Service Function Chain Proof of Transit

OpenDaylight (Carbon)
Ver. 0.6

Author	Date	Notes
Srihari Raghavan (srihari@cisco.com)	10 Jun 2016	Initial Version – v0.1
Srihari Raghavan (srihari@cisco.com)	28 Jun 2016	Feature name change
Srihari Raghavan (srihari@cisco.com)	30 Nov 2016	Feature updates

Table of Contents

1. Introduction

Service Function Chain (SFC) Proof of Transit as implemented in Carbon release of OpenDaylight is based on the concept of in-band OAM [1]. In-band OAM is an implementation study to record operational information in the packet while the packet traverses a path between two points in the network. More specifically, SFC Proof of Transit via in-band OAM is meant to prove that data or user traffic indeed follows the suggested path or a SFC.

To ensure that the data traffic follows a specified path or a function chain, meta-data is added to all user traffic. The meta-data is based on a 'share of a secret' and provisioned by opendaylight controller over a secure channel to each of the nodes that implement the service function chain. This meta-data is updated at each of the service-hop while a designated node called the verifier checks whether the collected meta-data allows the retrieval of the secret.

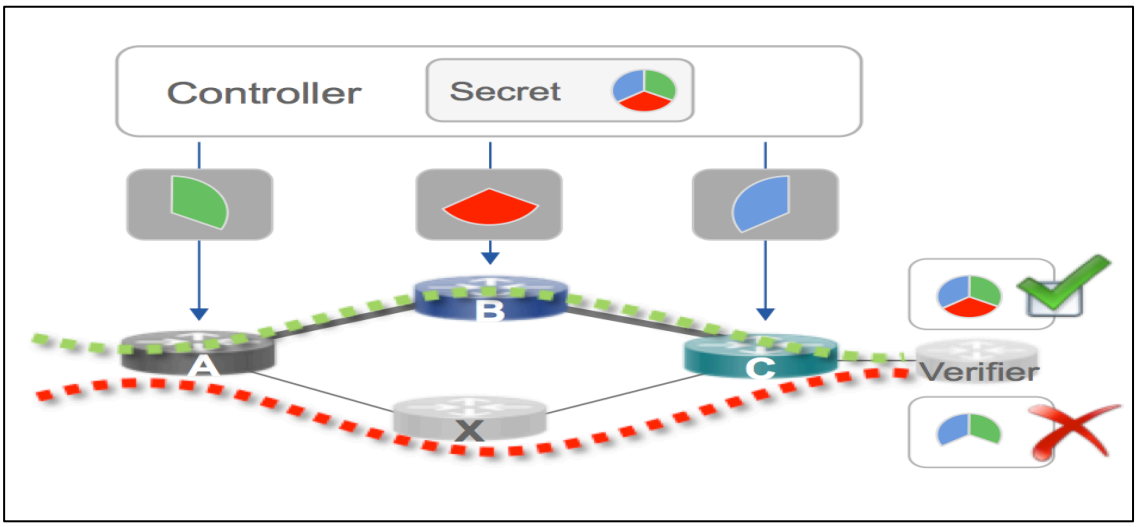

Fig 1: SFC Proof of Transit overview

The proof of transit via a SFC or a path via this solution utilizes Shamir's secret sharing algorithm [2], where each service is given a point on the curve and when the packet travels through each service, it collects these points (meta-data) and a verifier node tries to re-construct the curve using the collected points, thus verifying that the packet traversed through all the service functions along the chain.

Transport options for different protocols includes as a new TLV in SRH header for Segment Routing, Network Service Headers (NSH) Type-2 meta-data, IPv6 extension headers, IPv4 variants and for VXLAN-GPE as well.

2. SFC Proof of Transit and OpenDaylight

The meta-data for SFC Proof of Transit (SFC PoT) are provisioned via the OpenDaylight controller SFCV application. The configuration meta-data are partially user-configured while the rest of the configuration are auto generated based on the user specified criteria and sent down to the SFC nodes via Netconf/YANG based protocol. The meta-data might include service-chain hop

count (number of services in the chain), SFCV-key set, secret, prime-numbers, polynomial coefficients and share of a secret.

