Towards a distributed SDN control: Inter-platform signaling among flow processing platforms

SDN4FNS – EIT ICT Labs, Trento – 12 Nov 2013

Francesco Salvestrini (f.salvestrini@nextworks.it)
Gino Carrozzo (g.carrozzo@nextworks.it)
Nicola Ciulli (n.ciulli@nextworks.it)

Nextworks s.r.l., via Livornese 1027, Pisa, Italy
In the last decade networks, servers, storage technologies and applications have undergone significant changes

- virtualization, network overlays, orchestration, etc. eased the introduction of multiple network appliances
  - Firewalls, DPI, NAT, IDS, Traffic Scrubbers, etc.
- Improved network manageability and trend to new faster service deployments

Great part of these network functions are

- Implemented with (highly specialized) middleboxes
- Deployed at different points within the network
- Stateful / Flow-aware

Result is an Internet made of a concatenation of networks with distributed functions

- Services are typically implemented by the ordered combination of these service functions
Limitations

- **The current Internet is increasingly complex and rigid**
  - Vendor-locked hardware
  - Middleboxes strongly impacts reliability and manageability
  - Potentials to decrease network performance
  - Generally hard to manage

- **The effect is**
  - Hard to make dynamic network reconfiguration after service deployment
  - Difficult to customize net-architectures and functions based on user-demands changing over time

- **SDN, NFV, SFC and programmable network flow processing promise to unlock the current scenario**
  - However:
    - They are not yet fully standardized
    - Mostly support/focus on network functions for virtual-L2 switching over IP
      - E.g. VXLAN, GRE-NV, STT
    - Most SDN products adopt a centralized controller
Our goals

- **Design an architecture to**
  - Program the flow processing functions
  - Combine multiple flow processing functions
    » Conventional traffic flows can be processed at varying degrees of granularity
  - Control the end-to-end flow

- **The common ground is a Flow Processing Platform (FPP) by CHANGE project:**
  - Programmable
    » Can mangle data traffic related to multiple network flows
    » Can allow defining new processing primitives quickly
  - Scalable
    » Built from commodity hardware
  - With a common interface

- **We focused on the east-west interface**
  - To implement distributed end-to-end service provisioning among adjacent FPP
  - Where the reservations occur in the form of
    » Routing & switching rules
    » Flow processing “actions” (NF)
The Flow Processing Platform

- A set of Processing Modules (PMs)
- Each PM
  - They are lightweight VMs
    - No full-blown OS
    - Only a micro-kernel running the FP “application”
  - Performs a Flow Processing function
    - L2, L3, L4 …

- A set of Module Hosts (MHs)
- Each MH
  - Hosts multiple PMs
  - Provides fast backend primitives
    - PM ↔ PM comms
  - Is managed by a Control Module (CM)

- An OF switch connects the MHs

- A Platform Controller
  - Implements the platform control logic
    - Manages all the MHs (→ all PMs)
  - Provides the platform interface
The flow processing domain

- Various (domain local) identifiers exist:
  - Service Identifier (SID) → Univocally refers to a service
  - Platform Identifier (PID) → An unique identifier is assigned to each platform
  - Flow Identifier (FID) → The n-tuple used to identify a flow (entry in the OF switch)

- Flow (platform) → (PID, (FID-in, association, FID-out))
- Service (domain) → (SID, (PID, (FID-in, association, FID-out))+)
- Service (E2E) → (SID, (PID, (FID-in, association, FID-out)))+
The Flow Processing Route (FPR) encapsulates a concatenation of hops (FPROs).

It can be summarized as:

\[
\begin{align*}
\text{FPR} &::= \langle \text{SID} \rangle \ ( \langle \text{FPRO} \rangle \ | \ \langle \text{FPR} \rangle \ \langle \text{FPRO} \rangle ) \\
\text{FPRO} &::= \langle \text{PID} \rangle \ \langle \text{ACTIONS} \rangle \\
\text{ACTIONS} &::= ( \langle \text{ACTION} \rangle \ | \ \langle \text{ACTION} \rangle \ \langle \text{ACTIONS} \rangle ) \\
\text{ACTION} &::= ( \ \langle \text{ATTR\_FLOW} \rangle \ | \ \langle \text{REDIR\_FLOW} \rangle \ | \ \langle \text{PMS\_CONFIG} \rangle \ | \ \langle \text{OF\_CONFIG} \rangle \ | \\
&\quad \langle \text{TUN\_INGR} \rangle \ | \ \langle \text{TUN\_EGR} \rangle \ | \ ... \ ) \\
\text{PMS\_CFG} &::= ( \langle \text{PM\_CFG} \rangle \ | \ \langle \text{PMS\_CFG} \rangle \ \langle \text{PM\_CFG} \rangle ) \\
\text{PM\_CFG} &::= ... 
\end{align*}
\]

Supports the composition of platforms, actions and P2P bindings.
The inter-platform signaling arch

