Taming the transient while reconfiguring BGP

Abstract

BGP reconfigurations are a daily occurrence for most network operators, especially in large networks. Yet, performing safe and robust BGP reconfiguration changes is still an open problem. Few BGP reconfiguration techniques exist, and they are either (i) unsafe, because they ignore transient states, which can easily lead to invariant violations; or (ii) impractical, as they duplicate the entire routing and forwarding states, and require special hardware.

In this paper, we introduce Chameleon, the first BGP reconfiguration framework capable of maintaining correctness throughout a reconfiguration campaign while relying on standard BGP functionalities and minimizing state duplication. Akin to concurrency coordination in distributed systems, Chameleon models the reconfiguration process with happens-before relations. This modeling allows us to capture the safety properties of transient BGP states. We then use this knowledge to precisely control the BGP route propagation and convergence, so that input invariants are provably preserved at any time during the reconfiguration.

We fully implement Chameleon and evaluate it in both testbeds and simulations, on real-world topologies and large-scale reconfiguration scenarios. In most experiments, our system computes reconfiguration plans within a minute, and performs them from start to finish in a few minutes, with minimal overhead.

Keywords: Border Gateway Protocol (BGP), reconfiguration, network update, convergence, scheduling

1 Introduction

The introduction of "Taming the Transient while Reconfiguring BGP" by Schneider et al. addresses the critical issue of network reconfigurations, particularly focusing on Border Gateway Protocol (BGP) reconfigurations. It highlights the frequency and disruptive nature of these reconfigurations, citing examples like Alibaba's network outages and Microsoft Azure's service interruption due to BGP updates. Despite their commonality, existing frameworks for network reconfiguration fail to ensure both safety and practicality during BGP reconfigurations.

The paper points out the limitations of existing techniques like "Shadow Configurations" and "BGP Ships-in-the-Night," which duplicate routing and forwarding states, leading to impractical overheads and lack of router support. Another method, Snowcap, modifies BGP configurations in-place but fails to maintain transient state correctness, as illustrated in the paper through experiments like the Abilene network reconfiguration.

To address these challenges, the authors introduce Chameleon, a novel in-place BGP reconfiguration framework that guarantees correctness throughout the reconfiguration process, including both steady and transient states. Chameleon's development involved overcoming significant challenges, including creating a formal concurrency model for BGP networks, designing concurrency control mechanisms without controlling individual BGP message timings, and developing an efficient runtime controller for orchestrating the reconfiguration process.

A key innovation in Chameleon is the use of a "happens-before" model for BGP-specific concurrency. This model enables the framework to maintain network invariants across all concurrent executions through synchronization barriers, which are then compiled into a reconfiguration plan. This plan is executed using standard BGP commands, ensuring network-wide correctness.

The introduction concludes by acknowledging that ensuring such correctness extends the reconfiguration duration, as demonstrated in their experiments, but asserts that this trade-off is worthwhile for the increased correctness. Chameleon was extensively evaluated in testbeds and simulations on real-world topologies and scenarios, showing its capability to compute reconfiguration plans quickly and execute them with minimal overhead. The paper's main contributions include the development of the first practical BGP reconfiguration technique that preserves correctness throughout the entire process, a concurrency model for BGP reconfigurations, a complete implementation of Chameleon in Rust, and a comprehensive evaluation of its performance and overhead.

2 Related works

2.1 Systems Extending Software-Defined Networking (SDN)

These systems, represented by references [1] [2] [3], are primarily focused on SDN where a central controller modifies the forwarding tables of all nodes.

They offer robust guarantees such as per-packet consistency or congestion avoidance by directly controlling forwarding decisions.

2.2 Traditional Network Update Systems

2.2.1 Tools that Duplicate Control and Data Planes [4] [5]

These tools allow routers to run both initial and final configurations in parallel, gradually transitioning to the final configuration while avoiding forwarding loops [6] [7]. Despite their usefulness, they are often not used in practice due to their significant overhead.

2.2.2 In-Place Reconfiguration Tools [8] [9] [10]

These tools perform reconfigurations by partitioning changes into smaller units and applying them gradually while maintaining correctness.

They offer advantages over the first category, such as minimal overhead and no requirement for special router support.

Examples include techniques for avoiding forwarding loops during IGP reconfigurations. Snowcap

[10] is noted as the only in-place system for BGP reconfiguration but with limitations in ensuring correctness only in steady states.

2.3 Network Management Tools and Extensions to BGP

Research has also focused on network management tools [11] [12] [13]that can utilize systems like Chameleon for safe reconfiguration.

Additionally, there have been proposals for extensions to BGP [14] to ensure the absence of forwarding loops during convergence, but these have seen limited adoption due to the added complexity they introduce to BGP speakers.

3 Method

3.1 Overview

In a nutshell, Chameleon achieves safety by coordinating the forwarding state updates BGP makes during the reconfiguration. To practically coordinate the updates, Chameleon gradually introduces a temporary BGP configuration (which differs from both the initial and final configuration). This temporary configuration enforces a particular ordering of BGP messages that satisfies the specification.

Finding and implementing specific message orderings is far from trivial. The first challenge is capturing achievable orderings by introducing temporary BGP configurations. This is hard as many BGP-specific mechanisms—like route reflection [15]—limit route visibility. The second challenge is implementing a specific execution using temporary configurations; this requires both reasoning about the distributed routing state and synchronizing BGP computations.

