TEAS Working Group
Internet Draft
Intended status: Standard Track
Expires: January 2022

Italo Busi (Ed.)
Huawei
Sergio Belotti (Ed.)
Nokia
Victor Lopez
Nokia
Anurag Sharma
Google
Yan Shi
China Unicom

July 12, 2021

YANG Data Model for requesting Path Computation
draft-ietf-teas-yang-path-computation-14

Status of this Memo

Abstract

There are scenarios, typically in a hierarchical Software-Defined Networking (SDN) context, where the topology information provided by a Traffic Engineering (TE) network provider may be insufficient for its client to perform end-to-end path computation. In these cases the client would need to request the provider to calculate some (partial) feasible paths.

This document defines a YANG data model for a Remote Procedure Call (RPC) to request path computation. This model complements the solution, defined in RFC YYYY, to configure a TE tunnel path in "compute-only" mode.

[RFC EDITOR NOTE: Please replace RFC YYYY with the RFC number of draft-ietf-teas-yang-te once it has been published.

Moreover this document describes some use cases where a path computation request, via YANG-based protocols (e.g., NETCONF or RESTCONF), can be needed.

Table of Contents

1. Introduction

There are scenarios, typically in a hierarchical Software-Defined Networking (SDN) context, where the topology information provided by a Traffic Engineering (TE) network provider may be insufficient for its client to perform end-to-end path computation. In these cases the

client would need to request the provider to calculate some (partial) feasible paths, complementing his topology knowledge, to make his end-to-end path computation feasible.

This type of scenarios can be applied to different interfaces in different reference architectures:

- Application-Based Network Operations (ABNO) control interface [RFC7491], in which an Application Service Coordinator can request ABNO controller to take in charge path calculation (see Figure 1 in [RFC7491]).

- Abstraction and Control of TE Networks (ACTN) [RFC8453], where a controller hierarchy is defined, the need for path computation arises on the interface between Customer Network Controller (CNC) and Multi-Domain Service Coordinator (MDSC), called CNC-MDSC Interface (CMI), and on the interface between MSDC and Provisioning Network Controller (PNC), called MDSC-PNC Interface (MPI). [RFC8454] describes an information model for the Path Computation request.

Multiple protocol solutions can be used for communication between different controller hierarchical levels. This document assumes that the controllers are communicating using YANG-based protocols (e.g., NETCONF [RFC6241] or RESTCONF [RFC8040]).

Path Computation Elements (PCEs), controllers and orchestrators perform their operations based on Traffic Engineering Databases (TED). Such TEDs can be described, in a technology agnostic way, with the YANG data model for TE Topologies [RFC8795]. Furthermore, the technology specific details of the TED are modeled in the augmented TE topology models, e.g. [OTN-TOPO] for Optical Transport Network (OTN) Optical Data Unit (ODU) technologies.

The availability of such topology models allows the provisioning of the TED using YANG-based protocols (e.g., NETCONF or RESTCONF). Furthermore, it enables a PCE/controller performing the necessary abstractions or modifications and offering this customized topology to another PCE/controller or high level orchestrator.

The tunnels that can be provided over the networks described with the topology models can be also set-up, deleted and modified via YANG-based protocols (e.g., NETCONF or RESTCONF) using the TE tunnel YANG data model [TE-TUNNEL].

This document defines a YANG data model [RFC7950] for an RPC to request path computation, which complements the solution defined in [TE-TUNNEL], to configure a TE tunnel path in "compute-only" mode.

The YANG data model definition does not make any assumption about whether that the client or the server implement a "PCE" functionality, as defined in [RFC4655], and the Path Computation Element Communication Protocol (PCEP) protocol, as defined in [RFC5440].

Moreover, this document describes some use cases where a path computation request, via YANG-based protocols (e.g., NETCONF or RESTCONF), can be needed.

The YANG data model defined in this document conforms to the Network Management Datastore Architecture [RFC8342].

1.1. Terminology

TED: The traffic engineering database is a collection of all TE information about all TE nodes and TE links in a given network.

PCE: A Path Computation Element (PCE) is an entity that is capable of computing a network path or route based on a network graph, and of applying computational constraints during the computation. The PCE entity is an application that can be located within a network node or component, on an out-of-network server, etc. For example, a PCE would be able to compute the path of a TE Label Switched Path (LSP) by operating on the TED and considering bandwidth and other constraints applicable to the TE LSP service request. [RFC4655].

Domain: TE information is the data relating to nodes and TE links that is used in the process of selecting a TE path. TE information is usually only available within a network. We call such a zone of visibility of TE information a domain. An example of a domain may be an IGP area or an Autonomous System. [RFC7926]

The terminology for describing YANG data models is found in [RFC7950].

1.2. Tree Diagram

Tree diagrams used in this document follow the notation defined in [RFC8340].

1.3. Prefixes in Data Node Names

In this document, names of data nodes and other data model objects are prefixed using the standard prefix associated with the corresponding YANG imported modules, as shown in Table 1.

Prefix	YANG module	Reference
inet	ietf-inet-types	[RFC6991]
te-types	ietf-te-types	[RFC8776]
te	ietf-te	[TE-TUNNEL]
te-pc	ietf-te-path-computation	this document

Table 1: Prefixes and corresponding YANG modules

2. Use Cases

This section presents some use cases, where a client needs to request underlying SDN controllers for path computation.

The use of the YANG data model defined in this document is not restricted to these use cases but can be used in any other use case when deemed useful.

The presented uses cases have been grouped, depending on the different underlying topologies: a) Packet/Optical Integration; b) multi-domain Traffic Engineered (TE) Networks; and c) Data Center Interconnections. Use cases d) and e) respectively present how to apply this YANG data model for standard multi-domain PCE i.e. Backward Recursive Path Computation [RFC5441] and Hierarchical PCE [RFC6805].

2.1. Packet/Optical Integration

In this use case, an optical domain is used to provide connectivity to some nodes of a packet domain (see Figure 1).

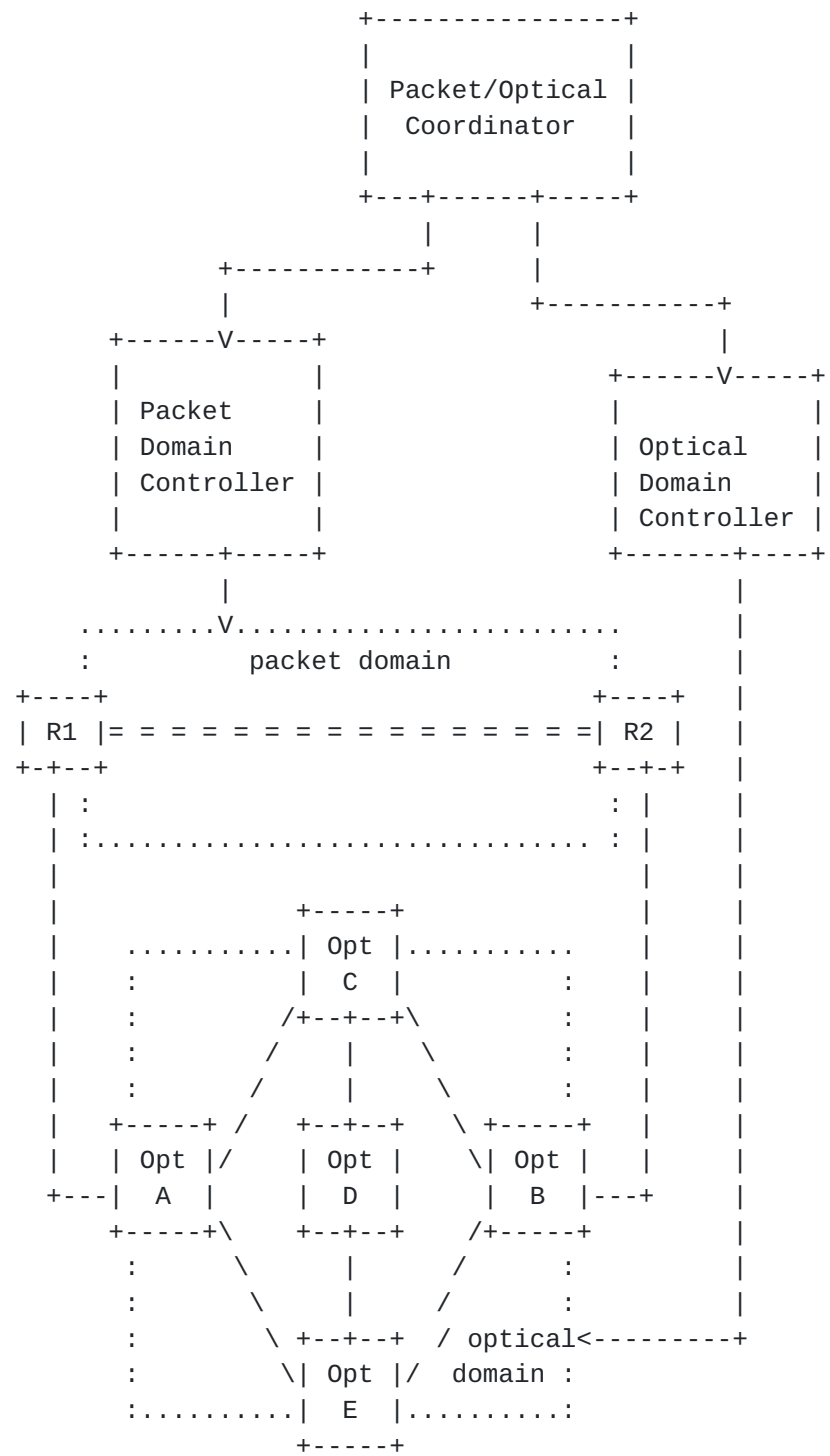


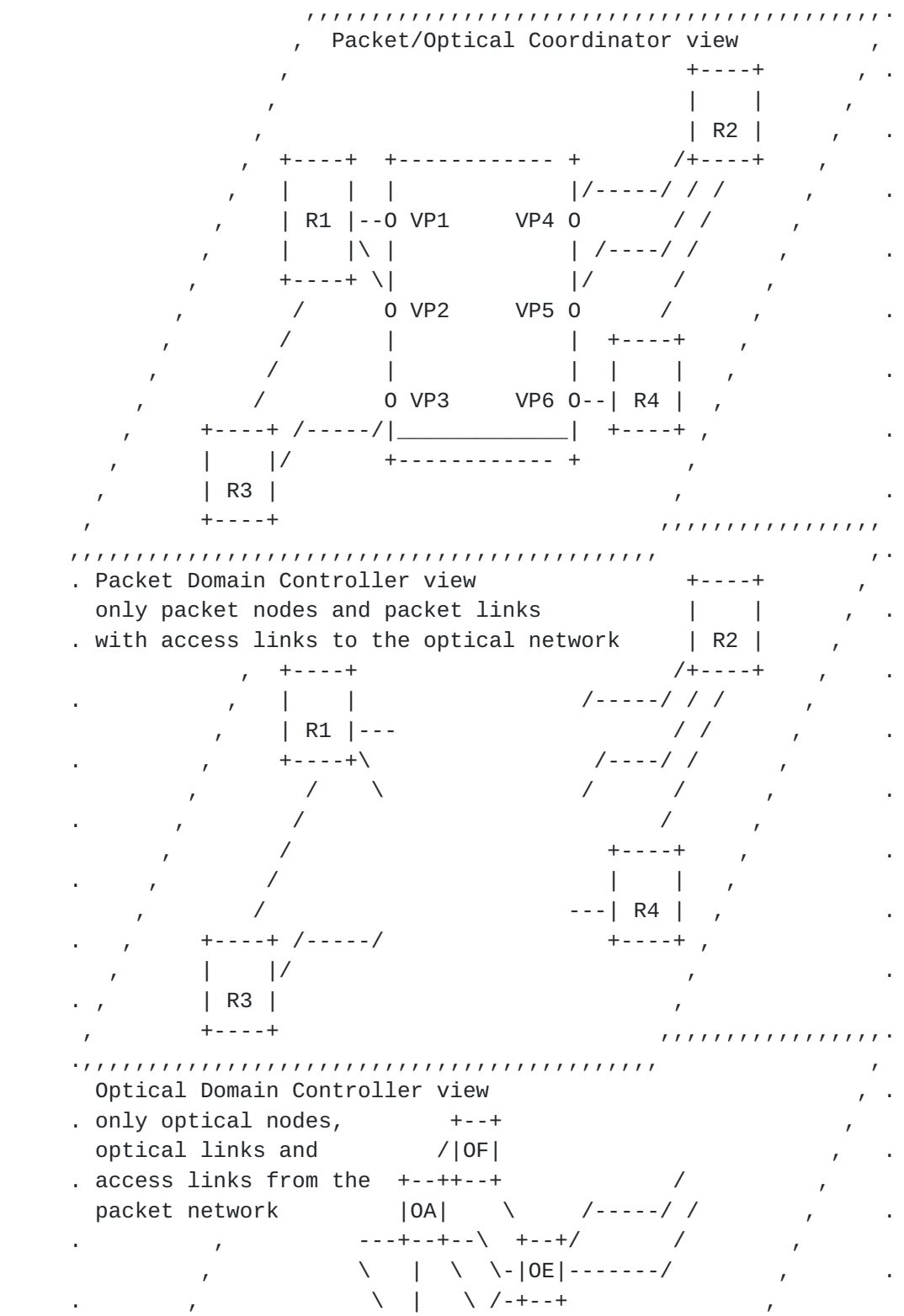
Figure 1 - Packet/Optical Integration use case

Figure 1 as well as Figure 2 below only show a partial view of the packet network connectivity, before additional packet connectivity is provided by the optical network.

It is assumed that the Optical Domain Controller provides to the Packet/Optical Coordinator an abstracted view of the optical network. A possible abstraction could be to represent the whole optical network as one "virtual node" with "virtual ports" connected to the access links, as shown in Figure 2.

It is also assumed that Packet Domain Controller can provide the Packet/Optical Coordinator the information it needs to set up connectivity between packet nodes through the optical network (e.g., the access links).

The path computation request helps the Packet/Optical Coordinator to know the real connections that can be provided by the optical network.


Packet/Optical Coordinator view
R2
R1
VP1
VP2
VP3
VP4
VP5
VP6
R4
R3
Packet Domain Controller view
only packet nodes and packet links
with access links to the optical network
R2
R1
R4
R3
Optical Domain Controller view
only optical nodes,
optical links and
access links from the
packet network
OF
OA
OE

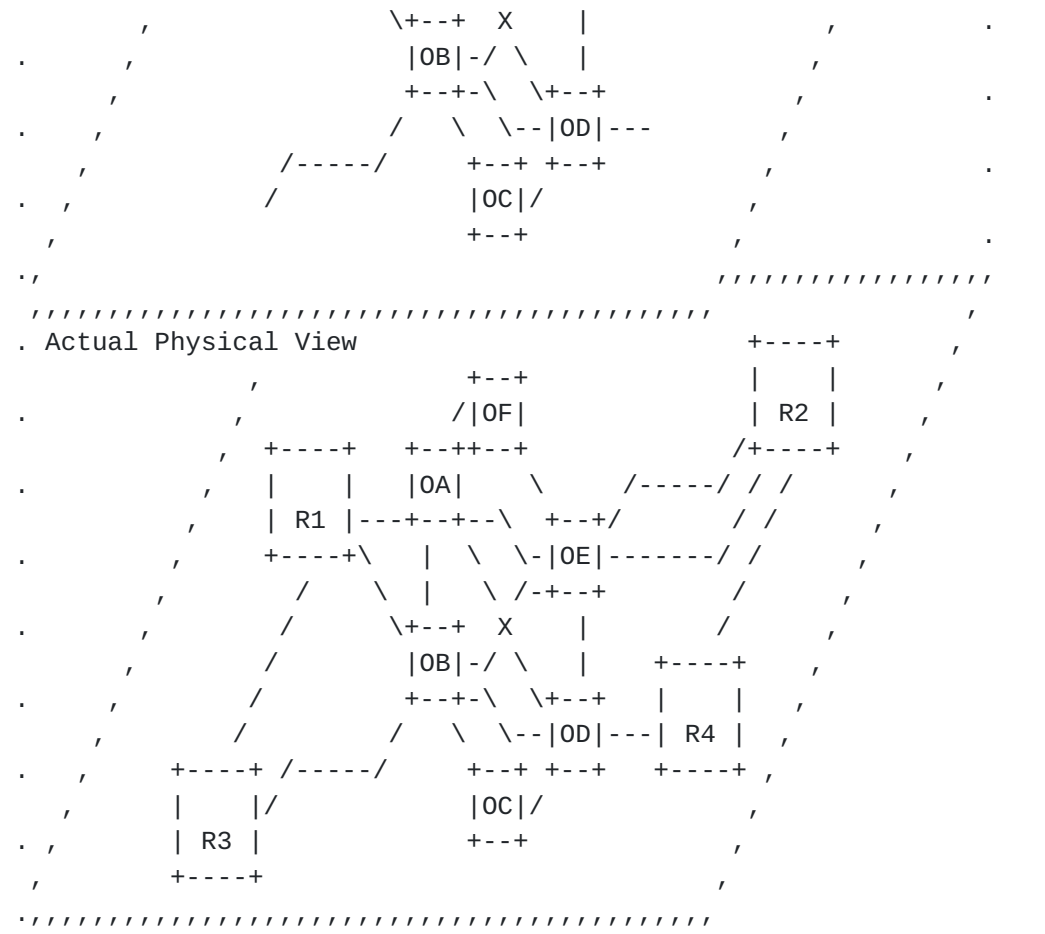


Figure 2 - Packet and Optical Topology Abstractions

In this use case, the Packet/Optical Coordinator needs to set up an optimal underlying path for an IP link between R1 and R2.

As depicted in Figure 2, the Packet/Optical Coordinator has only an "abstracted view" of the physical network, and it does not know the feasibility or the cost of the possible optical paths (e.g., VP1-VP4 and VP2-VP5), which depend on the current status of the physical resources within the optical network and on vendor-specific optical attributes.

The Packet/Optical Coordinator can request the underlying Optical Domain Controller to compute a set of potential optimal paths, taking into account optical constraints. Then, based on its own constraints, policy and knowledge (e.g. cost of the access links), it can choose

which one of these potential paths to use to set up the optimal end-to-end path crossing optical network.

