SPIRIT: A Framework for Profiling SDN

November, 10, 2015

Heedo Kang, Seungsoo Lee, Chanhee Lee, Changhoon Yoon, Seungwon Shin

Graduate School of Information Security, School of Computing, KAIST
Outline

• Motivation

• Challenges

• SPIRIT Design

• Evaluation

• Conclusions
Motivation: Current status of SDN

- All the underlying network devices
- SDN disadvantage
  - Potential bottleneck (Control plane)
- To avoid disadvantage
  - Support the enough performance
- What has to be improved?
- What are the features?

Fundamental problem of performance on the SDN controller:

SDN controller and its application Code
Motivation: Current status of SDN

- SDN controller is also one kind of software
- Software performance:
  - How it is designed & implemented?

Floodlight Test Result

<table>
<thead>
<tr>
<th>Application</th>
<th>Throughput</th>
</tr>
</thead>
<tbody>
<tr>
<td>Learning Switch</td>
<td>130,804.35 responses/sec</td>
</tr>
<tr>
<td>Forwarding (routing path)</td>
<td>48,328.79 responses/sec</td>
</tr>
</tbody>
</table>

How can we understand the performance of the controller and its applications in detail?
Motivation: Profiling

• Profiling methodology
• Many existing profiling tools: Jprofiler, Vtunes etc...

However..

It does not provide comfortable environment for profiling SDN!
Motivation: What is the problem?

These are very time consuming and complicated tasks..

Start

End

Repeat a few times

- Run controller
- Construct the network
- Generate events
- Attach profiling tool

Environment Setting for Profiling

Find out Critical Path

Identify Hotspot

Analyzing Context of Hotspot

Identify Hotspot

These are very time consuming and complicated tasks..
Outline

• Motivation

• Challenges

• SPIRIT Design

• Evaluation

• Conclusions
Challenges

We need
a new automated profiling framework for SDN!

Ultimate goal: Design automated profiling framework for SDN
Does not require much of manual work!

Challenges:

• The Network Composition
• The Language-Specific
• The Information Sharing
Outline

• Motivation

• Challenges

• SPIRIT Design

• Evaluation

• Conclusions
SPIRIT Design: Overall architecture

- Consist of seven modules
- Operational Scenario
  - User Interface
  - Command Parser & Executor
  - Profiling Tool Agent
  - Network Emulator Agent
  - Flow Generator
  - Data Analyzer
  - Report Generator
- Detail
  - Data Analyzer
SPIRIT Design: Operational Scenario

Configurations:
- Target controller
- Target application
- Network topology
- Network status
- Amount of flow
- Testing time
- Number of testing repetition

User Interface
- Command Parser & Executor
- User Interface
- Report Generator
- Data Analyzer
- Flow Generator
- Network Emulator Agent
- Profiling Tool Agent

Controller Machine
- App1
- App2
- App3
- App n

Generate events
SPIRIT Design: Data Analyzer

- Parse stored XML files
- Find Critical path and hotspot
  - From the entire call graph
- Analyze the basic context
  - Based on each class specification
- To find Critical path & Hotspot
  - Apply simple custom algorithm
SPIRIT Design: Data Analyzer

• Algorithm - Find Critical path

Following the Longest total execution time node
SPIRIT Design: Data Analyzer

- Algorithms - Identify specific hotspot

\[ \text{I\_Time: Inherent time} \]

**Small Network**

- Method 1: I\_Time: 500ms
- Method 2: I\_Time: 1000ms
- Method 5: I\_Time: 700ms
- Method 8: I\_Time: 300ms

**Large Network**

- Method 1: I\_Time: 1000ms
- Method 2: I\_Time: 2500ms
- Method 5: I\_Time: 1400ms
- Method 8: I\_Time: 900ms
- Method 10: I\_Time: 5000ms

Finding the Most time consuming node
Outline

• Motivation
• Challenges
• SPIRIT Design
• Evaluation
• Conclusions
Evaluation: Test Environment

Controller Machine
CPU: Intel i5, MEM: 16G, NET: 1Gb/s,

Floodlight  |  ONOS  |  SPIRIT

Network Operating System (NOS)

Mininet v2.0

Network Emulating Machine
CPU: Intel i7, NET: 1Gb/s, MEM: 8G

- Target controller – Floodlight v1.0, ONOS v1.1
  - Default Application on each controller
- Composed four test cases
  - Emulating 4, 8, 16, 32 switches
- Consistently changing network environment & Topology Application running
  - Emulate an unstable network
  - Periodically connect & disconnect links
- One distinct flow per second
- Duration of each test case: 30 sec
Evaluation: Use Case – Topology App

- **Floodlight**

  Hotspot

  - Calculate the shortest path whenever topology is changed
    - Using Dijkstra algorithm
    - Spent 1274 milliseconds in 30 seconds (Latency)
    - 32% (Resource consumption)
Evaluation: Use Case – Topology App

- ONOS

**Critical Path**

1. onosproject.fwd.ReactiveForwarding.ReactivePacketProcessor.process
2. onosproject.net.topology.impl.TopologyManager.getPaths
3. onosproject.store.trivial.impl.SimpleTopoloyStore.getPaths
4. onosproject.store.trivial.impl.DefualtTopology.getPaths
5. onlab.graph.DijkstraGraphSearch.search

**Hotspot**

- Calculate the shortest path whenever Packet-In comes up
  - Using Dijkstra algorithm
  - Spent 8977 milliseconds in 30 seconds (Latency)
  - 54% (Resource consumption)
Evaluation: Use Case – Topology App

- Latency time and Resource consumption of each controller hotspot

- **Latency time**
  - ONOS has the higher latency time than Floodlight

- **Resource Consumption**
  - Increase exponentially (ONOS)
  - Increase linearly (Floodlight)

Adopted solution on ONOS incur much higher overhead than Floodlight!
## Evaluation: Overhead

<table>
<thead>
<tr>
<th>Throughput / controller</th>
<th>Not Running</th>
<th>Running</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>ONOS</td>
<td>Floodlight</td>
</tr>
<tr>
<td>Min</td>
<td>67,682</td>
<td>114,852</td>
</tr>
<tr>
<td>Max</td>
<td>72,998</td>
<td>121,982</td>
</tr>
<tr>
<td>Avg</td>
<td>70,217</td>
<td>117,689</td>
</tr>
</tbody>
</table>

SPIRIT is a testing tool and not running on real network at real time. Also, this overhead does not affect SPIRIT’s accuracy.
Outline

• Motivation

• Challenges

• SPIRIT Design

• Evaluation

• Conclusions
Conclusions

• We investigate the performance of default applications in some well-known controllers – Floodlight and ONOS

• We suggest new automated SDN profiling framework – SPIRIT

• In the future..
  • Enhance the algorithms
  • More comprehensive SDN applications on various controllers
  • Extending SPIRIT (OpenDaylight)
  • Support commercial controller(HP VAN SDN)
Thank You!
Any Questions?