Chameleon computes a reconfiguration plan to transition from the old to the new configuration while satisfying the specification.Chameleon solves those challenges by following the following three consecutive steps. Chameleon's workflow is visualized in Fig. 1:

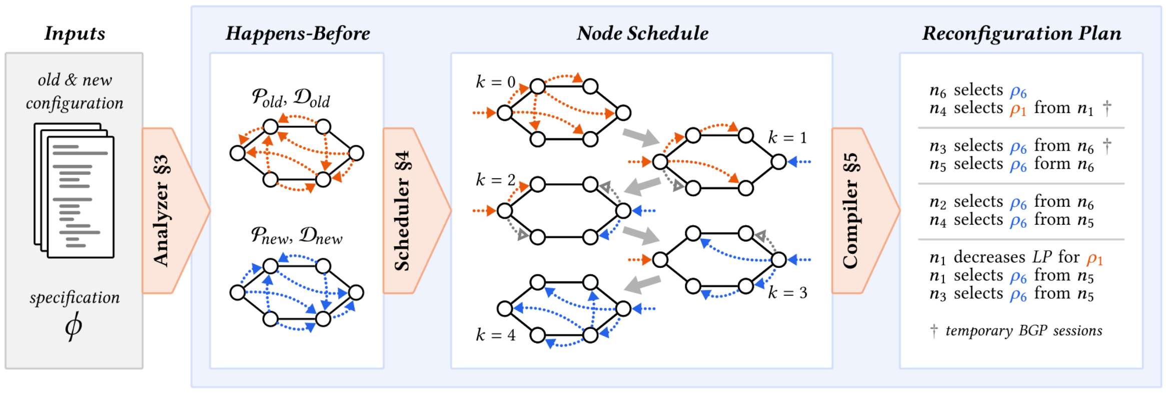

Inputs
old & new configuration
specification
ϕ
Analyzer §3
Happens-Before
$\mathcal{P}_{old}$, $\mathcal{D}_{old}$
$\mathcal{P}_{new}$, $\mathcal{D}_{new}$
Scheduler §4
Node Schedule
$k = 0$
$k = 1$
$k = 2$
$k = 3$
$k = 4$
Compiler §5
Reconfiguration Plan
n_6 selects ρ_6
n_4 selects ρ_1 from n_1 †
n_3 selects ρ_6 from n_6 †
n_5 selects ρ_6 form n_6
n_2 selects ρ_6 from n_6
n_4 selects ρ_6 from n_5
n_1 decreases LP for ρ_1
n_1 selects ρ_6 from n_5
n_3 selects ρ_6 from n_5
† temporary BGP sessions

Figure 1. Chameleon Workflow

Step 1: Analyzer. The analyzer extracts happens-before relations between selected routes in the network, encoding the propagation path of routes. These relations define the space of convergence processes that are realizable just using temporary BGP configurations. We obtain these relations by simulating the network and analyzing the resulting network state in a way that generalizes to any BGP configuration.

Step 2: Scheduler. We express the happens-before relations together with the specification as an Integer Linear Program (ILP). A solution to the ILP is a node schedule describing a BGP convergence process that meets the specification, in both steady and transient states. The scheduler allows concurrent updates if their relative order does not affect the specification. Further, Chameleon allows additional propagation paths to increase route visibility and ensure a solution exists. We maximize the number of concurrent updates (thus, minimizing the reconfiguration time) as a primary objective while reducing additional propagation paths as a secondary goal.

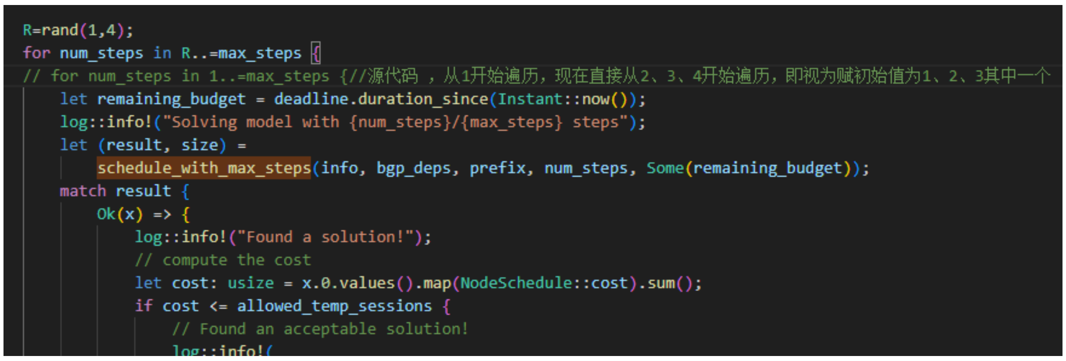
Step 3: Compiler. Chameleon computes a reconfiguration plan to implement the calculated schedule, along with local conditions to synchronize the update and guarantee correctness. Each temporary command only rewrites routing preferences by modifying route attributes such as weight [25] or local preference. The conditions assert a router knows or selects a specific route and can be checked locally by inspecting that router's routing table.

4 Implementation details

4.1 Comparing with the released source codes

This project refers to the source code of https://github.com/nsg-ethz/Chameleon/tree/main as the basic framework, and this project modifies and improves it to make it work properly, and to a certain extent, has some improvements.

The authors chose the program is the initial value of all assigned 0 value, this topology for the number of network devices will have a large number of initial traversal, I chose to randomize the initial value of all the devices as an integer not greater than 3.

