Traffic Management Applications for Stateful SDN Data Plane

Carmelo Cascone^, Luca Pollini*, Davide Sanvito*, Antonio Capone^
Goal

- Highlight shortcomings of current SDN-OpenFlow paradigm
- Present a new “stateful” data plane model
- Motivate this need with 2 application examples
  - Failure recovery
  - Forwarding consistency
OpenFlow recap

Logically-centralized control

Events from switches
- Topology changes,
- Traffic statistics,
- Arriving packets

Commands to switches
- (Un)install rules,
- Query statistics,
- Send packets

SMART!

DUMB!
Centralized control: we know the pros but...

• **Control latency**
  – Switch-controller RTT
  – Controller processing

• **Signaling overhead**
  – First packet to the controller (Internet dominated by very short flows)
  – Flow statistics gathering
Example: failure recovery in OpenFlow (1)

“Fast-failover”: Local reroute based on port status (OpenFlow 1.1+)

Weak! What if a local reroute in not available?
Example: failure recovery in OpenFlow (2)

- Can rely on controller intervention, but:
  - Long recovery latency (> 50ms)
    - detection + signaling + flow table update
  - Failure of control channel
  - Signaling congestion (e.g. multiple failures, disasters)
Towards a new behavioral data plane model

Stateless model (e.g. OpenFlow)

- Controller
  Global + local states

- Switch
  Stateless

Stateful (behavioral) model

- Controller
  Global states

- Switch
  Local states

Auto-adaption

Event notifications

1 static forwarding behavior

Multiple forwarding behaviors + adaptation rules

Control enforcing paradigm

Control delegation paradigm
Easier said than done

• We need a switch abstraction and API which is...
  – **High performance**: control tasks executed at wire-speed (packet-based events)
  – **Platform-independent**: consistent with vendors’ needs for closed platforms
  – **Low cost and immediately viable**: based on commodity HW

• Apparently, far beyond OpenFlow switches...

• Our finding: much closer to OpenFlow than expected
Our approach: OpenState

• **Idea: forward packets based on “flow states”**
  – *Maintained* by the switch
  – *Autonomously updated* as a consequence of local events (i.e. match, timers)

• **FSM-like forwarding model**

• **Minimal extension to OpenFlow**


OpenState: 2 table approach

Almost classic OpenFlow

<table>
<thead>
<tr>
<th>Key</th>
<th>State</th>
<th>Timeouts</th>
</tr>
</thead>
<tbody>
<tr>
<td>...</td>
<td>...</td>
<td>...</td>
</tr>
<tr>
<td>...</td>
<td>...</td>
<td>...</td>
</tr>
<tr>
<td>any</td>
<td>default</td>
<td>n/a</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>pkt</th>
<th>Match</th>
<th>Actions</th>
<th>Timeouts</th>
</tr>
</thead>
<tbody>
<tr>
<td>pkt</td>
<td>...</td>
<td>...</td>
<td>...</td>
</tr>
<tr>
<td>pkt</td>
<td>...</td>
<td>...</td>
<td>...</td>
</tr>
<tr>
<td>pkt</td>
<td>...</td>
<td>...</td>
<td>...</td>
</tr>
</tbody>
</table>

**set_state**(new_state, timeouts)
Flow key extractors

- **Used to match/access the state table**
  - Lookup or update phase

- **Scope = ordered list of header fields**
  - E.g. \{ip\_src\} $\rightarrow$ 32 bit flow key
  - E.g. \{eth\_src, eth\_dst\} $\rightarrow$ 96 bit flow key

```
State table
Key   State   Timeouts
...   ...     ...
...   ...     ...
any   default n/a
```

```
Flow table
Match   Actions   Timeouts
...     ...       ...
...     ...       ...
..      ...       ...
```

set state(new state, timeouts)
State table

• **Exact match on flow key**
  – Efficient implementation in RAM (vs. TCAM)

• **DEFAULT state if table miss**

• **Optional timeouts**
  – Idle or hard: equivalent to OpenFlow
  – \( \leq 1 \text{ms} \) granularity
  – Rollback state when timeout expires
  – Configured by `set_state()` action
Pipeline configuration

1) Set lookup-scope
2) Set update-scope
3) Populate flow table (FSM description)

Controller

Tables are stateless at switch boot. The controller can then configure one or more tables as stateful.

Classic OpenFlow table (stateless)

OpenState stateful stage
Open source: http://www.openstate-sdn.org

• **Running code: softswitch + controller**
  – Based on CPqD ofsoftswitch13, RYU
  – Initial support to Open vSwitch based on “learn()” action

• **Protocol specification**
  – OpenFlow 1.3 Experimenter Extension (PDF available)

• **Mininet-based application examples**
  – MAC learning, port knocking firewall, failure Recovery, DDoS detection and mitigation, load balancing

• **Download & try!**
Failure recovery
Failure recovery with OpenState

• Tags (e.g. MPLS labels) used to distinguish between **different forwarding behaviors**

• Upon failure, packets are **“bounced back”** with special tag – until matched against a node able to respond to that specific failure

