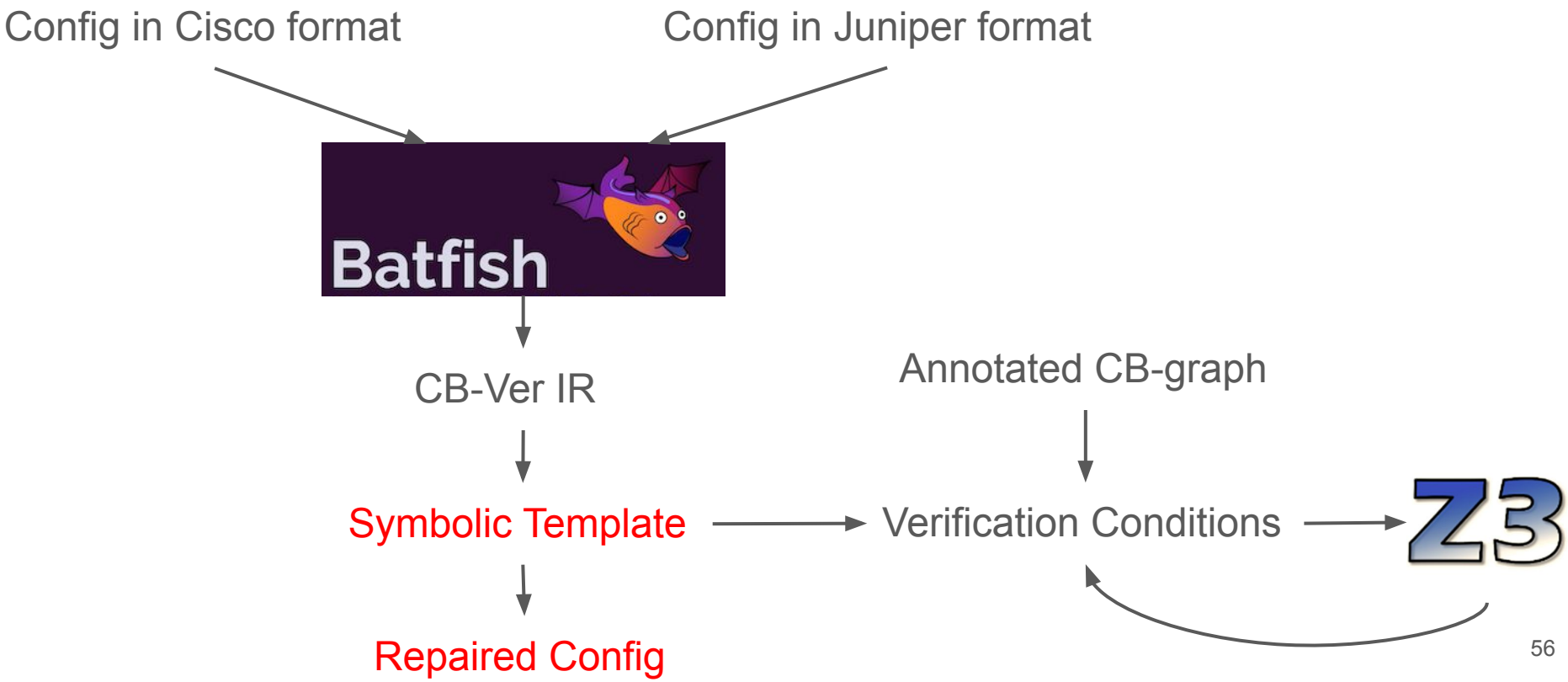
cost = 5

```
match dest IP=ip_A
    then set lp = val1
    and action2;
if bool3 then
    match dest IP=ip_E
        …;
otherwise action4;
```

cost = …

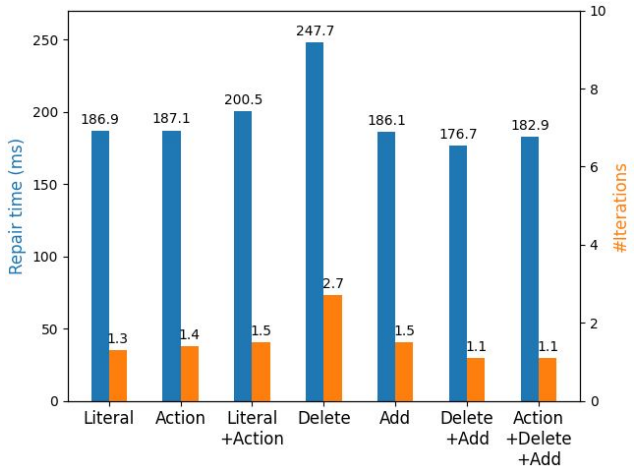
- Thanks to optimizing SMT solvers!

