31-03-2013

Open Call Deliverable OCH-DS1.1.1 Dynamic Path Computation Framework Final Report (DynPaC)

Open Call Deliverable OCH-DS1.1.1

Grant Agreement No.:	605243
Activity:	NA1
Task Item:	10
Nature of Deliverable:	R (Report)
Dissemination Level:	PU (Public)
Lead Partner:	EHU
Document Code:	GN3PLUS14-1290-47
Authors:	Jasone Astorga, Víctor Fuentes, Mariví Higuero, Eduardo Jacob, Alaitz Mendiola, Aitor Urtasun

The research leading to these results has received funding from the European Community's Seventh Framework Programme (FP7 2007–2013) under Grant Agreement No. 605243 (GN3plus).

Abstract

This document summarizes the main results and achievements of the DynPaC project. The main objective of the DynPaC project is to allow the deployment of a disruptive point-to-point Ethernet connectivity service, like GÉANT Plus, on top of a SDN/OpenFlow-pure infrastructure. Additionally, DynPaC framework allows maximizing network utilization, by means of a flow disaggregation mechanism which leverages the granularity in flow characterization and management provided by OpenFlow; and provides network resiliency, thanks to the pre-computation and reservation of completely disjoint backup paths.

Table of Contents

Table of Figures

Index of Tables

Executive Summary

The objective of this document is to summarize the major results and achievements of the DynPaC project. The general objective defined for the DynPaC project is to allow the deployment of a disruptive point-to-point Ethernet connectivity service, like GÉANT Plus, on top of a SDN/OpenFlow-pure infrastructure. Additionally, the following functionalities must also be guaranteed while providing these services:

- Resiliency in case of a link failure with quick recovery times.
- Efficient use of the network capacity.
- Reduction of the operational costs of the service management.
- Improvement of the network monitoring by gathering real time information.

In order to achieve the aforementioned objectives, the DynPaC project is focused on the design and implementation of a framework for dynamic path computation, which leverages the flexibility and granularity offered by SDN/OpenFlow technologies. Basically, the approach followed is to try to forward each flow using the shortest path in terms of number of hops between the given source and destination. However, when a new service cannot be provided maintaining the current routes for ongoing or scheduled services, the framework automatically tries to move ongoing flows in order to compute a network configuration which allows providing all the flows, including the newly requested one.

If after evaluating all the possible combinations of routes for all of the network flows, there is still no one that allows providing the newly requested service, the DynPaC framework goes a step further and makes use of the information provided by an external traffic analyser module in order to effectively disaggregate ongoing traffic flows. Thanks to the implemented disaggregation mechanisms it is possible to split a flow in two or more sub flows and to route each of these sub flows independently. The objective of the flow disaggregation functionality is to optimize network utilization, maximizing the options to route the newly requested service through the network.

Additionally, the DynPaC framework includes also scheduling mechanisms, so that services can be requested in advanced specifying a start and end time. As a result, the DynPaC framework maintains independent configuration information for each different combination of flows that can happen during time. These different combinations of flows during time are called network snapshot. Flow reordering and disaggregation functionalities are applied to affected snapshots in each case in an individual manner. Nevertheless, a “smooth snapshot change” functionality has also been implemented, guaranteeing that no unnecessary flow movements are performed between adjacent snapshots.

The first section of the present document provides an introduction to the report. The goal of this section is to clearly state the project objectives.

Section two corresponds to the related work section and its aim is to summarize the work done so far in related research fields, outlining the gap in the current state of the art to be covered by the DynPaC project. This section highlights the novelty of the flow disaggregation mechanism implemented in DynPaC in order to optimize network utilization.

The third section, Functional Overview, details at a high level perspective the main functionalities to be provided by the DynPaC framework, which correspond to the functionalities stated as the project objectives. More specifically, this section describes the following functionalities: service scheduling, resiliency, service disaggregation and monitoring.

The fourth section provides a detailed description of the DynPaC framework. In this section, the DynPaC framework is described in terms of the building blocks it consists of. The relationships or information exchanges between the different building blocks are also detailed.