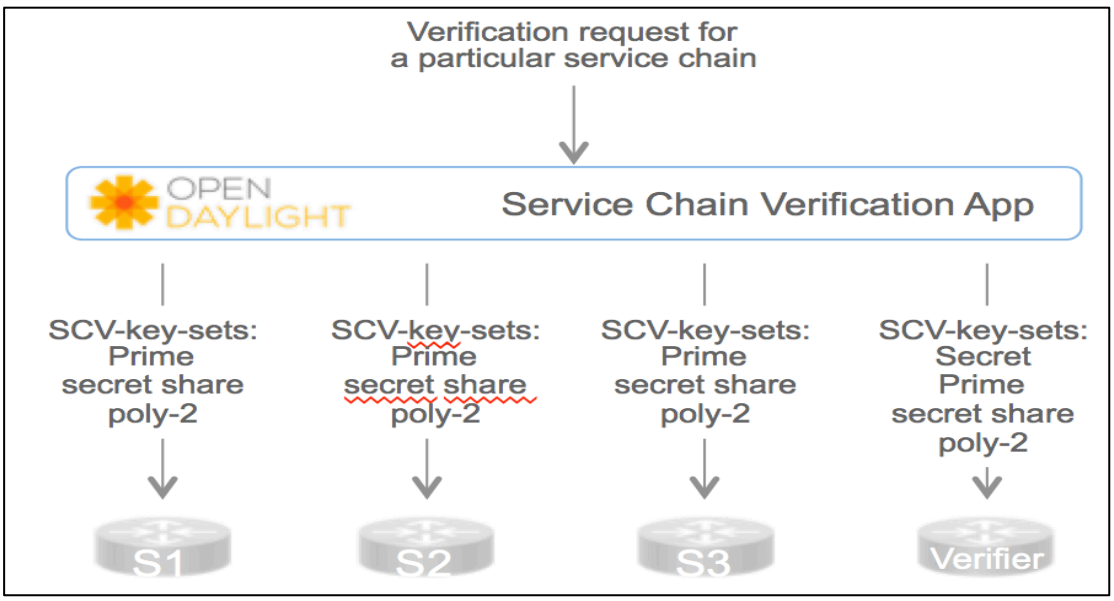

Fig 2: SFC Proof of Transit controller application overview

3. Pre-Requisites

- A native Ubuntu X distribution or running as a Virtualbox image with interface
- Java 8 and Maven
- SFC [3] ODL distribution as descripted in SFC 101 [3] document.
- Python 3.4.2 and all modules needed by SFC
    - Requests
    - Flask
    - Netifcaes
    - Paramiko
    - Netfilter module

