BYZANTINE FAULT TOLERANT SOFTWARE-DEFINED NETWORKING (SDN) CONTROLLERS

KARIM ELDEFRAWY* AND TYLER KACZMAREK**

* INFORMATION AND SYSTEMS SCIENCES LAB (ISSL), HRL LABORATORIES
** UNIVERSITY OF CALIFORNIA IRVINE (UCI), WORK CONDUCTED WHILE AT HRL.

KMELDEFRAWY@HRL.COM
TKACZMAR@UCI.EDU
Overview of Talk

• Introduction and Motivation
• SDN Tools
• BFT-SMaRT Consensus Protocol
• BFT SDN System Design
• BFT SDN Controller Prototype Performance Results
• Improvements and Future Direction
Software-Defined Networking (SDN)

- Network Function Virtualization (NFV)
- Decouples Forwarding Operations from Control Decisions
- Introduces centralized controller
  - Controller dictates forwarding rules
  - Routers act as dumb switches
- Openflow emerged as de facto SDN protocol

Typical SDN Architecture (Wikipedia, 2016)
Openflow Protocol: Messages

- Openflow introduces unique message types for SDN control
  - PACKET_IN
    - Switch sends captured packet to controller on 2 occasions
      - Miss in the flow table
      - Explicitly specified as per match rule
  - FLOW_MOD
    - Controller modifies state of switch
  - PACKET_OUT
    - Controller directly injects packet into switch’s data plane
Openflow Protocol: Installing a Flow

- **S1’s Flow Table**
  - Rule: Dest = H1
  - Action: FWD to H1

- **S2’s Flow Table**
  - Rule: Dest = H2
  - Action: FWD to H2

Ping H2

---

© 2016 HRL Laboratories, LLC. All Rights Reserved
Openflow Protocol: Installing a Flow

S1’s Flow Table

<table>
<thead>
<tr>
<th>Rule</th>
<th>Action</th>
</tr>
</thead>
<tbody>
<tr>
<td>Dest = H1</td>
<td>FWD to H1</td>
</tr>
</tbody>
</table>

S2’s Flow Table

<table>
<thead>
<tr>
<th>Rule</th>
<th>Action</th>
</tr>
</thead>
<tbody>
<tr>
<td>Dest = H2</td>
<td>FWD to H2</td>
</tr>
</tbody>
</table>

H1 -> S1 -> S2 -> H2

PACKET_IN: no rule for destination “H2”
Openflow Protocol: Installing a Flow

S1’s Flow Table

<table>
<thead>
<tr>
<th>Rule</th>
<th>Action</th>
</tr>
</thead>
<tbody>
<tr>
<td>Dest = H1</td>
<td>FWD to H1</td>
</tr>
</tbody>
</table>

S2’s Flow Table

<table>
<thead>
<tr>
<th>Rule</th>
<th>Action</th>
</tr>
</thead>
<tbody>
<tr>
<td>Dest = H2</td>
<td>FWD to H2</td>
</tr>
</tbody>
</table>

FLOW_MOD: ADD Rule: Dest = H2, Action: FWD to S2
Openflow Protocol: Installing a Flow

S1’s Flow Table

<table>
<thead>
<tr>
<th>Rule</th>
<th>Action</th>
</tr>
</thead>
<tbody>
<tr>
<td>Dest = H1</td>
<td>FWD to H1</td>
</tr>
</tbody>
</table>

S2’s Flow Table

<table>
<thead>
<tr>
<th>Rule</th>
<th>Action</th>
</tr>
</thead>
<tbody>
<tr>
<td>Dest = H2</td>
<td>FWD to H2</td>
</tr>
</tbody>
</table>

Rule established: forward to S2
Openflow Protocol: Installing a Flow

<table>
<thead>
<tr>
<th>Rule</th>
<th>Action</th>
</tr>
</thead>
<tbody>
<tr>
<td>Dest = H1</td>
<td>FWD to H1</td>
</tr>
</tbody>
</table>

S1’s Flow Table

S2’s Flow Table

<table>
<thead>
<tr>
<th>Rule</th>
<th>Action</th>
</tr>
</thead>
<tbody>
<tr>
<td>Dest = H2</td>
<td>FWD to H2</td>
</tr>
</tbody>
</table>

Rule established: forward to S2
Our Motivation

- SDN architecture introduces single point of failure

- Unique threats facing SDN (Kreutz et. al 2013)
  - Attacks on Control Plane
  - Attacks on Controller Vulnerabilities
  - Few mechanisms ensuring trust between controller and management applications

- Current implementations do not account for failing or corrupted controllers
Our Contributions

