MED: The Monitor-Emulator-Debugger for Software-Defined Networks

Quanquan Zhi and Wei Xu
Institute for Interdisciplinary Information Sciences
Tsinghua University
Software-Defined Networks (SDN): promises and challenges

• SDN will simplify future network design and operation

• Bugs are common
  — Controller
  — Switch software
  — Race conditions

• Network Ops -> Systems DevOps
  — Command line -> programs
  — Lacking of tools
  — Fast, repeatable
Monitor-Emulator-Debugger: A debug / testing tool for SDN DevOps

• A software Debugger
  — fast, repeatable, automated tools
  — addresses concurrency bugs

• Tightly coupled with physical network
  - Automatic physical network sync
MED architecture overview

Monitor

Emulator

Debugger
The monitor

• Snapshot (initialization)
  — Physical network topology (LLDP)
  — Initial forwarding table states

• Capture SDN state changes over time
  — Openflow messages to/from the SDN controller
  — E.g. packets-in, packets-out, rule installation/removal, and ports up/down events

• Sample data packets
  — Essential for replay/testing
The emulator: key ideas

• The key challenge
  — Emulating a blackbox controller from physical SDN

• Solution
  — Replay all Openflow messages captured => set to a time

• Question: In what order?
The emulator: operation

- **Online Operation**
  - Tracking mode
- **Offline Operation**
  - “Time Travel”

![Diagram of emulator operation]

- Initial setup → Tracking state (Online)
- Specified state (Offline)
- Replay
- State1 → StateN
The emulator: offline operations

- Set to a stable state at any time
- Emulate all possible ordering for concurrent events
The debugger

• A controller that injects messages into the replayed message stream

• “Apps” built on top of the emulator
  — Set to a specific time
  — An external controller interface

• Example debugger apps
  — Packet tracer
  — Loop and reachability checker
  — Forwarding table checker
  — Race conditions detector
Example debugger app 1: Packet Tracer (PT)

Outputs:
1. A packet’s entire path through the network
2. Which forwarding rule is used on each hop
Example debugger app 2: Loop and Reachability Checker (LRC)

Asserts:

• The packet forwarding has no loop

-- AND --

• The packet reaches the destination

• Works online or offline
Example debugger app 3: Race Condition Detector (RCD)

Asserts:

- In **ANY** possible concurrent state, there is no loop or blackhole

- Expensive? Can trivially run in parallel with multiple emulators
Example debugger app 4: Table Checker (TC)

Asserts:

• The forwarding tables on physical switches are the same as those in the emulator
Evaluation

• Performance
  - Emulator initialization
  - Packet Tracing (PT) performance

• Case studies
  - Bugs on physical switch software
  - Race condition analysis
Experiment setup

- 20 switches network, typical DCN topology
  - Pica8 P-3298
  - 30,000 OpenFlow total (~1,500 rules per switch)
Initial setup performance

Discover physical topo + setup emulator topo

Dump all flow tables from switches

Install all flow tables entries to Emulator (30K rules)

4.9 sec 0.54 sec 12.2 sec

State changed during the setup? Redo until done.
Packet Tracing (PT) performance

• Random routing

• Performance of tracing paths with different lengths

<table>
<thead>
<tr>
<th># hops</th>
<th>2</th>
<th>4</th>
<th>6</th>
<th>8</th>
<th>10</th>
</tr>
</thead>
<tbody>
<tr>
<td>% of test data</td>
<td>10.6%</td>
<td>13.2%</td>
<td>57.9%</td>
<td>16.2%</td>
<td>2.1%</td>
</tr>
<tr>
<td>Time taken (ms)</td>
<td>0.626</td>
<td>1.536</td>
<td>2.828</td>
<td>3.532</td>
<td>5.001</td>
</tr>
</tbody>
</table>
Real world bug in switch software

Pica8 switch flow table:

```
NXST_FLOW reply (xid=0x4):
None
```

MED OVS flow table:

```
NXST_FLOW reply (xid=0x4):
1)cookie=0x0, duration=4.723s, table=3, n_packets=n/a, n_bytes=204, priority=2,in_port=28,dl_dst=00:e0:ed:2e:12:86 actions=output:27
2)cookie=0x0, duration=4.714s, table=3, n_packets=n/a, n_bytes=102, priority=2,in_port=27,dl_dst=00:e0:ed:21:d8:be actions=output:28
3)cookie=0x0, duration=10.608s, table=3, n_packets=n/a, n_bytes=230, priority=0 actions=CONTROLLER:65535
```

Bug in PicOS-OVS 2.3

“A GRE port is injecting ARP request packets back to the same port. The expected results is to forward all packets except the GRE port.”

Non-deterministic states in the network due to concurrent messages

• Which switch processed the message first?
  — Sometimes we do not know
  — Can be ok, but can mean problems
Race condition example

Should we enforce the ordering?

Are we enforcing them correctly?

[1] Xin Jin, Hongqiang Harry Liu, Rohan Gandhi, Srikanth Kandula, Ratul Mahajan, Ming Zhang, Jennifer Rexford, Roger Wattenhofer, Dynamic Scheduling of Network Updates, SIGCOMM, 2014
## Race condition detector example (cont’d)

<table>
<thead>
<tr>
<th>Operation</th>
<th>Packet loss</th>
</tr>
</thead>
<tbody>
<tr>
<td>A-&gt;B-&gt;C</td>
<td>N</td>
</tr>
<tr>
<td>A-&gt;C-&gt;B</td>
<td>Y</td>
</tr>
<tr>
<td>B-&gt;A-&gt;C</td>
<td>N</td>
</tr>
<tr>
<td>B-&gt;C-&gt;A</td>
<td>Y</td>
</tr>
<tr>
<td>C-&gt;B-&gt;A</td>
<td>Y</td>
</tr>
<tr>
<td>C-&gt;A-&gt;B</td>
<td>Y</td>
</tr>
</tbody>
</table>
Conclusion

• A step bring in the software testing / debugging tools to SDN
  • Fast, reproducible
  • Single step tracing with packets
  • Debugging concurrency problems
• Emulates physical network
• Evaluation on an SDN with 20-switches

Wei Xu <weixu@tsinghua.edu.cn>
Backup slides
MED functions

MED: a useful tool to debug problems in SDN

- Create an emulator that can be set to the network state at any given point of time
- Trace the forwarding paths and the flow table entries used along the path, for each individual data packets
- Capture and find the cause of common SDN problems: Loop, Reachability failure and Race Conditions
Performance: inserting rules