Tutorial Service Function Chaining

Jump to bottom

Roberto Riggio edited this page on Aug 18, 2019 · 1 revision

Table of Contents

Objective

In this section we shall illustrate how to implement deploy a single LVNF and how to steer the HTTP traffic generated by the wireless clients through it.

The complete implementation of the Service Function Chaining Network application discussed in this tutorial can be found here.

Pre-requisites

This tutorial requires a network setup similar to the one described here. This means at least one WTP connected to a CPP. Moreover both the 5G-EmPOWER Controller and the Ryu Controller must be active.

Refer to this page for instructions about how to setup the Ryu controller. If any other switch is present between the WTP and the CPP that switch must be an OpenFlow switch and must point to the Ryu controller.

Launch method & Initializations Tasks

The Service Function Chaining class definition and initialization methods are reported below:

```
class LvnfChain(EmpowerApp):
    def __init__(self, **kwargs):
        EmpowerApp.__init__(self, **kwargs)
        self.cppup(callback=self.cpp_up_callback)
        self.lvapjoin(callback=self.lvap_join_callback)
        self.lvnfjoin(callback=self.lvnf_join_callback)
        self.lvnf = None
```

As it can be seen, the initialization tasks registers three triggers: *lvapjoin*, *lvnfjoin*, and *cppup*.

The launch method for the Mobility Manager application is shown below:

```
def launch(tenant_id, every=DEFAULT_PERIOD):
    """ Initialize the module. """

    return LvnfChain(tenant_id=tenant_id, every=every)
```

As it can be seen the *launch* method has two parameters: the mandatory *tenant_id* parameter, and the *every* parameter specifying the control task period.

Deploy the LVNFs

In this case the *cppup* trigger is used to deploy an LVNF when a CPP becomes available. The *cppup* callback method is reported below:

```
def cpp_up_callback(self, cpp):
    img = Image(vnf="in_0 -> ToDump(/tmp/lvnf.pcap) -> out_0")
    self.spawn_lvnf(img, cpp)
```

Notice how this LVNF essentially consists of a single Click element which simply saves all the traffic to a file.

The *lvnf_join_callback* method is reported below. This method is used in order to save a pointer th LVNF as soon as it becomes active.

```
def lvnf_join_callback(self, lvnf):
    self.lvnf = lvnf
```

Service Function Chaining

Finally, the *lvap_join_callback* method implements the service function chaining between any new wireless client and the LVNF:

```
def lvap_join_callback(self, lvap):
    if self.lvnf:
        lvap.ports[0].next["tp_dst=80"] = self.lvnf.ports[0]
```

At this point if you login using SSH into the CPP running the LVNF you will find a file named */tmp/lvnf.pap*. You can either open this file using tcpdumpp (if this software has been installed in the CPP) or you can copy it to your laptop and open it with a software like Wireshark. Inside the file you will find only the HTTP requests made by the client.

The LVNF can then be modified in order to run you own Click element implementing a more complex function, e.g. filtering out some HTTP requests.

▸ Pages 25

Getting Started

Using EmPOWER

Intent Based Networking

Downloads

Developers

Tutorials

- Mobility Manager (LTE)
- Service Function Chaining

Support

- Mailing List

Acknowledgements

- Acknowledgements

Clone this wiki locally

```
https://github.com/clicknf/clicknf.github.io.wiki.git
```