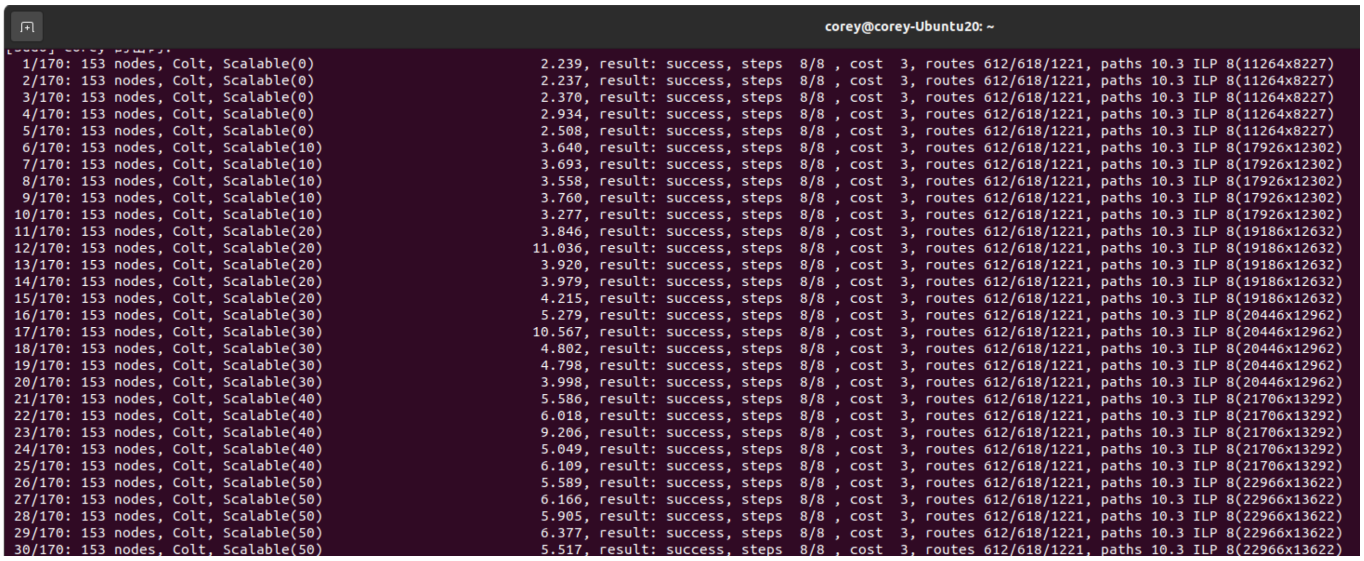


```
R=rand(1,4);
for num_steps in R..=max_steps {
// for num_steps in 1..=max_steps {//源代码 ，从1开始遍历，现在直接从2、3、4开始遍历，即视为赋初始值为1、2、3其中一个
    let remaining_budget = deadline.duration_since(Instant::now());
    log::info!("Solving model with {num_steps}/{max_steps} steps");
    let (result, size) =
        schedule_with_max_steps(info, bgp_deps, prefix, num_steps, Some(remaining_budget));
    match result {
        Ok(x) => {
            log::info!("Found a solution!");
            // compute the cost
            let cost: usize = x.0.values().map(NodeSchedule::cost).sum();
            if cost <= allowed_temp_sessions {
                // Found an acceptable solution!
                log::info!(
```

Figure 2. Experimental parameter modification

4.2 Experimental environment setup

This project uses Ubuntu 20.04 as the experimental environment. Docker was utilized to quickly configure the experimental environment built by the author.