4. SFC Proof of Transit Architecture

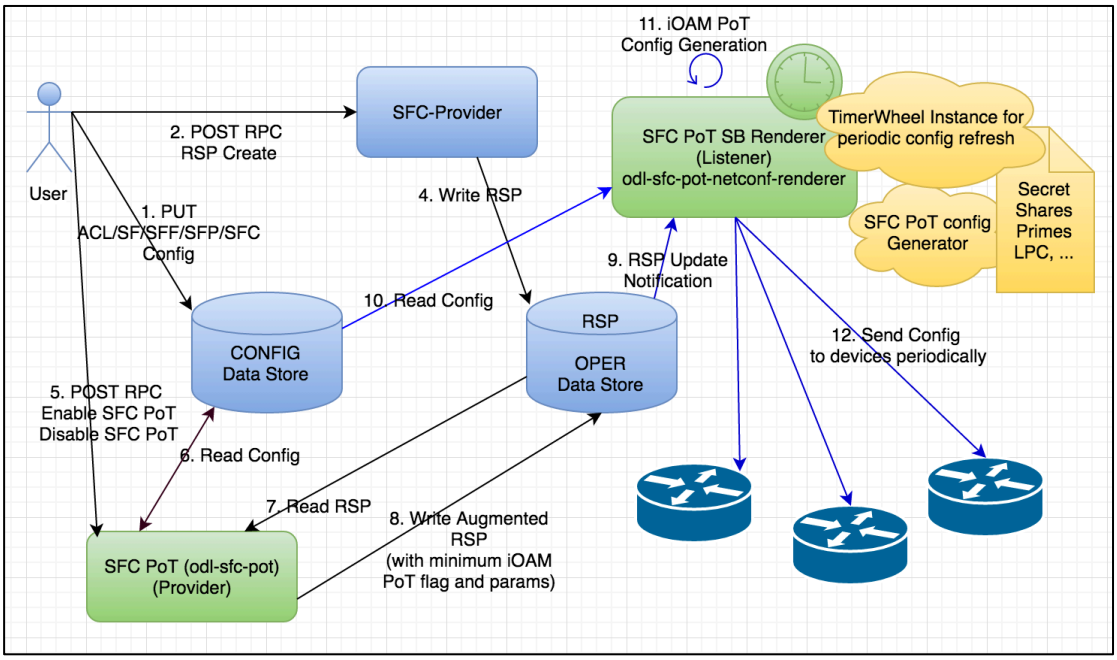

Fig 3: Opendaylight SFC PoT Architecture

The general architecture relies on the following aspects:

a. There is a generic north bound (NB) module that focuses on RPC aspects to enable and disable the feature and for storing the parameters to realize the Proof of Transit feature for the SFC under consideration.
b. There is a per-south-bound (SB) protocol renderer (to start with, a Netconf based renderer) that comprises the following aspects.
    i. Netconf specific aspects of the feature.
    ii. SB-specific yang files for configuration data.
    iii. Configuration generation API that creates the configuration data based on the yang files.
    iv. A configuration refresh timer to refresh profile configurations across SFCs running Proof of Transit feature.
c. The Proof of Transit feature is enabled by two different sub-features:
    i. Feature:install odl-sfc-pot
    ii. Feature:install odl-sfc-pot-netconf-renderer

When SFC Proof of Transit feature is installed the following aspects are done:

- The class (**SfcPotRpc**) sets up RPC handlers for enabling the feature.
- There are new RPC handlers for two new RPCs (**EnableSfcIoamPotRenderedPath** and **DisableSfcIoamPotRenderedPath**) and effected via **SfcPotRspProcessor** class.
- When a user configures via a POST RPC call to enable Proof of Transit on a particular SFC (via the Rendered Service Path), the configuration

drives the creation of necessary augmentations to the RSP (to modify the RSP) to effect the Proof of Transit configurations.

- The augmentation meta-data added to the RSP are defined in the **sfc-ioam-nb-pot.yang** file and are as below**. NOTE: There are no auto generated configuration parameters added to the RSP to avoid RSP bloat.**

Index	Field	Notes
1	Ioam-pot-enable	Enable iOAM PoT for this SFC
2	Ioam-pot-num-profiles	Number of iOAM PoT profiles per node of the SFC
3	Ioam-pot-bit-mask	Number of bits to use in iOAM configuration generation
4	Ioam-pot-refresh-period	Periodic configuration refresh timer value

- Adding SFC Proof of Transit meta-data to the RSP is done in **SfcPotRspProcessor** class.

- Once the RSP is updated, the RSP data listeners in the SB renderer modules (**odl-sfc-pot-netconf-renderer**) will listen to the RSP changes and send out configurations to the necessary SB nodes that are part of the SFC.
  - The configurations are handled mainly in the **SfcPotAPI, SfcPotConfigGenerator, SfcPotPolyAPI, SfcPotPolyClass and SfcPotPolyClassAPI** classes
  - There is a **sfc-ioam-sb-pot.yang** file that shows the format of the iOAM PoT configuration data sent to each node of the SFC.
  - A timer is started based on the "ioam-pot-refresh-period" value in the SB renderer module that handles configuration refresh periodically. This is detailed below.
- **Configuration Refresh Protocol**
  - The SB and timer handling are done in the odl-sfc-pot-netconf-renderer module. **Note: This is NOT done in the NB odl-sfc-pot module to avoid periodic updates to the RSP itself.**
  - Controller creates a new profile of a set of keys and secrets at a constant rate and updates the RSP data store with the configuration. The controller labels the configurations per RSP as "even" or "odd" – and the controller cycles between "even" and "odd" labeled profiles. The rate at which these profiles are communicated to the nodes is configurable and in future, could be automatic based on profile usage. Once the profile has been successfully communicated to all nodes (all Netconf transactions completed), the controller sends an "enable scv-profile" request to the ingress node. The controller or the non-renderer part (odl-sfc-pot) handles the periodic refresh of the configuration along with NB protocol handling.
  - The nodes are to maintain two profiles (an even and an odd scv-profile). One profile is currently active and in use, and one

