Testing Router Command Line Interfaces

Omshi Samal
Colgate University
osamal@colgate.edu

Aaron Gember-Jacobson
Colgate University
agemberjacobson@colgate.edu

ABSTRACT

Router software is prone to bugs triggered by commands entered in the command line interface (CLI). In particular, over two-thirds of the CLI-triggered bugs in FreeRangeRouting (FRR) occur due to interactions between a pair of commands. Consequently, we propose a router CLI testing framework based on combinatorial testing. Applying our prototype to FRR's access-list commands takes less than a minute, and uncovered a bug that has been confirmed by the developers.

1 INTRODUCTION

Router software tends to be highly complex due to the intricate nature of distributed routing protocols and extensive flexibility offered by router configuration languages. For example, FreeRangeRouting (FRR) [3]—a widely-used open-source routing software suite for Linux/Unix platforms—has ≈600K lines of C code. Furthermore, there are >300 million possible commands that can be entered in the FRR command line interface (CLI) in configuration mode—which doesn't even account for nested configuration commands (e.g., commands entered after a "`router bgp`" command) or possible command parameter values (e.g., IP addresses).

This complexity makes router software prone to bugs. In particular, Yin et al.'s study of bug reports from four router software suites found the two most common triggers of bugs are configuration changes and CLI commands, triggering 63% and 52% of bugs, respectively [14]. Furthermore, approximately half of the lines of code in router software are associated with router management interfaces, including parsing configurations, the CLI, and logging.

Our own study of bugs in FRR corroborate these findings. We manually studied 162 FRR issues which were reported in the years 2017 through 2020. From these issues, we identified 67 issues that were triggered by configuration commands. We categorized these issues based on the command name (e.g., `access-list`, `route-map`), action (e.g., addition, deletion, replacement), and faulty behavior (e.g., incorrect deletion, duplication, crash). The most frequent triggers were "`router bgp`" sub-commands—e.g., `neighbor` and `address-family` commands—and `access-list` commands. Moreover, over two-thirds of the issues triggered by configuration commands occurred due to interactions between a pair of commands. For example, running a command to add an access control list (ACL) rule

```
access-list 1 seq 10 permit any
```

followed by a command to modify the rule

```
access-list 1 seq 10 deny 7.0.0.1/32
```

incorrectly results in the ACL containing both rules [1]

```
access-list 1 seq 10 permit any
access-list 1 seq 10 deny 7.0.0.1/32
```

Although numerous tools have been developed to detect errors in router configurations (e.g., [4, 5, 12]), no tools have been designed to assess whether the configuration commands are properly processed by the router software. Researchers have developed general-purpose testing tools for configurable systems (e.g., [8, 9, 13]), but (to the best of our knowledge) these have not been applied to router software. Instead, router software developers rely on simulation [2, 11], fuzzing [2], and symbolic execution [7] to detect bugs.

We aim to close this gap through the design of a thorough and efficient testing framework for router CLIs. Based on our aforementioned observation that over two-thirds of the FRR issues triggered by configuration commands occurred due to interactions between a pair of commands, our preliminary design (Section 2) is based on combinatorial testing [6, 10]. Applying our framework to FRR's `access-list` commands takes less than a minute, and we uncovered a bug in FRR's CLI that has been confirmed by the developers (Section 3).

2 APPROACH

Our approach to detect CLI bugs is pairwise testing of commands generated using combinatorial testing. Combinatorial testing avoids the scalability and coverage limitations of current router software testing approaches, and aligns with our observation that many router CLI bugs are triggered by pairs of commands (Section 1).

Router configurations contained nested (ordered) sets of commands. At each level of nesting, there are a certain number of valid commands that can be entered. Most router CLIs provide an easy way to list the allowable commands in a templated form. For example, one of the `access-list` commands supported by FRR is

```
access-list WORD [seq (1-4294967295)]
    <deny|permit> <A.B.C.D/M [exact-match]|any>
```

where "`[seq (1-4294967295)]`" is an optional *command parameter* and "`<A.B.C.D/M [exact-match]|any>`" and "`<deny|permit>`" are required parameters. Furthermore, the

latter is a *few-valued* parameter, whereas the former parameters are *many-valued* parameters.

Having templates with command parameters allows us generate a set of valid commands using combinatorial testing. We do this at each level of configuration nesting for every command-template, giving us a set of commands which we can then test pairwise by entering them at the CLI. For many-valued parameters, such as sequence numbers and IP addresses, the number of possibilities is massive. Consequently, we focus on critical boundary values—e.g., `0.0.0.0/0`, `10.0.0.0/16`—as well as adjacent/overlapping values—e.g., `1.0.0.0/8` and `2.0.0.0/8`, or `1.0.0.0/8` and `1.2.0.0/16`.

We must also address the challenge of determining if a certain pair of commands triggered a bug. Since many bugs result in normal operation of the router [14], and we are trying to detect bugs triggered by entering configuration commands at the CLI, we adopt the simple approach of comparing the running configuration of the router, as reported by the CLI (e.g., "`show running-config`"), with the expected configuration after entering the commands. However, router software typically adds default values for excluded optional parameter—e.g., the next unused sequence number in an `access-list`—and transforms values to a standard form—e.g., FRR transforms "`1.2.3.4/16`" to "`1.2.0.0 255.255.0.0`". Consequently, the commands in the running configuration may not match the commands entered in the CLI. To address this issue, we ignore optional parameters and convert IP addresses to a canonical form when comparing the running configuration with the entered commands.