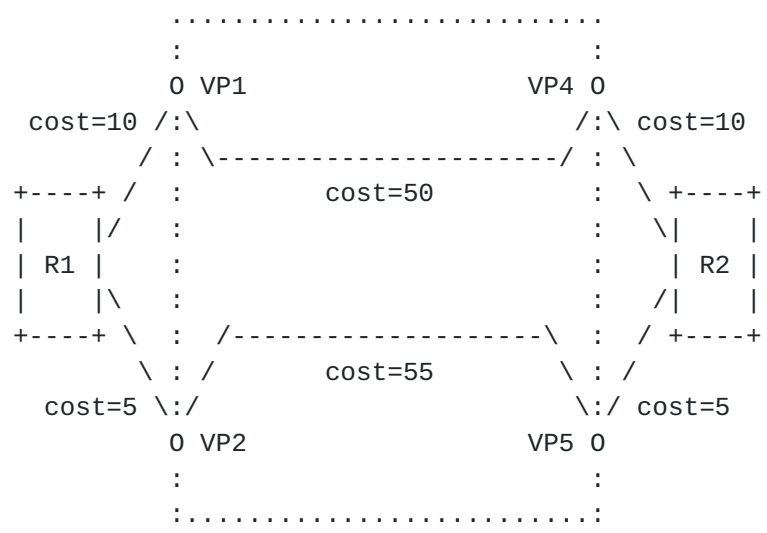


Figure 3 - Packet/Optical Integration path computation example

For example, in Figure 3, the Packet/Optical Coordinator can request the Optical Domain Controller to compute the paths between VP1-VP4 and VP2-VP5 and then decide to set up the optimal end-to-end path using the VP2-VP5 optical path even if this is not the optimal path from the optical domain perspective.

Considering the dynamicity of the connectivity constraints of an optical domain, it is possible that a path computed by the Optical Domain Controller when requested by the Packet/Optical Coordinator is no longer valid/available when the Packet/Optical Coordinator requests it to be set up. This is further discussed in section 3.3.

2.2. Multi-domain TE networks

In this use case there are two TE domains which are interconnected together by multiple inter-domains links.

A possible example could be a multi-domain optical network.

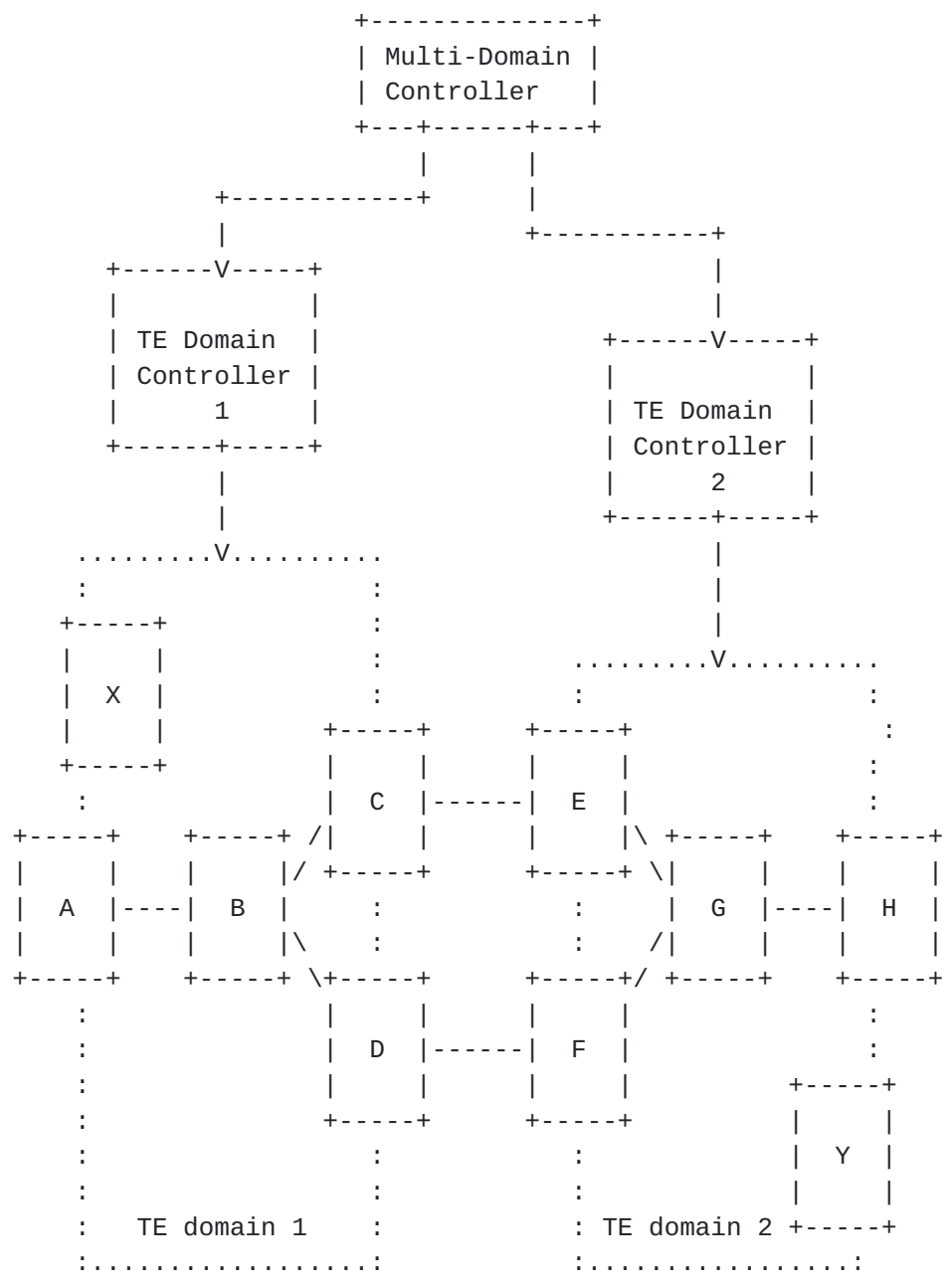


Figure 4 - Multi-domain multi-link interconnection

In order to set up an end-to-end multi-domain TE path (e.g., between nodes A and H), the Multi-Domain Controller needs to know the feasibility or the cost of the possible TE paths within the two TE domains, which depend on the current status of the physical resources within each TE domain. This is more challenging in case of optical

networks because the optimal paths depend also on vendor-specific optical attributes (which may be different in the two domains if they are provided by different vendors).

In order to set up a multi-domain TE path (e.g., between nodes A and H), the Multi-Domain Controller can request the TE Domain Controllers to compute a set of intra-domain optimal paths and take decisions based on the information received. For example:

- The Multi-Domain Controller asks TE Domain Controllers to provide set of paths between A-C, A-D, E-H and F-H
- TE Domain Controllers return a set of feasible paths with the associated costs: the path A-C is not part of this set (in optical networks, it is typical to have some paths not being feasible due to optical constraints that are known only by the Optical Domain Controller)
- The Multi-Domain Controller will select the path A-D-F-H since it is the only feasible multi-domain path and then request the TE Domain Controllers to set up the A-D and F-H intra-domain paths
- If there are multiple feasible paths, the Multi-Domain Controller can select the optimal path knowing the cost of the intra-domain paths (provided by the TE domain controllers) and the cost of the inter-domain links (known by the Multi-Domain Controller)

This approach may have some scalability issues when the number of TE domains is quite big (e.g. 20).

In this case, it would be worthwhile using the abstract TE topology information provided by the TE Domain Controllers to limit the number of potential optimal end-to-end paths and then request path computation from fewer TE Domain Controllers in order to decide what the optimal path within this limited set is.

For more details, see section 3.2.3.

2.3. Data Center Interconnections

In these use case, there is a TE domain which is used to provide connectivity between Data Centers which are connected with the TE domain using access links.

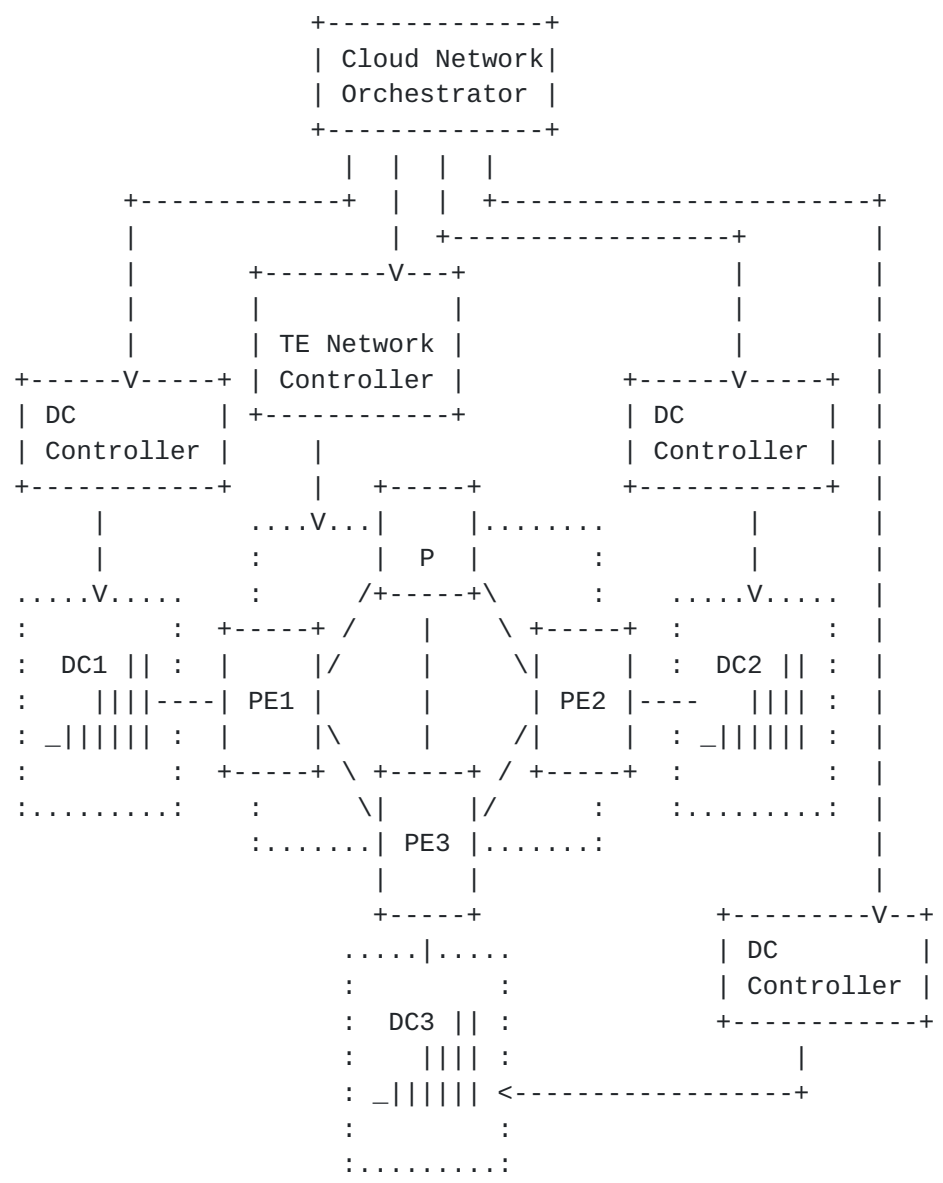


Figure 5 - Data Center Interconnection use case

In this use case, there is the need to transfer data from Data Center 1 (DC1) to either DC2 or DC3 (e.g. workload migration).

The optimal decision depends both on the cost of the TE path (DC1-DC2 or DC1-DC3) and of the Data Center resources within DC2 or DC3.

The Cloud Network Orchestrator needs to make a decision for optimal connection based on TE network constraints and Data Center resources.

It may not be able to make this decision because it has only an abstract view of the TE network (as in use case in 2.1).

The Cloud Network Orchestrator can request to the TE Network Controller to compute the cost of the possible TE paths (e.g., DC1-DC2 and DC1-DC3) and to the DC Controller to provide the information it needs about the required Data Center resources within DC2 and DC3 and then it can take the decision about the optimal solution based on this information and its policy.

2.4. Backward Recursive Path Computation scenario

[RFC5441] has defined the Virtual Source Path Tree (VSPT) TLV within PCE Reply Object in order to compute inter-domain paths following a "Backward Recursive Path Computation" (BRPC) method. The main principle is to forward the PCE request message up to the destination domain. Then, each PCE involved in the computation will compute its part of the path and send it back to the requester through PCE Response message. The resulting computation is spread from destination PCE to source PCE. Each PCE is in charge of merging the path it received with the one it calculated. At the end, the source PCE merges its local part of the path with the received one to achieve the end-to-end path.

Figure 6 below show a typical BRPC scenario where 3 PCEs cooperate to compute inter-domain paths.

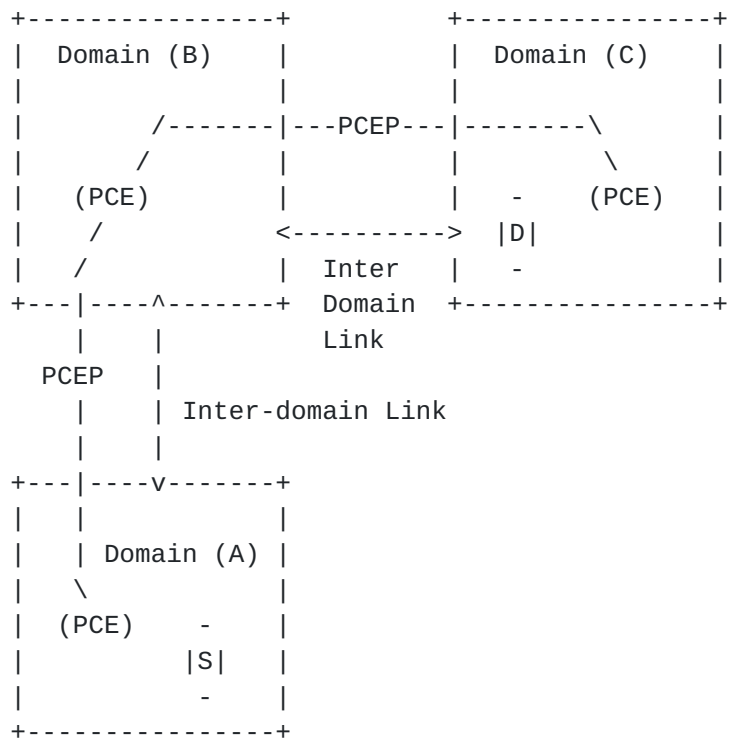


Figure 6 - BRPC Scenario

In this use case, a client can use the YANG data model defined in this document to request path computation from the PCE that controls the source of the tunnel. For example, a client can request to the PCE of domain A to compute a path from a source S, within domain A, to a destination D, within domain C. Then PCE of domain A will use PCEP protocol, as per [RFC5441], to compute the path from S to D and in turn gives the final answer to the requester.

2.5. Hierarchical PCE scenario

[RFC6805] has defined an architecture and extensions to the PCE standard to compute inter-domain path following a hierarchical method. Two new roles have been defined: parent PCE and child PCE. The parent PCE is in charge to coordinate the end-to-end path computation. For that purpose it sends to each child PCE involved in the multi-domain path computation a PCE Request message to obtain the local part of the path. Once received all answer through PCE Response message, the parent PCE will merge the different local parts of the path to achieve the end-to-end path.

Figure 7 below shows a typical hierarchical scenario where a parent PCE request end-to-end path to the different child PCE. Note that a PCE could take independently the role of child or parent PCE depending of which PCE will request the path.

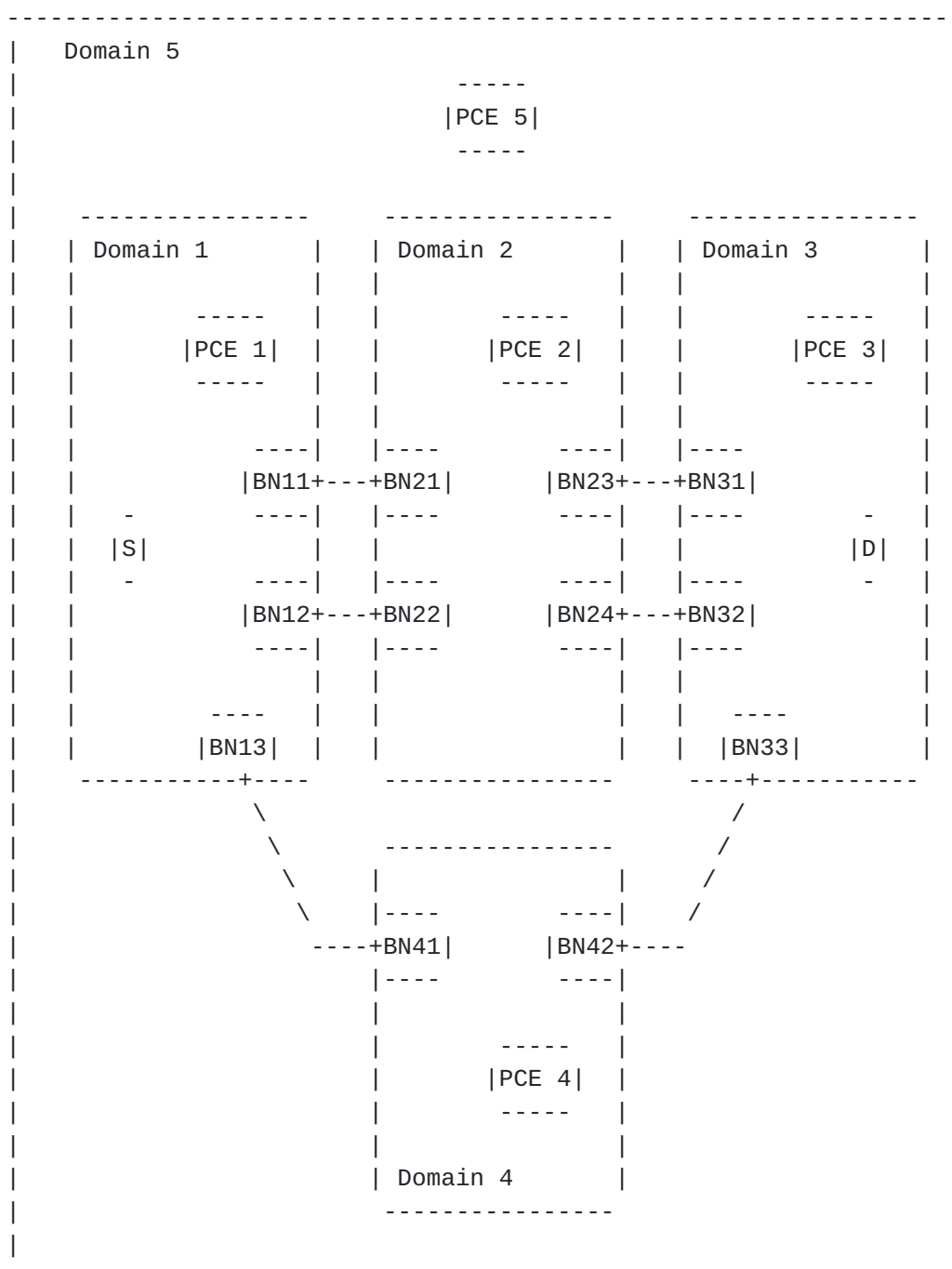


Figure 7 - Hierarchical domain topology from [RFC6805]

In this use case, a client can use the YANG data model defined in this document to request to the parent PCE a path from a source S to a destination D. The parent PCE will in turn contact the child PCEs

through PCEP protocol to compute the end-to-end path and then return the computed path to the client, using the YANG data model defined in this document. For example the YANG data model can be used to request to PCE5 acting as parent PCE to compute a path from source S, within domain 1, to destination D, within domain 3. PCE5 will contact child PCEs of domain 1, 2 and 3 to obtain local part of the end-to-end path through the PCEP protocol. Once received the PCE Response message, it merges the answers to compute the end-to-end path and send it back to the client.