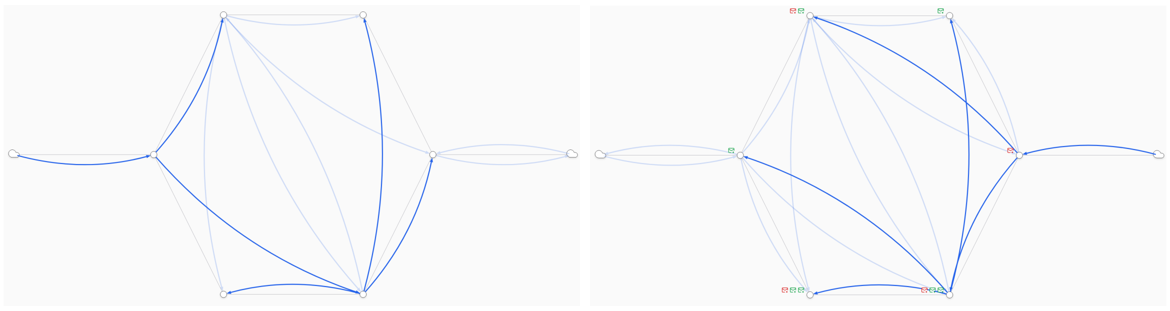
```
corey@corey-Ubuntu20: ~
  1/170: 153 nodes, Colt, Scalable(0)                          2.239, result: success, steps  8/8 , cost  3, routes 612/618/1221, paths 10.3 ILP 8(11264x8227)
  2/170: 153 nodes, Colt, Scalable(0)                          2.237, result: success, steps  8/8 , cost  3, routes 612/618/1221, paths 10.3 ILP 8(11264x8227)
  3/170: 153 nodes, Colt, Scalable(0)                          2.370, result: success, steps  8/8 , cost  3, routes 612/618/1221, paths 10.3 ILP 8(11264x8227)
  4/170: 153 nodes, Colt, Scalable(0)                          2.934, result: success, steps  8/8 , cost  3, routes 612/618/1221, paths 10.3 ILP 8(11264x8227)
  5/170: 153 nodes, Colt, Scalable(0)                          2.508, result: success, steps  8/8 , cost  3, routes 612/618/1221, paths 10.3 ILP 8(11264x8227)
  6/170: 153 nodes, Colt, Scalable(10)                         3.640, result: success, steps  8/8 , cost  3, routes 612/618/1221, paths 10.3 ILP 8(17926x12302)
  7/170: 153 nodes, Colt, Scalable(10)                         3.693, result: success, steps  8/8 , cost  3, routes 612/618/1221, paths 10.3 ILP 8(17926x12302)
  8/170: 153 nodes, Colt, Scalable(10)                         3.558, result: success, steps  8/8 , cost  3, routes 612/618/1221, paths 10.3 ILP 8(17926x12302)
  9/170: 153 nodes, Colt, Scalable(10)                         3.760, result: success, steps  8/8 , cost  3, routes 612/618/1221, paths 10.3 ILP 8(17926x12302)
 10/170: 153 nodes, Colt, Scalable(10)                         3.277, result: success, steps  8/8 , cost  3, routes 612/618/1221, paths 10.3 ILP 8(17926x12302)
 11/170: 153 nodes, Colt, Scalable(20)                         3.846, result: success, steps  8/8 , cost  3, routes 612/618/1221, paths 10.3 ILP 8(19186x12632)
 12/170: 153 nodes, Colt, Scalable(20)                        11.036, result: success, steps  8/8 , cost  3, routes 612/618/1221, paths 10.3 ILP 8(19186x12632)
 13/170: 153 nodes, Colt, Scalable(20)                         3.920, result: success, steps  8/8 , cost  3, routes 612/618/1221, paths 10.3 ILP 8(19186x12632)
 14/170: 153 nodes, Colt, Scalable(20)                         3.979, result: success, steps  8/8 , cost  3, routes 612/618/1221, paths 10.3 ILP 8(19186x12632)
 15/170: 153 nodes, Colt, Scalable(20)                         4.215, result: success, steps  8/8 , cost  3, routes 612/618/1221, paths 10.3 ILP 8(19186x12632)
 16/170: 153 nodes, Colt, Scalable(30)                         5.279, result: success, steps  8/8 , cost  3, routes 612/618/1221, paths 10.3 ILP 8(20446x12962)
 17/170: 153 nodes, Colt, Scalable(30)                        10.567, result: success, steps  8/8 , cost  3, routes 612/618/1221, paths 10.3 ILP 8(20446x12962)
 18/170: 153 nodes, Colt, Scalable(30)                         4.802, result: success, steps  8/8 , cost  3, routes 612/618/1221, paths 10.3 ILP 8(20446x12962)
 19/170: 153 nodes, Colt, Scalable(30)                         4.798, result: success, steps  8/8 , cost  3, routes 612/618/1221, paths 10.3 ILP 8(20446x12962)
 20/170: 153 nodes, Colt, Scalable(30)                         3.998, result: success, steps  8/8 , cost  3, routes 612/618/1221, paths 10.3 ILP 8(20446x12962)
 21/170: 153 nodes, Colt, Scalable(40)                         5.586, result: success, steps  8/8 , cost  3, routes 612/618/1221, paths 10.3 ILP 8(21706x13292)
 22/170: 153 nodes, Colt, Scalable(40)                         6.018, result: success, steps  8/8 , cost  3, routes 612/618/1221, paths 10.3 ILP 8(21706x13292)
 23/170: 153 nodes, Colt, Scalable(40)                         9.206, result: success, steps  8/8 , cost  3, routes 612/618/1221, paths 10.3 ILP 8(21706x13292)
 24/170: 153 nodes, Colt, Scalable(40)                         5.049, result: success, steps  8/8 , cost  3, routes 612/618/1221, paths 10.3 ILP 8(21706x13292)
 25/170: 153 nodes, Colt, Scalable(40)                         6.109, result: success, steps  8/8 , cost  3, routes 612/618/1221, paths 10.3 ILP 8(21706x13292)
 26/170: 153 nodes, Colt, Scalable(50)                         5.589, result: success, steps  8/8 , cost  3, routes 612/618/1221, paths 10.3 ILP 8(22966x13622)
 27/170: 153 nodes, Colt, Scalable(50)                         6.166, result: success, steps  8/8 , cost  3, routes 612/618/1221, paths 10.3 ILP 8(22966x13622)
 28/170: 153 nodes, Colt, Scalable(50)                         5.905, result: success, steps  8/8 , cost  3, routes 612/618/1221, paths 10.3 ILP 8(22966x13622)
 29/170: 153 nodes, Colt, Scalable(50)                         6.377, result: success, steps  8/8 , cost  3, routes 612/618/1221, paths 10.3 ILP 8(22966x13622)
 30/170: 153 nodes, Colt, Scalable(50)                         5.517, result: success, steps  8/8 , cost  3, routes 612/618/1221, paths 10.3 ILP 8(22966x13622)
```

Figure 3. Recording in experiments

4.3 Interface design

The authors used https://bgpsim.github.io/ to visualize the reconfiguration process.

I implemented running the application locally (http://127.0.0.1:8080/) for process visualization.



(a) Initial situation

(b) After BGP reconfiguration

Figure 4. Routing relationships before and after reconfiguration

4.4 Main contributions

Improvements are centered around two functions of the scheduler, schedule_smart and schedule_with_max_steps.

4.4.1 schedule_smart function

The goal of this function is to find a way to switch each router in the network from the old route to the new one in an optimal order. This "optimal order" means minimizing any confusion or instability in the network during the switchover.

The function tries different "steps", each of which can be interpreted as a unit of time, to see if it can find a good switching plan within that time. It will start with a small number of steps, and gradually increase the number of steps until it finds a plan that works, or reaches a set time limit.

If it finds a good plan, it will use that plan; if it doesn't find one when the time runs out, it will tell you that it didn't find a viable plan.