• **Periodic probe** to re-establish forwarding on the primary path

⇒ No extra signaling/packet loss after failure detection

⇒ **Controller not involved (besides initial provisioning)**
Behavioral model (FSM)

• Each flow (lookup-scope) has an associated state (tag)
  – 0 (default) → all good, forward on primary path
  – Fi node i unreachable → forward on detour i-th
  – Pi node i must be probed → send 1 probe to node i
Normal conditions (no failures)

**State table**

<table>
<thead>
<tr>
<th>Key</th>
<th>State</th>
</tr>
</thead>
<tbody>
<tr>
<td>...</td>
<td>...</td>
</tr>
<tr>
<td>...</td>
<td>...</td>
</tr>
<tr>
<td><em>(any)</em></td>
<td>0</td>
</tr>
</tbody>
</table>

**Flow table**

<table>
<thead>
<tr>
<th>Match</th>
<th>Instructions</th>
</tr>
</thead>
<tbody>
<tr>
<td>src=1, dst=6, state=0</td>
<td>fwd(3)</td>
</tr>
<tr>
<td>...</td>
<td>...</td>
</tr>
<tr>
<td>...</td>
<td>...</td>
</tr>
</tbody>
</table>

lookup-scope=[eth_src, eth_dst]
update-scope=[eth_src, eth_dst]

Failure recovery

Example

PKT
state = 0
primary path
Failure recovery

Example

Packets “bounced back” in case of failure

<table>
<thead>
<tr>
<th>Match</th>
<th>Instructions</th>
</tr>
</thead>
<tbody>
<tr>
<td>src=1, dst=6</td>
<td>Group(1)</td>
</tr>
<tr>
<td>...</td>
<td>...</td>
</tr>
<tr>
<td>...</td>
<td>...</td>
</tr>
</tbody>
</table>

Group table

<table>
<thead>
<tr>
<th>ID</th>
<th>Type</th>
<th>Action buckets</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>FAST-FAILOVER</td>
<td>&lt;output(2)&gt;,</td>
</tr>
<tr>
<td></td>
<td></td>
<td>&lt;push_tag(F4), output(1)&gt;,</td>
</tr>
<tr>
<td></td>
<td></td>
<td>... ... ... ... ... ... ... ... ...</td>
</tr>
</tbody>
</table>

match tag F4
state → F4

port down

1 2 3 4 5
7 8
6

PKT
match tag F4
state → F4
//
port down
Failure recovery

Example

State transition at a pre-determined reroute node

<table>
<thead>
<tr>
<th>Match</th>
<th>Instructions</th>
</tr>
</thead>
<tbody>
<tr>
<td>…</td>
<td>…</td>
</tr>
<tr>
<td>src=1, dst=6, state=0</td>
<td>fwd(3)</td>
</tr>
<tr>
<td>src=1, dst=6, tag=F4</td>
<td>set_state(F4, hard_to=10s,</td>
</tr>
<tr>
<td></td>
<td>hard_rollback=P4)</td>
</tr>
<tr>
<td>…</td>
<td>…</td>
</tr>
</tbody>
</table>

Any packet

- <fwd(primary path)>
- <fwd(detour i-th)>
- <push_tag(Fi), fwd(detour i-th)>
- tag=Fi
- tag=Pi
- <fwd(detour i-th)>
- hard_timeout=δ
- <drop()>>
- <push_tag(Fi), fwd(detour i-th)>
- <push_tag(Pi); fwd(primary path)>

Any packet
Failure recovery

Detour path enabled

Example

<table>
<thead>
<tr>
<th>Match</th>
<th>Instructions</th>
</tr>
</thead>
<tbody>
<tr>
<td>src=1, dst=6,</td>
<td>push_tag(F4), fwd(7)</td>
</tr>
<tr>
<td>state=F4</td>
<td></td>
</tr>
</tbody>
</table>

any packet  
<fwd(primary path)>  
any packet  
<push_tag(Fi), fwd(detour i-th)>  
any packet  
<push_tag(Pi), fwd(detour i-th)>  
any packet  
<push_tag(Pi); fwd(primary path)>
Failure recovery

Example

State hard timeout to generate probe packets

Match

| ... | ... |
| ... | ... |
| ... | ... |
| ... | ... |
| src=1, dst=6, state=P4 | set_state(F4, hard_to=10s, hard_rollback=P4), <push_tag(F4), fwd(7)>, <push_tag(P4), fwd(3)> |
Failure recovery

Example

Primary path re-established

Match

... ... ...
... ... ...
... ... ...
... ... ...
tag=P4 set_state(0), drop()

any packet
<fwd(primary path)>

any packet
<push_tag(Fi), fwd(detour i-th)>

tag=Fi
<fwd(detour i-th)>

hard_timeout=δ

any packet
<push_tag(Fi), fwd(detour i-th)>
<push_tag(Pi); fwd(primary path)>

0
...
Fi
(link i down)

P
(link i probe)

...
Failure recovery

Example

Failure solved

```
<table>
<thead>
<tr>
<th>Match</th>
<th>Instructions</th>
</tr>
</thead>
<tbody>
<tr>
<td>src=1, dst=6, state=0</td>
<td>fwd(3)</td>
</tr>
<tr>
<td>...</td>
<td>...</td>
</tr>
<tr>
<td>...</td>
<td>...</td>
</tr>
</tbody>
</table>
```

