**5GT MTP PA SERVER**

**Autor: CTTC/CERCA**

**Contact: ricardo.martinez@cttc.es**

Contents

[License 1](#_Toc6921895)

[Usage 1](#_Toc6921896)

[Building 2](#_Toc6921897)

[Executing 2](#_Toc6921898)

[REST API Server 2](#_Toc6921899)

[PA Algorithm 2](#_Toc6921900)

[Example 3](#_Toc6921901)

# License

Copyright 2019 Centre Tecnològic de Telecomunicacions de Catalunya (CTTC/CERCA) www.cttc.es

Licensed under the Apache License, Version 2.0 (the "License");

you may not use this file except in compliance with the License.

You may obtain a copy of the License at

http://www.apache.org/licenses/LICENSE-2.0

Unless required by applicable law or agreed to in writing, software

distributed under the License is distributed on an "AS IS" BASIS,

WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express or implied.

See the License for the specific language governing permissions and

limitations under the License.

# Usage

The mtp\_pa\_server deployment is devised and deployed within the context of the 5G TRANSFORMER WP2 project aiming at providing a REST API server to receive, process and compute networking inter-NfviPop routes. In other words, those logical links selected by the PA mechanism at the SO are expanded through *n* WAN domains via the MTP PA algorithm which is hosted and triggered at the MTP PA server.

The mtp\_pa\_server provides two specific functions:

1. REST API server to process the incoming REST API requests (POST method) sent by the MTP core (acting as REST API client)
2. Triggering and computing (if feasible) and end-to-end networking inter-NfviPop connection (associate to a selected logical link at the 5GT-SO)

NOTE: the description of the REST API is described and detailed within the 5GT D2.3

# Building

The mtp\_pa\_server is writing in C programming language using standard C libraries, glib-2.0 (<https://lazka.github.io/pgi-docs/GLib-2.0/index.html>) and the project cJSON (<https://github.com/DaveGamble/cJSON>)

Copy the source code (C and header files) to your local folder and/or project.

A Makefile is provided to compile the whole mtp\_pa\_server project

Once copied, run ‘make’ command in the directory where the source is to compile and create the binary mtp\_pat\_server

# Executing

Once the binary mtp\_pa\_server is created, it can be triggered via ./mtp\_pa\_server without any argument.

mtp\_pa\_serve.conf file

A conf file is called mtp\_pa\_server.conf and is read once the program is executed. The folder containing such file is /etc/mtp\_pa\_server/. This folder could be modified within the main C file mtp\_pa\_server.c as preferred.

The mtp\_pa\_server\_file contains the IP address of the REST API server and a flag enabling the REST API. Example:

MTP\_PA\_SERVER\_IPAddr 10.1.1.1

REST\_APIN\_ENABLED 1

mtp\_pa\_server.log file

The log file is found at the folder /var/log/mtp\_pa\_server.log. It can be modified as well in the main C file mtp\_pa\_server.c

# REST API Server

The REST API server (implemented at the mtp\_pa\_server.c) allows processing an incoming REST PA request (POST method) with JSON encoding as described in 5GT WP2 D23. The TCP port is 8081. For the sake of completeness, the REST API defines a pair of request (POST) and response (HTTP/1.1) messages. Once the request is processed by the REST API server, the selected PA algorithm (paid) is triggered.

# PA Algorithm

The implemented PA algorithm (paid set to pa1000) exclusively focuses on providing the unidirectional networking interconnection between a pair of NfviPops (source and destination, *src* and *dst*). Specifically, the algorithm should receive in the REST API request (described in 5GT Wp2 D23) as input information:

1. The name of the connection Identifier (*connectionId*) which is used then to associate the output of the PA execution
2. The *srcPE* and *dstPE* nodeIds of the Provider Edge (PE) nodes physically attached to the Gw at the srcNfviPop and dstNfviPop, respectively
3. The connection service constraints: requested bandwidth (*bandwidthConsValue*) and maximum tolerated end-to-end delay (*delayConsValue*)
4. The set of all the interWanLinks (and their attributes in terms of nodeIds endpoints, link Ids, available Bw, cost and delay). This information should be kept at the MTP core element.
5. The abstracted topology of all the WAN domains kept at the MTP. The topology information should provide the nodeIds and links (with their associated attributes) constituting each WAN domain