- Reference points: Service-UNI, Internal-NNI and Inter-AS NNI

- Functional Entities:
  - Service Manager
    » The flow processing domain orchestrator
    » Service instantiation and maintenance
    » Implements the Service-UNI
  - Service Composer
    » Service composition → domain FPR building
  - Signaling Manager
    » Implements the Internal-NNI
    » Relates to the Platform Controller
  - Service Broker
    » Implements the Inter-AS NNI
    » Applies policies for exporting information (FPROs) between different domains
  - Attraction Manager
    » FlowSpec, BGP and DNS based attractions
    » FlowSpec for matching level L3/L4 fields
  - Tunnel Broker
    » Data plane tunnels if not directly connected platforms
High level signaling messages

- Service-UNI: request/response[/notify] messages
- Internal-NNI:

<table>
<thead>
<tr>
<th>Activity</th>
<th>Message</th>
<th>Direction</th>
</tr>
</thead>
<tbody>
<tr>
<td>Setup</td>
<td>Service Setup Request</td>
<td>Up → Down</td>
</tr>
<tr>
<td></td>
<td>Service Setup Allocation</td>
<td>Down → Up</td>
</tr>
<tr>
<td></td>
<td>Service Setup Confirmation</td>
<td>Up → Down</td>
</tr>
<tr>
<td>Deletion</td>
<td>Service Deletion Request</td>
<td>Both</td>
</tr>
<tr>
<td></td>
<td>Service Deletion Response</td>
<td>Both</td>
</tr>
<tr>
<td>Modification</td>
<td>Service Modification Request</td>
<td>Both</td>
</tr>
<tr>
<td></td>
<td>Service Modification Response</td>
<td>Both</td>
</tr>
<tr>
<td>Notification</td>
<td>Notify</td>
<td>Both</td>
</tr>
<tr>
<td></td>
<td>Notify Acknowledge</td>
<td>Both</td>
</tr>
</tbody>
</table>

- Inter-AS NNI almost as Internal-NNI
  - Mangling, filtering and injection of protocol objects
Inter-platform signaling components

- Three components have been designed and developed
  - Service Manager
    - CLI based interface for FPR /signaling messages building
    - Flow processing composition
      - Embeds the Service Composer as a stand-alone application
    - Internal-NNI
  - Signaling Manager
    - Internal-NNI
    - Interacts with the platform
  - Service Broker
    - Inter-AS NNI
    - CLI based interface for service mangling specification
      - FPROs suppression, injection and filtering rules specification

- All the components have a two layers architecture
  - Upper layer → component-specific logic
  - Lower layer → common to all components
Control Channels / Sign. Adjacencies

- **The lower layer**
  - Separates the signaling application from transport
  - Handles
    - Reliability
    - Fragmentation
    - Congestion control
    - Message integrity
  - Manages the control channels (CC)
    - On-demand setup/tear-down
  - Manages the signaling adjacencies (SA)
    - For easy reasoning about NHOP/PHOP
The split design in practice

- We adopted NSIS ([nsis-ka.org](http://nsis-ka.org))
  - Layered architecture: NSLP + NTLP

- **GIST (General Internet Signaling Transport)**
  - Purposes:
    - Node discovery
      - Next node discovery decoupled from messages delivery
    - Message
      - Routing
      - Transport
  - Provides transport & security to NSLP

- **NTLP → as it is**

- **NSLP → from scratch over GIST**
Signaling Manager – High Level Arch

- **Two layers:**
  - `signald`
  - `transportd`

- **FPR Mgmt:**
  - FPR/FPROs handling

- **Service Management**
  - Implements the data-model
    - Stores
      - Services objects
      - FPROs

- **Adaptation logic**
  - Interacts with the Platform Controller
    - FP Services instantiations / removal
    - Reacts to events, e.g.
      - PMs failures

- **FSMs:**
  - Protocol-FSM → `signald`
  - CC-SA-FSM → `transportd`
Prototype software

- Each component code is loosely split into different layers
  - signaling application / transport
  - The transport has various “implementations”
    - The CC-SA NSLP
    - A simulation layer used for testing and debugging purposes

- The SW entities can be easily extended:
  - Signaling applications → Python
    - Python objects provide automatic message / objects ser-des
  - Transport layer → C++
  - C++/Python bindings produced with SWIG (www.swig.org)
    - Almost no-SWIG directions introduced → Python bindings / transport APIs mapped 1:1

- Signaling Manager
  - Hooks are already available, e.g.
    - Routing & Service helpers
    - Platform APIs
  - The (total) FSM is spread into the layers → reduced complexity
    - The Protocol-FSM has 25 states, 11 of them implementing the “core” functionalities

- The prototype is under verification in the CHANGE testbed
- Once consolidated, it will be released as GPLv2
  - Info made available via CHANGE website
Thanks for your attention

Questions?

For any further info:
info@nextworks.it
www.nextworks.it
via Livornese, 1027, 56122 Pisa (Italy)
Tel:  +39-050-3871600
Fax:  +39-050-3871601

Nextworks R&D on inter-platform signaling is partly funded by the EC through the FP7 Project #257422 “CHANGE: Enabling Innovation in the Internet Architecture through Flexible Flow-Processing Extensions

http://change-project.eu