Metha:
Network Verifiers Need To Be Correct Too!

Rüdiger Birkner*, Tobias Brodmann*, Petar Tsankov, Laurent Vanbever, Martin Vechev
*equal contribution

nsg.ee.ethz.ch

NSDI’21
April, 12 2021

ETH zürich
With the rise of network analysis and verification tools, network outages should soon be a relic of the past...
With the rise of network analysis and verification tools, network outages should soon be a relic of the past…

…provided these tools make no mistakes
Building an accurate network analysis tool is extremely difficult (…if not impossible)

one has to accurately capture all protocols and their features

BGP, OSPF, IS-IS, EIGRP, …

for all vendors, devices and OSes

Cisco, Juniper, Arista, …

How can we help building accurate tools?
Metha systematically tests network analysis tools through automated configuration generation.
Metha: Automated Testing of Network Analyzers

1. Sensible configurations
   satisfying configuration dependencies

2. Systematic exploration
   covering the search space thoroughly

3. Evaluation
   finding bugs in the wild
Metha: Automated Testing of Network Analyzers

1. **Sensible configurations**
   satisfying configuration dependencies

   **Systematic exploration**
   covering the search space thoroughly

**Evaluation**
finding bugs in the wild
For effective testing, configurations must be **syntactically** and **semantically** valid

Configs need to adhere to a configuration syntax such that the devices/tools can parse them. Be consistent and coherent such that used resources are also defined. Allow for control-plane computations such that routes are exchanged.
Metha takes a two-stage approach to generate semantically valid configurations

```plaintext
interface FastEthernet0/0
  ip address 1.1.1.1/24

router bgp 100
  distance bgp 100 100 100
  redistribute static
  neighbor 1.1.1.2 remote-as 50
  neighbor 1.1.1.2 route-map XYZ out
  neighbor 1.1.1.2 next-hop-self

route-map XYZ permit 10
  match ip address prefixList
```

define a **base configuration**
set up basic infrastructure
provision resources

randomly add config features
activate **features**
choose **parameters**
Metha: Automated Testing of Network Analyzers

1. Sensible configurations
   satisfying configuration dependencies

2. Systematic exploration
   covering the search space thoroughly

3. Evaluation
   finding bugs in the wild
The search space of all configurations is prohibitively large

```ini
interface FastEthernet0/0
    ip address 1.1.1.1/24

router bgp 100
distance bgp 100 100 100
redistribute static
neighbor 1.1.1.2 remote-as 50
neighbor 1.1.1.2 route-map XYZ out
neighbor 1.1.1.2 next-hop-self

route-map XYZ permit 10
    match ip address prefixList
```

~16.5 million different options
To cope with the huge search space
Metha restricts it to few representative configurations

#1  boundary value reduction
    restrict parameter values

#2  combinatorial testing
    restrict feature combinations
#1  boundary value reduction
restrict parameter values

#2  combinatorial testing
restrict feature combinations
For every parameter, Metha considers only few representative values.

**boundary value reduction**

Reduces all parameters to boundary values:
- minimum, middle and maximum

Restricts the space by orders of magnitude:
- 8bit int needs 3 values instead of 256

Helps to actively test all features:
- unlike randomly choosing the values
#1  boundary value reduction
restrict parameter values

#2  combinatorial testing
restrict feature combinations
Metha creates a test suite that covers all pairwise feature interactions

Combinatorial testing defines a testing strategy, which is the input for config generation

tests pairwise feature interactions, but considers all of these interactions

```
  test #1 #2 #3 #4 #n
  ip ospf cost 1
  set metric 150
```
Metha: Automated Testing of Network Analyzers

1. Sensible configurations
   satisfying configuration dependencies

2. Systematic exploration
   covering the search space thoroughly

3. Evaluation
   finding bugs in the wild
Question #1
Does Metha manage to find real bugs?

Question #2
How do the components contribute to Metha’s effectiveness?
Implementation

7k lines of Python

github.com/nsg-ethz/Metha

Features

static routes, OSPF, BGP, route-maps
covering most common features

Oracle

virtualised GNS3 network with 4 routers
using Cisco 7200 and Juniper vMX images
Metha found bugs in all of the three tested tools

<table>
<thead>
<tr>
<th>Tool</th>
<th># bugs</th>
</tr>
</thead>
<tbody>
<tr>
<td>Batfish</td>
<td>29</td>
</tr>
<tr>
<td>C-BGP</td>
<td>3</td>
</tr>
<tr>
<td>NV</td>
<td>30</td>
</tr>
</tbody>
</table>
Only few bugs lead to crashes, while the majority leads to false analyses

<table>
<thead>
<tr>
<th></th>
<th># bugs</th>
<th>crash</th>
<th>silent</th>
</tr>
</thead>
<tbody>
<tr>
<td>Batfish</td>
<td>29</td>
<td>5</td>
<td>24</td>
</tr>
<tr>
<td>C-BGP</td>
<td>3</td>
<td>0</td>
<td>3</td>
</tr>
<tr>
<td>NV</td>
<td>30</td>
<td>5</td>
<td>25</td>
</tr>
</tbody>
</table>
feature coverage
as % of feature pairs

# discovered bugs

combinatorial testing: 20
boundary values: 17
semantic generation: 16
syntactic generation: 3
By definition, combinatorial testing achieves complete feature coverage.

![Graph showing feature coverage as % of feature pairs versus number of tests.]

- 20 combinational testing
d- 17 boundary values
- 16 semantic generation
- 3 syntactic generation

# discovered bugs

feature coverage as % of feature pairs

# tests
Semantic configuration generation is critical for Metha’s effectiveness.
Metha: Automated Testing of Network Analyzers

Metha generates semantically valid configs using a two-stage approach systematically covers the search space through restricting the search space provides actionable bug reports including a minimal config example