profile is about to get used. A flag in the packet is indicating whether the odd or even scv-profile is to be used by a node. This is to ensure that during profile change we're not disrupting the service. I.e. if the "odd" profile is active, the controller can communicate the "even" profile to all nodes – and only if all the nodes have received it, the controller will tell the ingress node to switch to the "even" profile. Given that the indicator travels within the packet, all nodes will switch to the "even" profile. The "even" profile gets active on all nodes – and nodes are ready to receive a new "odd" profile.

- Unless the ingress node receives a request to switch profiles, it'll continue to use the active profile. If a profile is "used up" the ingress node will recycle the active profile and start over (this could give rise to replay attacks in theory – but with 2^32 or 2^64 packets this isn't really likely in reality).

- **Timer-handling in the SB module:**
  - **HashedTimerWheel** implementation is used to support the periodic configuration refresh. The default refresh is 5 seconds to start with.
  - Depending on the last updated profile, the odd or the even profile is updated in the fresh timer pop and the configurations are sent down appropriately.
  - **SfcPotTimerQueue, SfcPotTimerWheel, SfcPotTimerTask , SfcPotTimerData and SfcPotTimerThread** are the classes that handle the Proof of Transit protocol profile refresh implementation.
- **Scale aspects**
  - The RSP data store is NOT being changed periodically and the timer and configuration refresh modules are present in the SB renderer module handler and hence there are are no scale or RSP churn issues affecting the design.

The following sequence diagram gives the overview.