3. Motivations

This section provides the motivation for the YANG data model defined in this document.

Section 3.1 describes the motivation for a YANG data model to request path computation.

Section 3.2 describes the motivation for a YANG data model which complements the TE topology YANG data model defined in [RFC8795].

Section 3.3 describes the motivation for a YANG RPC which complements the TE tunnel YANG data model defined in [TE-TUNNEL].

3.1. Motivation for a YANG Model

3.1.1. Benefits of common data models

The YANG data model for requesting path computation is closely aligned with the YANG data models that provide (abstract) TE topology information, i.e., [RFC8795] as well as that are used to configure and manage TE tunnels, i.e., [TE-TUNNEL].

There are many benefits in aligning the data model used for path computation requests with the YANG data models used for TE topology information and for TE tunnels configuration and management:

- There is no need for an error-prone mapping or correlation of information.

- It is possible to use the same endpoint identifiers in path computation requests and in the topology modeling.

- The attributes used for path computation constraints are the same as those used when setting up a TE tunnel.

3.1.2. Benefits of a single interface

The system integration effort is typically lower if a single, consistent interface is used by controllers, i.e., one data modeling language (i.e., YANG) and a common protocol (e.g., NETCONF or RESTCONF).

Practical benefits of using a single, consistent interface include:

1. Simple authentication and authorization: The interface between different components has to be secured. If different protocols have different security mechanisms, ensuring a common access control model may result in overhead. For instance, there may be a need to deal with different security mechanisms, e.g., different credentials or keys. This can result in increased integration effort.

2. Consistency: Keeping data consistent over multiple different interfaces or protocols is not trivial. For instance, the sequence of actions can matter in certain use cases, or transaction semantics could be desired. While ensuring consistency within one protocol can already be challenging, it is typically cumbersome to achieve that across different protocols.

3. Testing: System integration requires comprehensive testing, including corner cases. The more different technologies are involved, the more difficult it is to run comprehensive test cases and ensure proper integration.

4. Middle-box friendliness: Provider and consumer of path computation requests may be located in different networks, and middle-boxes such as firewalls, NATs, or load balancers may be deployed. In such environments it is simpler to deploy a single protocol. Also, it may be easier to debug connectivity problems.

5. Tooling reuse: Implementers may want to implement path computation requests with tools and libraries that already exist in controllers and/or orchestrators, e.g., leveraging the rapidly growing eco-system for YANG tooling.

3.1.3. Extensibility

Path computation is only a subset of the typical functionality of a controller. In many use cases, issuing path computation requests comes along with the need to access other functionality on the same system. In addition to obtaining TE topology, for instance also configuration of services (set-up/modification/deletion) may be required, as well as:

1. Receiving notifications for topology changes as well as integration with fault management

2. Performance management such as retrieving monitoring and telemetry data

3. Service assurance, e.g., by triggering OAM functionality

4. Other fulfilment and provisioning actions beyond tunnels and services, such as changing QoS configurations

YANG is a very extensible and flexible data modeling language that can be used for all these use cases.

3.2. Interactions with TE topology

The use cases described in section 2 have been described assuming that the topology view exported by each underlying controller to its client is aggregated using the "virtual node model", defined in [RFC7926].

TE topology information, e.g., as provided by [RFC8795], could in theory be used by an underlying controller to provide TE information to its client thus allowing a PCE available within its client to perform multi-domain path computation on its own, without requesting path computations to the underlying controllers.

In case the client does not implement a PCE function, as discussed in section 1, it could not perform path computation based on TE topology information and would instead need to request path computation from the underlying controllers to get the information it needs to find the optimal end-to-end path.

In case the client implements a PCE function, as discussed in section 1, the TE topology information needs to be complete and accurate, which would bring to scalability issues.

Using TE topology to provide a "virtual link model" aggregation, as described in [RFC7926], may be insufficient, unless the aggregation provides a complete and accurate information, which would still cause scalability issues, as described in sections 3.2.1 below.

Using TE topology abstraction, as described in [RFC7926], may lead to compute an unfeasible path, as described in [RFC7926] in section 3.2.2 below.

Therefore when computing an optimal multi-domain path, there is a scalability trade-off between providing complete and accurate TE information and the number of path computation requests to the underlying controllers.

The TE topology information used, in a complimentary way, to reduce the number for path computation requests to the underlying controllers, are described in section 3.2.3 below.

3.2.1. TE topology aggregation

Using the TE topology model, as defined in [RFC8795], the underlying controller can export the whole TE domain as a single TE node with a "detailed connectivity matrix" (which provides specific TE attributes, such as delay, Shared Risk Link Groups (SRLGs) and other TE metrics, between each ingress and egress links).

The information provided by the "detailed connectivity matrix" would be equivalent to the information that should be provided by "virtual link model" as defined in [RFC7926].

For example, in the Packet/Optical Integration use case, described in section 2.1, the Optical Domain Controller can make the information shown in Figure 3 available to the Packet/Optical Coordinator as part of the TE topology information and the Packet/Optical Coordinator could use this information to calculate on its own the optimal path between R1 and R2, without requesting any additional information to the Optical Domain Controller.

However, when designing the amount of information to provide within the "detailed connectivity matrix", there is a tradeoff to be

considered between accuracy (i.e., providing "all" the information that might be needed by the PCE available within the client) and scalability.

Figure 8 below shows another example, similar to Figure 3, where there are two possible Optical paths between VP1 and VP4 with different properties (e.g., available bandwidth and cost).

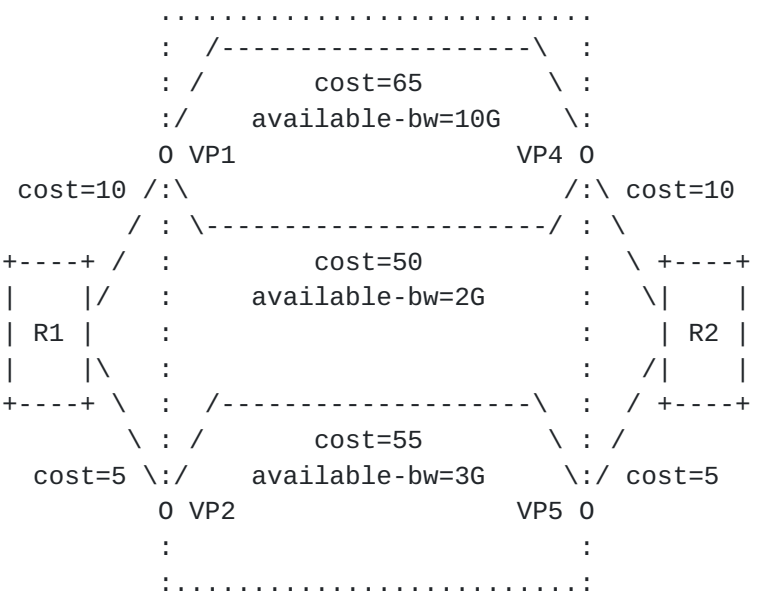


Figure 8 - Packet/Optical Integration path computation example with multiple choices

If the information in the "detailed connectivity matrix" is not complete/accurate, we can have the following drawbacks:

- If only the VP1-VP4 path with available bandwidth of 2 Gb/s and cost 50 is reported, the client's PCE will fail to compute a 5 Gb/s path between routers R1 and R2, although this would be feasible;

- If only the VP1-VP4 path with available bandwidth of 10 Gb/s and cost 60 is reported, the client's PCE will compute, as optimal, the 1 Gb/s path between R1 and R2 going through the VP2-VP5 path within the optical domain while the optimal path would actually be the one going thought the VP1-VP4 sub-path (with cost 50) within the optical domain.

Reporting all the information, as in Figure 8, using the "detailed connectivity matrix", is quite challenging from a scalability perspective. The amount of this information is not just based on number of end points (which would scale as N-square), but also on many other parameters, including client rate, user constraints/policies for the service, e.g. max latency < N ms, max cost, etc., exclusion policies to route around busy links, min Optical Signal to Noise Ratio (OSNR) margin, max pre-Forward Error Correction (FEC) Bit Error Rate (BER) etc. All these constraints could be different based on connectivity requirements.

Examples of how the "detailed connectivity matrix" can be dimensioned are described in Appendix A.

It is also worth noting that the "connectivity matrix" has been originally defined in Wavelength Switched Optical Networks (WSON), [RFC7446], to report the connectivity constrains of a physical node within the Wavelength Division Multiplexing (WDM) network: the information it contains is pretty "static" and therefore, once taken and stored in the TE data base, it can be always being considered valid and up-to-date in path computation request.

The "connectivity matrix" is sometimes confused with "optical reach table" that contain multiple (e.g. k-shortest) regen-free reachable paths for every A-Z node combination in the network. Optical reach tables can be calculated offline, utilizing vendor optical design and planning tools, and periodically uploaded to the Controller: these optical path reach tables are fairly static. However, to get the connectivity matrix, between any two sites, either a regen free path can be used, if one is available, or multiple regen free paths are concatenated to get from the source to the destination, which can be a very large combination. Additionally, when the optical path within optical domain needs to be computed, it can result in different paths based on input objective, constraints, and network conditions. In summary, even though "optical reach table" is fairly static, which regen free paths to build the connectivity matrix between any source and destination is very dynamic, and is done using very sophisticated routing algorithms.

Using the "basic connectivity matrix" with an abstract node to abstract the information regarding the connectivity constraints of an Optical domain, would make this information more "dynamic" since the connectivity constraints of an optical domain can change over time because some optical paths that are feasible at a given time may

become unfeasible at a later time when e.g., another optical path is established.

The information in the "detailed connectivity matrix" is even more dynamic since the establishment of another optical path may change some of the parameters (e.g., delay or available bandwidth) in the "detailed connectivity matrix" while not changing the feasibility of the path.

There is therefore the need to keep the information in the "detailed connectivity matrix" updated which means that there another tradeoff between the accuracy (i.e., providing "all" the information that might be needed by the client's PCE) and having up-to-date information. The more the information is provided and the longer it takes to keep it up-to-date which increases the likelihood that the client's PCE computes paths using not updated information.

It seems therefore quite challenging to have a "detailed connectivity matrix" that provides accurate, scalable and updated information to allow the client's PCE to take optimal decisions by its own.

Considering the example in Figure 8 with the approach defined in this document, the client, when it needs to set up an end-to-end path, it can request the Optical Domain Controller to compute a set of optimal paths (e.g., for VP1-VP4 and VP2-VP5) and take decisions based on the information received:

- When setting up a 5 Gb/s path between routers R1 and R2, the Optical Domain Controller may report only the VP1-VP4 path as the only feasible path: the Packet/Optical Coordinator can successfully set up the end-to-end path passing though this optical path;

- When setting up a 1 Gb/s path between routers R1 and R2, the Optical Domain Controller (knowing that the path requires only 1 Gb/s) can report both the VP1-VP4 path, with cost 50, and the VP2-VP5 path, with cost 65. The Packet/Optical Coordinator can then compute the optimal path which is passing thought the VP1-VP4 sub-path (with cost 50) within the optical domain.

3.2.2. TE topology abstraction

Using the TE topology model, as defined in [RFC8795], the underlying controller can export to its client an abstract TE topology, composed

by a set of TE nodes and TE links, representing the abstract view of the topology under its control.

For example, in the multi-domain TE network use case, described in section 2.2, the TE Domain Controller 1 can export a TE topology encompassing the TE nodes A, B, C and D and the TE links interconnecting them. In a similar way, the TE Domain Controller 2 can export a TE topology encompassing the TE nodes E, F, G and H and the TE links interconnecting them.

In this example, for simplicity reasons, each abstract TE node maps with each physical node, but this is not necessary.

In order to set up a multi-domain TE path (e.g., between nodes A and H), the Multi-Domain Controller can compute by its own an optimal end-to-end path based on the abstract TE topology information provided by the domain controllers. For example:

- Multi-Domain Controller can compute, based on its own TED data, the optimal multi-domain path being A-B-C-E-G-H, and then request the TE Domain Controllers to set up the A-B-C and E-G-H intra-domain paths

- But, during path set-up, the TE Domain Controller may find out that A-B-C intra-domain path is not feasible (as discussed in section 2.2, in optical networks it is typical to have some paths not being feasible due to optical constraints that are known only by the Optical Domain Controller), while only the path A-B-D is feasible

- So what the Multi-Domain Controller has computed is not good and it needs to re-start the path computation from scratch

As discussed in section 3.2.1, providing more extensive abstract information from the TE Domain Controllers to the Multi-Domain Controller may lead to scalability problems.

In a sense this is similar to the problem of routing and wavelength assignment within an optical domain. It is possible to do first routing (step 1) and then wavelength assignment (step 2), but the chances of ending up with a good path is low. Alternatively, it is possible to do combined routing and wavelength assignment, which is known to be a more optimal and effective way for optical path set-up. Similarly, it is possible to first compute an abstract end-to-end

path within the Multi-Domain Controller (step 1) and then compute an intra-domain path within each optical domain (step 2), but there are more chances not to find a path or to get a suboptimal path than performing multiple per-domain path computations and then stitch them.

3.2.3. Complementary use of TE topology and path computation

As discussed in section 2.2, there are some scalability issues with path computation requests in a multi-domain TE network with many TE domains, in terms of the number of requests to send to the TE Domain Controllers. It would therefore be worthwhile using the abstract TE topology information provided by the TE Domain Controllers to limit the number of requests.

An example can be described considering the multi-domain abstract TE topology shown in Figure 9. In this example, an end-to-end TE path between domains A and F needs to be set up. The transit TE domain should be selected between domains B, C, D and E.

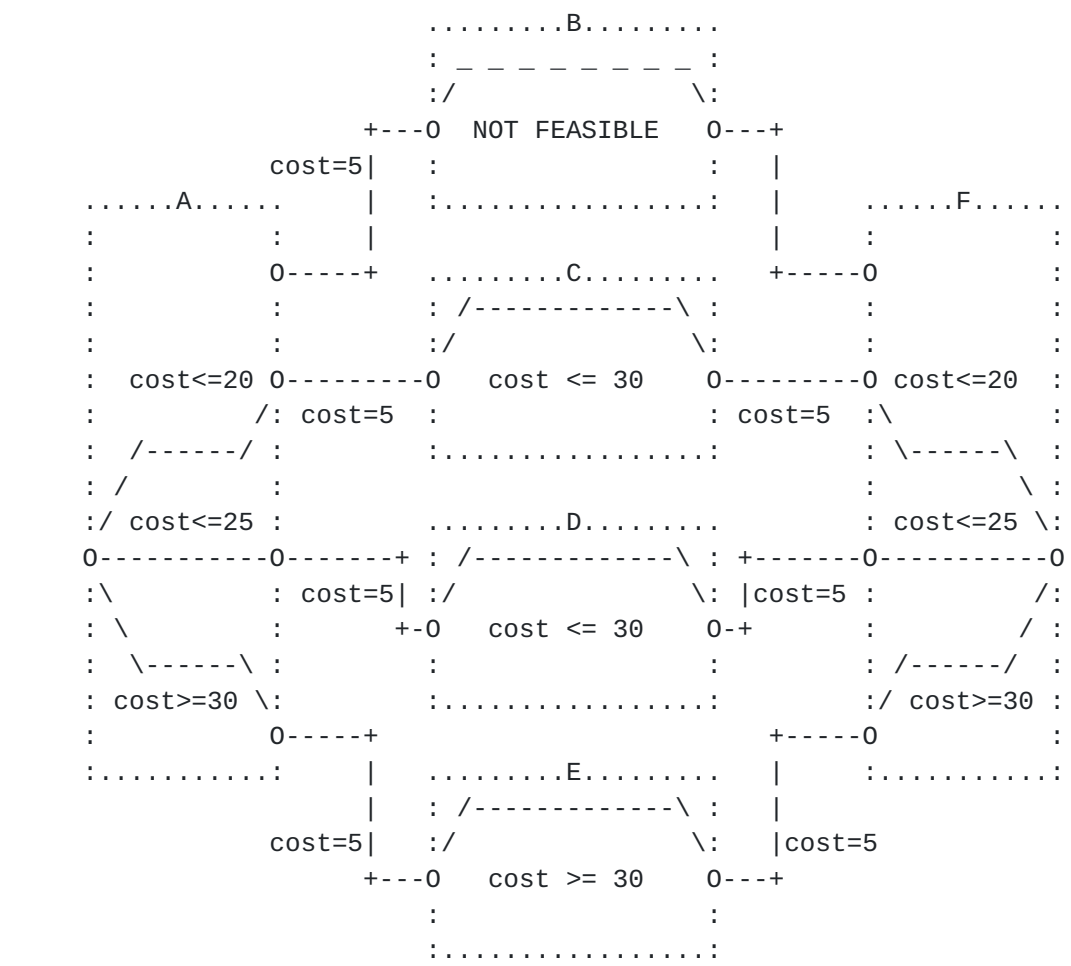


Figure 9 - Multi-domain with many domains (Topology information)

The actual cost of each intra-domain path is not known a priori from the abstract topology information. The Multi-Domain Controller only knows, from the TE topology provided by the underlying domain controllers, the feasibility of some intra-domain paths and some upper-bound and/or lower-bound cost information. With this information, together with the cost of inter-domain links, the Multi-Domain Controller can understand by its own that:

- Domain B cannot be selected as the path connecting domains A and F is not feasible;

- Domain E cannot be selected as a transit domain since it is known from the abstract topology information provided by domain controllers that the cost of the multi-domain path A-E-F (which is 100, in the best case) will be always be higher than the cost of the multi-domain paths A-D-F (which is 90, in the worst case) and A-C-F (which is 80, in the worst case).