The PA algorithm implementation is written in the C file mtp\_pa\_server\_pa1000.c. This algorithm executes a K-CSPF algorithm based on the Yen algorithm (see <https://en.wikipedia.org/wiki/Yen%27s_algorithm>). In a nutshell, the algorithm computes up to K shortest paths aiming at accommodating the connectionId between *srcPE* and *dstPE* nodeIds satisfying the imposed constraints (i.e., *bandwidthConsValue* and *delayConsValue*). The selected path is the one with the lowest path cost (adding the linkCost of every traversed link) ensuring the demanded constraints. If two or more paths are feasible, the tie is broken selecting the path with the lowest end-to-end delay.

The output of the PA execution (if succeeds) for a given *connectionId* contains for the selected path:

1. The resulting available bandwidth for the entire computed path (that may traverse a number of WAN domains)
2. An array containing the set of traversed interWanLinks(nodeIds and linkIds) being part of the computed end-to-end path
3. An array per each traversed WAN domain specifying the computed intra-WAN path segments (nodeIds and linkIds)

# Example

The following illustrates an example to execute and validate the functionalities of the implemented mtp\_pa\_server.

To this end, once the mtp\_pa\_server is triggered and running, it waits for incoming REST API requests that are supposed to be sent by the MTP core REST API client. In this example, the MTP core REST API is emulated using curl. An example of a REST API request using curl is:

curl -X POST -H "Expect:" -H "Content-Type: application/json" http://IPAddressRESTAPIServer:8081/compRoute/Connectivity1000 -d '{"paId":1000, "srcPEId":"00.00.00.00.00.1a", "dstPEId":"00.00.00.00.00.1b", "interWanLinks":[{"aWimId":"WIM\_A", "zWimId":"WIM\_C", "aPEId":"00.00.00.00.00.2a", "zPEId":"00.00.00.00.00.3c", "aLinkId":1001, "zLinkId":1002, "netwLinkQoS":{"linkCost":"linkCost", "linkCostValue":10, "linkDelay":"linkDelay", "linkDelayValue":1, "linkAvailBw":"linkAvailBw", "linkAvailBwValue":20}}, {"aWimId":"WIM\_C", "zWimId":"WIM\_B", "aPEId":"00.00.00.00.00.1c", "zPEId":"00.00.00.00.00.2b", "aLinkId":1001, "zLinkId":1002, "netwLinkQoS":{"linkCost":"linkCost", "linkCostValue":10, "linkDelay":"linkDelay", "linkDelayValue":1, "linkAvailBw":"linkAvailBw", "linkAvailBwValue":20}}], "absWanTopo": [{"wimId":"WIM\_A", "nodes":[{"nodeId":"00.00.00.00.00.1a"}, {"nodeId":"00.00.00.00.00.2a"}, {"nodeId":"00.00.00.00.00.3a"}], "edges":[{"aNodeId":"00.00.00.00.00.1a", "zNodeId":"00.00.00.00.00.2a", "aLinkId":1, "zLinkId":2, "netwLinkQoS":{"linkCost":"linkCost", "linkCostValue":10, "linkDelay":"linkDelay", "linkDelayValue":1, "linkAvailBw":"linkAvailBw", "linkAvailBwValue":20}, "networkLayer":"psc"}, {"aNodeId":"00.00.00.00.00.1a", "zNodeId":"00.00.00.00.00.3a", "aLinkId":2, "zLinkId":1, "netwLinkQoS":{"linkCost":"linkCost", "linkCostValue":10, "linkDelay":"linkDelay", "linkDelayValue":1, "linkAvailBw":"linkAvailBw", "linkAvailBwValue":20}, "networkLayer":"psc"}, {"aNodeId":"00.00.00.00.00.3a", "zNodeId":"00.00.00.00.00.2a", "aLinkId":2, "zLinkId":1, "netwLinkQoS":{"linkCost":"linkCost", "linkCostValue":10, "linkDelay":"linkDelay", "linkDelayValue":1, "linkAvailBw":"linkAvailBw", "linkAvailBwValue":20}, "networkLayer":"psc"}]}, {"wimId":"WIM\_C", "nodes":[{"nodeId":"00.00.00.00.00.3c"}, {"nodeId":"00.00.00.00.00.1c"}, {"nodeId":"00.00.00.00.00.2c"}], "edges":[{"aNodeId":"00.00.00.00.00.3c", "zNodeId":"00.00.00.00.00.1c", "aLinkId":1, "zLinkId":2, "netwLinkQoS":{"linkCost":"linkCost", "linkCostValue":10, "linkDelay":"linkDelay", "linkDelayValue":1, "linkAvailBw":"linkAvailBw", "linkAvailBwValue":20}, "networkLayer":"psc"}, {"aNodeId":"00.00.00.00.00.3c", "zNodeId":"00.00.00.00.00.2c", "aLinkId":2, "zLinkId":1, "netwLinkQoS":{"linkCost":"linkCost", "linkCostValue":10, "linkDelay":"linkDelay", "linkDelayValue":1, "linkAvailBw":"linkAvailBw", "linkAvailBwValue":20}, "networkLayer":"psc"}, {"aNodeId":"00.00.00.00.00.2c", "zNodeId":"00.00.00.00.00.1c", "aLinkId":2, "zLinkId":1, "netwLinkQoS":{"linkCost":"linkCost", "linkCostValue":10, "linkDelay":"linkDelay", "linkDelayValue":1, "linkAvailBw":"linkAvailBw", "linkAvailBwValue":20}, "networkLayer":"psc"}]}, {"wimId":"WIM\_B", "nodes":[{"nodeId":"00.00.00.00.00.1b"}, {"nodeId":"00.00.00.00.00.2b"}, {"nodeId":"00.00.00.00.00.3b"}], "edges":[{"aNodeId":"00.00.00.00.00.2b", "zNodeId":"00.00.00.00.00.1b", "aLinkId":1, "zLinkId":2, "netwLinkQoS":{"linkCost":"linkCost", "linkCostValue":10, "linkDelay":"linkDelay", "linkDelayValue":1, "linkAvailBw":"linkAvailBw", "linkAvailBwValue":20}, "networkLayer":"psc"}, {"aNodeId":"00.00.00.00.00.2b", "zNodeId":"00.00.00.00.00.3b", "aLinkId":2, "zLinkId":1, "netwLinkQoS":{"linkCost":"linkCost", "linkCostValue":10, "linkDelay":"linkDelay", "linkDelayValue":1, "linkAvailBw":"linkAvailBw", "linkAvailBwValue":20}, "networkLayer":"psc"}, {"aNodeId":"00.00.00.00.00.3b", "zNodeId":"00.00.00.00.00.1b", "aLinkId":2, "zLinkId":1, "netwLinkQoS":{"linkCost":"linkCost", "linkCostValue":10, "linkDelay":"linkDelay", "linkDelayValue":1, "linkAvailBw":"linkAvailBw", "linkAvailBwValue":20}, "networkLayer":"psc"}]}], "qosCons": {"bandwidthCons":"bandwidthCons", "bandwidthConsValue":10, "delayCons":"delayCons", "delayConsValue":5}}'

The above REST API POST request is demanding to the mtp\_pa\_server (at IPAddressRESTAPIServer)the computation (for the connectionId: *Connectivity1000*) of a path between *srcPE* (00.00.00.00.00.1a) and *dstPE* (00.00.00.00.00.1b) with *bandwidthConsValue* being 10 and *delayCons* set to 5. Additionally, the request carries specific information of the both interWanLink and topologies of the abstracted WAN domains. Such a topological information is depicted in the following picture:



There are three WAN domains being interconnected by two interWanLinks. For each unidirectional link (inter- or intra-WAN link), it is specified required link attributes in terms of linkCost, linkAvailBw and linkDelay. These attributes are used by the K-CSPF (being K set to 5) PA algorithm to derive a feasible end-to-end path. In the considered example, the PA computes a path which is encoded (using JSON) and then passed to the REST API client (i.e., MTP core). The contents of the response representing the computed path are:

