LINC – Open Source, Enterprise, Full-Functional OpenFlow Switch, written on Erlang

Dmitry Orekhov, Epam Systems
OpenFlow switch and Controller

OF Controller

OF Protocol

OF Switch

Secure channel

Group Table

Flow Table 0

Flow Table N

PORTS

OpenFlow Resource

OpenFlow Resource
Packet forwarding inside OpenFlow switch

- Packet may transferred to other table
- Packet header may be modified
- Packet may be forwarded to given port or just dropped
- Packet may be applied to given QoS

In-Packet → Out-Packet → Out-Packet → Out-Packet
Flow table entry: key elements

<table>
<thead>
<tr>
<th>Match Fields</th>
<th>Priority</th>
<th>Counters</th>
<th>Timeout</th>
<th>Cookies</th>
<th>Instruction set</th>
</tr>
</thead>
</table>

Match criteria:
- Ingress-port
- Ethernet MAC
- ARP
- IPv4 and IPv6
- TCP ports
- VLAN, MPLS etc.

Instruction:
- Go-To Table
- Modify Metadata
- Action Set {forward, apply QoS, drop, Apply to Group}
OpenFlow can be compared to the instruction set of a CPU. It specifies basic primitives that can be used by an external software application to program the forwarding plane of network devices, just like the instruction set of a CPU would program a computer system.
**Matching**

- **Packet In**
  - Start at table 0

- **Match in table n?**
  - Yes: Update counters, Execute instructions:
    - update action set
    - update packet/match set fields
    - update metadata
  - No: Goto-Table n?

- **Goto-Table n?**
  - Yes: Execute action set
  - No: Based on table configuration, do one:
    - send to controller
    - drop
    - continue to next table
## Group Table: “Aspects” of OpenFlow

<table>
<thead>
<tr>
<th>Group Identifier</th>
<th>Group Type</th>
<th>Counters</th>
<th>Action bucket</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>All</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>Select</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>Indirect</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>Fast Failover</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Groups represent sets of actions for flooding, as well as more complex forwarding semantics (e.g. multipath, fast reroute, and link aggregation). As a general layer of indirection, groups also enable multiple flows to forward to a single identifier (e.g. IP forwarding to a common next hop). This abstraction allows common output actions across flows to be changed efficiently.
OF Config – the new concept of OpenFlow Capable Switch

OF Capable Switch

logical ports

{ port 0 

{ lo

{ eth0

{ eth7

{ tap3

capable ports
**Layer**

- Content
- Operations
- RPC
- Transport Protocol

**Example**

- `<capable-switch>…</capable-switch>`
- `<get-config>,<set-config>,<notification>`
- `<rpc>,<rpc-reply>`
- SSH, TLS, BEEP, SOAP
Example

```xml
<capable-switch>
    <id>CapableSwitch0</id>

    <configuration-points>
        ...
    </configuration-points>

    <resources>
        ...
    </resources>

    <logical-switches>
        ...
    </logical-switches>
</capable-switch>
```
So what do we really have?

- OpenFlow capable switch looks like a container of many (probably thousands and tens of thousands) processes which are totally independent.
- Processes can be created/terminated in runtime, always, always!
- Connections – probably millions of them!
- One process must not crash another – no way!
- What about support new incoming OpenFlow versions? Do we need stop our switches?
- What about scalability? Who does take care of this?

- Last but not the least: Binary encoding/decoding process is too boring!
Erlang: Processes

✓ Process creation – spawn(fun task/0).
   It takes microseconds and down to hundreds bytes. Tens of thousands processes can be created per seconds.

✓ Process isolation.
   Every process is isolated inside Erlang VM.
   Processes communicate using queues of messages.
   You’re able to use global variables, but you won’t!

✓ Processes as an object.
   Local variables as an internal state
   Messages as methods
Erlang: Crash handling

✓ The main principle: “Let it fall”

✓ Don’t use try-catch

✓ Supervisors – special processes controlling another processes.
decode_match_field(<<Header:4/bytes, Binary/bytes>>) ->
  <<ClassInt:16, FieldInt:7, HasMaskInt:1,
  Length:8>> = Header,
Erlang: Development and Deployment

✓ OTP – A really reach library. Supervisors, evens handler, final-state machine – this is OTP.

✓ Erlang designed considering the fact that developers put bugs in the code – and try to stop developers to do it! Did I tell about immutability?

✓ Modules can be fixed and replaced in runtime – but don’t ask me how!
LINC switch

OF Configuration Point

OF-Config

LINC

Userspace implementation

OF Protocol

API (gen-switch)

HW

Kernel mode implementation
Is LINC REALLY able?

- 10,000 connections benchmark – Erlang looks great.
- All OpenFlow 1.3 features are implemented.
- ONF PlugFest – LINC was tested in topologies, together with enterprise switches and controllers.

- But we really didn’t test it as a switch, under high load.
- But we really out of System Integration Testing for LINC.
How you can try it?

- Linux box with Erlang, scons and pcap library
- git clone https://github.com/FlowForwarding/LINC-Switch
- cd LINC-Switch
- make rel

- You can refer to README on GitHub. Also, wiki contains document with simple examples and topologies
OpenNetworking Foundation (OpenFlow documents)
https://www.opennetworking.org/about/onf-documents

FlowForwarding
http://www.flowforwarding.org/

GitHub repository:
https://github.com/FlowForwarding/LINC-Switch

Testing framework for OpenFlow:
http://onlab.us/testing.html

And me, Dmitry Orekhov (Dmitry_Orekhov@epam.com)