Therefore, the Multi-Domain Controller can understand by its own that the optimal multi-domain path could be either A-D-F or A-C-F but it cannot know which one of the two possible option actually provides the optimal end-to-end path.

The Multi-Domain Controller can therefore request path computation only to the TE Domain Controllers A, D, C and F (and not to all the possible TE Domain Controllers).

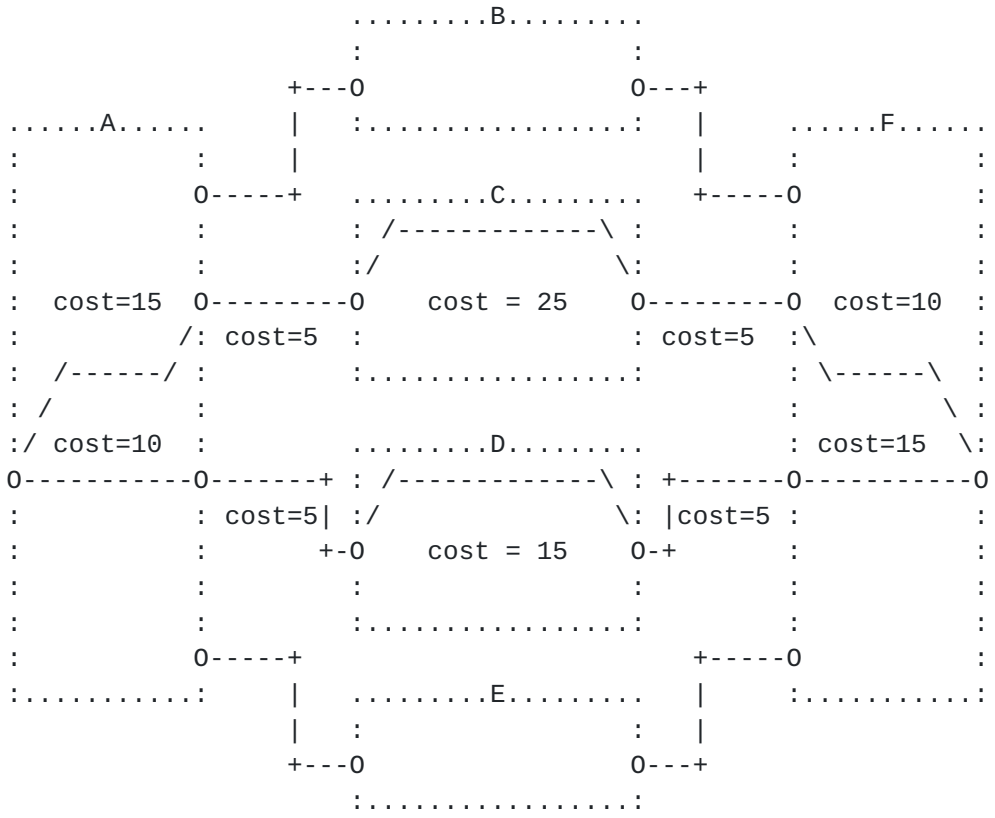


Figure 10 - Multi-domain with many domains (Path Computation information)

Based on these requests, the Multi-Domain Controller can know the actual cost of each intra-domain paths which belongs to potential optimal end-to-end paths, as shown in Figure 10, and then compute the optimal end-to-end path (e.g., A-D-F, having total cost of 50, instead of A-C-F having a total cost of 70).

3.3. Path Computation RPC

The TE tunnel YANG data model, defined in [TE-TUNNEL], can support the need to request path computation, as described in section 5.1.2 of [TE-TUNNEL].

This solution is stateful since the state of each created "compute-only" TE tunnel path needs to be maintained, in the YANG datastores (at least in the running datastore and operational datastore), and updated, when underlying network conditions change.

The RPC mechanism allows requesting path computation using a simple atomic operation, without creating any state in the YANG datastores, and it is the natural option/choice, especially with stateless PCE.

It is very useful to provide both the options of using an RPC as well as of setting up TE tunnel paths in "compute-only" mode. It is suggested to use the RPC as much as possible and to rely on "compute-only" TE tunnel paths, when really needed.

Using the RPC solution would imply that the underlying controller (e.g., a PNC) computes a path twice during the process to set up an LSP: at time T1, when its client (e.g., an MDSC) sends a path computation RPC request to it, and later, at time T2, when the same client (MDSC) creates a TE tunnel requesting the set-up of the LSP. The underlying assumption is that, if network conditions have not changed, the same path that has been computed at time T1 is also computed at time T2 by the underlying controller (e.g. PNC) and therefore the path that is set up at time T2 is exactly the same path that has been computed at time T1.

However, since the operation is stateless, there is no guarantee that the returned path would still be available when path set-up is requested: this does not cause major issues when the time between path computation and path set-up is short (especially if compared with the time that would be needed to update the information of a very detailed connectivity matrix).

In most of the cases, there is even no need to guarantee that the path that has been set up is the exactly same as the path that has been returned by path computation, especially if it has the same or even better metrics. Depending on the abstraction level applied by the server, the client may also not know the actual computed path.

The most important requirement is that the required global objectives (e.g., multi-domain path metrics and constraints) are met. For this reason a path verification phase is always necessary to verify that the actual path that has been set up meets the global objectives (for example in a multi-domain network, the resulting end-to-end path meets the required end-to-end metrics and constraints).

In most of the cases, even if the path being set up is not exactly the same as the path returned by path computation, its metrics and constraints are "good enough" and the path verification passes successfully. In the few corner cases where the path verification fails, it is possible repeat the whole process (path computation, path set-up and path verification).

In case it is required to set up at T2 exactly the same path computed at T1, the RPC solution should not be used and, instead, a "compute-only" TE tunnel path should be set up, allowing also notifications in case the computed path has been changed.

In this case, at time T1, the client (MDSC) creates a TE tunnel in a compute-only mode in the running datastore and later, at time T2, changes the configuration of that TE tunnel (not to be any more in a compute-only mode) to trigger the set-up of the LSP over the path which have been computed at time T1 and reported in the operational datastore.

It is worth noting that also using the "compute-only" TE tunnel path, although increasing the likelihood that the computed path is available at path set-up, does not guarantee that because notifications may not be reliable or delivered on time. Path verification is needed also in this case.

The solution based on "compute-only" TE tunnel path has also the following drawbacks:

- Several messages required for any path computation

- Requires persistent storage in the underlying controller

- Need for garbage collection for stranded paths
- Process burden to detect changes on the computed paths in order to provide notifications update

3.3.1. Temporary reporting of the computed path state

This section describes an optional extension to the stateless behavior of the path computation RPC, where the underlying controller, after having received a path computation RPC request, maintains some "transient state" associated with the computed path, allowing the client to request the set-up of exactly that path, if still available.

This is similar to the "compute-only" TE tunnel path solution but, to avoid the drawbacks of the stateful approach, is leveraging the path computation RPC and the separation between configuration and operational datastore, as defined in the NMDA architecture [RFC8342].

The underlying controller, after having computed a path, as requested by a path computation RPC, also creates a TE tunnel instance within the operational datastore, to store that computed path. This would be similar to a "compute-only" TE tunnel path, with the only difference that there is no associated TE tunnel instance within the running datastore.

Since the underlying controller stores in the operational datastore the computed path based on an abstract topology it exposes, it also remembers, internally, which is the actual native path (physical path), within its native topology (physical topology), associated with that compute-only TE tunnel instance.

Afterwards, the client (e.g., MDSC) can request the set-up of that specific path by creating a TE tunnel instance (not in compute-only mode) in the running datastore using the same tunnel-name of the existing TE tunnel in the operational datastore: this will trigger the underlying controller to set up that path, if still available.

There are still cases where the path being set up is not exactly the same as the path that has been computed:

- When the tunnel is configured with path constraints which are not compatible with the computed path;

- When the tunnel set-up is requested after the resources of the computed path are no longer available;

- When the tunnel set-up is requested after the computed path is no longer known (e.g. due to a server reboot) by the underlying controller.

In all these cases, the underlying controller should compute and set up a new path.

Therefore the "path verification" phase, as described in section 3.3 above, is always needed to check that the path that has been set up is still "good enough".

Since this new approach is not completely stateless, garbage collection is implemented using a timeout that, when it expires, triggers the removal of the computed path from the operational datastore. This operation is fully controlled by the underlying controller without the need for any action to be taken by the client that is not able to act on the operational datastore. The default value of this timeout is 10 minutes but a different value may be configured by the client.

In addition, it is possible for the client to tag each path computation request with a transaction-id allowing for a faster removal of all the paths associated with a transaction-id, without waiting for their timers to expire.

The underlying controller can remove from the operational datastore all the paths computed with a given transaction-id which have not been set up either when it receives a Path Delete RPC request for that transaction-id or, automatically, right after the set-up up of a path that has been previously computed with that transaction-id.

This possibility is useful when multiple paths are computed but, at most, only one is set up (e.g., in multi-domain path computation scenario scenarios). After the selected path has been set up (e.g, in one domain during multi-domain path set-up), all the other alternative computed paths can be automatically deleted by the underlying controller (since no longer needed). The client can also request, using the Path Delete RPC request, the underlying controller to remove all the computed paths, if none of them is going to be set up (e.g., in a transit domain not being selected by multi-domain path computation and so not being automatically deleted).

This approach is complimentary and not alternative to the timer which is always needed to avoid stranded computed paths being stored in the operational datastore when no path is set up and no explicit Path Delete RPC request is received.

4. Path computation and optimization for multiple paths

There are use cases, where it is advantageous to request path computation for a set of paths, through a network or through a network domain, using a single request [RFC5440].

In this case, sending a single request for multiple path computations, instead of sending multiple requests for each path computation, would reduce the protocol overhead and it would consume less resources (e.g., threads in the client and server).

In the context of a typical multi-domain TE network, there could multiple choices for the ingress/egress points of a domain and the Multi-Domain Controller needs to request path computation between all the ingress/egress pairs to select the best pair. For example, in the example of section 2.2, the Multi-Domain Controller needs to request the TE Network Controller 1 to compute the A-C and the A-D paths and to the TE Network Controller 2 to compute the E-H and the F-H paths.

It is also possible that the Multi-Domain Controller receives a request to set up a group of multiple end to end connections. The Multi-Domain Controller needs to request each TE domain Controller to compute multiple paths, one (or more) for each end to end connection.

There are also scenarios where it can be needed to request path computation for a set of paths in a synchronized fashion.

One example could be computing multiple diverse paths. Computing a set of diverse paths in an unsynchronized fashion, leads to the possibility of not being able to satisfy the diversity requirement. In this case, it is preferable to compute a sub-optimal primary path for which a diversely routed secondary path exists.

There are also scenarios where it is needed to request optimizing a set of paths using objective functions that apply to the whole set of paths, see [RFC5541], e.g. to minimize the sum of the costs of all the computed paths in the set.

5. YANG data model for requesting Path Computation

This document define a YANG RPC to request path computation as an "augmentation" of tunnel-rpc, defined in [TE-TUNNEL]. This model provides the RPC input attributes that are needed to request path computation and the RPC output attributes that are needed to report the computed paths.

```
  augment /te:tunnels-path-compute/te:input/te:path-compute-info:
    +-- path-request* [request-id]
    |  +-- request-id                          uint32
    |  ...........

  augment /te:tunnels-path-compute/te:output/te:path-compute-result:
    +--ro response* [response-id]
       +--ro response-id                       uint32
       +--ro computed-paths-properties
       |  +--ro computed-path-properties* [k-index]
       |     +--ro k-index            uint8
       |     +--ro path-properties
       |     ...........
```

This model extensively re-uses the grouping defined in [TE-TUNNEL] and [RFC8776] to ensure maximal syntax and semantics commonality.

This YANG data model allows one RPC to include multiple path requests, each path request being identified by a request-id. Therefore, one RPC can return multiple responses, one for each path request, being identified by a response-id equal to the corresponding request-id. Each response reports one or more computed paths, as requested by the k-requested-paths attribute. By default, each response reports one computed path.

5.1. Synchronization of multiple path computation requests

The YANG data model permits the synchronization of a set of multiple path requests (identified by specific request-id) all related to a "svec" container emulating the syntax of the Synchronization VECtor (SVEC) PCEP object, defined in [RFC5440].

```
    +-- synchronization* []
       +-- svec
```

```
|  +-- relaxable?           boolean
|  +-- disjointness?        te-path-disjointness
|  +-- request-id-number*   uint32
+-- svec-constraints
|  +-- path-metric-bound* [metric-type]
|     +-- metric-type    identityref
|     +-- upper-bound?   uint64
+-- path-srlgs-lists
|  +-- path-srlgs-list* [usage]
|     +-- usage     identityref
|     +-- values*   srlg
+-- path-srlgs-names
|  +-- path-srlgs-name* [usage]
|     +-- usage    identityref
|     +-- names*   string
+-- exclude-objects
|  +-- excludes* []
|     +-- (type)?
|        +--:(numbered-node-hop)
|        |  +-- numbered-node-hop
|        |     +-- node-id     te-node-id
|        |     +-- hop-type?   te-hop-type
|        +--:(numbered-link-hop)
|        |  +-- numbered-link-hop
|        |     +-- link-tp-id    te-tp-id
|        |     +-- hop-type?     te-hop-type
|        |     +-- direction?    te-link-direction
|        +--:(unnumbered-link-hop)
|        |  +-- unnumbered-link-hop
|        |     +-- link-tp-id    te-tp-id
|        |     +-- node-id       te-node-id
|        |     +-- hop-type?     te-hop-type
|        |     +-- direction?    te-link-direction
|        +--:(as-number)
|        |  +-- as-number-hop
|        |     +-- as-number    inet:as-number
|        |     +-- hop-type?    te-hop-type
|        +--:(label)
|           +-- label-hop
```

```
          |              +-- te-label
          |                 +-- (technology)?
          |                 |  +--:(generic)
          |                 |     +-- generic?
          |                 |              rt-types:generalized-label
          |                 +-- direction?       te-label-direction
          +-- optimizations
             +-- (algorithm)?
                +--:(metric) {te-types:path-optimization-metric}?
                |  +-- optimization-metric* [metric-type]
                |     +-- metric-type    identityref
                |     +-- weight?        uint8
                +--:(objective-function)
                          {te-types:path-optimization-objective-
   function}?
                   +-- objective-function
                      +-- objective-function-type?   identityref
```

The model, in addition to the metric types, defined in [RFC8776], which can be applied to each individual path request, supports also additional metric types, which apply to a set of synchronized requests, as referenced in [RFC5541]. These additional metric types are defined by the following YANG identities:

- svec-metric-type: base YANG identity from which cumulative metric types identities are derived.

- svec-metric-cumul-te: cumulative TE cost metric type, as defined in [RFC5541].

- svec-metric-cumul-igp: cumulative IGP cost metric type, as defined in [RFC5541].

- svec-metric-cumul-hop: cumulative Hop metric type, representing the cumulative version of the Hop metric type defined in [RFC8776].

- svec-metric-aggregate-bandwidth-consumption: aggregate bandwidth consumption metric type, as defined in [RFC5541].

- svec-metric-load-of-the-most-loaded-link: load of the most loaded link metric type, as defined in [RFC5541].

5.2. Returned metric values

This YANG data model provides a way to return the values of the metrics computed by the path computation in the output of RPC, together with other important information (e.g. srlg, affinities, explicit route), emulating the syntax of the "C" flag of the "METRIC" PCEP object [RFC5440]:

```
|     +--ro path-properties
|        +--ro path-metric* [metric-type]
|        |  +--ro metric-type           identityref
|        |  +--ro accumulative-value?   uint64
|        +--ro path-affinities-values
|        |  +--ro path-affinities-value* [usage]
|        |     +--ro usage    identityref
|        |     +--ro value?   admin-groups
|        +--ro path-affinity-names
|        |  +--ro path-affinity-name* [usage]
|        |     +--ro usage            identityref
|        |     +--ro affinity-name* [name]
|        |        +--ro name    string
|        +--ro path-srlgs-lists
|        |  +--ro path-srlgs-list* [usage]
|        |     +--ro usage     identityref
|        |     +--ro values*   srlg
|        +--ro path-srlgs-names
|        |  +--ro path-srlgs-name* [usage]
|        |     +--ro usage    identityref
|        |     +--ro names*   string
|        +--ro path-route-objects
|        ...........
```

It also allows the client to request which information (metrics, srlg and/or affinities) should be returned:

```
|  +-- request-id                           uint32
|  ...........
|  +-- requested-metrics* [metric-type]
|  |  +-- metric-type    identityref
|  +-- return-srlgs?                        boolean
```

```
|  +-- return-affinities?                  boolean
|  ...........
```

This feature is essential for path computation in a multi-domain TE network as described in section 2.2. In this case, the metrics returned by a path computation requested to a given underlying controller must be used by the client to compute the best end-to-end path. If they are missing, the client cannot compare different paths calculated by the underlying controllers and choose the best one for the optimal end-to-end (e2e) path.

5.3. Multiple Paths Requests for the same TE tunnel

The YANG data model allows including multiple requests for different paths intended to be used within the same tunnel or within different tunnels.

When multiple requested paths are intended to be used within the same tunnel (e.g., requesting path computation for the primary and secondary paths of a protected tunnel), the set of attributes that are intended to be configured on per-tunnel basis rather than on per-path basis are common to all these path requests. These attributes includes both attributes which can be configured only a per-tunnel basis (e.g., tunnel-name, source/destination TTP, encoding and switching-type) as well attributes which can be configured both on a per-tunnel and on a per-path basis (e.g., the te-bandwidth or the associations).

Therefore, a tunnel-attributes list is defined, within the path computation request RPC:

```
+-- tunnel-attributes* [tunnel-name]
|  +-- tunnel-name              string
|  +-- encoding?                identityref
|  +-- switching-type?          identityref
|  ...........
```

The path requests that are intended to be used within the same tunnel should reference the same entry in the tunnel-attributes list. This allows:

- avoiding repeating the same set of per-tunnel parameters on multiple requested paths;

- the server to understand what attributes are intended to be configured on a per-tunnel basis (e.g., the te-bandwidth configured in the tunnel-attributes) and what attributes are intended to be configured on a per-path basis(e.g., the te-bandwidth configured in the path-request). This could be useful especially when the server also creates a TE tunnel instance within the operational datastore to report the computed paths, as described in section 3.3.1: in this case, the tunnel-name is also used as the suggested name for that TE tunnel instance.

The YANG data model allows also including requests for paths intended to modify existing tunnels (e.g., adding a protection path for an existing un-protected tunnel). In this case, the per-tunnel attributes are already provided in an existing TE tunnel instance and do not need to be re-configured in the path computation request RPC. Therefore, these requests should reference an existing TE tunnel instance.

It is also possible to request computing paths without indicating in which tunnel they are intended to be used (e.g., in case of an unprotected tunnel). In this case, the per-tunnel attributes could be provided together with the per-path attributes in the path request, without using the tunnel-attributes list.

The choices below are defined to distinguish the cases above:

- whether the per-tunnel attributes are configured by reference (providing a leafref), to:
  - either a TE tunnel instance, if it exists;
  - or to an entry of the tunnel-attributes list, if the TE tunnel instance does not exist;
- or by value, providing the set of tunnel attributes within the path request:

```
|  +-- (tunnel-attributes)?
|  |  +--:(reference)
|  |  |  +-- tunnel-reference
|  |  |     +-- (tunnel-exist)?
|  |  |     |  +--:(tunnel-ref)
|  |  |     |  |  +-- tunnel-ref                te:tunnel-ref
|  |  |     |  +--:(tunnel-attributes-ref)
```

```
      |  |  |     |     +-- tunnel-attributes-ref     leafref
      |  |  ...........
      |  |  +--:(value)
      |  |     +-- tunnel-name?                    string
      |  |     ...........
      |  |     +-- encoding?                       identityref
      |  |     +-- switching-type?                 identityref
      |  |     ...........
```

5.3.1. Tunnel attributes specified by value

The (value) case provides the set of attributes that are configured only on per-tunnel basis (e.g., tunnel-name, source/destination TTP, encoding and switching-type).

In this case, it is assumed that the requested path will be the only path within a tunnel.

If the requested path is a transit segment of a multi-domain end-to-end path, it can be of any type (primary, secondary, reverse-primary or reverse-secondary), as specified by the (path-role) choice:

```
      |  |     +-- (path-role)?
      |  |     |  +--:(primary-path)
      |  |     |  +--:(secondary-path)
      |  |     |  |  +-- secondary-path!
      |  |     |  |     +-- primary-path-name?   string
      |  |     |  +--:(primary-reverse-path)
      |  |     |  |  +-- primary-reverse-path!
      |  |     |  |     +-- primary-path-name?   string
      |  |     |  +--:(secondary-reverse-path)
      |  |     |     +-- secondary-reverse-path!
      |  |     |        +-- primary-path-name?           string
      |  |     |        +-- primary-reverse-path-name?   string
```

In all the other cases, the requested path can only be a primary path.

The secondary-path, the primary-reverse-path and the secondary-reverse-path are presence container used to indicate the role of the path: by default, the path is assumed to be a primary path.

They optionally can contain the name of the primary-path under which the requested path could be associated within the YANG tree structure defined in [TE-TUNNEL], which could be useful when the server also creates a TE tunnel instance within the operational datastore to report the computed paths, as described in section 3.3.1: in this case, the tunnel-name and the path names are also used as the suggested name for that TE tunnel and path instances.

5.3.2. Tunnel attributes specified by reference

The (reference) case provides the information needed to associate multiple path requests that are intended to be used within the same tunnel.

In order to indicate the role of the path being requested within the intended tunnel (e.g., primary or secondary path), the (path-role) choice is defined:

```
|  |  |     +-- (path-role)
|  |  |        +--:(primary-path)
|  |  |        |  +-- primary-path!
|  |  |        |     ...........
|  |  |        +--:(secondary-path)
|  |  |        |  +-- secondary-path
|  |  |        |     ...........
|  |  |        +--:(primary-reverse-path)
|  |  |        |  +-- primary-reverse-path
|  |  |        |     ...........
|  |  |        +--:(secondary-reverse-path)
|  |  |           +-- secondary-reverse-path
|  |  |              ...........
```

The primary-path is a presence container used to indicate that the requested path is intended to be used as a primary path. It can also contain some attributes which are configured only on primary paths (e.g., the k-requested-paths).

The secondary-path container indicates that the requested path is intended to be used as a secondary path and it contains at least references to one or more primary paths which can use it as a candidate secondary path:

```
      |  |  |        |  +-- secondary-path
      |  |  |        |     ...........
      |  |  |        |     +-- primary-path-ref* []
      |  |  |        |        +-- (primary-path-exist)?
      |  |  |        |           +--:(path-ref)
      |  |  |        |           |  +-- primary-path-ref    leafref
      |  |  |        |           +--:(path-request-ref)
      |  |  |        |              +-- path-request-ref    leafref
```

A requested secondary path can reference any requested primary paths, and, in case they are intended to be used within an existing TE tunnel, it could also reference any existing primary-paths.

The secondary-path container can also contain some attributes which are configured only on secondary paths (e.g., the protection-type).

The primary-reverse-path container indicates that the requested path is intended to be used as a primary reverse path and it contains only the reference to the primary path which is intended to use it as a reverse path:

```
      |  |  |        |  +-- primary-reverse-path
      |  |  |        |     +-- (primary-path-exist)?
      |  |  |        |        +--:(path-ref)
      |  |  |        |        |  +-- primary-path-ref    leafref
      |  |  |        |        +--:(path-request-ref)
      |  |  |        |           +-- path-request-ref    leafref
```

A requested primary reverse path can reference either a requested primary path, or, in case it is intended to be used within an existing TE tunnel, an existing primary-path.

The secondary-reverse-path container indicates that the requested path is intended to be used as a secondary reverse path and it contains at least references to one or more primary paths, whose primary reverse path can use it as a candidate secondary reverse path:

```
      |  |  |           +-- secondary-reverse-path
      |  |  |              ...........
      |  |  |              +-- primary-reverse-path-ref* []
      |  |  |                 +-- (primary-reverse-path-exist)?
```

```
|  |  |                    +--:(path-ref)
|  |  |                    |  +-- primary-path-ref    leafref
|  |  |                    +--:(path-request-ref)
|  |  |                       +-- path-request-ref    leafref
```

A requested secondary reverse path can reference any requested primary paths, and, in case they are intended to be used within an existing TE tunnel, it could reference also existing primary-paths.

The secondary-reverse-path container can also contain some attributes which are configured only on secondary reverse paths (e.g., the protection-type).

In case the requested path is a transit segment of a multi-domain end-to-end path, the primary-path may not be needed to be setup/computed. However, the request for path computation of a secondary-path or a primary-reverse or of a secondary-reverse-path requires that the primary-path exists or it is requested within the same RPC request. In the latter case, the path request for the primary-path should have an empty ERO to indicate to the server that path computation is not requested and no path properties will therefore be returned in the RPC response.

5.4. Multi-Layer Path Computation

The models supports requesting multi-layer path computation following the same approach based on dependency tunnels, as defined in [TE-TUNNEL].

The tunnel-attributes of a given client-layer path request can reference server-layer TE tunnels which can already exist in the YANG datastore or be specified in the tunnel-attributes list, within the same RPC request:

```
|     +-- dependency-tunnels
|     |  +-- dependency-tunnel* [name]
|     |  |  +-- name              -> /te:te/tunnels/tunnel/name
|     |  |  +-- encoding?         identityref
|     |  |  +-- switching-type?   identityref
|     |  +-- dependency-tunnel-attributes* [name]
|     |     +-- name              leafref
|     |     +-- encoding?         identityref
|     |     +-- switching-type?   identityref
```

In a similar way as in [TE-TUNNEL], the server-layer tunnel attributes should provide the information of what would be the dynamic link in the client layer topology supported by that tunnel, if instantiated:

```
|     +-- hierarchical-link
|        +-- local-te-node-id?         te-types:te-node-id
|        +-- local-te-link-tp-id?      te-types:te-tp-id
|        +-- remote-te-node-id?        te-types:te-node-id
|        +-- te-topology-identifier
|           +-- provider-id?   te-global-id
|           +-- client-id?     te-global-id
|           +-- topology-id?   te-topology-id
```

It is worth noting that since path computation RPC is stateless, the dynamic hierarchical links configured for the server-layer tunnel attributes cannot be used for path computation of any client-layer path unless explicitly referenced in the dependency-tunnel-attributes list within the same RPC request.