• Designed and prototyped a Byzantine Fault Tolerant (BFT) SDN controller
  – Able to handle $f \leq 1$ corrupted controller out of $3f+1 \geq 4$ replica instances
  – Integrated into both OpenFlowJ and Beacon SDN controllers
  – Relies on BFT state machine replication (BFT-SMaRT)
OpenflowJ and Beacon

• OpenflowJ
  – Basic controller implementation
  – Close implementation of Openflow SDN standard
  – Java-based SDN controller

• Beacon
  – Optimizations on top of OpenflowJ
    – Leverages Parallelism
    – Modular
    – Extensible
BFT-SMaRT

- Developed by Bessani et. al, (Technical Report, 2013)
- Implemented in Java ([https://github.com/bft-smart/library](https://github.com/bft-smart/library))
- Close implementation of Lamport’s original schema
- Accepts $f$ faults with $3f+1$ total replicas
- Accepts batching
- Used for base-level implementation
BFT-SMaRT Consensus Protocol
• Request – Client multicasts request
• Pre-prepare – Primary orders request and multicasts unique number
BFT-SMaRT Consensus Protocol - Prepare

- Prepare – All replica multicast out Prepare
• Commit – Replicas multicast Commit; request added to committed certificate
BFT-SMaRT Consensus Protocol - Reply

- Reply – Replicas compute request and reply to client, result accepted once $f+1$ identical responses received
BFT Switch Design

• Based on Open vSwitch

• Same switch design used for both controller designs

• Introduced “Client Proxy” to interface with BFT Controllers
  – Switch sends PACKET_IN requests to Proxy, which formats into REQUEST messages as per BFT-SMaRT
  – Client Proxy formats RESPONSE messages from BFT controller into either PACKET_OUT or FLOW_MOD messages
BFT Controller Design – SimpleBFT

• Extension of OpenflowJ control logic
  – Responds to individual PACKET_IN messages
  – Issues identical PACKET_OUT and FLOW_MOD responses to OpenFlowJ

• Interfaces with Client Proxy instead of directly with switch
  – Reaches consensus on each given message
  – Primary changes after primary found faulty
• Identical modifications made to Beacon as OpenflowJ

• Extension of Beacon control logic
  – Responds to individual PACKET_IN messages
  – Issues identical PACKET_OUT and FLOW_MOD responses to OpenFlowJ

• Interfaces with Client Proxy instead of directly with switch
  – Reaches consensus on each given message
  – Primary changes after primary found faulty
BFT Controller Design - Architecture Changes

- Typical Architecture (2 switches, 2 hosts)
BFT Controller Design – Architecture Changes

• Modified Architecture (2 switches, 2 hosts)

H1 → S1 → Proxy 1 → Controller Replicas R1-R4 → Proxy 2 → S2 → H2
Performance Testing

- Tested in Cbench on an HP EliteBook 8560W
  - Windows 7, 64 bit
  - 4096MB RAM
  - Intel Core i7
    - 2.7GHz
    - 4 CPUs

- Simulated binary tree network of depth 5 in mininet
## Performance Results

<table>
<thead>
<tr>
<th>Controller</th>
<th>End-to-End Setup Duration</th>
<th>Flow Setup Delay</th>
<th>Flow Setup Rate (flow mods/second)</th>
</tr>
</thead>
<tbody>
<tr>
<td>OpenflowJ</td>
<td>376ms</td>
<td>9.44ms</td>
<td>106.9 fm/sec</td>
</tr>
<tr>
<td>SimpleBFT</td>
<td>775ms</td>
<td>31.7ms</td>
<td>59.3 fm/sec</td>
</tr>
<tr>
<td>Beacon</td>
<td>77ms</td>
<td>0.5ms</td>
<td>550.6 fm/sec</td>
</tr>
<tr>
<td>BeaconBFT</td>
<td>475ms</td>
<td>14.5ms</td>
<td>87.0 fm/sec</td>
</tr>
</tbody>
</table>
Performance Issues

- Slowdown for OpenFlowJ much smaller than for Beacon
- Beacon leverages parallelism for performance gains
- BeaconBFT unable to leverage parallelism
  - Total ordering required for BFT operation
Speeding up the Switch

• Batch Client requests
  – Minimize overhead per PACKET_IN message

• Integrate BFT Switch with client
  – Removes reliance on proxy for communication
  – 1 less step in communication
Speeding up the Controller

- **Alternative BFT protocols**
  - Naïve implementation maximizes communication

- **Spinning BFT**
  - Allows multiple views
  - Greater parallelism support

- **Speculative BFT**
  - $f+1$ replicas running until fault occurs
  - Reduces messages needed by a factor of 3 in PREPARE and COMMIT steps
Speeding up the Network

- Scalability issues even with non-BFT SDN controllers in reactive mode

- Convert to proactive controller
  - Greatest costs are in pre-computing many flows
  - Minimize active PACKET_IN messages generated
  - Less real-time computation
Questions?