Safe and Flexible Controller Upgrades for SDNs

Karla Saur - University of Maryland (now at Intel Labs)
Joseph Collard - UMass Amherst
Nate Foster - Cornell University
Arjun Guha - UMass Amherst
Laurent Vanbever - ETH Zurich
Michael Hicks - University of Maryland
SDN Software Updates

SDN controllers must be periodically upgraded:

- Add features
- Improve performance
- Fix bugs

Updating controllers presents many issues:

- Downtime
- Incorrect behavior
- Loss of state
SDN Software Updates

- State stored in a controller determines behavior
- Ex: store assigned replicas for hosts using a load balancer

<table>
<thead>
<tr>
<th>Host</th>
<th>tcpSrc</th>
<th>Replica</th>
</tr>
</thead>
<tbody>
<tr>
<td>Client A</td>
<td>6782</td>
<td>1</td>
</tr>
<tr>
<td>Client B</td>
<td>8153</td>
<td>1</td>
</tr>
<tr>
<td>Client C</td>
<td>8728</td>
<td>2</td>
</tr>
</tbody>
</table>

Active Flows

![Active Flows Diagram](image-url)

- Server Replicas
- Switch
- External Hosts

![Bandwidth Graph](image-url)

- Time (seconds)
- Bandwidth (Mbits/s)
Existing Approaches

- **Simple restart**: Halt old controller and begin executing a fresh copy of the new controller
  - Existing state lost, rules on the switches must be wiped

### Active Flows

<table>
<thead>
<tr>
<th>Host</th>
<th>tcpSrc</th>
<th>Replica</th>
</tr>
</thead>
<tbody>
<tr>
<td>Client A</td>
<td>6782</td>
<td>1</td>
</tr>
<tr>
<td>Client B</td>
<td>8153</td>
<td>1</td>
</tr>
<tr>
<td>Client C</td>
<td>8728</td>
<td>2</td>
</tr>
</tbody>
</table>

![Bandwidth Graph]

**Server Replicas**

**Switch**

**External Hosts**
Existing Approaches

- **Simple restart**: Halt old controller and begin executing a fresh copy of the new controller
  - This may cause disruption or incorrect behavior

### Active Flows

<table>
<thead>
<tr>
<th>Host</th>
<th>tcpSrc</th>
<th>Replica</th>
</tr>
</thead>
<tbody>
<tr>
<td>Client A</td>
<td>6782</td>
<td>2</td>
</tr>
<tr>
<td>Client B</td>
<td>8153</td>
<td>1</td>
</tr>
<tr>
<td>Client C</td>
<td>8728</td>
<td>2</td>
</tr>
</tbody>
</table>

---

![Diagram of Server Replicas and External Hosts](image-url)

![Bandwidth Graph](image-url)

**Update**

Bandwidth

Time (seconds)
Existing Approaches

- **Simple restart**: Halt old controller and begin executing a fresh copy of the new controller.
- **Record and replay**: Stored network events are replayed to “warm up” the new controller.

Replaying the recorded events causes the connections to be evenly distributed across replicas.
Existing Approaches

- **Simple restart**: Halt old controller and begin executing a fresh copy of the new controller

- **Record and replay**: Stored network events are replayed to “warm up” the new controller

- **Rule-sourced reconstruction**: State is initialized from service-specific code that queries rules on switches
  - Works in some cases (when all necessary controller state is encoded into switch rules), but laborious and error prone
  - Does not work in some cases:
    - Switch failure
    - Incomplete information on switches, such as if the message history needed for stateful firewalling is absent
State in Software Updates

- To avoid disruption, controllers must have correct post-upgrade execution state
- State may have different representation in new version
Dynamic Updates with Morpheus

- Morpheus: *update by state transfer*

- Externalizes the state, and insures apps always have a consistent view of the state, even during upgrade

- Provides a DSL to assist generating the state transformation functions when necessary

- State transformation is performed lazily

- Experiments show seamless upgrades at low overhead and little programmer effort
Morpheus Architecture

