P4 User Plane Function (P4-UPF)
SD-Fabric Tutorial – Part 3
Part 3 Agenda

- P4-UPF architecture and pipeline design
- Hands-on lab
  - Configure P4-UPF
  - Generate traffic
  - Observe GTP-U termination performed by switches
Switch-Based P4-UPF

- **Frees up CPU resources**
  - To be used by edge applications
  - UPF data path fully offloaded to switches

- **Addresses Industry 4.0 requirements**
  - Ultra low latency (<1.5µs) and jitter (<4ns)
  - Tbps throughput

- **Tailored for enterprise and IoT use cases**
  - GTP-U termination (incl. 5G extensions)
  - Application filtering (ACL)
  - Slicing & QoS
  - Usage reporting
  - Idle-mode buffering (cloud-native service)

- **INT visibility for SLA validation**
  - Monitor flows inside GTP-U tunnels
  - Support UPF-specific drop reasons

BESS: Berkeley Extensible Software Switch
Distributed UPF Data Path

- **Minimum latency**
  - Tunnels terminated at the ingress leaf, without detouring through additional devices

- **Fast failover**
  - With paired-ToRs, if one switch fails, the other can take over as it is already programmed with the same rules.

- **Fabric-wide QoS**
  - Packets are classified as soon as they hit the first leaf. We then use a custom DSCP-based marking to enforce the same QoS at each hop.
Integration with Mobile Core
Via One-Big-UPF Abstraction

- Mobile Core Control Plane (5G SMF)
- PFCP Agent
- UP4 App: One-Big-UPF abstraction
- Trellis Control Apps: Routing, ECMP, MPLS, etc.
- ONOS - SDN Controller (Highly Available)
- SD-Fabric
- UPF data path
- Switch (virtual-upf.p4)
- Stratum Spine
- Stratum Leaf
- Stratum Spine

PFCP: Packet Forwarding Control Protocol (3GPP standard interface)

Allows fabric topology to scale independently of mobile core control plane.
Role of UP4 App

**virtual-upf.p4**
Defines only UPF tables, not optimized for any HW target

**fabric.p4**
Optimized for Tofino. Defines tables for UPF, routing, ECMP, MPLS, INT, etc.

---

https://github.com/omec-project/up4
Role of PFCP Agent

- Go-based micro-service
- Implement complex PFCP protocol once, for many data paths
- Main functions:
  - PFCP session handling
  - UE IP address allocation
  - Volume/time-based triggers for Usage Reporting Rules (URR)
  - Etc.
- Support multiple southbound protocols via plug-in mechanism

https://github.com/omec-project/upf
UPF P4 Pipeline Design

With an aside on fabric.p4
UP4 Logical Pipeline

PFCP Agent

UP4 Plug-in

Packet Detection Rule (PDR)
Forwarding Action Rules (FAR)
QoS Enforcement Rules (QER)
Etc...

Applications
<Inner IP prefix, port range>

Identifies the application

Terminations
<Session ID, App ID>

Drop
(app not allowed)

GTP-U encap/decap

Source interfaces
<Outer IP addr>

Identifies the UPF logical interface (N3, N6)

Sessions
<TEID/IP addr>

Identifies the UE

Subset of PFCP capabilities tailored for enterprise use cases

UP4 plug-in validates and decomposes PFCP entities into P4Runtime table entries

P4Runtime table entries

PFCP defines a very flexible match-action abstraction. Hard to implement in HW without knowing pattern of rules from SMF.

virtual-upf.p4

Packet

Drop (app not allowed)

An Aside: Fabric.p4 Design Rationale

Trellis Control Apps

ONOS FlowObjective API (Java)
3-stage logical pipeline

Filtering
Permit/deny
VLAN-based port admission

Forwarding
Forwarding information base (bridging, IPv4/6 routing, MPLS, etc.)

Next-ID
Apply forwarding actions (rewrite headers, push/pop VLAN/MPLS, ECMP, multicast, etc.)