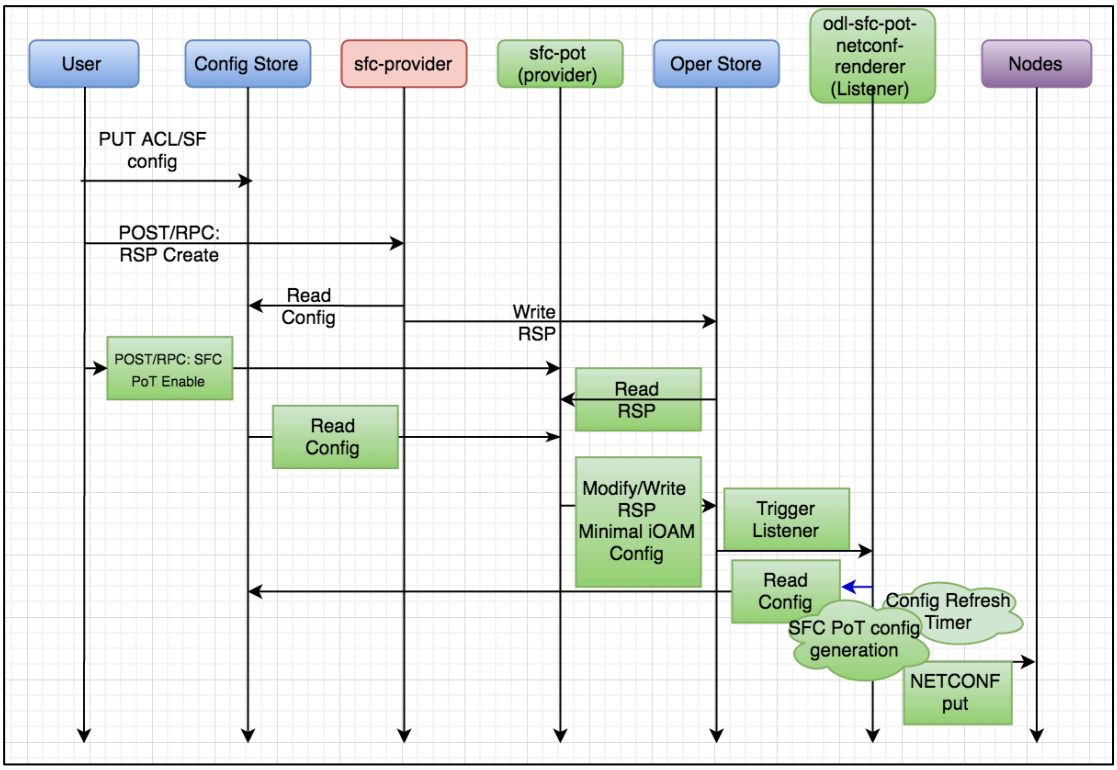

Fig 4: SFC Proof of Transit Sequence Diagram Overview

5. SFC Proof of Transit Architecture – Details

a. ODL-SFC-POT Overview

SFC Proof of Transit is implemented in OpenDaylight by the "**odl-sfc-pot**" feature and "**odl-sfc-pot-netconf-renderer"** features and is intended to be part of the base **SFC.git** and intended to be **optional by design**. To enable the feature, one must do **"feature:install odl-sfc-pot"** and "**feature:install odl-sfc-pot-netconf-renderer**".

The feature is hosted under "**sfc-pot"** directory of SFC.git. Related features.xml and pom.xml files are updated to handle the feature.

The following table provides the details of the various yang files and classes that comprise the feature and a description of the same.

Index	File Name	Purpose	LoC	Notes
1	Sfc-ioam-nb-pot.yang (NB)	Tracks iOAM enable flag and minimal iOAM PoT parameters to be augmented to the RSP. It also contains RPCs to enable and disable SFC Proof of transit.	~100	This file is internal to ODL SFC PoT application.
2	SfcPotRpc (NB)	This registers with the system to be a handler for the EnableSfcIoamPotRenderedPath and Disable RPCs.	~100	@see SfcPotRspProcessor
3	SfcPotRspProcessor (NB)	Handles the RPC calls to enable or disable SFC Proof of transit to modify the RSP and write it back to the data store.	~250	
Index	**File Name**	**Purpose**	**LoC**	**Notes**
1	SfcPotPolyClass(SB)	This is the class that forms the template of the configuration stored for SFC Proof of Transit per RSP.	~50	@see SfcPotPolyAPI
2	SfcPotPolyAPI(SB)	This is the class that interacts with the configuration generator and handles the modalities around storage of the created, configured and auto generated per-RSP configuration.	~250	@see SfcPotConfig Generator
3	SfcPotConfig Generator(SB)	This is the main class that creates the necessary SFC proof of transit related meta-data and shares, prime-numbers etc. to effect the SFC proof of transit.	~250	
4	SfcPotTimerQueue (SB)	Timer queue for holding the RSP meta-data key.	~50	

5	SfcPotTimerWheel(SB)	Timer wheel implementation.	~50	
6	SfcPotTimerThread (SB)	Timer worker thread implementation to periodically refresh the configuration and send to the SFC nodes.	~60	
7	SfcPotTimerTask (SB)	Timer task implementation that works with SFC PoT API to create the configuration.	~30	
8	SfcPotTimerData (SB)	Handles the SFC PoT meta-data information keyed by RSP name with value-data including current active index, timer value etc., to be utilized on timer pops.	~150	
9	SfcPotNetconfIoamAPI(SB)	SB handler that sends down the configuration to the SFC nodes via Netconf protocol		

6. Glossary

SF - Service Function
SFF - Service Function Forwarder
SFC - Service Function Chain
SFP - Service Function Path
RSP - Rendered Service Path
SFC PoT - Service Function Chain Proof of Transit
iOAM – In-band OAM

7. References

1. In-band OAM (iOAM) - https://github.com/ciscodevnet/ioam
2. Shamir's secret sharing - Link
3. Service Function Chaining - Link