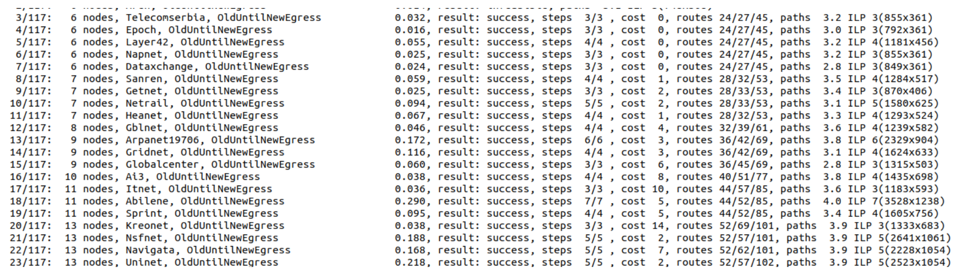
4.4.2 schedule_with_max_steps function

This function is a tool used by schedule_smart to try to find a switching schedule. Given a number of steps (i.e. a time limit), it will try to find a good switching order within that limit.

The function uses a mathematical method called Integer Linear Programming (ILP) to solve this problem. In simple terms, it's like a complex math puzzle with lots of rules (e.g., how the router switches is effective, how switching causes problems, etc.), and ILP is used to find a best solution given all these rules.

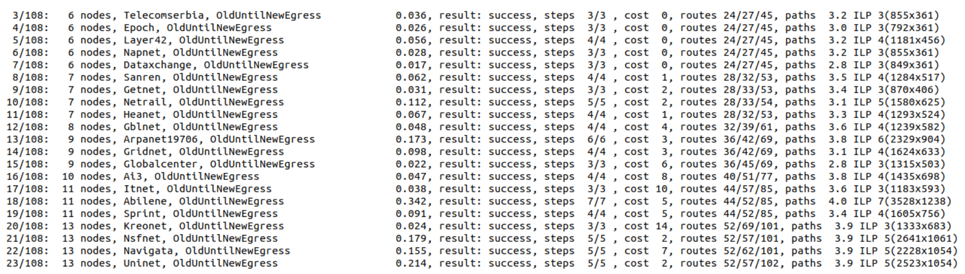
If a good solution is found within a given number of steps, the function returns that plan; if it doesn't, it tells you that it can't find a solution within that number of steps.

5 Results and analysis



```
 3/117:   6 nodes, Telecomserbia, OldUntilNewEgress          0.032, result: success, steps  3/3 , cost  0, routes 24/27/45, paths  3.2 ILP 3(855x361)
 4/117:   6 nodes, Epoch, OldUntilNewEgress                  0.016, result: success, steps  3/3 , cost  0, routes 24/27/45, paths  3.0 ILP 3(792x361)
 5/117:   6 nodes, Layer42, OldUntilNewEgress                0.055, result: success, steps  4/4 , cost  0, routes 24/27/45, paths  3.2 ILP 4(1181x456)
 6/117:   6 nodes, Napnet, OldUntilNewEgress                 0.025, result: success, steps  3/3 , cost  0, routes 24/27/45, paths  3.2 ILP 3(855x361)
 7/117:   6 nodes, Dataxchange, OldUntilNewEgress            0.024, result: success, steps  3/3 , cost  0, routes 24/27/45, paths  2.8 ILP 3(849x361)
 8/117:   7 nodes, Sanren, OldUntilNewEgress                 0.059, result: success, steps  4/4 , cost  1, routes 28/32/53, paths  3.5 ILP 4(1284x517)
 9/117:   7 nodes, Getnet, OldUntilNewEgress                 0.025, result: success, steps  3/3 , cost  2, routes 28/33/53, paths  3.4 ILP 3(870x406)
10/117:   7 nodes, Netrail, OldUntilNewEgress                0.094, result: success, steps  5/5 , cost  2, routes 28/33/53, paths  3.1 ILP 5(1580x625)
11/117:   7 nodes, Heanet, OldUntilNewEgress                 0.067, result: success, steps  4/4 , cost  1, routes 28/32/53, paths  3.3 ILP 4(1293x524)
12/117:   8 nodes, Gblnet, OldUntilNewEgress                 0.046, result: success, steps  4/4 , cost  4, routes 32/39/61, paths  3.6 ILP 4(1239x582)
13/117:   9 nodes, Arpanet19706, OldUntilNewEgress           0.172, result: success, steps  6/6 , cost  3, routes 36/42/69, paths  3.8 ILP 6(2329x904)
14/117:   9 nodes, Gridnet, OldUntilNewEgress                0.116, result: success, steps  4/4 , cost  3, routes 36/42/69, paths  3.1 ILP 4(1624x633)
15/117:   9 nodes, Globalcenter, OldUntilNewEgress           0.060, result: success, steps  3/3 , cost  6, routes 36/45/69, paths  2.8 ILP 3(1315x503)
16/117:  10 nodes, Ai3, OldUntilNewEgress                    0.038, result: success, steps  4/4 , cost  8, routes 40/51/77, paths  3.8 ILP 4(1435x698)
17/117:  11 nodes, Itnet, OldUntilNewEgress                  0.036, result: success, steps  3/3 , cost 10, routes 44/57/85, paths  3.6 ILP 3(1183x593)
18/117:  11 nodes, Abilene, OldUntilNewEgress                0.290, result: success, steps  7/7 , cost  5, routes 44/52/85, paths  4.0 ILP 7(3528x1238)
19/117:  11 nodes, Sprint, OldUntilNewEgress                 0.095, result: success, steps  4/4 , cost  5, routes 44/52/85, paths  3.4 ILP 4(1605x756)
20/117:  13 nodes, Kreonet, OldUntilNewEgress                0.038, result: success, steps  3/3 , cost 14, routes 52/69/101, paths  3.9 ILP 3(1333x683)
21/117:  13 nodes, Nsfnet, OldUntilNewEgress                 0.188, result: success, steps  5/5 , cost  2, routes 52/57/101, paths  3.9 ILP 5(2641x1061)
22/117:  13 nodes, Navigata, OldUntilNewEgress               0.168, result: success, steps  5/5 , cost  7, routes 52/62/101, paths  3.9 ILP 5(2228x1054)
23/117:  13 nodes, Uninet, OldUntilNewEgress                 0.218, result: success, steps  5/5 , cost  2, routes 52/57/102, paths  3.9 ILP 5(2523x1054)
```