Next
To port(s)

Filtering tables

Forwarding tables

Next tables

Traffic manager (replication, buffering)

Egress next tables

Ingress pipe

Egress pipe

fabric.p4 (Tofino Native Architecture)
Fabric.p4 Tables (Simplified)

Filtering
- In-port + VLAN filtering table
  - Permit with internal VLAN

Forwarding
- Forwarding classifier
  - Bridging
  - IPv4 routing
  - IPv6 routing (WIP)
  - MPLS

ACL
  - Drop or punt to CPU (ONOS)

Next
- Next ID mapping
  - Hashed (ECMP)
  - Multicast
  - Next VLAN
Compile-Time Profiles

- Same P4 program, multiple profiles
- Choose which capabilities to include via p4c preprocessor flags

<table>
<thead>
<tr>
<th>Profile name</th>
<th>p4c preprocessor flags</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>fabric</td>
<td>None</td>
<td>Basic fabric profile</td>
</tr>
<tr>
<td>fabric-upf</td>
<td>-DWITH_UPF</td>
<td>With UPF tables</td>
</tr>
<tr>
<td>fabric-int</td>
<td>-DWITH_INT</td>
<td>With Inband-Network Telemetry (INT) spec v0.5</td>
</tr>
<tr>
<td>fabric-upf-int</td>
<td>-DWITH_UPF -DWITH_INT</td>
<td>With both UPF and INT functions</td>
</tr>
</tbody>
</table>

https://github.com/stratumproject/fabric-tna
UPF Integration with Fabric.p4

Similar tables to virtual-upf.p4 but optimized for Tofino

Routing on modified IP header (if encap/decap)

Traffic manager (replication, buffering)

Egress counters, etc.

Filtering  UPF Ingress  Forwarding  Next  Traffic manager  Next Egress  UPF Egress

Ingress pipe

Egress pipe

P4-UPF Summary

What we talked about

- Distributed UPF data path

- Integration with 5G mobile core via:
  - PFCP-Agent: multiple southbound plug-ins
  - UP4 ONOS app: One-Big-UPF abstraction

- Two P4 programs:
  - Virtual-upf.p4: logical, API data model for UP4
  - Fabric.p4: runs on Tofino

What we didn’t talk about

- Idle-mode buffering

- Slicing & QoS
  - Dedicated tutorial session soon

- INT integration
  - Dedicated tutorial sessions soon

- Further reading:
  - [docs.sd-fabric.org/master/advanced/p4-upf.html](https://docs.sd-fabric.org/master/advanced/p4-upf.html)
  - R. MacDavid et al. *A P4-based 5G User Plane Function*, SOSR 2021
Exercise 2

GTP-U Tunnel Termination with P4-UPF
Same 2x2 leaf-spine fabric as in Exercise 1. We will use only two hosts: gNodeB (emulated) and app host.
Exercise 2 Overview

User Equipment (UE)

Base Station (gNodeB)

App host (h4)

UPF function distributed on leaf1 and leaf2 (using fabric-upf pipeconf)
Exercise 2 Overview

User Equipment (UE)

192.168.0.1

172.16.1.99

Base Station (gNodeB)

spine1

leaf1

spine2

leaf2

App host (h4)

172.16.4.1

Dst IP: 192.168.0.1 (UE)
Src IP: 172.16.4.1 (app)
Exercise 2 Overview

Base Station (gNodeB)

User Equipment (UE)

App host (h4)

192.168.0.1

172.16.1.99

172.16.4.1

Dst IP: 172.16.1.99 (gNodeB)
Src IP: 172.16.1.254 (UPF)

Dst IP: 192.168.0.1 (UE)
Src IP: 172.16.4.1 (app)

