ODL BGPCEP Magnesium Retrospective and Roadmap for Aluminium

Dr. Olivier Dugeon, Orange Labs

olivier.dugeon@orange.com

Outline

- PCE Overview
- Magnesium Retrospective
  - New features: Graph & Algorithms
  - RFC 5440 compliancy
- Demo
- Aluminium Roadmap
  - RFCs compliancy
  - Graph & algorithms improvements
  - Path Manager
- And Beyond
  - Inter-domain controller use case
  - Free Range Routing (FRR) integration

PCE OVERVIEW

Remind Me: What is PCE & PCEP?

From RFC 4655

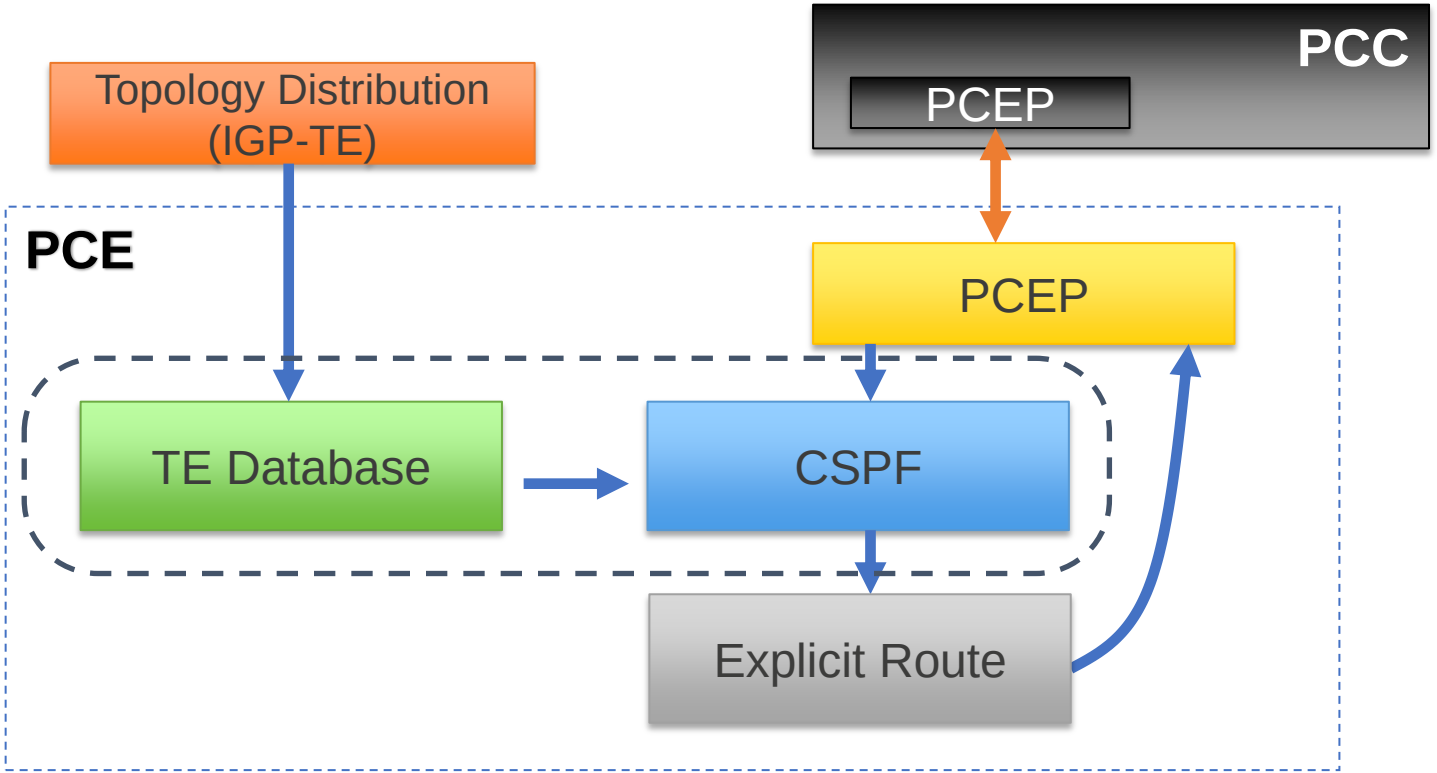
“PCE: Path Computation Element. An entity (component, application, or network node) that is capable of computing a network path or route based on a network graph and applying computational constraints”