(a) Iterate from constant 1

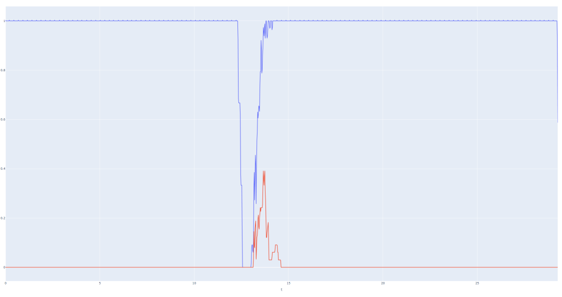


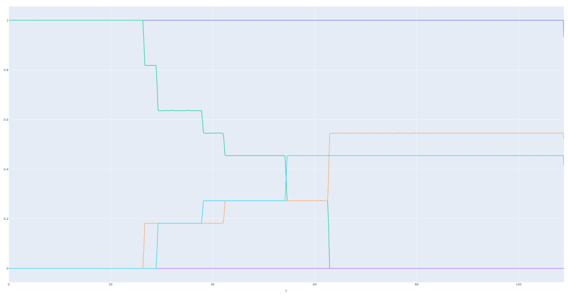
```
 3/108:   6 nodes, Telecomserbia, OldUntilNewEgress          0.036, result: success, steps  3/3 , cost  0, routes 24/27/45, paths  3.2 ILP 3(855x361)
 4/108:   6 nodes, Epoch, OldUntilNewEgress                  0.026, result: success, steps  3/3 , cost  0, routes 24/27/45, paths  3.0 ILP 3(792x361)
 5/108:   6 nodes, Layer42, OldUntilNewEgress                0.056, result: success, steps  4/4 , cost  0, routes 24/27/45, paths  3.2 ILP 4(1181x456)
 6/108:   6 nodes, Napnet, OldUntilNewEgress                 0.028, result: success, steps  3/3 , cost  0, routes 24/27/45, paths  3.2 ILP 3(855x361)
 7/108:   6 nodes, Dataxchange, OldUntilNewEgress            0.017, result: success, steps  3/3 , cost  0, routes 24/27/45, paths  2.8 ILP 3(849x361)
 8/108:   7 nodes, Sanren, OldUntilNewEgress                 0.062, result: success, steps  4/4 , cost  1, routes 28/32/53, paths  3.5 ILP 4(1284x517)
 9/108:   7 nodes, Getnet, OldUntilNewEgress                 0.031, result: success, steps  3/3 , cost  2, routes 28/33/53, paths  3.4 ILP 3(870x406)
10/108:   7 nodes, Netrail, OldUntilNewEgress                0.112, result: success, steps  5/5 , cost  2, routes 28/33/54, paths  3.1 ILP 5(1580x625)
11/108:   7 nodes, Heanet, OldUntilNewEgress                 0.067, result: success, steps  4/4 , cost  1, routes 28/32/53, paths  3.3 ILP 4(1293x524)
12/108:   8 nodes, Gblnet, OldUntilNewEgress                 0.048, result: success, steps  4/4 , cost  4, routes 32/39/61, paths  3.6 ILP 4(1239x582)
13/108:   9 nodes, Arpanet19706, OldUntilNewEgress           0.173, result: success, steps  6/6 , cost  3, routes 36/42/69, paths  3.8 ILP 6(2329x904)
14/108:   9 nodes, Gridnet, OldUntilNewEgress                0.098, result: success, steps  4/4 , cost  3, routes 36/42/69, paths  3.1 ILP 4(1624x633)
15/108:   9 nodes, Globalcenter, OldUntilNewEgress           0.022, result: success, steps  3/3 , cost  6, routes 36/45/69, paths  2.8 ILP 3(1315x503)
16/108:  10 nodes, Ai3, OldUntilNewEgress                    0.047, result: success, steps  4/4 , cost  8, routes 40/51/77, paths  3.8 ILP 4(1435x698)
17/108:  11 nodes, Itnet, OldUntilNewEgress                  0.038, result: success, steps  3/3 , cost 10, routes 44/57/85, paths  3.6 ILP 3(1183x593)
18/108:  11 nodes, Abilene, OldUntilNewEgress                0.342, result: success, steps  7/7 , cost  5, routes 44/52/85, paths  4.0 ILP 7(3528x1238)
19/108:  11 nodes, Sprint, OldUntilNewEgress                 0.091, result: success, steps  4/4 , cost  5, routes 44/52/85, paths  3.4 ILP 4(1605x756)
20/108:  13 nodes, Kreonet, OldUntilNewEgress                0.024, result: success, steps  3/3 , cost 14, routes 52/69/101, paths  3.9 ILP 3(1333x683)
21/108:  13 nodes, Nsfnet, OldUntilNewEgress                 0.179, result: success, steps  5/5 , cost  2, routes 52/57/101, paths  3.9 ILP 5(2641x1061)
22/108:  13 nodes, Navigata, OldUntilNewEgress               0.155, result: success, steps  5/5 , cost  7, routes 52/62/101, paths  3.9 ILP 5(2228x1054)
23/108:  13 nodes, Uninet, OldUntilNewEgress                 0.214, result: success, steps  5/5 , cost  2, routes 52/57/102, paths  3.9 ILP 5(2523x1054)
```