6. YANG data model for TE path computation

6.1. Tree diagram

Figure 11 below shows the tree diagram of the YANG data model defined in module ietf-te-path-computation.yang.

```
module: ietf-te-path-computation

  augment /te:tunnels-path-compute/te:input/te:path-compute-info:
    +-- path-request* [request-id]
    |  +-- request-id                              uint32
    |  +-- (tunnel-attributes)?
    |  |  +--:(reference)
    |  |  |  +-- tunnel-reference
    |  |  |     +-- (tunnel-exist)?
    |  |  |     |  +--:(tunnel-ref)
    |  |  |     |  |  +-- tunnel-ref                 te:tunnel-ref
    |  |  |     |  +--:(tunnel-attributes-ref)
    |  |  |     |     +-- tunnel-attributes-ref     leafref
```

```
|  |  |     +-- path-name?                      string
|  |  |     +-- (path-role)
|  |  |        +--:(primary-path)
|  |  |        |  +-- primary-path!
|  |  |        |     +-- preference?          uint8
|  |  |        |     +-- k-requested-paths?   uint8
|  |  |        +--:(secondary-path)
|  |  |        |  +-- secondary-path
|  |  |        |     +-- preference?           uint8
|  |  |        |     +-- protection-type?      identityref
|  |  |        |     +-- restoration-type?     identityref
|  |  |        |     +-- restoration-scheme?   identityref
|  |  |        |     +-- primary-path-ref* []
|  |  |        |        +-- (primary-path-exist)?
|  |  |        |           +--:(path-ref)
|  |  |        |           |  +-- primary-path-ref    leafref
|  |  |        |           +--:(path-request-ref)
|  |  |        |              +-- path-request-ref    leafref
|  |  |        +--:(primary-reverse-path)
|  |  |        |  +-- primary-reverse-path
|  |  |        |     +-- (primary-path-exist)?
|  |  |        |        +--:(path-ref)
|  |  |        |        |  +-- primary-path-ref    leafref
|  |  |        |        +--:(path-request-ref)
|  |  |        |           +-- path-request-ref    leafref
|  |  |        +--:(secondary-reverse-path)
|  |  |           +-- secondary-reverse-path
|  |  |              +-- preference?                 uint8
|  |  |              +-- protection-type?            identityref
|  |  |              +-- restoration-type?           identityref
|  |  |              +-- restoration-scheme?         identityref
|  |  |              +-- primary-reverse-path-ref* []
|  |  |                 +-- (primary-reverse-path-exist)?
|  |  |                    +--:(path-ref)
|  |  |                    |  +-- primary-path-ref    leafref
|  |  |                    +--:(path-request-ref)
|  |  |                       +-- path-request-ref    leafref
|  |  +--:(value)
|  |     +-- tunnel-name?                   string
```

```
|  |     +-- path-name?                      string
|  |     +-- (path-role)?
|  |     |  +--:(primary-path)
|  |     |  +--:(secondary-path)
|  |     |  |  +-- secondary-path!
|  |     |  |     +-- primary-path-name?   string
|  |     |  +--:(primary-reverse-path)
|  |     |  |  +-- primary-reverse-path!
|  |     |  |     +-- primary-path-name?   string
|  |     |  +--:(secondary-reverse-path)
|  |     |     +-- secondary-reverse-path!
|  |     |        +-- primary-path-name?          string
|  |     |        +-- primary-reverse-path-name?   string
|  |     +-- k-requested-paths?              uint8
|  |     +-- encoding?                       identityref
|  |     +-- switching-type?                 identityref
|  |     +-- source?                         te-types:te-node-id
|  |     +-- destination?                    te-types:te-node-id
|  |     +-- src-tunnel-tp-id?               binary
|  |     +-- dst-tunnel-tp-id?               binary
|  |     +-- bidirectional?                  boolean
|  |     +-- te-topology-identifier
|  |        +-- provider-id?   te-global-id
|  |        +-- client-id?     te-global-id
|  |        +-- topology-id?   te-topology-id
|  +-- association-objects
|  |  +-- association-object* [association-key]
|  |  |  +-- association-key    string
|  |  |  +-- type?              identityref
|  |  |  +-- id?                uint16
|  |  |  +-- source
|  |  |     +-- id?     te-gen-node-id
|  |  |     +-- type?   enumeration
|  |  +-- association-object-extended* [association-key]
|  |     +-- association-key    string
|  |     +-- type?              identityref
|  |     +-- id?                uint16
|  |     +-- source
|  |     |  +-- id?     te-gen-node-id
```

```
       |  |     |  +-- type?   enumeration
       |  |     +-- global-source?     uint32
       |  |     +-- extended-id?       yang:hex-string
       |  +-- optimizations
       |  |  +-- (algorithm)?
       |  |     +--:(metric) {path-optimization-metric}?
       |  |     |  +-- optimization-metric* [metric-type]
       |  |     |  |  +-- metric-type                      identityref
       |  |     |  |  +-- weight?                          uint8
       |  |     |  |  +-- explicit-route-exclude-objects
       |  |     |  |  |  +-- route-object-exclude-object* [index]
       |  |     |  |  |     +-- index                       uint32
       |  |     |  |  |     +-- (type)?
       |  |     |  |  |        +--:(numbered-node-hop)
       |  |     |  |  |        |  +-- numbered-node-hop
       |  |     |  |  |        |     +-- node-id     te-node-id
       |  |     |  |  |        |     +-- hop-type?   te-hop-type
       |  |     |  |  |        +--:(numbered-link-hop)
       |  |     |  |  |        |  +-- numbered-link-hop
       |  |     |  |  |        |     +-- link-tp-id    te-tp-id
       |  |     |  |  |        |     +-- hop-type?     te-hop-type
       |  |     |  |  |        |     +-- direction?    te-link-direction
       |  |     |  |  |        +--:(unnumbered-link-hop)
       |  |     |  |  |        |  +-- unnumbered-link-hop
       |  |     |  |  |        |     +-- link-tp-id    te-tp-id
       |  |     |  |  |        |     +-- node-id       te-node-id
       |  |     |  |  |        |     +-- hop-type?     te-hop-type
       |  |     |  |  |        |     +-- direction?    te-link-direction
       |  |     |  |  |        +--:(as-number)
       |  |     |  |  |        |  +-- as-number-hop
       |  |     |  |  |        |     +-- as-number    inet:as-number
       |  |     |  |  |        |     +-- hop-type?    te-hop-type
       |  |     |  |  |        +--:(label)
       |  |     |  |  |        |  +-- label-hop
       |  |     |  |  |        |     +-- te-label
       |  |     |  |  |        |        +-- (technology)?
       |  |     |  |  |        |        |  +--:(generic)
       |  |     |  |  |        |        |     +-- generic?
```

```
      |  |     |  |  |        |        |             rt-
   types:generalized-label
      |  |     |  |  |        |        +-- direction?
      |  |     |  |  |        |                te-label-direction
      |  |     |  |  |        +--:(srlg)
      |  |     |  |  |           +-- srlg
      |  |     |  |  |              +-- srlg?   uint32
      |  |     |  |  +-- explicit-route-include-objects
      |  |     |  |     +-- route-object-include-object* [index]
      |  |     |  |        +-- index                        uint32
      |  |     |  |        +-- (type)?
      |  |     |  |           +--:(numbered-node-hop)
      |  |     |  |           |  +-- numbered-node-hop
      |  |     |  |           |     +-- node-id     te-node-id
      |  |     |  |           |     +-- hop-type?   te-hop-type
      |  |     |  |           +--:(numbered-link-hop)
      |  |     |  |           |  +-- numbered-link-hop
      |  |     |  |           |     +-- link-tp-id    te-tp-id
      |  |     |  |           |     +-- hop-type?     te-hop-type
      |  |     |  |           |     +-- direction?    te-link-direction
      |  |     |  |           +--:(unnumbered-link-hop)
      |  |     |  |           |  +-- unnumbered-link-hop
      |  |     |  |           |     +-- link-tp-id    te-tp-id
      |  |     |  |           |     +-- node-id       te-node-id
      |  |     |  |           |     +-- hop-type?     te-hop-type
      |  |     |  |           |     +-- direction?    te-link-direction
      |  |     |  |           +--:(as-number)
      |  |     |  |           |  +-- as-number-hop
      |  |     |  |           |     +-- as-number    inet:as-number
      |  |     |  |           |     +-- hop-type?    te-hop-type
      |  |     |  |           +--:(label)
      |  |     |  |              +-- label-hop
      |  |     |  |                 +-- te-label
      |  |     |  |                    +-- (technology)?
      |  |     |  |                    |  +--:(generic)
      |  |     |  |                    |     +-- generic?
      |  |     |  |                    |             rt-
   types:generalized-label
      |  |     |  |                    +-- direction?
```

```
|  |     |  |                          te-label-direction
|  |     |  +-- tiebreakers
|  |     |     +-- tiebreaker* [tiebreaker-type]
|  |     |        +-- tiebreaker-type    identityref
|  |     +--:(objective-function)
|  |              {path-optimization-objective-function}?
|  |        +-- objective-function
|  |           +-- objective-function-type?   identityref
|  +-- named-path-constraint?               leafref
|  |       {te-types:named-path-constraints}?
|  +-- te-bandwidth
|  |  +-- (technology)?
|  |     +--:(generic)
|  |        +-- generic?   te-bandwidth
|  +-- link-protection?                     identityref
|  +-- setup-priority?                      uint8
|  +-- hold-priority?                       uint8
|  +-- signaling-type?                      identityref
|  +-- path-metric-bounds
|  |  +-- path-metric-bound* [metric-type]
|  |     +-- metric-type    identityref
|  |     +-- upper-bound?   uint64
|  +-- path-affinities-values
|  |  +-- path-affinities-value* [usage]
|  |     +-- usage    identityref
|  |     +-- value?   admin-groups
|  +-- path-affinity-names
|  |  +-- path-affinity-name* [usage]
|  |     +-- usage            identityref
|  |     +-- affinity-name* [name]
|  |        +-- name    string
|  +-- path-srlgs-lists
|  |  +-- path-srlgs-list* [usage]
|  |     +-- usage     identityref
|  |     +-- values*   srlg
|  +-- path-srlgs-names
|  |  +-- path-srlgs-name* [usage]
|  |     +-- usage    identityref
|  |     +-- names*   string
```

```
|  +-- disjointness?                          te-path-disjointness
|  +-- explicit-route-objects-always
|  |  +-- route-object-exclude-always* [index]
|  |  |  +-- index                        uint32
|  |  |  +-- (type)?
|  |  |     +--:(numbered-node-hop)
|  |  |     |  +-- numbered-node-hop
|  |  |     |     +-- node-id     te-node-id
|  |  |     |     +-- hop-type?   te-hop-type
|  |  |     +--:(numbered-link-hop)
|  |  |     |  +-- numbered-link-hop
|  |  |     |     +-- link-tp-id    te-tp-id
|  |  |     |     +-- hop-type?     te-hop-type
|  |  |     |     +-- direction?    te-link-direction
|  |  |     +--:(unnumbered-link-hop)
|  |  |     |  +-- unnumbered-link-hop
|  |  |     |     +-- link-tp-id    te-tp-id
|  |  |     |     +-- node-id       te-node-id
|  |  |     |     +-- hop-type?     te-hop-type
|  |  |     |     +-- direction?    te-link-direction
|  |  |     +--:(as-number)
|  |  |     |  +-- as-number-hop
|  |  |     |     +-- as-number    inet:as-number
|  |  |     |     +-- hop-type?    te-hop-type
|  |  |     +--:(label)
|  |  |        +-- label-hop
|  |  |           +-- te-label
|  |  |              +-- (technology)?
|  |  |              |  +--:(generic)
|  |  |              |     +-- generic?
|  |  |              |             rt-types:generalized-label
|  |  |              +-- direction?       te-label-direction
|  |  +-- route-object-include-exclude* [index]
|  |     +-- explicit-route-usage?        identityref
|  |     +-- index                        uint32
|  |     +-- (type)?
|  |        +--:(numbered-node-hop)
|  |        |  +-- numbered-node-hop
|  |        |     +-- node-id     te-node-id
```

```
       |  |        |     +-- hop-type?   te-hop-type
       |  |        +--:(numbered-link-hop)
       |  |        |  +-- numbered-link-hop
       |  |        |     +-- link-tp-id    te-tp-id
       |  |        |     +-- hop-type?     te-hop-type
       |  |        |     +-- direction?    te-link-direction
       |  |        +--:(unnumbered-link-hop)
       |  |        |  +-- unnumbered-link-hop
       |  |        |     +-- link-tp-id    te-tp-id
       |  |        |     +-- node-id       te-node-id
       |  |        |     +-- hop-type?     te-hop-type
       |  |        |     +-- direction?    te-link-direction
       |  |        +--:(as-number)
       |  |        |  +-- as-number-hop
       |  |        |     +-- as-number    inet:as-number
       |  |        |     +-- hop-type?    te-hop-type
       |  |        +--:(label)
       |  |        |  +-- label-hop
       |  |        |     +-- te-label
       |  |        |        +-- (technology)?
       |  |        |        |  +--:(generic)
       |  |        |        |     +-- generic?
       |  |        |        |             rt-types:generalized-label
       |  |        |        +-- direction?       te-label-direction
       |  |        +--:(srlg)
       |  |           +-- srlg
       |  |              +-- srlg?   uint32
       |  +-- path-in-segment!
       |  |  +-- label-restrictions
       |  |     +-- label-restriction* [index]
       |  |        +-- restriction?    enumeration
       |  |        +-- index           uint32
       |  |        +-- label-start
       |  |        |  +-- te-label
       |  |        |     +-- (technology)?
       |  |        |     |  +--:(generic)
       |  |        |     |     +-- generic?   rt-types:generalized-label
       |  |        |     +-- direction?       te-label-direction
       |  |        +-- label-end
```

```
|  |        |  +-- te-label
|  |        |     +-- (technology)?
|  |        |     |  +--:(generic)
|  |        |     |     +-- generic?   rt-types:generalized-label
|  |        |     +-- direction?       te-label-direction
|  |        +-- label-step
|  |        |  +-- (technology)?
|  |        |     +--:(generic)
|  |        |        +-- generic?   int32
|  |        +-- range-bitmap?   yang:hex-string
|  +-- path-out-segment!
|  |  +-- label-restrictions
|  |     +-- label-restriction* [index]
|  |        +-- restriction?    enumeration
|  |        +-- index           uint32
|  |        +-- label-start
|  |        |  +-- te-label
|  |        |     +-- (technology)?
|  |        |     |  +--:(generic)
|  |        |     |     +-- generic?   rt-types:generalized-label
|  |        |     +-- direction?       te-label-direction
|  |        +-- label-end
|  |        |  +-- te-label
|  |        |     +-- (technology)?
|  |        |     |  +--:(generic)
|  |        |     |     +-- generic?   rt-types:generalized-label
|  |        |     +-- direction?       te-label-direction
|  |        +-- label-step
|  |        |  +-- (technology)?
|  |        |     +--:(generic)
|  |        |        +-- generic?   int32
|  |        +-- range-bitmap?   yang:hex-string
|  +-- requested-metrics* [metric-type]
|  |  +-- metric-type    identityref
|  +-- return-srlgs?                        boolean
|  +-- return-affinities?                   boolean
|  +-- requested-state!
|     +-- timer?           uint16
|     +-- transaction-id?   string
```

```
      +-- tunnel-attributes* [tunnel-name]
      |  +-- tunnel-name               string
      |  +-- encoding?                 identityref
      |  +-- switching-type?           identityref
      |  +-- source?                   te-types:te-node-id
      |  +-- destination?              te-types:te-node-id
      |  +-- src-tunnel-tp-id?         binary
      |  +-- dst-tunnel-tp-id?         binary
      |  +-- bidirectional?            boolean
      |  +-- association-objects
      |  |  +-- association-object* [association-key]
      |  |  |  +-- association-key    string
      |  |  |  +-- type?              identityref
      |  |  |  +-- id?                uint16
      |  |  |  +-- source
      |  |  |     +-- id?     te-gen-node-id
      |  |  |     +-- type?   enumeration
      |  |  +-- association-object-extended* [association-key]
      |  |     +-- association-key    string
      |  |     +-- type?              identityref
      |  |     +-- id?                uint16
      |  |     +-- source
      |  |     |  +-- id?     te-gen-node-id
      |  |     |  +-- type?   enumeration
      |  |     +-- global-source?     uint32
      |  |     +-- extended-id?       yang:hex-string
      |  +-- protection-type?          identityref
      |  +-- restoration-type?         identityref
      |  +-- restoration-scheme?       identityref
      |  +-- te-topology-identifier
      |  |  +-- provider-id?   te-global-id
      |  |  +-- client-id?     te-global-id
      |  |  +-- topology-id?   te-topology-id
      |  +-- te-bandwidth
      |  |  +-- (technology)?
      |  |     +--:(generic)
      |  |        +-- generic?   te-bandwidth
      |  +-- link-protection?          identityref
      |  +-- setup-priority?           uint8
```

```
|  +-- hold-priority?           uint8
|  +-- signaling-type?          identityref
|  +-- hierarchy
|     +-- dependency-tunnels
|     |  +-- dependency-tunnel* [name]
|     |  |  +-- name              -> /te:te/tunnels/tunnel/name
|     |  |  +-- encoding?         identityref
|     |  |  +-- switching-type?   identityref
|     |  +-- dependency-tunnel-attributes* [name]
|     |     +-- name              leafref
|     |     +-- encoding?         identityref
|     |     +-- switching-type?   identityref
|     +-- hierarchical-link
|        +-- local-te-node-id?         te-types:te-node-id
|        +-- local-te-link-tp-id?      te-types:te-tp-id
|        +-- remote-te-node-id?        te-types:te-node-id
|        +-- te-topology-identifier
|           +-- provider-id?   te-global-id
|           +-- client-id?     te-global-id
|           +-- topology-id?   te-topology-id
+-- synchronization* []
   +-- svec
   |  +-- relaxable?           boolean
   |  +-- disjointness?        te-path-disjointness
   |  +-- request-id-number*   uint32
   +-- svec-constraints
   |  +-- path-metric-bound* [metric-type]
   |     +-- metric-type    identityref
   |     +-- upper-bound?   uint64
   +-- path-srlgs-lists
   |  +-- path-srlgs-list* [usage]
   |     +-- usage     identityref
   |     +-- values*   srlg
   +-- path-srlgs-names
   |  +-- path-srlgs-name* [usage]
   |     +-- usage    identityref
   |     +-- names*   string
   +-- exclude-objects
   |  +-- excludes* []
```

```
          |     +-- (type)?
          |        +--:(numbered-node-hop)
          |        |  +-- numbered-node-hop
          |        |     +-- node-id     te-node-id
          |        |     +-- hop-type?   te-hop-type
          |        +--:(numbered-link-hop)
          |        |  +-- numbered-link-hop
          |        |     +-- link-tp-id    te-tp-id
          |        |     +-- hop-type?     te-hop-type
          |        |     +-- direction?    te-link-direction
          |        +--:(unnumbered-link-hop)
          |        |  +-- unnumbered-link-hop
          |        |     +-- link-tp-id    te-tp-id
          |        |     +-- node-id       te-node-id
          |        |     +-- hop-type?     te-hop-type
          |        |     +-- direction?    te-link-direction
          |        +--:(as-number)
          |        |  +-- as-number-hop
          |        |     +-- as-number    inet:as-number
          |        |     +-- hop-type?    te-hop-type
          |        +--:(label)
          |           +-- label-hop
          |              +-- te-label
          |                 +-- (technology)?
          |                 |  +--:(generic)
          |                 |     +-- generic?
          |                 |             rt-types:generalized-label
          |                 +-- direction?       te-label-direction
          +-- optimizations
             +-- (algorithm)?
                +--:(metric) {te-types:path-optimization-metric}?
                |  +-- optimization-metric* [metric-type]
                |     +-- metric-type    identityref
                |     +-- weight?        uint8
                +--:(objective-function)
                          {te-types:path-optimization-objective-
   function}?
                   +-- objective-function
                      +-- objective-function-type?   identityref
```

```
augment /te:tunnels-path-compute/te:output/te:path-compute-result:
  +--ro response* [response-id]
     +--ro response-id                          uint32
     +--ro computed-paths-properties
     |  +--ro computed-path-properties* [k-index]
     |     +--ro k-index            uint8
     |     +--ro path-properties
     |        +--ro path-metric* [metric-type]
     |        |  +--ro metric-type           identityref
     |        |  +--ro accumulative-value?   uint64
     |        +--ro path-affinities-values
     |        |  +--ro path-affinities-value* [usage]
     |        |     +--ro usage    identityref
     |        |     +--ro value?   admin-groups
     |        +--ro path-affinity-names
     |        |  +--ro path-affinity-name* [usage]
     |        |     +--ro usage            identityref
     |        |     +--ro affinity-name* [name]
     |        |        +--ro name    string
     |        +--ro path-srlgs-lists
     |        |  +--ro path-srlgs-list* [usage]
     |        |     +--ro usage     identityref
     |        |     +--ro values*   srlg
     |        +--ro path-srlgs-names
     |        |  +--ro path-srlgs-name* [usage]
     |        |     +--ro usage    identityref
     |        |     +--ro names*   string
     |        +--ro path-route-objects
     |        |  +--ro path-route-object* [index]
     |        |     +--ro index                        uint32
     |        |     +--ro (type)?
     |        |        +--:(numbered-node-hop)
     |        |        |  +--ro numbered-node-hop
     |        |        |     +--ro node-id     te-node-id
     |        |        |     +--ro hop-type?   te-hop-type
     |        |        +--:(numbered-link-hop)
     |        |        |  +--ro numbered-link-hop
     |        |        |     +--ro link-tp-id    te-tp-id
     |        |        |     +--ro hop-type?     te-hop-type
```

```
          |        |        |     +--ro direction?    te-link-direction
          |        |        +--:(unnumbered-link-hop)
          |        |        |  +--ro unnumbered-link-hop
          |        |        |     +--ro link-tp-id    te-tp-id
          |        |        |     +--ro node-id       te-node-id
          |        |        |     +--ro hop-type?     te-hop-type
          |        |        |     +--ro direction?    te-link-direction
          |        |        +--:(as-number)
          |        |        |  +--ro as-number-hop
          |        |        |     +--ro as-number    inet:as-number
          |        |        |     +--ro hop-type?    te-hop-type
          |        |        +--:(label)
          |        |           +--ro label-hop
          |        |              +--ro te-label
          |        |                 +--ro (technology)?
          |        |                 |  +--:(generic)
          |        |                 |     +--ro generic?
          |        |                 |             rt-types:generalized-
   label
          |        |                 +--ro direction?
          |        |                         te-label-direction
          |        +--ro te-bandwidth
          |        |  +--ro (technology)?
          |        |     +--:(generic)
          |        |        +--ro generic?   te-bandwidth
          |        +--ro disjointness-type?
          |                 te-types:te-path-disjointness
          +--ro computed-path-error-infos
          |  +--ro computed-path-error-info* []
          |     +--ro error-description?   string
          |     +--ro error-timestamp?     yang:date-and-time
          |     +--ro error-reason?        identityref
          +--ro tunnel-ref?                         te:tunnel-ref
          +--ro (path-role)?
             +--:(primary)
             |  +--ro primary-path-ref?             leafref
             +--:(primary-reverse)
             |  +--ro primary-reverse-path-ref?     leafref
             +--:(secondary)
```

```
            |  +--ro secondary-path-ref?          leafref
            +--:(secondary-reverse)
               +--ro secondary-reverse-path-ref?   leafref
    augment /te:tunnels-actions/te:input/te:tunnel-info/te:filter-type:
      +--:(path-compute-transactions)
         +-- path-compute-transaction-id*   string
    augment /te:tunnels-actions/te:output:
      +--ro path-computed-delete-result
         +--ro path-compute-transaction-id*   string
```

