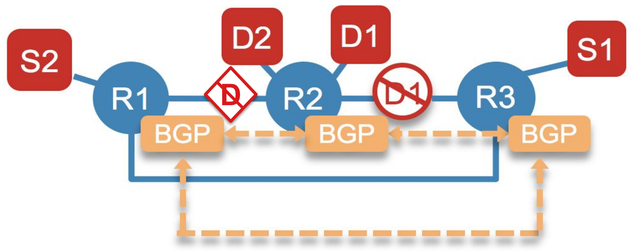
Localizing Router Configuration Errors Using Unsatisfiable Cores

Ruchit Shrestha, Xiaolin Sun, and Aaron Gember-Jacobson (Colgate University)

Motivation

Networks relying on distributed routing protocols often have complex router configurations. This complexity makes it difficult for operators to update configurations and locate errors in configurations.

Example network



Policy*	Satisfied	Counterexample
S1⇒D1	X	R3→Blocked
S1⇏D2	X	R3→R2→D2
S2⇒D1	X	R1→R3→Blocked
S2⇏D2	X	R1→R3→R2→D2

**should hold even under single link failure*

Current verification/repair tools

State-of-the-art network verifiers [1, 3, 5] do not indicate:

- which portions of the configurations influenced the computation of the forwarding path
- whether routers on the path, off the path, or both are at fault
- whether violations of the same requirement may manifest in different ways under different failure scenarios
- whether violations of different requirements are related

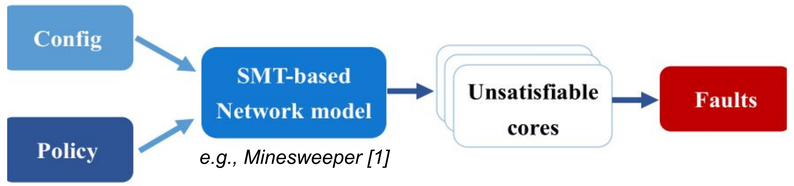
State-of-the-art network repair tools [2, 4] do not scale to networks with many routers or policies, because they consider all possible repairs.

Fault localization

Software fault localization is the process of identifying which lines of a program likely cause certain test cases to fail. Our goal is to design a technique for accurately localizing errors in network configurations, thus paving the way for faster network repair.

Our Approach

Our key insight is to localize configuration faults using *unsatisfiable cores* generated from SMT-based models of network configurations' semantics.



Network model

Models like Minesweeper [1] abstract away the details of router control software and encode a network's distributed decision process as a single system of logical formulas.

N: $BGP_{R2\to R3}$ = if Match(Dst,D1) then {D1,0} else {D2,0}

$(\wedge_{n=R1,R2}\ BestAd \leqslant BGP_{n\to R3}) \wedge (\vee_{n=R1,R2}\ BestAd == BGP_{n\to R3})$

$ControlFwd_{R3\to R2}$ = $(BestAd == BGP_{R2\to R3})$

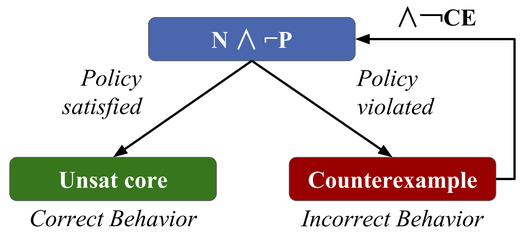
$DataFwd_{R3\to R2}$ = $ControlFwd_{R3\to R2} \wedge \neg$Match(Dst, D1)

$Reach_{R3\to R2}$ = $DataFwd_{R3\to R2} \vee (DataFwd_{R3\to R1} \wedge Reach_{R1\to R2})$

...

P: Src = S1 $\wedge$ Dst = D1 $\wedge$ $Reach_{R3\to R2}$

Obtaining unsatisfiable cores



Faults = N - Unsat Core

Multiple, minimal unsatisfiable cores

Challenges:

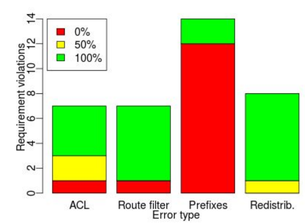
- Unsat core may not be minimal ⇒ core includes constraints that do not contribute to correct behavior ⇒ under estimate faults
- Solver produces one, out of many, unsat cores ⇒ overlook constraints that contribute to good behavior ⇒ over estimate faults

Solution:

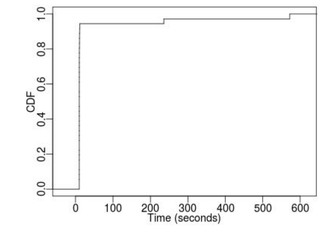
- Compute *all* minimal unsat cores using MARCO [6]
- Faults = N - union of all minimal unsat cores

Results

We implement our approach atop Minesweeper [1] and test it on 6 synthetic networks of varying size that use a combination of OSPF, BGP, static routes and route redistribution to enable reachability between different pairs of subnets connected to different routers. We introduce errors into these configurations by adding ACLs, adding route filters, removing advertised prefixes, and disabling route redistribution.



(a) Recall



(b) Performance

Figure (a) shows for each type of error the fraction of reachability requirement violations for which our technique identified none, half or all of the SMT constraints associated with faulty configuration stanzas. Currently, our approach effectively detects three of four types of errors that lead to reachability violations.

Figure (b) demonstrates that our approach requires less than 10 seconds to localize faults for 90% of the network scenarios.

Future Work

- Develop domain-specific heuristics—e.g., failure equivalence classes—to speed-up computation of unsat cores
- Evaluate our approach on real network configurations and additional types of errors
- Determine how to localize faults at a sub-constraint granularity to identify faults in individual lines of configuration, rather than stanzas

References & Acknowledgements

1. R. Beckett, A. Gupta, R. Mahajan, D. Walker. A general approach to network configuration verification. *SIGCOMM*, 2017.
2. A. El-Hassany, P. Tsankov, L. Vanbever, M. Vechev. Netcomplete: Practical network-wide configuration synthesis with autocompletion. *NSDI*, 2018.
3. S. K. Fayaz, T. Sharma, A. Fogel, R. Mahajan, T. D. Millstein, V. Sekar, G. Varghese. Efficient network reachability analysis using a succinct control plane representation. *OSDI*, 2016.
4. A. Gember-Jacobson, A. Akella, R. Mahajan, and H. Liu. Automatically repairing network control planes using an abstract representation. *SOSP*, 2017.
5. A. Gember-Jacobson, R. Viswanathan, A. Akella, and R. Mahajan. Fast control plane analysis using an abstract representation. *SIGCOMM*, 2016.
6. M. H. Liffiton, A. Previti, A. Malik, and J. Marques-Silva. Fast, flexible MUS enumeration. *Constraints*, 21(2), 2016.

This work is funded by National Science Foundation Grant CCF-1637427 and Colgate University.