From RFC 5440

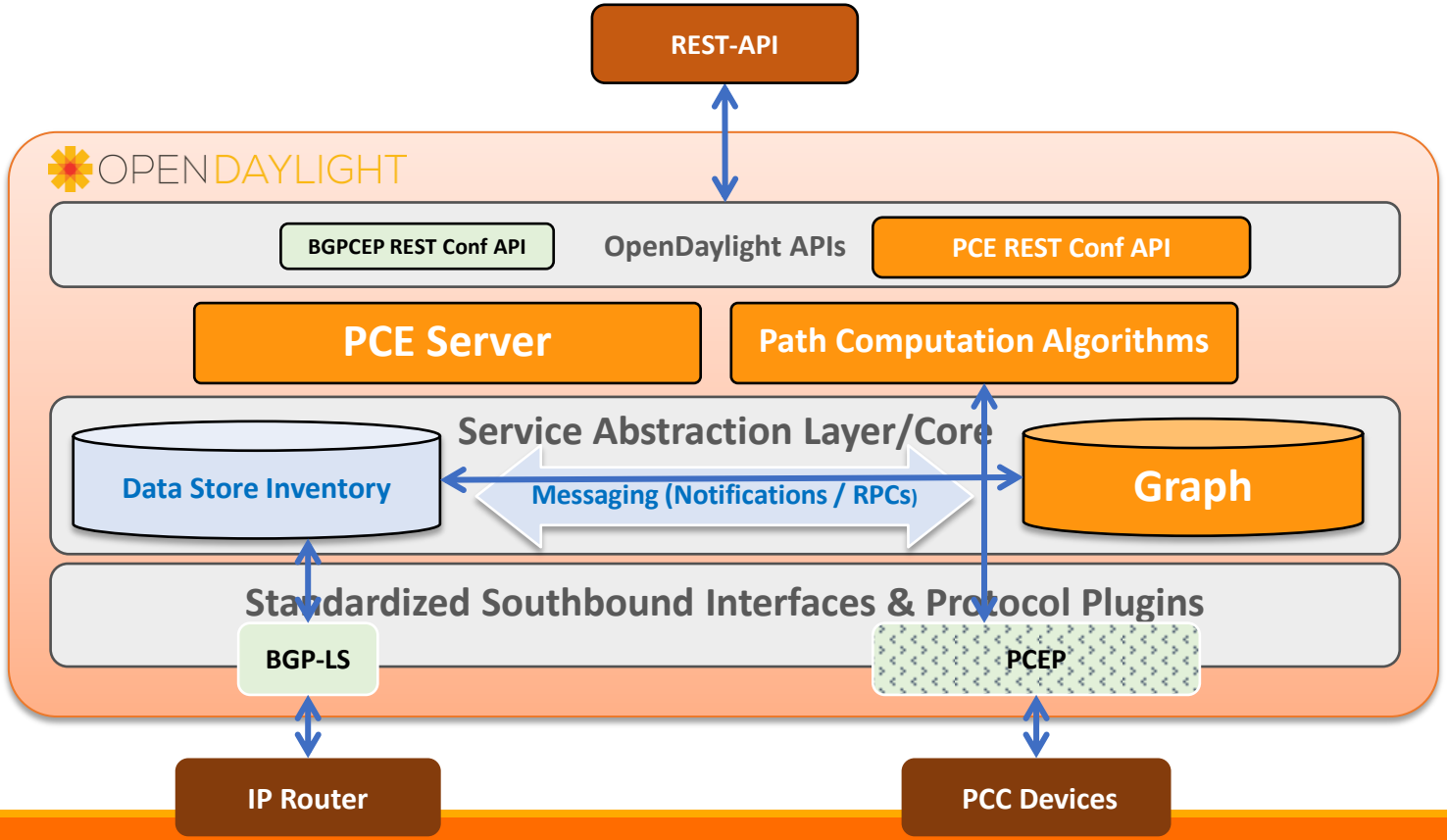
“This document specifies the Path Computation Element (PCE) Communication Protocol (PCEP) for communications between a Path Computation Client (PCC) and a PCE, or between two PCEs”