Performs GTP-U encapsulation
P4-UPF Workflow

1. Set UPF switches
   - Set UPF switches
   - Set UPF IP address (N3)
   - Set UP4 P4Runtime server

2. PFCP Association
   - PFCP Association
   - PFCP Session
   - Establishment/Modification/Deletion

3. PFCP Sim
   - PFCP Sim
   - PFCP Agent
   - P4Runtime
   - Trellis Control Apps
   - ONOS

• Set UPF switches

• Set UE subnet

• Set UPF IP address (N3)

• Set UP4 P4Runtime server

PFCP:
Packet Forwarding Control Protocol (3GPP standard interface)

\[ etcfg-up4.json \]

pfcp-agent.json

\[ netcfg-up4.json \]

PFCP Association

PFCP Session

Establishment/Modification/Deletion

ONOS

P4Runtime & gNMI

Stratum

Spine 1

Leaf 1

\[ \ldots \]

Stratum

Spine 2

Leaf 2

\[ \ldots \]
PFCP Sim

- Emulates 5G SMF
- CLI interface to manually set up UE sessions

```
pfcpcctl service associate
pfcpcctl session create --ue-pool 192.168.0.0/16 --gnb-addr 172.16.1.99
pfcpcctl session modify --ue-pool 192.168.0.0/16 --gnb-addr 172.16.1.99
```

Handles keepalives, session bookkeeping, etc.

---

https://github.com/omec-project/pfcpsim

https://github.com/opennetworkinglab/sdfabric-tutorial
Environment Overview

Useful Commands
- make deps
- make start
- make start-upf
- make netcfg
- make onos-cli
- make onos-log
- make mn-cli
- make mn-log
- make mn-pcap
- make pfcp-log

ONOS single instance

P4RT, gNMI

BMv2/Stratum Driver

IPv4 hosts (Linux net namespace)

IPv4 hosts

Mininet script

topo.py

CLI
Log
PCAP

sdfabric-onos Docker container

Trellis Control
(underlay forwarding)

UP4
(5G UPF)

INT
(INT Watchlist)

LLDP
Provider
(link discovery)

Host
Provider
(host discovery)

PFCP Agent

PFCP Sim

pfcp-agent Docker container

pfcp-sim Docker container

netcfg.json

REST

REGISTER

pipeconf

stratum_bmv2

stratum_bmv2

stratum_bmv2

P4RT

CLI
Log

P4RT, gNMI

 make deps
make start
make start-upf
make netcfg
make onos-cli
make onos-log
make mn-cli
make mn-log
make mn-pcap
make pfcp-log

github.com/opennetworkinglab/sdfabric-tutorial
Exercise 2 Steps

- Modify configuration files
- Start PFCP Agent
- Use pfcpctl to set up UE session
- Use Python scripts to generate and sniff traffic
- Verify that switch is performing GTP-U encapsulation as expected
Exercise 2: Get Started

- Open lab README on GitHub
  - [http://github.com/opennetworkinglab/sdfabric-tutorial](http://github.com/opennetworkinglab/sdfabric-tutorial)

- Or open in text editor
  - sdfabric-tutorial/README.md
  - sdfabric-tutorial/EXERCISE-2.md

- Solution
  - sdfabric-tutorial/solution
That’s All For Now!

- Part 1 – Introduction to SD-Fabric: motivation, architecture, use cases
- Part 2 – Basics & Configuration + hands-on lab
- Part 3 – P4 User Plane Function (UPF) + hands-on lab
- Part 4 – In-band Network Telemetry (INT)
- Part 5 – Extending SD-Fabric
- Part 6 – Slicing & QoS
- Part 7 – Advanced Connectivity
- And more...

More sessions and labs on the way!
Make sure to watch the GitHub repo github.com/opennetworkinglab/sdfabric-tutorial
Notices & Disclaimers

- Intel technologies may require enabled hardware, software or service activation.
- No product or component can be absolutely secure.
- Your costs and results may vary.
- © Intel Corporation. Intel, the Intel logo, and other Intel marks are trademarks of Intel Corporation or its subsidiaries. Other names and brands may be claimed as the property of others.