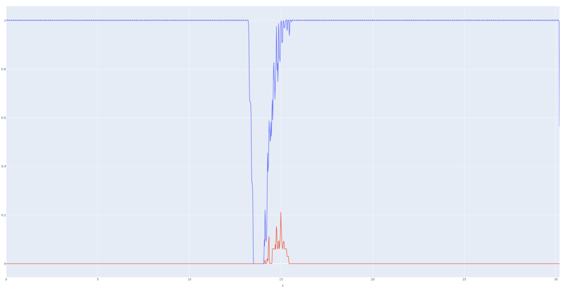
(b) Iterate from rand(1,3)

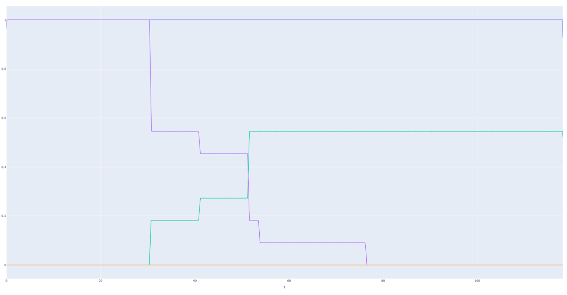
Figure 5. Effect Comparison

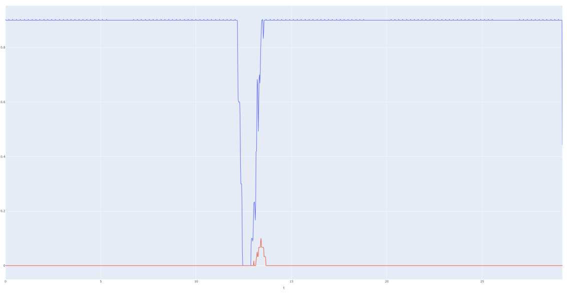
ANALYSIS: By comparing the effect of the modified scheme with that of the source code, it is found that when the number of nodes becomes larger when searching for the minimum number of rounds, the effect of the random initial value will be slightly better, but this does not apply to all cases; in addition, because of the randomness of the random, so the same topology results in the same topology results of repeated experiments may not be the same results. Besides,I have reproduced the original article, which compares the results of using snowcap with Chameleon for different topologies, as follows:

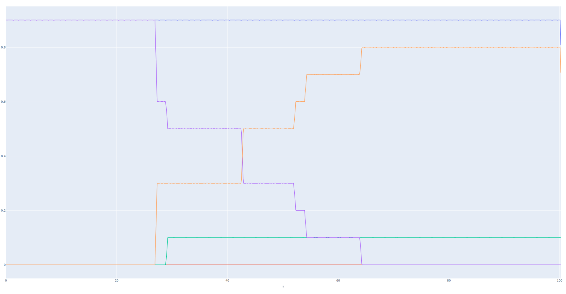

(a) Compuserve_Snowcap

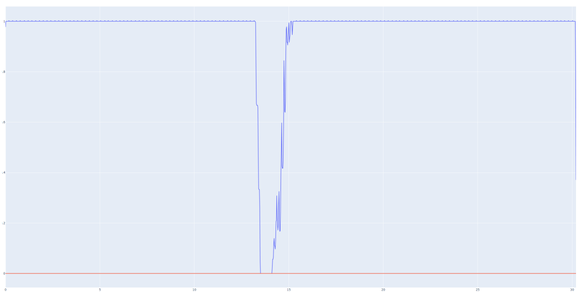

(b) Compuserve_Chameleon

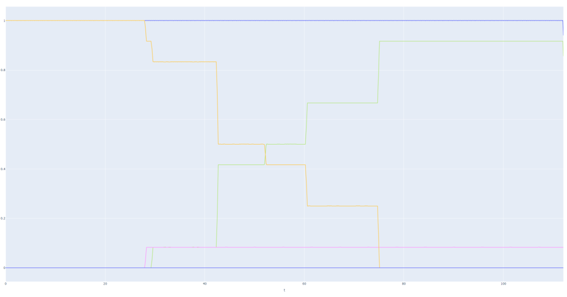

(c) Hibernia Canada_Snowcap

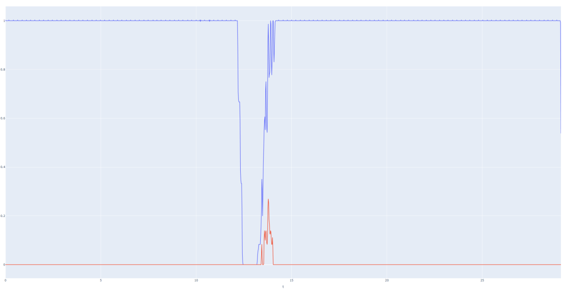

(d) Hibernia Canada_Chameleon


(e) Sprint_Snowcap


(f) Sprint_Chameleon


(g) JGN2plus_Snowcap


(h) JGN2plus_Chameleon


(i) EEnet_Snowcap

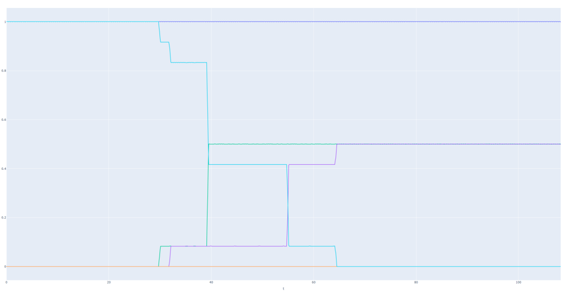

(j) EEnet_Chameleon

Figure 6. Chameleon consistently and safely reconfigures a network, while Snowcap [10] triggers black holes, and violates waypoint specifications.