“Such interactions include path computation requests and path computation replies as well as notifications of specific states related to the use of a PCE in the context of Multiprotocol Label Switching (MPLS) and Generalized MPLS (GMPLS) Traffic Engineering”

Path Computation Element Building Blocks



PCE architecture



Magnesium Retrospective

New Graph feature

Target is RFC 5440 compliancy

- Be able to answer to a PCE Request message
- Provides an Explicit Route Object (ERO) as result of a Path Computation

PCE Path Computation algorithm works with a Traffic Engineering Database (TED)

- TED must reflect the underlying Network Topology
- Which model? Yang of course!
- But yang doesn't allow cross reference
  - Node could reference Link and reciprocally, but not both
  - Using a flat database drastically reduce path computation performance at large scale (> 1000 nodes)
- Even if primary designed for IP, keep it flexible for other network topology (e.g. OpenFlow)

Introduce new yang model named Graph

- Algorithms are based on graph theory
- Gives the possibility to fulfil a graph manually or automatically
- Complete the yang model with a connected graph stored in memory for performance

Graph Model

```
+--rw graph-topology
   +--rw graph* [name]
      +--rw name              string
      +--rw domain-scope?     enumeration
      +--rw asn?              uint32
      +--rw vertex* [vertex-id]
      |  +--rw vertex-id      uint64
      |  +--rw name?          string
      |  +--rw router-id?     inet:ip-address
      |  +--rw vertex-type?   enumeration
      |  +--rw srgb
      |  |  +--rw lower-bound?   uint32
      |  |  +--rw range-size?    uint32
      |  +--rw asn?           uint32
      +--rw edge* [edge-id]
      |  +--rw edge-id              uint64
      |  +--rw local-vertex-id?     uint64
      |  +--rw remote-vertex-id?    uint64
      |  +--rw name?                string
      |  +--rw edge-attributes
      |     +--rw metric?                       uint32
      |     +--rw te-metric?                    uint32
      |     +--rw admin-group?                  uint32
      |     +--rw local-address?                inet:ip-address
      |     +--rw remote-address?               inet:ip-address
      |     +--rw local-identifier?             uint32
      |     +--rw remote-identifier?            uint32
      |     +--rw max-link-bandwidth?           decimal-bandwidth
      |     +--rw max-resv-link-bandwidth?      decimal-bandwidth
      |     +--rw unreserved-bandwidth* [class-type]
      |     |  +--rw class-type     uint8
      |     |  +--rw bandwidth?     decimal-bandwidth
      |     +--rw delay?                        delay
      |     +--rw min-max-delay
      |     |  +--rw min-delay?    delay
      |     |  +--rw max-delay?    delay
      |     +--rw jitter?                       delay
      |     +--rw loss?                         loss
      |     +--rw residual-bandwidth?           decimal-bandwidth
      |     +--rw available-bandwidth?          decimal-bandwidth
      |     +--rw utilized-bandwidth?           decimal-bandwidth
      |     +--rw adj-sid?                      uint32
      |     +--rw backup-adj-sid?               uint32
      |     +--rw srlgs*                        uint32
      +--rw prefix* [prefix]
      +--rw prefix          inet:ip-prefix
      +--rw prefix-sid?     uint32
      +--rw node-sid?       boolean
      +--rw vertex-id?      uint64
```