Implementation

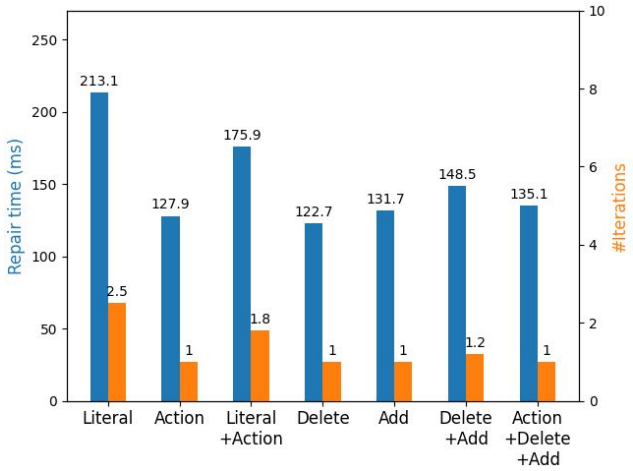


Evaluation

- Synthetic fattrees
  - Ability to repair different types of errors (using the same template!)

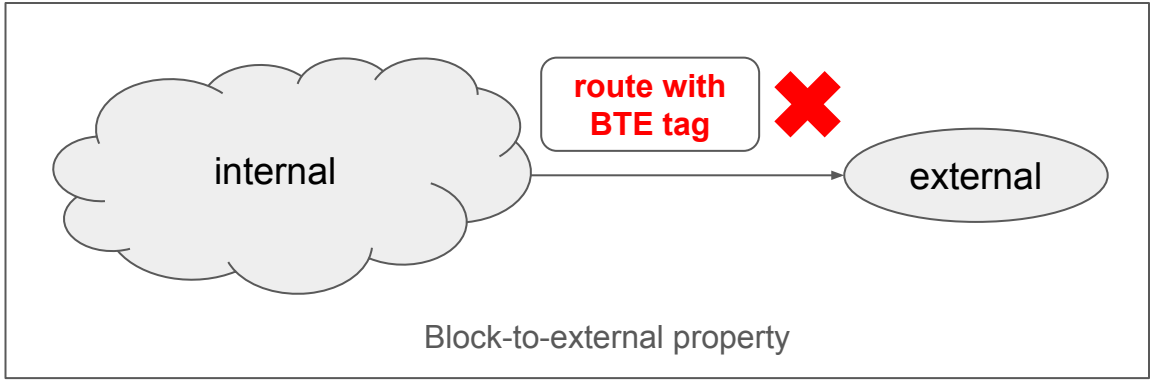


Valley-free



Hijack

Evaluation



Block-to-external property

- Internet2
  - Gives trivial repair when specifying BTE only
  - BTE + reachability gives meaningful repair

```
match dest IP=ip_A
    …;
match dest IP=ip_E
    …;
otherwise deny;
```

erroneous config

```
match anything
    then deny;
match dest IP=ip_A
    …;
match dest IP=ip_E
    …;
otherwise deny;
```

trivial repair

```
match BTE tag
    then deny;
match dest IP=ip_A
    …;
match dest IP=ip_E
    …;
otherwise deny;
```

meaningful repair

Related Works

- Control plane verification
  - Bagpipe, Minesweeper, …
  - Lightyear
  - Timepiece
  - CB-Graphs (our work)
- Configuration synthesis & repair
  - Propane/AT, Avenir, CPR, Hoyan…

Conclusion

- We introduce CB-graphs, which
  - Utilizes modularity to achieve scalability
  - Improves usability so that it
    - Can express key network properties
    - Minimizes user burden
    - Automates the configuration repair

Thank you!