Comparison experiment on five additional topologies in Topology Zoo [21], all of which contain 11 or 12 routers. The left column shows total traffic (and violations) for Snowcap, and the right column shows throughput for Chameleon.

ANALYSIS: From the resultant graphs, it is clear that Chameleon is able to keep the total throughput approximately constant in reconfiguration in all of these topologies.

6 Conclusion and future work

6.1 Conclusion

Chameleon is the first system to perform BGP reconfiguration and maintain correctness throughout the reconfiguration process. The authors present a framework for describing the BGP convergence process and a technique for generating a reconfiguration plan that seamlessly transitions a network from an old configuration to a new one. Demonstrates how Chameleon can schedule and execute large-scale reconfigurations in real networks in minutes while meeting complex specifications.There are no ethical issues associated with this work.

By reviewing this paper, I think there are two main reasons why this paper was able to win SIGCOMM:

1. The authors discovered a blank area of previous research: solving the problem of oscillations during BGP reconfiguration;

2. the authors transformed an NPC problem into a constrained optimization problem, and prioritized the logical relationships between the topologies using logical notation.

6.2 Future Work

By reproducing this post, I have three plans:

1, broaden the perspective of looking at problems, not only limit them to the level of representation, try to logicize complex problems, and then analyze and solve them according to logical relationships.

2, learn optimization-related knowledge, including but not limited to ILP, Big-M and other methods; this should be very helpful for my future research on routing and communication quality problems and no longer let the vision is limited to machine learning and other black-box type work.

3 Picture beautification, essay writing and other capabilities continue to strengthen the improvement.

References

[1] Chi-Yao Hong, Srikanth Kandula, Ratul Mahajan, Ming Zhang, Vijay Gill, Mohan Nanduri, and Roger Wattenhofer. Achieving high utilization with software-driven wan. In Proceedings of the ACM SIGCOMM 2013 Conference on SIGCOMM, pages 15–26, 2013.

[2] Ratul Mahajan and Roger Wattenhofer. On consistent updates in software defined networks. In Proceedings of the Twelfth ACM Workshop on Hot Topics in Networks, pages 1–7, 2013.

[3] Mark Reitblatt, Nate Foster, Jennifer Rexford, and David Walker. Consistent updates for software-defined networks: Change you can believe in! In Proceedings of the 10th ACM workshop on hot topics in networks, pages 1–6, 2011.

[4] Richard Alimi, Ye Wang, and Y Richard Yang. Shadow configuration as a network management primitive. In Proceedings of the ACM SIGCOMM 2008 conference on Data communication, pages 111–122, 2008.

[5] Gonzalo Gomez Herrero and Jan Antón Bernal Van der Ven. Network Mergers and Migrations: Junos Design and Implementation. John Wiley & Sons, 2011.

[6] Laurent Vanbever, Stefano Vissicchio, Cristel Pelsser, Pierre Francois, and Olivier Bonaventure. Seamless network-wide igp migrations. In Proceedings of the ACM SIGCOMM 2011 Conference, pages 314–325, 2011.

[7] Stefano Vissicchio, Laurent Vanbever, Cristel Pelsser, Luca Cittadini, Pierre Francois, and Olivier Bonaventure. Improving network agility with seamless bgp reconfigurations. IEEE/ACM Transactions on Networking, 21(3):990–1002, 2012.

[8] Francois Clad, Stefano Vissicchio, Pascal Mérindol, Pierre Francois, and Jean-Jacques Pansiot. Computing minimal update sequences for graceful router-wide reconfigurations. IEEE/ACM Transactions on Networking, 23(5):1373–1386, 2014.

[9] Pierre Francois, Mike Shand, and Olivier Bonaventure. Disruption free topology reconfiguration in ospf networks. In IEEE INFOCOM 2007-26th IEEE International Conference on Computer Communications, pages 89–97. IEEE, 2007.

[10] Tibor Schneider, Rüdiger Birkner, and Laurent Vanbever. Snowcap: synthesizing network-wide configuration updates. In Proceedings of the 2021 ACM SIGCOMM 2021 Conference, pages 33–49, 2021.

[11] Hongqiang Harry Liu, Xin Wu, Wei Zhou, Weiguo Chen, Tao Wang, Hui Xu, Lei Zhou, Qing Ma, and Ming Zhang. Automatic life cycle management of network configurations. In Proceedings of the Afternoon Workshop on Self-Driving Networks, pages 29–35, 2018.

[12] Peng Sun, Ratul Mahajan, Jennifer Rexford, Lihua Yuan, Ming Zhang, and Ahsan Arefin. A network-state management service. In Proceedings of the 2014 ACM Conference on SIGCOMM, pages 563–574, 2014.

[13] Yu-Wei Eric Sung, Xiaozheng Tie, Starsky HY Wong, and Hongyi Zeng. Robotron: Top-down network management at facebook scale. In Proceedings of the 2016 ACM SIGCOMM Conference, pages 426–439, 2016.

[14] Nikola Gvozdiev, Brad Karp, and Mark Handley. {LOUP}: The principles and practice of {Intra-Domain} route dissemination. In 10th USENIX Symposium on Networked Systems Design and Implementation (NSDI 13), pages 413–426, 2013.