Although we have found this approach is effective for detecting bugs (Section 3), there may be a mismatch between the running configuration reported by the router and the router's internal state, leading to missed/false bugs. More sophisticated analysis of commands' impact on routers' internal state is an important part of our future work (Section 4).

3 PRELIMINARY RESULTS

We have implemented a prototype of our testing framework in ≈450 lines of Python code, and applied it to FRR's `access-list` configuration commands, since this category of commands was one of the most frequent triggers of bugs in FRR (Section 1).

FRR has three `access-list` command templates:

```
access-list WORD [seq (1-4294967295)]
 <deny|permit> <A.B.C.D/M [exact-match]|any>
access-list WORD [seq (1-4294967295)]
 <deny|permit> <[host] A.B.C.D|A.B.C.D A.B.C.D>
access-list WORD [seq (1-4294967295)]
 <deny|permit> ip <A.B.C.D A.B.C.D|host A.B.C.D|
  any> <A.B.C.D A.B.C.D|host A.B.C.D|any>
```

We generated a set of 13530 valid `access-list` commands from these templates using combinatorial testing. After pairwise testing the set of generated commands, we found one bug which affects interactions between 294 pairs of access-list commands. The bug causes the incorrect modification of access-list commands where the old and the new forms of the command differ by the presence of an IP host address. The entire process from command-generation to testing for expected output took about 50 seconds, with the primary time sink being the entering and processing of commands in the CLI. Without combinatorial testing, we would have needed to check 339076 pairs of commands—and significantly more if we considered all possible network addresses and masks.

4 FUTURE WORK

For immediate future work, we will focus on extending our code to apply to other configuration commands (e.g., `route-map`, `(bgp) neighbor`). Furthermore, we will investigate other ways to detect a bug, especially in scenarios where the running configuration matches the expected configuration but the router has incorrect behavior. This includes examining other representations of the internal state of the router (e.g., 'show ip route') or sending probe-packets and comparing router behavior against expected behavior. Lastly, we will apply our tool on different routing software suites to demonstrate its generality, efficiency, and accuracy.

REFERENCES

[1] [n.d.]. Access-lists: change in behavior for sequence numbers and crash on delete - Issue #6747 - FRRouting. https://github.com/FRRouting/frr/issues/6747.

[2] [n.d.]. FRRouting Developer's Guide. https://docs.frrouting.org/projects/dev-guide.

[3] [n.d.]. FRRouting Project. https://frrouting.org.

[4] Anubhavnidhi Abhashkumar, Aaron Gember-Jacobson, and Aditya Akella. 2020. Tiramisu: Fast Multilayer Network Verification. In *USENIX Symposium on Networked Systems Design and Implementation (NSDI)*.

[5] Ryan Beckett, Aarti Gupta, Ratul Mahajan, and David Walker. 2017. A General Approach to Network Configuration Verification. In *SIGCOMM*.

[6] David M. Cohen, Siddhartha R. Dalal, Michael L. Fredman, and Gardner C. Patton. 1997. The AETG System: An Approach to Testing Based on Combinatiorial Design. *IEEE Trans. Software Eng.* 23, 7 (1997).

[7] Mihai Dobrescu and Katerina J. Argyraki. 2014. Software Dataplane Verification. In *11th USENIX Symposium on Networked Systems Design and Implementation (NSDI)*.

[8] Christopher Henard, Mike Papadakis, Gilles Perrouin, Jacques Klein, Patrick Heymans, and Yves Le Traon. 2014. Bypassing the Combinatorial Explosion: Using Similarity to Generate and Prioritize T-Wise Test Configurations for Software Product Lines. *IEEE Trans. Software Eng.* 40, 7 (2014), 650–670.

[9] Chang Hwan Peter Kim, Darko Marinov, Sarfraz Khurshid, Don S. Batory, Sabrina Souto, Paulo Barros, and Marcelo d'Amorim. 2013. SPLat: lightweight dynamic analysis for reducing combinatorics in

testing configurable systems. In *Joint Meeting of the European Software Engineering Conference and the ACM SIGSOFT Symposium on the Foundations of Software Engineering, (ESEC/FSE)*.

[10] D. Richard Kuhn, Dolores R. Wallace, and Albert M. Gallo. 2004. Software Fault Interactions and Implications for Software Testing. *IEEE Trans. Software Eng.* 30, 6 (2004), 418–421.

[11] Hongqiang Harry Liu, Yibo Zhu, Jitu Padhye, Jiaxin Cao, Sri Tallapragada, Nuno P. Lopes, Andrey Rybalchenko, Guohan Lu, and Lihua Yuan. 2017. CrystalNet: Faithfully Emulating Large Production Networks. In *26th Symposium on Operating Systems Principles (SOSP)*.

[12] Siva Kesava Reddy, Alan Tang, Ryan Beckett, Karthick Jayaraman, Todd D. Millstein, Yuval Tamir, and George Varghese. 2020. Finding Network Misconfigurations by Automatic Template Inference. In *USENIX Symposium on Networked Systems Design and Implementation (NSDI)*.

[13] Xudong Sun, Runxiang Cheng, Jianyan Chen, Elaine Ang, Owolabi Legunsen, and Tianyin Xu. 2020. Testing Configuration Changes in Context to Prevent Production Failures. In *USENIX Symposium on Operating Systems Design and Implementation (OSDI)*.

[14] Zuoning Yin, Matthew Caesar, and Yuanyuan Zhou. 2010. Towards understanding bugs in open source router software. *Comput. Commun. Rev.* 40, 3 (2010).