HotSwap: Correct and Efficient Controller Upgrades for Software-Defined Networks

Laurent Vanbever
vanbever@cs.princeton.edu

HotSDN
August, 16 2013

Joint work with
Joshua Reich, Theophilus Benson, Nate Foster and Jennifer Rexford
HotSwap: Correct and Efficient Controller Upgrades for Software-Defined Networks

1. Today’s upgrades
   disruptive & incorrect

2. The HotSwap system
   record, replay, swap

3. Scalability & correctness
   filter & specify
HotSwap: Correct and Efficient Controller
Upgrades for Software-Defined Networks

1 Today’s upgrades
   disruptive & incorrect

The HotSwap system
   record, replay, swap

Scalability & correctness
   filter & specify
As any piece of complex software, SDN controller must be frequently upgraded

SDN controllers must be upgraded to

- fix bugs
- improve performance
- deploy new features or applications
As any piece of complex software, SDN controller must be **frequently** upgraded

<table>
<thead>
<tr>
<th>SDN controller</th>
<th># releases</th>
<th># commits</th>
<th>(over 2 years)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Pox</td>
<td>3*</td>
<td>1349</td>
<td></td>
</tr>
<tr>
<td>Floodlight</td>
<td>7</td>
<td>2106</td>
<td></td>
</tr>
<tr>
<td>Ryu</td>
<td>15</td>
<td>897</td>
<td></td>
</tr>
<tr>
<td>Trema</td>
<td>33</td>
<td>2670</td>
<td></td>
</tr>
</tbody>
</table>

source: GitHub

* Pox uses branches instead of releases
As any piece of complex software, SDN controller must be frequently upgraded

<table>
<thead>
<tr>
<th>SDN controller</th>
<th># releases</th>
<th># commits</th>
<th>(over 2 years)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Ryu</td>
<td>15</td>
<td>897</td>
<td></td>
</tr>
<tr>
<td>Trema</td>
<td>33</td>
<td>2670</td>
<td></td>
</tr>
</tbody>
</table>

source: GitHub

How is it done today?
SDN controllers are usually upgraded by rebooting the controller on the new version
SDN controllers are usually upgraded by restarting the controller on the new version.

During a controller restart, any
- network failure
- rule timeout
- diverted packet

is ignored.
SDN controllers are usually upgraded by restarting the controller on the new version.

After a restart, the controller:

- resets all network forwarding state to prevent inconsistencies leading to **losses** and **delays**
- recreates its state according to the *current* network traffic leading to **bugs**
SDN controllers are usually upgraded by rebooting the controller on the new version. After a reboot, the controller leads to losses and delays by resets all network forwarding state to prevent inconsistencies, recreates its state according to the current network traffic leading to bugs. Is it really a problem?
Restarting a controller can create network-wide disruption
We stop the controller after 15 seconds
We restart it controller after 20 seconds
Soon after the controller restart, the network suffered from important *network-wide* losses.
Restarting a controller can create bugs
Let’s restart a controller running a stateful firewall which only allows connection initiated from the inside.
Let’s restart a controller running a stateful firewall which only allows connection initiated from the inside.

![Diagram showing network setup with a controller, host 1, host 2, and forwarding table entries.]

- **Controller**
- **Host 1**
- **Host 2**
- **Forwarding table**
  - 10 H1 → H2 fwd
  - 05 H2 → H1 fwd

**Internet**

---

The diagram illustrates the network setup with a controller acting as a stateful firewall, allowing connections initiated from the inside only.
Upon restart, the controller wipes out all the forwarding entries.
Upon restart, the controller wipes out all the forwarding entries.
Ongoing flows for which externally originated packets are received first will get dropped by the controller.
Ongoing flows for which externally originated packets are received first will get dropped by the controller.
Ongoing flows for which externally originated packets are received first will get dropped by the controller.

Restarting the controller can cause **allowed** traffic to be **blocked**.
Ongoing flows for which externally originated packets are received first will get dropped by the controller.

Restarting the controller can also cause **forbidden** traffic to be **allowed**.
HotSwap: Correct and Efficient Controller Upgrades for Software-Defined Networks

Today’s upgrades
- disruptive & incorrect

2 The HotSwap system
- record, replay, swap

Scalability & correctness
- filter & specify
HotSwap warms up the upgraded controller before giving it control over the network.

Recreate state in the upgraded controller in a controlled fashion, guaranteeing correctness.

Keeping as much traffic in the network avoiding network-wide disruptions.

Tolerating different control and forwarding behavior between the new and old controller.
HotSwap is a hypervisor that sits between the network and the controller.
HotSwap proceeds in four stages: *record, replay, compare & replace*
In the *record* stage, HotSwap maintains a copy of the network state.
When an upgrade is initiated, HotSwap sets the upgraded controller as slave.

Only the master controller can write to the network.
HotSwap then *replays* the recorded network events against the upgraded controller.
During the *replay*, HotSwap records the forwarding rules generated by the upgraded controller.
Once the *replay* is completed, HotSwap computes the deltas between the initial and upgraded rules.
In the *replace* stage, HotSwap sets the upgraded controller as master and installs the deltas.
HotSwap finally removes the initial controller and re-enters the *record* stage.
HotSwap performs upgrade in a disruption-free manner
Using HotSwap, not a single packet is lost during the upgrade.
HotSwap: Correct and Efficient Controller Upgrades for Software-Defined Networks

Today’s upgrades
disruptive & incorrect

The HotSwap system
record, replay, swap

3  Scalability & correctness
filter & specify
Recording all network events does not scale
Recording all network events does not scale

... but is not **needed**!
Most stateful controllers only require \textit{some} events to be replayed
The number and type of events to be recorded depend on the controller category ...
... whether their state depend on the actual traffic being exchanged
... whether their state depend on the last network event or on an history of events.

Event dependency

Last          History

Network-Traffic Dependency

Yes

No
Network-Traffic Dependency

- Yes
  - Learning-Switch
  - Stateful Firewall

- No
  - Shortest-Path Routing
  - Reliable Routing

Event dependency:
- Last
- History
HotSwap provides a query language to filter stream of events at record and replay time.
What does it mean for an upgrade to be correct?
When we upgrade from v1 to v2,

We would like the network to behave as if v2 had been running since the beginning
When we upgrade from v1 to v2,

We would like the network to behave as if v2 had been running since the beginning.

What does it mean?
When we upgrade from v1 to v2,

We would like the network to behave as if v2 had been running since the beginning.

What does it mean?  
- same forwarding rules?  
- same forwarding semantic?  
- eventual semantic consistency?
same forwarding rules?

It depends ...

same forwarding semantic?

eventual semantic consistency?
HotSwap verifies if the desired correctness criteria is met before swapping controllers.

The operator defines a relation that captures the acceptable differences on the controller outputs.

- $=$ same forwarding rules?
- $\cong$ same forwarding semantic?
- $\diamond$ eventual semantic consistency?
HotSwap: Correct and Efficient Controller
Upgrades for Software-Defined Networks

Today’s upgrades
disruptive & incorrect

The HotSwap system
record, replay, swap

Scalability & correctness
query language
HotSwap enables disruption-free and correct SDN controller upgrade

HotSwap

- works in practice
  first implementation on top of FlowVisor

- is highly general
  no assumption on the controller or on the application

- is easy to use
  minimum input from the network operator
HotSwap: Correct and Efficient Controller Upgrades for Software-Defined Networks

Laurent Vanbever
www.vanbever.eu

HotSDN
August, 16 2013

Joint work with
Joshua Reich, Theophilus Benson, Nate Foster and Jennifer Rexford