- Graph is identified by its name (unique) and contains domain-scope, AS number and List of Vertices, Edges and Prefix
- Vertices are identified by a unique ID and contains node information including Segment Routing and AS number (for inter-domain)

Edges are identified by a unique ID and contains all Traffic Engineering Parameters including Segment Routing and SRLG

Note: other parameter (e.g. Optical & Transport) could be easily added

Prefixes allow a fast search of Source and Destination in the Graph and provides Segment Routing support

Connected Graph associated to the Graph model

Connected Graph

```
/* List of Connected Vertics that composed this Connected Graph */
private final HashMap<Long, ConnectedVertexImpl> vertices = new HashMap<>();

/* List of Connected Edges that composed this Connected Graph */
private final HashMap<Long, ConnectedEdgeImpl> edges = new HashMap<>();

/* List of IP prefix attached to Vertices */
private final HashMap<IpPrefix, Prefix> prefixes = new HashMap<>();

/* Reference to the non connected Graph stored in DataStore */
private Graph graph;
```

Connected Vertex

```
/* Reference to input and output Connected Edge within the Connected Graph */
private ArrayList<ConnectedEdgeImpl> input = new ArrayList<>();
private ArrayList<ConnectedEdgeImpl> output = new ArrayList<>();

/* List of Prefixes announced by this Vertex */
private ArrayList<Prefix> prefixes = new ArrayList<>();

/* Reference to the Vertex of the standard Graph associated to the Connected Graph */
private Vertex vertex = null;
```

Connected Edge

```
/* Reference to Source and Destination Connected Vertex within the Connected Graph */
private ConnectedVertexImpl source;
private ConnectedVertexImpl destination;

/* Reference to the Edge within the Graph associated to the Connected Graph */
private Edge edge;
```

New RESTCONF API to manage Graph

Config RestConf API

- Create, Read, Update and Delete Graph, Vertices, Edges and Prefix from config datastore
  - Graph URL: restconf/config/graph:graph-topology
  - Vertex URL: restconf/config/graph:graph-topology/graph/<name_of_graph>/vertex/<vertex_id>
  - Edge URL: restconf/config/graph:graph-topology/graph/<name_of_graph>/edge/<edge_id>
  - Prefix URL: restconf/config/graph:graph-topology/graph/<name_of_graph>/prefix/<prefix_value>

Operational RestConf API

- Read only as usual
  - Graph URL: restconf/operational/graph:graph-topology

BGP Link State & Graph

- Create a new BGP session with a BGP-LS router as usual
- Operational Graph will be automatically fulfil

New Algo feature

3 Point-to-point Algorithms are proposed

- Simple Shortest Path First (SPF) with standard metric (hop count) only
- Constrained Shortest Path First (CSPF) which take care of Bandwidth and TE metric
- Self Adaptive Multiple Constraints Routing Algorithm (SAMCRA) which combines all metrics and bandwidth

All are using the same principle

- A pruning function to eliminate edges that don’t respect constraints
- A priority queue to explore all possible paths with path length as sorting key
- A relax function to select the best paths amongst all solutions

New RPC API to request path computation

- URL: restconf/operations/path-computation:get-constrained-path

RFC 5440 Compliancy – PCE server in ODL

Path Computation Element needs to compute a Constrained Path

- PCEP Request Message convey all requirements
  - Only EndPoint, Metric (TE, Delay), Bandwidth and Address family are supported right now
- Answer delivered as an Explicit Route Object (ERO) in PCEP Response message
  - ERO is replaced by a NO_PATH Object if no path has been found by the algorithm
- Path Computation is automatically triggered when a PCEP Request is received
  - Algorithm selection is based on constraints presence
- Special graph named ‘ted://xxxx’ is used
  - Need to automate graph retrieval from PCEP session point of view

PCEP Initiate integration

- Use same URL as usual: /restconf/operations/network-topology-pcep:add-lsp
- Don’t specify the Explicit Route Object to trigger automatically the path computation
- Use PST to trigger Segment Routing instead of RSVP-TE
  - Only Node SID NAI is supported

DEMO

Aluminium Roadmap & Beyond

Aluminium Roadmap (1/2)

Bug Tracking

- Several bugs have been identified
  - Connected Graph not correctly created if empty graph is not previously created
  - PCEP session crash if path computation crash ➔ Add Future class to avoid that problem
  - Path computation crash if bad parameters are provided (no graph, debug & unknown EndPoint, …)
  - Features-graph & features-algo are not part of ODL distribution while in BPCEP project
  - BGP-LS with IPv6 seems buggy to fulfil the graph

New constraints from PCEP Request Message

- Exclude (ERO) & Include (IRO) Route Object
- Objective Function
- Loose / Strict to provide Explicit Route Object
  - ERO is a list of Router ID in loose and a list of interface for Strict path
- Segment Routing Capability N flag
  - Automatic NAI configuration: Node SID for loose / Adjacency for strict with Index (N flag set) or MPLS label

Aluminium Roadmap (2/2)

RFC conformity

- RFC8231 (Stateful) ➔ rename all reference to old Stateful07 and update options
- RFC 8281 (Initiated) ➔ rename all reference to old Crabbe & Initiated01 and update options
- RFC 8408 (PST) ➔ some options are missing
- RFC 8664 (Segment Routing) ➔ update PCEP SR Capability (missing option)

Integration of Path Computation with PCEP Update Message

- Empty ERO with RPC API could also trigger Path Computation
- Trigger Path Computation if a PCEP Report message is received with an empty ERO
  - Even if RFC8231 recommended to use PCReq message, in some scenario, it is not possible i.e. when path are created on the router before PCEP session is established

Path Manager with new yang model to create/read/update/delete paths in config datastore

- Allow to recreate / update LSP – Segment Path after a PCC reboot
  - After a PCC reboot, ODL lost all PCIntiated tunnels setup previously on the router
- With persistency, re-sync PCC tunnels when ODL restart
  - After restart, ODL doesn't reconfigure PCIntiated tunnels that are automatically remove by the PCCs

And beyond

Inter-domain Path Computation

- Support of PCE – PCE session for distributed or hierarchical Path Computation
  - Add new PCEP capability to distinguish PCE / PCE session from PCC / PCE session
  - Add VSPT (Virtual Source Path Tree) handler

Implement Backward Recursive Path Computation (RFC5441)

- Distribute path computation amongst selected set of PCE
  - Each PCE computes its part of the path
  - And recursively mono-domain paths are merge to from end-to-end multi-domain path

Implement Stateful Inter-domain Path

- Same Recursive method is used to propagate Stitching Label
  - Ease inter-operator operation avoiding usage of in-band signalling
  - Keep each operator free of the underlying technology (RSVP-TE, SR-MPLS, LDP …)
  - draft-dugeon-pce-stateful-interdomain-03.txt (version 04 is on-going)

Free Range Routing (FRR)

FRRouting is an IP routing protocol suite https://frrouting.org/

- Fork from Quagga
- Current release 7.3 and source available on github: https://github.com/FRRouting
- Active community
- New PCELib and Pathd (Segment Routing path) are on-going
  - Strong collaboration to debug PCEP & Path Computation

Good candidate to automate some BGPCEP tests with real devices

- FRR used dedicated topotest feature for its CI/CD chain
  - See tests/topotests directory
  - Based on pytest & mininet
  - Other tools like netgen are available
- Many FRR instance could be deployed on small footprint
- Both PCEP & BGP protocols could be tested

Special thanks to Philippe Niger & Philippe Cadro for their contributions to Graph & Algo features