<table>
<thead>
<tr>
<th>NIB</th>
</tr>
</thead>
<tbody>
<tr>
<td>fw_allowed</td>
</tr>
<tr>
<td>10.0.0.1:3456</td>
</tr>
<tr>
<td>10.0.0.2:80</td>
</tr>
</tbody>
</table>
Morpheus Architecture

```
NIB
fw_allowed
10.0.0.1:3456
10.0.0.2:80
```

Morpheus

Routing
Topology
Firewall

Get
Put
Morpheus Architecture

Morpheus

NIB

fw_allowed
10.0.0.1:3456
10.0.0.2:80

Get

Put

Routing

Topology

Firewall

Platform

Platform

Switch

Switch

Switch

_configs

Events
Enabling Dynamic Updates

• Programmer must modify apps to be preemptible
  • Apps must exit quickly when notified of update
  • Ex: Blocking I/O or long running sleeps may prevent apps from promptly responding to notifications from update coordinator

• Programmer explicitly transforms old state with a DSL
  • DSL generates state transformation code
  • Add, Delete, Update, Rename JSON fields
  • Can also opt to write state transformer manually
Stateful Firewall Update

Example: Switch connected to trusted internal hosts and untrusted external hosts.

Firewall Policy:

• If the controller receives a packet from a trusted internal host, it records pair and allows flow

• If the controller receives a packet from an external host first, it logs the connection attempt and drops
Example: Stateful Firewall Update

Simple restart: service disruptions/correctness issues:

- Old state lost on restart, may not be accurately recreated
Example: Stateful Firewall Update

Simple restart: service disruptions/correctness issues:

- Old state lost on restart, may not be accurately recreated
- Traffic allowed by old controller can be blocked by the new

<table>
<thead>
<tr>
<th>Connections</th>
</tr>
</thead>
<tbody>
<tr>
<td>Src</td>
</tr>
<tr>
<td>Client A</td>
</tr>
<tr>
<td>Client A</td>
</tr>
<tr>
<td>Client X</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Connections</th>
</tr>
</thead>
<tbody>
<tr>
<td>Src</td>
</tr>
<tr>
<td>Client A</td>
</tr>
<tr>
<td>Client A</td>
</tr>
<tr>
<td>Client X</td>
</tr>
</tbody>
</table>

Incoming from Client Z, existing rule lost in update
Example: Stateful Firewall Update

Simple restart: service disruptions/correctness issues:

- Old state lost on restart, may not be accurately recreated
- Traffic allowed by old controller can be blocked by the new
- For correctness, must delete flows from all switches

<table>
<thead>
<tr>
<th>Src</th>
<th>Dst</th>
<th>Allow?</th>
</tr>
</thead>
<tbody>
<tr>
<td>Client A</td>
<td>Client Y</td>
<td>Y</td>
</tr>
<tr>
<td>Client A</td>
<td>Client Z</td>
<td>Y</td>
</tr>
<tr>
<td>Client X</td>
<td>Client A</td>
<td>N</td>
</tr>
</tbody>
</table>

Connections

<table>
<thead>
<tr>
<th>Src</th>
<th>Dst</th>
<th>Allow?</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Connections

<table>
<thead>
<tr>
<th>Src</th>
<th>Dst</th>
<th>Allow?</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Switch Rules

Switch Rules

Switch Rules

Switch (empty)

Switch (empty)

Switch (empty)
Update by state transfer with Morpheus:

- Morpheus provides direct access to critical state
- Programmer may provide a state transformer function
- Updates happen lazily as the data is accessed

### Example: Stateful Firewall Update

<table>
<thead>
<tr>
<th>Src</th>
<th>Dst</th>
<th>Allow?</th>
</tr>
</thead>
<tbody>
<tr>
<td>Client A</td>
<td>Client Y</td>
<td>Y</td>
</tr>
<tr>
<td>Client A</td>
<td>Client Z</td>
<td>Y</td>
</tr>
<tr>
<td>Client X</td>
<td>Client A</td>
<td>N</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Src</th>
<th>Dst</th>
<th>Allow?</th>
</tr>
</thead>
<tbody>
<tr>
<td>Client A</td>
<td>Client Y</td>
<td>Y</td>
</tr>
<tr>
<td>Client A</td>
<td>Client Z</td>
<td>Y</td>
</tr>
<tr>
<td>Client X</td>
<td>Client A</td>
<td>N</td>
</tr>
</tbody>
</table>
Morpheus DSL Example

Firewall update adds the ability to timeout connections (and uninstall their forwarding rules) after inactivity between the two hosts.

**Firewall Entry Version 1**

```json
{"trusted_ip": "10.0.0.1",
"trusted_port": 3456,
"untrusted_ip": "10.0.0.2",
"untrusted_port": 80 }
```

**Firewall Entry Version 2**

```json
{"trusted_ip": "10.0.0.1",
"trusted_port": 3456,
"untrusted_ip": "10.0.0.2",
"untrusted_port": 80,
"last_count": 0,
"time_created": 1426167581 }
```
Morpheus DSL Example

Firewall Entry Version 1

```json
{"trusted_ip": "10.0.0.1",
"trusted_port": 3456,
"untrusted_ip": "10.0.0.2",
"untrusted_port": 80 }
```

Firewall Entry Version 2

```json
{"trusted_ip": "10.0.0.1",
"trusted_port": 3456,
"untrusted_ip": "10.0.0.2",
"untrusted_port": 80,
"last_count": 0,
"time_created": 1426167581 }
```

Generates code to add and initialize "last_count" and "time_created" JSON fields
Updates and State Transformation

- DSL assists programmer in transforming state if necessary, usually only a few lines
- No state transformer necessary in some cases
- Looked at GitHub commits from 2012–2014 for OpenDaylight and POX controllers:
  - Many updates would involve adding and initializing a small number of new fields (due to state change)
  - Most updates were logic changes only and would not require data transformation (no state change)
Coordinated Updates with Morpheus

Example: Routing and Topology share some state

- Vers 1: Routing computes routes based on Topology data

- Vers 2: Topology queries the switches for stats and stores link capacity data. Routing uses this data to optimize paths.

Coordination Necessary:

- Routing will now expect the Topology stats information to be present in the new version

- Without coordination, Routing may see old data
Coordinated Updates with Morpheus

Update protocol coordinates updates across components

1) Notify apps to exit gracefully. Pause rule updates.

2) Install the update in the NIB.


4) Resume operation. NIB entries update as accessed.
Results: Routing and Topology Update

Tested with applications sharing the same data:

• Routing: computes per-source/destination routes
• Topology: stores information about hosts, switches, links, etc
Results: Routing and Topology Update

TCP iperf server session with two connected hosts, using Mininet HiFi with 1MBPS, 5ms delayed links:

- Both hosts down 10+ seconds vs single host down ~5 seconds
- Size of the outage dependent on topology
TCP iperf server session with two connected hosts, using Mininet HiFi with 1MBPS, 5ms delayed links:

Quiescence of update coordination:

<table>
<thead>
<tr>
<th>Start</th>
<th>Apps Exit</th>
<th>Restart Begins</th>
<th>Routing Push</th>
<th>Topology Push</th>
<th>Platform Resume</th>
</tr>
</thead>
<tbody>
<tr>
<td>0.00s</td>
<td>0.05s</td>
<td>0.11s</td>
<td>1.67s</td>
<td>1.68s</td>
<td>1.70s</td>
</tr>
</tbody>
</table>
Experiments Summary

• Less disruption compared to cold restart and replay methods
  • Load balancer example showed no dropped connections with Morpheus vs dropped connections with other methods
Experiments Summary

- Less disruption compared to cold restart and replay methods
- Demonstrated no disruption or reconnections during firewall update
  - Prevents incorrect behavior due to lost state
Experiments Summary

- Less disruption compared to cold restart and replay methods
- Demonstrated no disruption or reconnections during firewall update
- Successfully demonstrated updating apps using shared program state data (ex: Routing and Topology)
  - Ensures all applications see the newest data only
Morpheus Conclusions

Supports updates to SDN controller by state transfer:

- Provides direct access to the relevant state in the running controller; initializes the new controller’s state as a function of the existing state.

- Experiments show seamless upgrades at low overhead and little programmer effort, while prior approaches would result in disruption, incorrect behavior, or both.