Figure 11 - TE path computation tree diagram

6.2. YANG module

```
<CODE BEGINS>file "ietf-te-path-computation@2021-07-12.yang"
module ietf-te-path-computation {
  yang-version 1.1;
  namespace "urn:ietf:params:xml:ns:yang:ietf-te-path-computation";
  prefix te-pc;

  import ietf-te {
    prefix te;
    reference
      "RFCYYYY: A YANG Data Model for Traffic Engineering Tunnels
       and Interfaces";
  }

  /* Note: The RFC Editor will replace YYYY with the number assigned
     to the RFC once draft-ietf-teas-yang-te becomes an RFC.*/

  import ietf-te-types {
    prefix te-types;
    reference
      "RFC8776: Common YANG Data Types for Traffic Engineering.";
  }

  organization
    "Traffic Engineering Architecture and Signaling (TEAS)
     Working Group";
  contact
```

```
       "WG Web:   <http://tools.ietf.org/wg/teas/>
        WG List:  <mailto:teas@ietf.org>

        Editor:   Italo Busi
                  <mailto:italo.busi@huawei.com>

        Editor:   Sergio Belotti
                  <mailto:sergio.belotti@nokia.com>

        Editor:   Victor Lopez
                  <mailto: victor.lopez@nokia.com>

        Editor:   Oscar Gonzalez de Dios
                  <mailto:oscar.gonzalezdedios@telefonica.com>

        Editor:   Anurag Sharma
                  <mailto:ansha@google.com>

        Editor:   Yan Shi
                  <mailto:shiyan49@chinaunicom.cn>

        Editor:   Ricard Vilalta
                  <mailto:ricard.vilalta@cttc.es>

        Editor:   Karthik Sethuraman
                  <mailto:karthik.sethuraman@necam.com>

        Editor:   Michael Scharf
                  <mailto:michael.scharf@gmail.com>

        Editor:   Daniele Ceccarelli
                  <mailto:daniele.ceccarelli@ericsson.com>

       ";
     description
       "This module defines a YANG data model for requesting Traffic
        Engineering (TE) path computation. The YANG model defined in
        this document is based on RPCs augmenting the RPCs defined in
        the generic TE module (ietf-te).
```

```
        The model fully conforms to the
        Network Management Datastore Architecture (NMDA).

        Copyright (c) 2021 IETF Trust and the persons
        identified as authors of the code.  All rights reserved.

        Redistribution and use in source and binary forms, with or
        without modification, is permitted pursuant to, and subject
        to the license terms contained in, the Simplified BSD License
        set forth in Section 4.c of the IETF Trust's Legal Provisions

        Relating to IETF Documents
        (http://trustee.ietf.org/license-info).

        This version of this YANG module is part of RFC XXXX; see
        the RFC itself for full legal notices.";

    // RFC Ed.: replace XXXX with actual RFC number and remove
    // this note
    // replace the revision date with the module publication date
    // the format is (year-month-day)

    revision 2021-07-12 {
      description
        "Initial revision";
      reference
        "RFC XXXX: YANG Data Model for requesting Path Computation";
    }

    // RFC Ed.: replace XXXX with actual RFC number and remove
    // this note

    /*
     * Identities
     */

    identity svec-metric-type {
      description
        "Base identity for SVEC metric type.";
```

```
      reference
        "RFC5541: Encoding of Objective Functions in the Path
         Computation Element Communication Protocol (PCEP).";
    }

    identity svec-metric-cumul-te {
      base svec-metric-type;
      description
        "Cumulative TE cost.";
      reference
        "RFC5541: Encoding of Objective Functions in the Path
         Computation Element Communication Protocol (PCEP).";
    }

    identity svec-metric-cumul-igp {
      base svec-metric-type;
      description
        "Cumulative IGP cost.";
      reference
        "RFC5541: Encoding of Objective Functions in the Path
         Computation Element Communication Protocol (PCEP).";
    }

    identity svec-metric-cumul-hop {
      base svec-metric-type;
      description
        "Cumulative Hop path metric.";
      reference
        "RFC8776: Common YANG Data Types for Traffic Engineering.";
    }

    identity svec-metric-aggregate-bandwidth-consumption {
      base svec-metric-type;
      description
        "Aggregate bandwidth consumption.";
      reference
        "RFC5541: Encoding of Objective Functions in the Path
         Computation Element Communication Protocol (PCEP).";
    }
```

```
    identity svec-metric-load-of-the-most-loaded-link {
      base svec-metric-type;
      description
        "Load of the most loaded link.";
      reference
        "RFC5541: Encoding of Objective Functions in the Path
         Computation Element Communication Protocol (PCEP).";
    }

    identity tunnel-action-path-compute-delete {
      base te:tunnel-actions-type;
      description
        "Action type to delete the transient states
         of computed paths, as described in section 3.3.1 of
         RFC XXXX.";
      reference
        "RFC XXXX: YANG Data Model for requesting Path Computation";
    }

    /*
     * Groupings
     */

    grouping protection-restoration-properties {
      description
        "This grouping defines the restoration and protection types
         for a path in the path computation request.";
      leaf protection-type {
        type identityref {
          base te-types:lsp-protection-type;
        }
        default "te-types:lsp-protection-unprotected";
        description
          "LSP protection type.";
      }
      leaf restoration-type {
        type identityref {
          base te-types:lsp-restoration-type;
```

```
        }
        default "te-types:lsp-restoration-restore-any";
        description
          "LSP restoration type.";
      }
      leaf restoration-scheme {
        type identityref {
          base te-types:restoration-scheme-type;
        }
        default "te-types:restoration-scheme-preconfigured";
        description
          "LSP restoration scheme.";
      }
    } // grouping protection-restoration-properties

    grouping requested-info {
      description
        "This grouping defines the information (e.g., metrics)
         which is requested, in the path computation request, to be
         returned in the path computation response.";
      list requested-metrics {
        key "metric-type";
        description
          "The list of the requested metrics.
           The metrics listed here must be returned in the response.
           Returning other metrics in the response is optional.";
        leaf metric-type {
          type identityref {
            base te-types:path-metric-type;
          }
          description
            "The metric that must be returned in the response";
        }
      }
      leaf return-srlgs {
        type boolean;
        default "false";
        description
          "If true, path srlgs must be returned in the response.
```

```
            If false, returning path srlgs in the response optional.";
        }
        leaf return-affinities {
          type boolean;
          default "false";
          description
            "If true, path affinities must be returned in the response.
             If false, returning path affinities in the response is
             optional.";
        }
      } // grouping requested-info

      grouping requested-state {
        description
          "Configuration for the transient state used
           to report the computed path";
        container requested-state {
          presence
            "Request temporary reporting of the computed path state";
          description
            "Configures attributes for the temporary reporting of the
             computed path state (e.g., expiration timer).";
          leaf timer {
            type uint16;
            units "minutes";
            default "10";
            description
              "The timeout after which the transient state reporting
              the computed path should be removed.";
          }
          leaf transaction-id {
            type string;
            description
              "The transaction-id associated with this path computation
              to be used for fast deletion of the transient states
              associated with multiple path computations.

              This transaction-id can be used to explicitly delete all
              the transient states of all the computed paths associated
```

```
            with the same transaction-id.

            When one path associated with a transaction-id is setup,
            the transient states of all the other computed paths
            with the same transaction-id are automatically removed.

            If not specified, the transient state is removed only
            when the timer expires (when the timer is specified)
            or not created at all (stateless path computation,
            when the timer is not specified).";
        }
      }
    } // grouping requested-state

    grouping reported-state {
      description
        "This grouping defines the information, returned in the path
         computation response, reporting the transient state related
         to the computed path";
      leaf tunnel-ref {
        type te:tunnel-ref;
        description
          "
           Reference to the tunnel that reports the transient state
           of the computed path.

           If no transient state is created, this attribute is
           omitted.
          ";
      }
      choice path-role {
        description
          "The transient state of the computed path can be reported
           as a primary, primary-reverse, secondary or
           a secondary-reverse path of a te-tunnel";
        case primary {
          leaf primary-path-ref {
            type leafref {
              path "/te:te/te:tunnels/"
```

```
                + "te:tunnel[te:name=current()/../tunnel-ref]/"
                + "te:primary-paths/te:primary-path/"
                + "te:name";
            }
            must '../tunnel-ref' {
              description
                "The primary-path name can only be reported
                 if also the tunnel name is reported.";
            }
            description
              "
               Reference to the primary-path that reports
               the transient state of the computed path.

               If no transient state is created,
               this attribute is omitted.
              ";
          }
        } // case primary
        case primary-reverse {
          leaf primary-reverse-path-ref {
            type leafref {
              path "/te:te/te:tunnels/"
                 + "te:tunnel[te:name=current()/../tunnel-ref]/"
                 + "te:primary-paths/te:primary-path/"
                 + "te:name";
            }
            must '../tunnel-ref' {
              description
                "The primary-reverse-path name can only be reported
                 if also the tunnel name is reported.";
            }
            description
              "
               Reference to the primary-reverse-path that reports
               the transient state of the computed path.

               If no transient state is created,
               this attribute is omitted.
```

```
            ";
          }
        } // case primary-reverse
        case secondary {
          leaf secondary-path-ref {
            type leafref {
              path "/te:te/te:tunnels/"
                 + "te:tunnel[te:name=current()/../tunnel-ref]/"
                 + "te:secondary-paths/te:secondary-path/"
                 + "te:name";
            }
            must '../tunnel-ref' {
              description
                "The secondary-path name can only be reported
                 if also the tunnel name is reported.";
            }
            description
              "
               Reference to the secondary-path that reports
               the transient state of the computed path.

               If no transient state is created,
               this attribute is omitted.
              ";
          }
        } // case secondary
        case secondary-reverse {
          leaf secondary-reverse-path-ref {
            type leafref {
              path "/te:te/te:tunnels/"
                 + "te:tunnel[te:name=current()/../tunnel-ref]/"
                 + "te:secondary-reverse-paths/"
                 + "te:secondary-reverse-path/te:name";
            }
            must '../tunnel-ref' {
              description
                "The secondary-reverse-path name can only be reported
                 if also the tunnel name is reported.";
            }
```

```
            description
              "
               Reference to the secondary-reverse-path that reports
               the transient state of the computed path.

               If no transient state is created,
               this attribute is omitted.
              ";
          }
        } // case secondary
      } // choice path
    } // grouping reported-state

    grouping synchronization-constraints {
      description
        "Global constraints applicable to synchronized path
         computation requests.";
      container svec-constraints {
        description
          "global svec constraints";
        list path-metric-bound {
          key "metric-type";
          description
            "list of bound metrics";
          leaf metric-type {
            type identityref {
              base svec-metric-type;
            }
            description
              "SVEC metric type.";
            reference
              "RFC5541: Encoding of Objective Functions in the Path
              Computation Element Communication Protocol (PCEP).";
          }
          leaf upper-bound {
            type uint64;
            description
              "Upper bound on SVEC metric";
          }
```

```
        }
      }
      uses te-types:generic-path-srlgs;
      container exclude-objects {
        description
          "Resources to be excluded";
        list excludes {
          description
            "List of Explicit Route Objects to always exclude
             from synchronized path computation";
          uses te-types:explicit-route-hop;
        }
      }
    } // grouping synchronization-constraints

    grouping synchronization-optimization {
      description
        "Optimizations applicable to synchronized path
         computation requests.";
      container optimizations {
        description
          "The objective function container that includes attributes
           to impose when computing a synchronized set of paths";
        choice algorithm {
          description
            "Optimizations algorithm.";
          case metric {
            if-feature "te-types:path-optimization-metric";
            list optimization-metric {
              key "metric-type";
              description
                "svec path metric type";
              leaf metric-type {
                type identityref {
                  base svec-metric-type;
                }
                description
                  "TE path metric type usable for computing a set of
                  synchronized requests";
```

```
              }
              leaf weight {
                type uint8;
                description
                  "Metric normalization weight";
              }
            }
          }
          case objective-function {
            if-feature
              "te-types:path-optimization-objective-function";
            container objective-function {
              description
                "The objective function container that includes
                 attributes to impose when computing a TE path";
              leaf objective-function-type {
                type identityref {
                  base te-types:objective-function-type;
                }
                default "te-types:of-minimize-cost-path";
                description
                  "Objective function entry";
              }
            }
          }
        }
      }
    } // grouping synchronization-optimization

    grouping synchronization-info {
      description
        "Information for synchronized path computation requests.";
      list synchronization {
        description
          "List of Synchronization VECtors.";
        container svec {
          description
            "Synchronization VECtor";
          leaf relaxable {
```

```
            type boolean;
            default "true";
            description
              "If this leaf is true, path computation process is
               free to ignore svec content.
               Otherwise, it must take into account this svec.";
          }
          uses te-types:generic-path-disjointness;
          leaf-list request-id-number {
            type uint32;
            description
              "This list reports the set of path computation
               requests that must be synchronized.";
          }
        }
        uses synchronization-constraints;
        uses synchronization-optimization;
      }
    } // grouping synchronization-info

    grouping encoding-and-switching-type {
      description
        "Common grouping to define the LSP encoding and
         switching types";
      leaf encoding {
        type identityref {
          base te-types:lsp-encoding-types;
        }
        default "te-types:lsp-encoding-packet";
        description
          "LSP encoding type.";
        reference
          "RFC3945";
      }
      leaf switching-type {
        type identityref {
          base te-types:switching-capabilities;
        }
        default "te-types:switching-psc1";
```

```
        description
          "LSP switching type.";
        reference
          "RFC3945";
      }
    }

    grouping tunnel-common-attributes {
      description
        "Common grouping to define the TE tunnel parameters";
      uses encoding-and-switching-type;
      leaf source {
        type te-types:te-node-id;
        description
          "TE tunnel source node ID.";
      }
      leaf destination {
        type te-types:te-node-id;
        description
          "TE tunnel destination node identifier.";
      }
      leaf src-tunnel-tp-id {
        type binary;
        description
          "TE tunnel source termination point identifier.";
      }
      leaf dst-tunnel-tp-id {
        type binary;
        description
          "TE tunnel destination termination point identifier.";
      }
      leaf bidirectional {
        type boolean;
        default "false";
        description
          "Indicates a bidirectional co-routed LSP.";
      }
    }
```

```
     /*
      * Augment TE RPCs
      */

     augment "/te:tunnels-path-compute/te:input/te:path-compute-info" {
       description
         "Path Computation RPC input";
       list path-request {
         key "request-id";
         description
           "The list of the requested paths to be computed";
         leaf request-id {
           type uint32;
           mandatory true;
           description
             "Each path computation request is uniquely identified
              within the RPC request by the request-id-number.";
         }
         choice tunnel-attributes {
           default "value";
           description
             "Whether the tunnel attributes are specified by value
              within this path computation request or by reference.
              The reference could be either to an existing te-tunnel
              or to an entry in the tunnel-attributes list";
           case reference {
             container tunnel-reference {
               description
                 "Attributes for a requested path that belongs to
                 either an exiting te-tunnel or to an entry in the
                 tunnel-attributes list.";
               choice tunnel-exist {
                 description
                   "Whether the tunnel reference is to an existing
                   te-tunnel or to an entry in the tunnel-attributes
                   list";
                 case tunnel-ref {
                   leaf tunnel-ref {
                     type te:tunnel-ref;
```

```
                mandatory true;
                description
                  "The referenced te-tunnel instance";
              }
            } // case tunnel-ref
            case tunnel-attributes-ref {
              leaf tunnel-attributes-ref {
                type leafref {
                  path "/te:tunnels-path-compute/"
                    + "te:path-compute-info/"
                    + "te-pc:tunnel-attributes/te-pc:tunnel-name";
                }
                mandatory true;
                description
                  "The referenced te-tunnel instance";
              }
            } // case tunnel-attributes-ref
          } // choice tunnel-exist
          leaf path-name {
            type string;
            description
              "TE path name.";
          }
          choice path-role {
            mandatory true;
            description
              "Whether this path is a primary, or a reverse
              primary, or a secondary, or a reverse secondary
              path.";
            case primary-path {
              container primary-path {
                presence "Indicates that the requested path
                          is a primary path";
                description
                  "TE primary path";
                uses te:path-preference;
                uses te:k-requested-paths;
              } // container primary-path
            } // case primary-path
```

```
                case secondary-path {
                  container secondary-path {
                    description
                      "TE secondary path";
                    uses te:path-preference;
                    uses protection-restoration-properties;
                    list primary-path-ref {
                      min-elements 1;
                      description
                        "The list of primary paths that reference
                        this path as a candidate secondary path";
                      choice primary-path-exist {
                        description
                          "Whether the path reference is to an existing
                          te-tunnel path or to another path request";
                        case path-ref {
                          leaf primary-path-ref {
                            type leafref {
                              path "/te:te/te:tunnels/te:tunnel"
                                + "[te:name=current()/../../../"
                                + "tunnel-ref]/te:primary-paths/"
                                + "te:primary-path/te:name";
                            }
                            must '../../../tunnel-ref' {
                              description
                                "The primary-path can be referenced
                                if also the tunnel is referenced.";
                            }
                            mandatory true;
                            description
                              "The referenced primary path";
                          }
                        } // case path-ref
                        case path-request-ref {
                          leaf path-request-ref {
                            type leafref {
                              path "/te:tunnels-path-compute/"
                                + "te:path-compute-info/"
                                + "te-pc:path-request/"
```

```
                        + "te-pc:request-id";
                    }
                    mandatory true;
                    description
                      "The referenced primary path request";
                  }
                } // case path-request-ref
              } // choice primary-path-exist
            } // list primary-path-ref
          } // container secondary-path
        } // case secondary-path
        case primary-reverse-path {
          container primary-reverse-path {
            description
              "TE primary reverse path";
            choice primary-path-exist {
              description
                "Whether the path reference to the primary
                paths for which this path is the reverse-path
                is to an existing te-tunnel path or to
                another path request.";
              case path-ref {
                leaf primary-path-ref {
                  type leafref {
                    path "/te:te/te:tunnels/te:tunnel[te:name"
                      + "=current()/../../tunnel-ref]/"
                      + "te:primary-paths/te:primary-path/"
                      + "te:name";
                  }
                  must '../../tunnel-ref' {
                    description
                      "The primary-path can be referenced
                      if also the tunnel is referenced.";
                  }
                  mandatory true;
                  description
                    "The referenced primary path";
                }
              } // case path-ref
```

```
                    case path-request-ref {
                      leaf path-request-ref {
                        type leafref {
                          path "/te:tunnels-path-compute/"
                            + "te:path-compute-info/"
                            + "te-pc:path-request/"
                            + "te-pc:request-id";
                        }
                        mandatory true;
                        description
                          "The referenced primary path request";
                      }
                    } // case path-request-ref
                  } // choice primary-path-exist
                } // container primary-reverse-path
              } // case primary-reverse-path
              case secondary-reverse-path {
                container secondary-reverse-path {
                  description
                    "TE secondary reverse path";
                  uses te:path-preference;
                  uses protection-restoration-properties;
                  list primary-reverse-path-ref {
                    min-elements 1;
                    description
                      "The list of primary reverse paths that
                      reference this path as a candidate
                      secondary reverse path";
                    choice primary-reverse-path-exist {
                      description
                        "Whether the path reference is to an existing
                        te-tunnel path or to another path request";
                      case path-ref {
                        leaf primary-path-ref {
                          type leafref {
                            path "/te:te/te:tunnels/te:tunnel"
                              + "[te:name=current()/../../../"
                              + "tunnel-ref]/te:primary-paths/"
                              + "te:primary-path/te:name";
```

```
                        }
                        must '../../../tunnel-ref' {
                          description
                            "The primary-path can be referenced
                            if also the tunnel is referenced.";
                        }
                        mandatory true;
                        description
                          "The referenced primary path";
                      }
                    } // case path-ref
                    case path-request-ref {
                      leaf path-request-ref {
                        type leafref {
                          path "/te:tunnels-path-compute/"
                            + "te:path-compute-info/"
                            + "te-pc:path-request/"
                            + "te-pc:request-id";
                        }
                        mandatory true;
                        description
                          "The referenced primary reverse path
                          request";
                      }
                    } // case path-request-ref
                  } // choice primary-reverse-path-exist
                } // list primary-reverse-path-ref
              } // container secondary-reverse-path
            } // case secondary-reverse-path
          } // choice tunnel-path-role
        }
      } // case reference
      case value {
        leaf tunnel-name {
          type string;
          description
            "TE tunnel name.";
        }
        leaf path-name {
```

```
            type string;
            description
              "TE path name.";
          }
          choice path-role {
            when 'not (./source) and not (./destination) and
                  not (./src-tunnel-tp-id) and
                  not (./dst-tunnel-tp-id)' {
              description
                "When the tunnel attributes are specified by value
                within this path computation, it is assumed that the
                requested path will be the only path of a tunnel.

                If the requested path is a transit segment path, it
                could be of any type. Otherwise it could only be a
                primary path.";
            }
            default primary-path;
            description
              "Indicates whether the requested path is a primary
              path, a secondary path, a reverse primary path or a
              reverse secondary path.";
            case primary-path {
              description
                "The requested path is a primary path.";
            }
            container secondary-path {
              presence
                "Indicates that the requested path is a secondary
                path.";
              description
                "The name of the primary path which the requested
                primary reverse path belongs to.";
              leaf primary-path-name {
                type string;
                description
                  "TE primary path name.";
              }
            } // container secondary-path
```

```
            container primary-reverse-path {
              presence
                "Indicates that the requested path is a primary
                reverse path.";
              description
                "The name of the primary path which the requested
                primary reverse path belongs to.";
              leaf primary-path-name {
                type string;
                description
                  "TE primary path name.";
              }
            } // container primary-reverse-path
            container secondary-reverse-path {
              presence
                "Indicates that the requested path is a secondary
                reverse path.";
              description
                "The names of the primary path and of the primary
                reverse path which the requested secondary reverse
                path belongs to.";
              leaf primary-path-name {
                type string;
                description
                  "TE primary path name.";
              }
              leaf primary-reverse-path-name {
                type string;
                description
                  "TE primary reverse path name.";
              }
            } // container primary-reverse-path
          } // choice path-role
          uses te:k-requested-paths;
          uses tunnel-common-attributes;
          uses te-types:te-topology-identifier;
        } // case value
      } // choice tunnel-attributes
      uses te:path-compute-info;
```

```
        uses requested-info;
        uses requested-state;
      }
      list tunnel-attributes {
        key "tunnel-name";
        description
          "Tunnel attributes common to multiple request paths";
        leaf tunnel-name {
          type string;
          description
            "TE tunnel name.";
        }
        uses tunnel-common-attributes;
        uses te:tunnel-associations-properties;
        uses protection-restoration-properties;
        uses te-types:tunnel-constraints;
        uses te:tunnel-hierarchy-properties {
          augment "hierarchy/dependency-tunnels" {
            description
              "Augment with the list of dependency tunnel requests.";
            list dependency-tunnel-attributes {
              key "name";
              description
                "A tunnel request entry that this tunnel request can
                 potentially depend on.";
              leaf name {
                type leafref {
                  path "/te:tunnels-path-compute/"
                     + "te:path-compute-info/te-pc:tunnel-attributes/"
                     + "te-pc:tunnel-name";
                }
                description
                  "Dependency tunnel request name.";
              }
              uses encoding-and-switching-type;
            }
          }
        }
      }
```

```
      uses synchronization-info;
    } // path-compute rpc input

    augment "/te:tunnels-path-compute/te:output/"
          + "te:path-compute-result" {
      description
        "Path Computation RPC output";
      list response {
        key "response-id";
        config false;
        description
          "response";
        leaf response-id {
          type uint32;
          description
            "The response-id has the same value of the
             corresponding request-id.";
        }
        uses te:path-computation-response;
        uses reported-state;
      }
    } // path-compute rpc output

    augment "/te:tunnels-actions/te:input/te:tunnel-info/"
          + "te:filter-type" {
      description
        "Augment Tunnels Action RPC input filter types";
      case path-compute-transactions {
        when "derived-from-or-self(../te:action-info/te:action, "
           + "'tunnel-action-path-compute-delete')";
        description
          "Path Delete Action RPC";
        leaf-list path-compute-transaction-id {
          type string;
          description
            "The list of the transaction-id values of the
             transient states to be deleted";
        }
      }
```

```
    } // path-delete rpc input

    augment "/te:tunnels-actions/te:output" {
      description
        "Augment Tunnels Action RPC output with path delete result";
      container path-computed-delete-result {
        description
          "Path Delete RPC output";
        leaf-list path-compute-transaction-id {
          type string;
          description
            "The list of the transaction-id values of the
             transient states that have been successfully deleted";
        }
      }
    } // path-delete rpc output
  }
  <CODE ENDS>
```

Figure 12 - TE path computation YANG module

7. Security Considerations

This document describes use cases of requesting Path Computation using YANG data models, which could be used at the ABNO Control Interface [RFC7491] and/or between controllers in ACTN [RFC8453]. As such, it does not introduce any new security considerations compared to the ones related to YANG specification, ABNO specification and ACTN Framework defined in [RFC7950], [RFC7491] and [RFC8453].

The YANG module defined in this draft is designed to be accessed via the NETCONF protocol [RFC6241] or RESTCONF protocol [RFC8040]. The lowest NETCONF layer is the secure transport layer, and the mandatory-to-implement secure transport is Secure Shell (SSH) [RFC6242]. The lowest RESTCONF layer is HTTPS, and the mandatory-to-implement secure transport is TLS [RFC8446].

This document also defines common data types using the YANG data modeling language. The definitions themselves have no security impact on the Internet, but the usage of these definitions in concrete YANG modules might have. The security considerations spelled out in the YANG specification [RFC7950] apply for this document as well.

The NETCONF access control model [RFC8341] provides the means to restrict access for particular NETCONF or RESTCONF users to a preconfigured subset of all available NETCONF or RESTCONF protocol operations and content.

Note - The security analysis of each leaf is for further study.

8. IANA Considerations

This document registers the following URIs in the "ns" subregistry within the "IETF XML registry" [RFC3688].

```
URI: urn:ietf:params:xml:ns:yang:ietf-te-path-computation
Registrant Contact:  The IESG.
XML: N/A, the requested URI is an XML namespace.
```

This document registers a YANG module in the "YANG Module Names" registry [RFC7950].

```
name:      ietf-te-path-computation
namespace: urn:ietf:params:xml:ns:yang:ietf-te-path-computation
prefix:    te-pc
reference: this document
```

9. References

9.1. Normative References

[RFC3688] Mealling, M., "The IETF XML Registry", RFC 3688, January 2004.

[RFC3945] Mannie, E., Ed., "Generalized Multi-Protocol Label Switching (GMPLS) Architecture", RFC 3945, DOI 10.17487/RFC3945, October 2004, <https://www.rfc-editor.org/info/rfc3945>.

[RFC5440] Vasseur, JP., Le Roux, JL. et al., "Path Computation Element (PCE) Communication Protocol (PCEP)", RFC 5440, March 2009.

[RFC5441] Vasseur, JP., Ed., Zhang, R., Bitar, N., and JL. Le Roux, "A Backward-Recursive PCE-Based Computation (BRPC) Procedure to Compute Shortest Constrained Inter-Domain Traffic Engineering Label Switched Paths", RFC 5441, DOI 10.17487/RFC5441, April 2009, <https://www.rfc-editor.org/info/rfc5441>.

[RFC5541] Le Roux, JL. et al., "Encoding of Objective Functions in the Path Computation Element Communication Protocol (PCEP)", RFC 5541, June 2009.

[RFC6241] Enns, R., Ed., Bjorklund, M., Ed., Schoenwaelder, J., Ed., and A. Bierman, Ed., "Network Configuration Protocol (NETCONF)", RFC 6241, June 2011.

[RFC6242] Wasserman, M., "Using the NETCONF Protocol over Secure Shell (SSH)", RFC 6242, June 2011.

[RFC6991] Schoenwaelder, J., "Common YANG Data Types", RFC 6991, July 2013.

[RFC8040] Bierman, A., Bjorklund, M., and K. Watsen, "RESTCONF Protocol", RFC 8040, January 2017.

[RFC8341] Bierman, A., and M. Bjorklund, "Network Configuration Access Control Model", RFC 8341, March 2018.

[RFC7926] Farrel, A. et al., "Problem Statement and Architecture for Information Exchange Between Interconnected Traffic Engineered Networks", RFC 7926, July 2016.

[RFC7950] Bjorklund, M., "The YANG 1.1 Data Modeling Language", RFC 7950, August 2016.

[RFC8040] Bierman, A., Bjorklund, M., and K. Watsen, "RESTCONF Protocol", RFC 8040, January 2017.

[RFC8340] Bjorklund, M. and L. Berger, Ed., "YANG Tree Diagrams", BCP 215, RFC 8340, March 2018.

[RFC8446] Rescorla, E., "The Transport Layer Security (TLS) Protocol Version 1.3", RFC 8446, August 2018.

[RFC8776] Saad, T., Gandhi, R., Liu, X., Beeram, V., and I. Bryskin, "Common YANG Data Types for Traffic Engineering", RFC8776, June 2020.

[RFC8795] Liu, X. et al., " Liu, X. et al., "YANG Data Model for Traffic Engineering (TE) Topologies", RFC8795, August 2020.

[TE-TUNNEL] Saad, T. et al., "A YANG Data Model for Traffic Engineering Tunnels and Interfaces", draft-ietf-teas-yang-te, work in progress.

9.2. Informative References

[RFC4655] Farrel, A. et al., "A Path Computation Element (PCE)-Based Architecture", RFC 4655, August 2006.

[RFC6805] King, D., Ed. and A. Farrel, Ed., "The Application of the Path Computation Element Architecture to the Determination of a Sequence of Domains in MPLS and GMPLS", RFC 6805, DOI 10.17487/RFC6805, November 2012, <https://www.rfc-editor.org/info/rfc6805>.

[RFC7139] Zhang, F. et al., "GMPLS Signaling Extensions for Control of Evolving G.709 Optical Transport Networks", RFC 7139, March 2014.

[RFC7446] Lee, Y. et al., "Routing and Wavelength Assignment Information Model for Wavelength Switched Optical Networks", RFC 7446, February 2015.

[RFC7491] Farrel, A., King, D., "A PCE-Based Architecture for Application-Based Network Operations", RFC 7491, March 2015.

[RFC8233] Dhody, D. et al., "Extensions to the Path Computation Element Communication Protocol (PCEP) to Compute Service-Aware Label Switched Paths (LSPs)", RFC 8233, September 2017

[RFC8342] Bjorklund,M. et al. "Network Management Datastore Architecture (NMDA)", RFC 8342, March 2018

[RFC8453] Ceccarelli, D., Lee, Y. et al., "Framework for Abstraction and Control of TE Networks (ACTN)", RFC8453, August 2018.

[RFC8454] Lee, Y. et al., "Information Model for Abstraction and Control of TE Networks (ACTN)", RFC8454, September 2018.

[OTN-TOPO] Zheng, H. et al., "A YANG Data Model for Optical Transport Network Topology", draft-ietf-ccamp-otn-topo-yang, work in progress.

[ITU-T G.709-2016] ITU-T Recommendation G.709 (06/16), "Interface for the optical transport network", June 2016.

Appendix A. Examples of dimensioning the "detailed connectivity matrix"

In the following table, a list of the possible constraints, associated with their potential cardinality, is reported.

The maximum number of potential connections to be computed and reported is, in first approximation, the multiplication of all of them.

Constraint	Cardinality
End points	N(N-1)/2 if connections are bidirectional (OTN and WDM), N(N-1) for unidirectional connections.
Bandwidth	In WDM networks, bandwidth values are expressed in GHz. On fixed-grid WDM networks, the central frequencies are on a 50GHz grid and the channel width of the transmitters are typically 50GHz such that each central frequency can be used, i.e., adjacent channels can be placed next to each other in terms of central frequencies. On flex-grid WDM networks, the central frequencies are on a 6.25GHz grid and the channel width of the transmitters can be multiples of 12.5GHz. For fixed-grid WDM networks typically there is only one possible bandwidth value (i.e., 50GHz) while for flex-grid WDM networks typically there are 4 possible bandwidth values (e.g., 37.5GHz, 50GHz, 62.5GHz, 75GHz). In OTN (ODU) networks, bandwidth values are expressed as pairs of ODU type and, in case of ODUflex, ODU rate in bytes/sec as described in section 5 of [RFC7139]. For "fixed" ODUk types, 6 possible bandwidth values are possible (i.e., ODU0, ODU1, ODU2, ODU2e, ODU3, ODU4). For ODUflex(GFP), up to 80 different bandwidth values can be specified, as defined in Table 7-8 of [ITU-T G.709-2016].

For other ODUflex types, like ODUflex(CBR), the number of possible bandwidth values depends on the rates of the clients that could be mapped over these ODUflex types, as shown in Table 7.2 of [ITU-T G.709-2016], which in theory could be a countinuum of values. However, since different ODUflex bandwidths that use the same number of TSs on each link along the path are equivalent for path computation purposes, up to 120 different bandwidth ranges can be specified.

Ideas to reduce the number of ODUflex bandwidth values in the detailed connectivity matrix, to less than 100, are for further study.

Bandwidth specification for ODUCn is currently for further study but it is expected that other bandwidth values can be specified as integer multiples of 100Gb/s.

In IP we have bandwidth values in bytes/sec. In principle, this is a countinuum of values, but in practice we can identify a set of bandwidth ranges, where any bandwidth value inside the same range produces the same path.
The number of such ranges is the cardinality, which depends on the topology, available bandwidth and status of the network. Simulations (Note: reference paper submitted for publication) show that values for medium size topologies (around 50-150 nodes) are in the range 4-7 (5 on average) for each end points couple.

Metrics IGP, TE and hop number are the basic objective metrics defined so far. There are also the 2 objective functions defined in [RFC5541]: Minimum Load Path (MLP) and Maximum Residual Bandwidth Path (MBP). Assuming that one only metric or objective function can be optimized at once, the total cardinality here is 5.

With [RFC8233], a number of additional metrics are defined, including Path Delay metric, Path Delay Variation metric and Path Loss metric, both for point-to-point and point-to-multipoint paths. This increases the cardinality to 8.

Bounds	Each metric can be associated with a bound in order to find a path having a total value of that metric lower than the given bound. This has a potentially very high cardinality (as any value for the bound is allowed). In practice there is a maximum value of the bound (the one with the maximum value of the associated metric) which results always in the same path, and a range approach like for bandwidth in IP should produce also in this case the cardinality. Assuming to have a cardinality similar to the one of the bandwidth (let say 5 on average) we should have 6 (IGP, TE, hop, path delay, path delay variation and path loss; we don't consider here the two objective functions of [RFC5541] as they are conceived only for optimization)*5 = 30 cardinality.
Technology constraints	For further study
Priority	We have 8 values for set-up priority, which is used in path computation to route a path using free resources and, where no free resources are available, resources used by LSPs having a lower holding priority.
Local prot	It's possible to ask for a local protected service, where all the links used by the path are protected with fast reroute (this is only for IP networks, but line protection schemas are available on the other technologies as well). This adds an alternative path computation, so the cardinality of this constraint is 2.
Administrative Colors	Administrative colors (aka affinities) are typically assigned to links but when topology abstraction is used affinity information can also appear in the detailed connectivity matrix. There are 32 bits available for the affinities. Links can be tagged with any combination of these bits, and path computation can be constrained to include or exclude any or all of them. The relevant cardinality is 3 (include-any, exclude-any, include-all) times 2^32 possible values. However, the number of possible values used in real networks is quite small.

Included Resources

A path computation request can be associated to an ordered set of network resources (links, nodes) to be included along the computed path. This constraint would have a huge cardinality as in principle any combination of network resources is possible. However, as far as the client doesn't know details about the internal topology of the domain, it shouldn't include this type of constraint at all (see more details below).

Excluded Resources

A path computation request can be associated to a set of network resources (links, nodes, SRLGs) to be excluded from the computed path. Like for included resources, this constraint has a potentially very high cardinality, but, once again, it can't be actually used by the client, if it's not aware of the domain topology (see more details below).

As discussed above, the client can specify include or exclude resources depending on the abstract topology information that the underlying controller exposes:

- In case the underlying controller exposes the entire domain as a single abstract TE node with his own external terminations and detailed connectivity matrix (whose size we are estimating), no other topological details are available, therefore the size of the detailed connectivity matrix only depends on the combination of the constraints that the client can use in a path computation request to its underlying controller. These constraints cannot refer to any details of the internal topology of the domain, as those details are not known to the client and so they do not impact size of the detailed connectivity matrix exported.

- Instead in case the underlying controller exposes a topology including more than one abstract TE nodes and TE links, and their attributes (e.g. SRLGs, affinities for the links), the client knows these details and therefore could compute a path across the domain referring to them in the constraints. The detailed connectivity matrixes, whose size need to be estimated here, are the ones relevant to the abstract TE nodes exported to the client. These detailed connectivity matrixes and therefore theirs sizes, while cannot depend on the other abstract TE nodes and TE links, which are external to the given abstract node, could depend to SRLGs (and other attributes, like affinities) which could be present also in the portion of the topology represented by the abstract nodes, and therefore contribute to the size of the related detailed connectivity matrix.

We also don't consider here the possibility to ask for more than one path in diversity or for point-to-multi-point paths, which are for further study.

Considering for example an IP domain without considering SRLG and affinities, we have an estimated number of paths depending on these estimated cardinalities:

Endpoints = N*(N-1), Bandwidth = 5, Metrics = 6, Bounds = 20, Priority = 8, Local prot = 2

The number of paths to be pre-computed by each IP domain is therefore 24960 * N(N-1) where N is the number of domain access points.

This means that with just 4 access points we have nearly 300000 paths to compute, advertise and maintain (if a change happens in the domain, due to a fault, or just the deployment of new traffic, a substantial number of paths need to be recomputed and the relevant changes advertised to the client).

This seems quite challenging. In fact, if we assume a mean length of 1K for the json describing a path (a quite conservative estimate), reporting 300000 paths means transferring and then parsing more than 300 Mbytes for each domain. If we assume that 20% (to be checked) of this paths change when a new deployment of traffic occurs, we have 60 Mbytes of transfer for each domain traversed by a new end-to-end path. If a network has, let say, 20 domains (we want to estimate the load for a non-trivial domain set-up) in the beginning a total initial transfer of 6Gigs is needed, and eventually, assuming 4-5

domains are involved in mean during a path deployment we could have 240-300 Mbytes of changes advertised to the client.

Further bare-bone solutions can be investigated, removing some more options, if this is considered not acceptable; in conclusion, it seems that an approach based only on the information provided by the detailed connectivity matrix is hardly feasible, and could be applicable only to small networks with a limited meshing degree between domains and renouncing to a number of path computation features.

Acknowledgments

The authors would like to thank Igor Bryskin and Xian Zhang for participating in the initial discussions that have triggered this work and providing valuable insights.

The authors would like to thank the authors of the TE tunnel YANG data model [TE-TUNNEL], in particular Igor Bryskin, Vishnu Pavan Beeram, Tarek Saad and Xufeng Liu, for their inputs to the discussions and support in having consistency between the Path Computation and TE tunnel YANG data models.

The authors would like to thank Adrian Farrel, Dhruv Dhody, Igor Bryskin, Julien Meuric and Lou Berger for their valuable input to the discussions that has clarified that the path being set up is not necessarily the same as the path that has been previously computed and, in particular to Dhruv Dhody, for his suggestion to describe the need for a path verification phase to check that the actual path being set up meets the required end-to-end metrics and constraints.

The authors would like to thank Aihua Guo, Lou Berger, Shaolong Gan, Martin Bjorklund and Tom Petch for their useful comments on how to define XPath statements in YANG RPCs.

The authors would like to thank Haomian Zheng, Yanlei Zheng, Tom Petch, Aihua Guo and Martin Bjorklund for their review and valuable comments to this document.

This document was prepared using 2-Word-v2.0.template.dot.

Contributors

Dieter Beller
Nokia
Email: dieter.beller@nokia.com

Gianmarco Bruno
Ericsson
Email: gianmarco.bruno@ericsson.com

Francesco Lazzeri
Ericsson
Email: francesco.lazzeri@ericsson.com

Young Lee
Samsung Electronics
Email: younglee.tx@gmail.com

Carlo Perocchio
Ericsson
Email: carlo.perocchio@ericsson.com

Olivier Dugeon
Orange Labs
Email: olivier.dugeon@orange.com

Julien Meuric
Orange Labs
Email: julien.meuric@orange.com

Authors' Addresses

Italo Busi (Editor)
Huawei
Email: italo.busi@huawei.com

Sergio Belotti (Editor)
Nokia
Email: sergio.belotti@nokia.com

Victor Lopez
Nokia
Email: victor.lopez@nokia.com

Oscar Gonzalez de Dios
Telefonica
Email: oscar.gonzalezdedios@telefonica.com

Anurag Sharma
Google
Email: ansha@google.com

Yan Shi
China Unicom
Email: shiyan49@chinaunicom.cn

Ricard Vilalta
CTTC
Email: ricard.vilalta@cttc.es

Karthik Sethuraman
NEC
Email: karthik.sethuraman@necam.com

Michael Scharf
Nokia
Email: michael.scharf@gmail.com

Daniele Ceccarelli
Ericsson
Email: daniele.ceccarelli@ericsson.com