PKT
state = 0
primary path
Load balancing
Load balancing in OpenFlow

• OpenFlow SELECT group entry
  – Packets forwarded using only one of multiple defined action buckets
  – Implementation left out to vendors (e.g. round robin, hash-based, etc)

• Usually implemented with ECMP-like hash-based schemes
  – Can’t decide on which header fields
  – Two or more elephant flows can collide on their hash, using the same path, hence creating a bottleneck
  – **Current OF solutions:**
    • reactive allocation (first packet to controller)
    • detection and relocation based on periodic flow statistic gathering
Better idea: flowlet-based load balancing

• **Originally introduced in FLARE (2007)***
  – Based on the idea of **switching bursts of packets** (flowlets) instead of pinning the whole flow to one path
  – **No packet reordering** if the idle time between bursts is larger than the maximum delay difference between parallels paths
  – No need to worry about elephant flows (burden shared among all paths)

\[
\text{No packet reordering if } \text{idle}_\text{time} > |\text{delay}_1 - \text{delay}_2|
\]

OpenState-based implementation

- States used to **distinguish between consecutive bursts/instances of the same flow**
- State idle timeouts to define the **lifetime of a forwarding decision**
  – sub-RTT scales for flowlet switching

**Legend:**
- **Event**
- **<per-packet actions>**

**State table**

<table>
<thead>
<tr>
<th>Key</th>
<th>State</th>
<th>Timeouts</th>
<th>Match</th>
<th>Instructions</th>
</tr>
</thead>
<tbody>
<tr>
<td>A, B, x, y</td>
<td>1</td>
<td>idle_to=δ</td>
<td>ip_dst=A, state=0</td>
<td>group(1)</td>
</tr>
<tr>
<td>...</td>
<td>...</td>
<td>...</td>
<td>ip_dst=B, state=0</td>
<td>group(2)</td>
</tr>
<tr>
<td></td>
<td>0</td>
<td>n/a</td>
<td>state=1</td>
<td>output(1)</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>state=2</td>
<td>output(2)</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>...</td>
<td>...</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>state=N</td>
<td>output(N)</td>
</tr>
</tbody>
</table>

**Flow table**

<table>
<thead>
<tr>
<th>Key</th>
<th>State</th>
<th>Timeouts</th>
<th>Match</th>
<th>Instructions</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>1</td>
<td>idle_to=δ</td>
<td>ip_dst=A, state=0</td>
<td>group(1)</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>ip_dst=B, state=0</td>
<td>group(2)</td>
</tr>
<tr>
<td></td>
<td>0</td>
<td>n/a</td>
<td>state=1</td>
<td>output(1)</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>state=2</td>
<td>output(2)</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>...</td>
<td>...</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>state=N</td>
<td>output(N)</td>
</tr>
</tbody>
</table>

**Group table**

<table>
<thead>
<tr>
<th>Group ID</th>
<th>Type</th>
<th>Action buckets</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>SELECT</td>
<td>&lt;set_state(1, idle_to=δ), output(1)&gt;,</td>
</tr>
<tr>
<td></td>
<td></td>
<td>&lt;set_state(2, idle_to=δ), output(2)&gt;,</td>
</tr>
<tr>
<td></td>
<td></td>
<td>...</td>
</tr>
<tr>
<td>2</td>
<td>SELECT</td>
<td>...</td>
</tr>
</tbody>
</table>
Example results: failure recovery

- **OF**: OpenFlow-based reactive approach, controller establishes backup path (with different switch-controller RTTs)
- **OS**: OpenState-based approach, packets bounced back upon failure

Optimal routing that minimizes bounce path based on:
A. Capone, C. Cascone, A. Q. Nguyen, and B. Sansò. “Detour planning for fast and reliable failure recovery in SDN with OpenState”.
In IEEE Design of Reliable Communication Networks (DRCN), March 2015
Example results: load balancing

- **OF**: controller-based reactive approach, new connections allocated by controller
- **OVS**: same as OF, but with faster switch (Open vSwitch)
- **OS**: OpenState-based approach

![Graph showing switch processing time vs. new connections rate with OF, OVS, and OS]
Conclusions

• **New stateful data plane model → OpenState**
  – Control «decided» at controller, «execution» delegated to switches’ data plane)

  – Openflow 1.3 extension

• **Failure recovery**
  – Switches pre-loaded with backup routing
  – MPLS labels use to perform failure signaling/path probing
  – Almost 0 packets lost after failure detection

• **Load balancing**
  – Can implement flowlet-based scheme
  – No need for elephant flows handling
  – Controller initially configure group table with optimal state idle timeouts
• Started January 2015

• Technical plans:
  – Propose OpenState for standardization
  – SW switch acceleration + HW prototype
  – Advanced security, forwarding and monitoring applications
  – Data plane verification
  – Real field large scale experimentation

http://www.beba-project.eu
Thanks!
carmelo.cascone@polimi.it