Section five summarizes the main factors related to the specific GN3plus infrastructure in which DynPaC framework is to be deployed and which directly affect the design of the DynPaC framework. More specifically, this section details the constraints imposed by two GN3plus infrastructures: the Géant OpenFlow Facility (GOFF) and the Géant Testbed Service (GTS).

Section six details the low level design of the different DynPaC modules. For each of the modules that compose the DynPaC framework, its basic operation and implementation details are provided, relying in flow algorithms for the description. Additionally, this section includes the description of some key concepts on which DynPaC is based, such as how a service is defined within the DynPaC context and the definition of the "network snapshot" concept, a term used to distinguish different network service demands during time.

Section seven includes the functional evaluation. This section consists of a detailed description of all the tests that have been carried out in order to assert that the DynPaC framework performs as expected and that the all the functionalities defined as the project objectives have been met.

Finally, section eight includes the concluding remarks and the planned future research actions that will give continuity to the work carried out in the DynPaC project.

As shown along the different sections of this document, the project has achieved the set goals. In short, the DynPaC framework allows providing bidirectional L2 services with a guaranteed bandwidth from a given source to a given destination and during a predefined time frame, on top of SDN/OpenFlow infrastructure. This way operational costs relative to service management can be reduced, as the DynPaC framework automates all the service life cycle in the network and thus, no operators are needed to manually provision the services. Additionally, network utilization is optimized by means of flow reordering and disaggregation mechanisms, network resiliency is improved based on the allocation of completely disjoint backup paths and network monitorization is enhanced by gathering real time information with a flow-level granularity.

1 Introduction

This document describes the work carried out in the Dynamic Path Computation (DynPaC) framework project. The outcome of the project is a software framework designed to provide bandwidth scheduling and resiliency, while it minimizes the network utilization by means of a flow disaggregation mechanism. The DynPaC framework is a flexible and modular software framework with well-defined interfaces between the modules it consists of.

In a nutshell, the purpose of the DynPaC project is to deploy disruptive Ethernet connectivity services, like Géant Plus, on top of a SDN/OpenFlow infrastructure [1]. This novel approach to provide connectivity services would benefit the network and its administrators in numerous ways. First, having a centralized controller aware of the state of the entire network will improve network monitoring and it will make possible to use the network capabilities more efficiently. The resiliency and the fast recovery capabilities will also be enhanced, as it is easier to detect failures on the network and to react upon them. In addition, one of the main benefits of having a SDN infrastructure is that the operational costs of the service management are reduced. This is a consequence of the network programmability capabilities of SDN, which make possible to reduce cost as operations can be automated and less manual operators are needed.

The DynPaC framework has been built extending the Open Daylight Project (ODP) [2] and reusing some of its modules. DynPaC consists of four custom modules and three modules that are part of ODP. The framework has been designed to be modular enough to support the integration of new modules easily. For instance, the Path Computation Element (PCE) that DynPaC includes by default to optimize network utilization based on a flow disaggregation mechanism could be easily changed by another PCE focused on the optimization of another performance objective.

Furthermore, the DynPaC framework can be applied over multiple transport technologies. For the moment, the framework has been developed to operate on top of an OpenFlow infrastructure. However, being based on the ODP, it could easily be adapted to work with other network technologies. The ODP supports a wide range of southbound protocols, that is, the protocols used to communicate the control plane and the forwarding plane. Current versions of ODP already support Path Computation Element Protocol (PCEP), which means that it could be possible to adapt the DynPaC framework to operate on top, for instance, on an MPLS network.

To the best of our knowledge, the DynPaC framework is the first available solution that deals simultaneously with network optimization based on traffic aggregation and disaggregation mechanisms, bandwidth scheduling, resiliency and flow monitoring.

This document is structured as follows. Section 2 presents the related work, analysing topics such as BoD systems and monitoring in OpenFlow. Section 3 provides a functional overview of the DynPaC framework. The architecture of the DynPaC framework and how the modules are integrated with the Open Daylight project is presented in Section 4. Then, Section 5 introduces the constraints imposed by the GN3plus infrastructures. Section 6 describes the framework in detail, providing the low-level details of the four modules that comprise the framework. Section 7 presents the obtained results and the achievements, describing the testbed and the evaluation procedures. Finally, Section 8 summarizes the conclusion and the future work.

2 Related work

As outlined in the previous section, the DynPaC framework is a very ambitious tool that aims to optimize the network utilization while it provides on-demand resilient services with QoS constraints to users. As a consequence, this section briefly overviews the related work con a variety of topics such as novel Bandwidth on Demand (BoD) systems, network resiliency, flow aggregation and disaggregation and traffic monitoring in OpenFlow networks [3].

Currently, several research proposals and products are focused on the on-demand service provisioning to users. In fact, many National Research and Education Networks (NREN) and Internet Service Providers (ISP) have started to provide BoD services in their portfolio, in order to satisfy the increasing demand of this type of services, especially from the scientific community. For instance, Géant provides BoD to its users with AutoBAHN [4], a scalable and distributed tool. With AutoBAHN, users are able to request a connectivity service for a given period of time that satisfies certain QoS constraints using a simple web browser. However, it is a service that is highly dependent of the underlying hardware. Even if it currently supports several technologies such as Ethernet, MPLS or SDH, it cannot be applied on top of an OpenFlow network. In addition, it does not provide any dynamicity in case of failures, as it does not include any resiliency mechanism.

A very novel approach is the one followed by ESNet (Energy Sciences Network), which provides BoD services to their users with the On-Demand Secure Circuits and Advance Reservation System (OSCARS) [5]. One of the most interesting features of OSCARS is that it is based on the PCE-based architecture [6]. In this architecture, the computation of the paths is performed in an external entity, which can considerably improve the performance of the solution. There have been some efforts to adapt the OSCARS system, originally build for MPLS networks, to operate over OpenFlow networks integrating it with the FloodLight [7] controller. This approach has helped overcome one of the major limitations of OSCARS [8], adding topology discovery mechanisms and thus increasing the dynamicity of the system.

However, the solutions that deal with the integration of OSCARS and OpenFlow are based on NOX [9] or FloodLight controllers, that is, OpenFlow 1.0 controllers. This imposes great challenges when it comes to QoS enforcement. In the version 1.0 of the OpenFlow protocol, there was a very limited support of QoS features. In fact, the protocol allows assigning flows to queues but it does not allow to create the queues on the underlying devices. To be able to create queues in the devices, another protocol is required, such as OF-CONFIG [10] or OVSDB [11], which are not supported by the previously mentioned controllers. Some of the most novel SDN controllers, such as the Open Daylight Project or Cisco ONE [12] overcome this limitations, by integrating several southbound interfaces, including OVSDB and OpenFlow 1.3, with enhanced support for QoS through the metering messages. In fact, aware of the benefits that it provides, the solution proposed by the OLiMPS project has started to upgrade their solution to be based on Open Daylight, while Cisco is already providing BoD in MPLS networks using their Cisco WAN automation engine based on Cisco ONE [13].

As mentioned before, DynPaC is not just a BoD tool, it also includes resiliency mechanisms to guarantee the SLA even in case of failures. When talking about resiliency, it must be taken into account that it is not a trivial problem. Reacting upon changing conditions involves the failure detection, the failure notification and the reaction against it [14]. This highly increases the time needed to react against failures, which must be keep below the 50 ms to meet the carrier grade requirements [15]. The OpenFlow protocol in its latest versions

[16] does include fast failover mechanisms. By using the group table, it is possible to specify multiple actions to be executed in a specific order, that is, it is possible to setup a backup action. This solution makes possible to reduce considerably the time needed to react upon the failure, but it requires to compute a backup path and reserve some resources to it. There are also other methods that are able to restore the disrupted service both proactively and dynamically. These approaches usually involve running an algorithm when the failure occurs, which usually takes a lot of time and involves a lot of computing. The approach followed by the DynPaC framework consists on the pre-computation of the primary path and its associated totally disjoint secondary path and the reservation of resources for both of them.

One of the main pillars of the DynPaC framework is that it supports traffic disaggregation to optimize network utilization. Minimizing network congestion is a classic traffic engineering problem that is usually approached following the minimum cost Multi-Commodity Flow problem. Many of the solutions trying to solve this problem are based on the traffic splitting [17, 18]. Nevertheless, only a few of these approaches are really deployed at production level, being one of these few successful examples the B4 private WAN of Google, which uses multipath forwarding to optimize the network utilization [19]. With the appearance of the OpenFlow protocol and the high granularity that it provides, these proposals could be more easily deployed in real networks. Actually, there are commercial products such as Secure Application Enablement from Palo Alto Networks [20] able to provide information about the traffic in the network by aggregating and disaggregating flows. Traffic aggregation and disaggregation has been used in OpenFlow networks for a variety of purposes. For instance, Kim et al. [21] propose a traffic aggregation and disaggregation mechanism to optimize the flow entries of a flow table, which traditionally has been a very limited resource. Meanwhile, Kudou et al. [22] proposed a solution based on different flow granularities to route the traffic which are applied depending on the network conditions. However, there are no known solutions that use the traffic aggregation and disaggregation to optimize the network utilization. In such a scenario, the DynPaC framework is to the best of our knowledge the first available solution that uses flow disaggregation to optimize that performance objective.

Finally, the framework also includes a monitoring tool in order to guarantee that the SLAs are being fulfilled. Network monitoring in OpenFlow networks is not new, they are a lot of proposals and products available. To cite a few, FlowSense [23] allows to gather statistics from the flows in an OpenFlow network, but it does not provide more advance information such as the occupied bandwidth or the latency between two nodes. A very interesting monitoring tool is the one proposed by Argyropoulos et al. [24], called PaFloMon, which leverages passive network monitoring protocols such as sFlow, SNMP or Netflow. Even more, the ODP also includes some monitoring capabilities that allow to keep track of the flows installed in the network. However, none of the solutions are meant to obtain the real bandwidth occupied by a flow neither to react upon the value of this parameter. As a consequence, the DynPaC framework has its custom monitoring module able to measure the occupied bandwidth of the flows.

3 Functional Overview

3.1 Service Management

DynPaC framework allows increasing the maximum number of services that can coexist, guaranteeing for all of them the requested bandwidth and a completely disjoint backup path in case of link failure.

3.1.1 Service Request / Delete

DynPaC framework allows a user to request services from a web user interface based on the standard ODL web template. However, a REST API can also be used to reserve services. Both mechanisms require authentication.

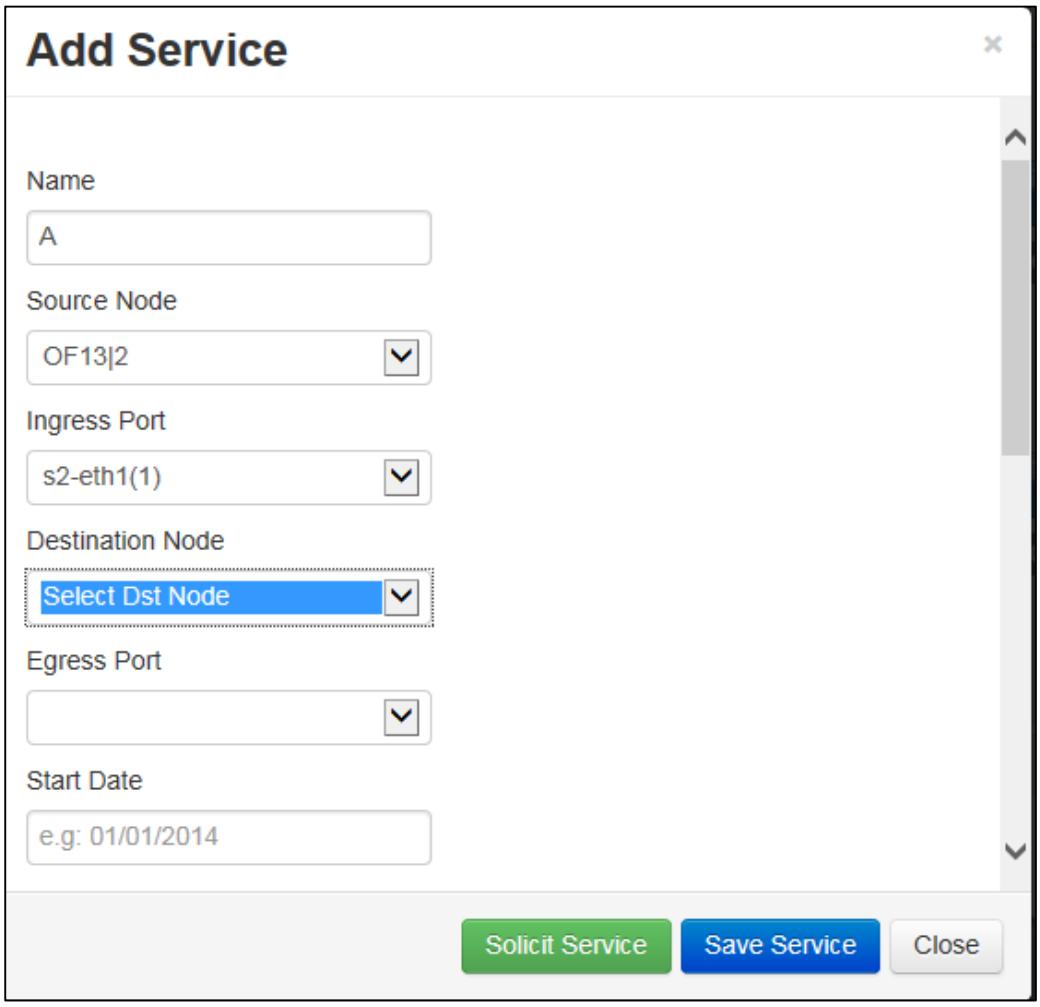

Figure 1: Service reservation form

In order to request a service, some parameters are mandatory, such as start and end time, name, bandwidth required and ingress and egress node and ports. If no gold feature is selected, service is marked as regular. MAC addresses can also be set for the service; however, they are not mandatory. Not using MAC addresses would decrease drastically the number of concurrent services running, as the port cannot be shared among different services. Figure 1 shows the web interfaces developed to allow service request.

It must also be noted that that the paths used by the DynPaC framework are not exposed to the experimenter / user. Only the STATUS is available from the user side. Figure 2 shows the information provided to the experimenter / user regarding the service status.

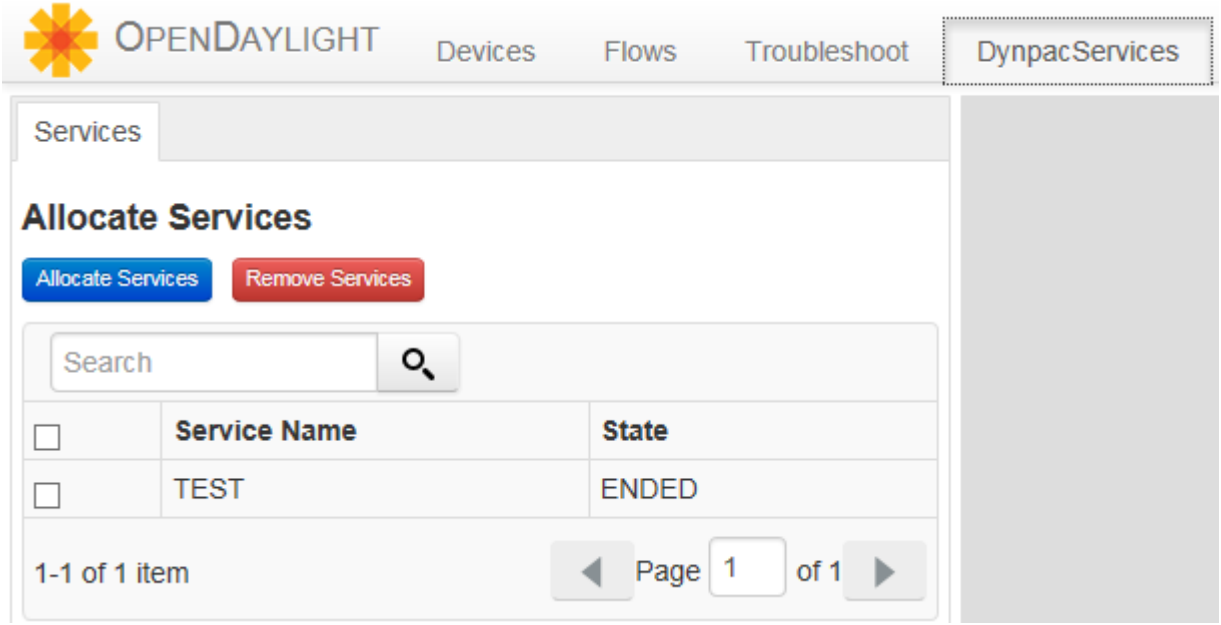

Figure 2: Service expired shown on DynPaC GUI

As an example of the process that takes place when a new service is requested, Figure 3 represents the output of the processing when a regular service A with 8 GB of bandwidth is requested from node 1 to node 4. As the network is empty, even being a regular service enough network resources are available to guarantee a backup path and, therefore, the network configurations that include a backup path are chosen as first alternatives.

```
Requesting Service to DSM...
New Service Creation requested
Disagregable = false
Golden = null
INFO successfully parsed...
1311673395766624256
Overlapping computed
No overlapping with other services
Calling PCE
Available paths obtained...
0
[(OF13|3@OF13|1->OF13|2@OF13|6),(OF13|4@OF13|6->OF13|3@OF13|4)]
[(OF13|2@OF13|1->OF13|2@OF13|2),(OF13|3@OF13|2->OF13|1@OF13|3),(OF13|2@OF13|3->OF13|4@OF13|4)]
[8000, 0, 8000, 0, 8000, 8000, 0, 8000]
1
[(OF13|2@OF13|1->OF13|2@OF13|2),(OF13|4@OF13|2->OF13|1@OF13|5),(OF13|3@OF13|5->OF13|2@OF13|4)]
[(OF13|3@OF13|1->OF13|2@OF13|6),(OF13|4@OF13|6->OF13|3@OF13|4)]
[8000, 8000, 0, 0, 8000, 0, 8000, 8000]
2
[(OF13|3@OF13|1->OF13|2@OF13|6),(OF13|3@OF13|6->OF13|2@OF13|5),(OF13|3@OF13|5->OF13|2@OF13|4)]
[(OF13|2@OF13|1->OF13|2@OF13|2),(OF13|3@OF13|2->OF13|1@OF13|3),(OF13|2@OF13|3->OF13|4@OF13|4)]
[8000, 8000, 8000, 8000, 8000, 8000, 0, 0]
3
[(OF13|3@OF13|1->OF13|2@OF13|6),(OF13|4@OF13|6->OF13|3@OF13|4)]
[8000, 0, 0, 0, 0, 0, 0, 8000]
4
[(OF13|2@OF13|1->OF13|2@OF13|2),(OF13|4@OF13|2->OF13|1@OF13|5),(OF13|3@OF13|5->OF13|2@OF13|4)]
[0, 8000, 0, 0, 8000, 0, 8000, 0]
5
[(OF13|3@OF13|1->OF13|2@OF13|6),(OF13|3@OF13|6->OF13|2@OF13|5),(OF13|3@OF13|5->OF13|2@OF13|4)]
[8000, 8000, 0, 8000, 0, 0, 0, 0]
6
[(OF13|2@OF13|1->OF13|2@OF13|2),(OF13|3@OF13|2->OF13|1@OF13|3),(OF13|2@OF13|3->OF13|4@OF13|4)]
[0, 0, 8000, 0, 8000, 8000, 0, 0]
7
[(OF13|2@OF13|1->OF13|2@OF13|2),(OF13|4@OF13|2->OF13|1@OF13|5),(OF13|2@OF13|5-
>OF13|3@OF13|6),(OF13|4@OF13|6->OF13|3@OF13|4)]
[0, 0, 0, 8000, 8000, 0, 8000, 8000]
8
[(OF13|3@OF13|1->OF13|2@OF13|6),(OF13|3@OF13|6->OF13|2@OF13|5),(OF13|1@OF13|5-
>OF13|4@OF13|2),(OF13|3@OF13|2->OF13|1@OF13|3),(OF13|2@OF13|3->OF13|4@OF13|4)]
[8000, 0, 8000, 8000, 0, 8000, 8000, 0]

ID: 1
Starts: 09-06-2015
Ends: 30-09-2015
ServiceList: A
Alternatives: 9
```

Figure 3: DynPaC log output when processing a request

3.1.2 Service Load / unload

DynPaC Core is synchronized with the system time. In this way, it is able to automatically start and end bandwidth reservations for the different services requested, according to their scheduling and lifetime. Figure 4 shows the log messages printed by the DynPaC framework when a new network configuration is loaded into the network in order to allow the provision of a newly requested service.

```
Current Time updated...1427123220001
Loading SNAPSHOT...1
Loading Service... TEST
Vector to find[30, 0, 30, 0, 30, 30, 0, 30]
Comparing to...[30, 0, 30, 0, 30, 30, 0, 30]
Found!
  Loading Pattern... 0
  Loading Primary Path... [(OF13|3@OF13|1->OF13|2@OF13|6),(OF13|4@OF13|6->OF13|3@OF13|4)]
  Loading Secondary Path...[(OF13|2@OF13|1->OF13|2@OF13|2),(OF13|3@OF13|2->OF13|1@OF13|3),(OF13|2@OF13|3->OF13|4@OF13|4)]
Marking service as active...
Getting flowpattern to be activated...
Pushing FlowMods...
Building common match...
Service cookie1234000100000000
Forward cookie1234000100004000
Registering cookie1234000100004000
Building common match...
Service cookie1234000100000000
Registering cookie1234000100000000
Adding cookie 1234000100000000
Adding cookie 1234000100008000
Adding cookie 1234000100000000
Adding cookie 1234000100008000
TRACKED FLOWS
NODE: OF13|1 // FLOW INFO: ID - 1311673395766624256 LAST TIME - 0 BYTECOUNT - 0 MAX BW - 30 MEASURED BW - 0
NODE: OF13|1 // FLOW INFO: ID - 1311673395766640640 LAST TIME - 0 BYTECOUNT - 0 MAX BW - 30 MEASURED BW - 0
```

Figure 4: DynPaC log output when installing a new service

3.2 Resiliency

Another feature implemented in DynPaC framework is resiliency. This feature allows the fast reallocation of the services running over the network and their adaptation to a suitable configuration when a network failure occurs.

Depending on the kind of service and the rest of the concurrent services running at a certain moment, DynPac resiliency module tries to load backup paths that have already been computed to give a backup solution to each currently established service. The computation of the backup path is performed at the moment when system tries to allocate a new service. The computed backup path is always a path totally disjoint from the primary path. This way if any of the links of the primary path, or a combination of them, fails, the backup path is always valid.

It should be highlighted at this point that the DynPaC framework considers the existence of two different types of services: gold services and regular services. Services defined as gold are always guaranteed a totally disjoint bakup path. In the case of regular services, however, the backup path is only provided if there

are enough free resources in the network to provide it, but there is no guarantee of the backup path being available at any time or under any network condition.

When a new network configuration is loaded and pushed into the network, simultaneously, resiliency mechanisms are also loaded according to the currently provided services and their properties. Links used per each service are cached and a backup path is set in case of a certain link failure.

3.3 Traffic shaping and flow disaggregation

Flow disaggregation is an advanced feature implemented in DynPaC framework that allows maximizing the network utilization. In case of not availability of bandwidth for a new service given the current configuration of services, the flow disaggregation functionality allows reordering current services or even splitting running services into several sub-flows, which can then be forwarded according to a different network configuration. This way, it is possible to leave enough free bandwidth to allow the establishment of the newly requested service. In order to allow an efficient splitting of flows, an external entity, named traffic pattern analyzer, gathers traffic profile information and provides suitable alternatives to split traffic.

3.3.1 Traffic reordering and disaggregation feature

This feature provides a mechanism to maximize the usage of physical resources of the network. When a service cannot be accepted due to the lack of enough guaranteed bandwidth given the current services configuration, the DynPaC framework tries to find an alternative configuration of services which would allow providing the newly requested services.

The first approach followed by the DynPaC framework is to try to find an alternative configuration of services which allows providing all the requested services (including the new one) without splitting any flow. With this aim, the DynPaC framework evaluates all the possible combinations of paths for each of the requested services that would allow providing all these services at the same time. If this analysis does not provide a positive solution, the DynPaC framework gathers advanced information about running services, with the aim of splitting one or more running services into smaller ones that, now, can be moved to alternative paths, freeing enough bandwidth for the new service to be installed. This mechanism allows increasing the number of concurrent services in some circumstances, depending on the splitting patterns provided, source and destination nodes and bandwidth.

As an example if a service that cannot be fit into the network is requested, DynPaC analyzes running traffic information to split service into sub-flows according to the information of an external entity known as the Traffic Pattern Analyzer, as represented in Figure 5.

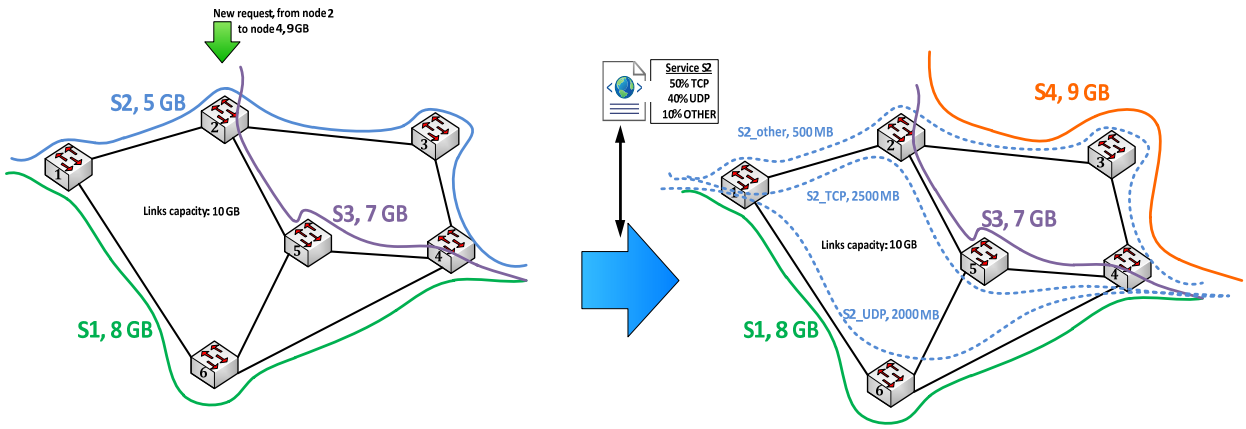

Figure 5: Example of traffic disaggregation feature

3.3.2 Traffic shaping

In order to enable the disaggregation feature, detailed service characterization information is needed. This information is provided by an external entity known as the Traffic Pattern Analyzer. The Traffic Pattern Analyzer characterizes communication sub-flows inside the communication flow that corresponds to a network service. Sub-flows are defined in a more specific way than the general service flow. More specifically, the Traffic Pattern Analyzer provides sub-flow characterization with a granularity that considers all packet header fields from link layer to application layer of the TCP/IP model. Additionally, the Traffic Pattern Analyzer specifies also the amount of bandwidth of the total network service flow that corresponds to each of the identified sub-flows.

3.4 Monitoring

A monitoring service has been added to check whether a user exceeds the bandwidth agreed, informing to the DSM to carry out corresponding actions or inform to the user about this over usage of the network.

4 The DynPaC framework

This section overviews the design of the DynPaC framework, presenting its architecture and how the framework is integrated within the Open Daylight Project.

4.1 Architecture

The DynPaC framework has been designed to provide the functionalities described in Section 3. It consists of four custom modules: the DynPaC Graphic User Interface (GUI), the Path Computation Element (PCE), the Monitoring module and the DynPaC Service Manager (DSM), which acts as the coordinator of the framework. Besides, the framework relies on three modules of the Open Daylight Project: the Flow Programmer, the Topology Manager and Switch Manager. The full architecture of the framework is depicted in Figure 6.

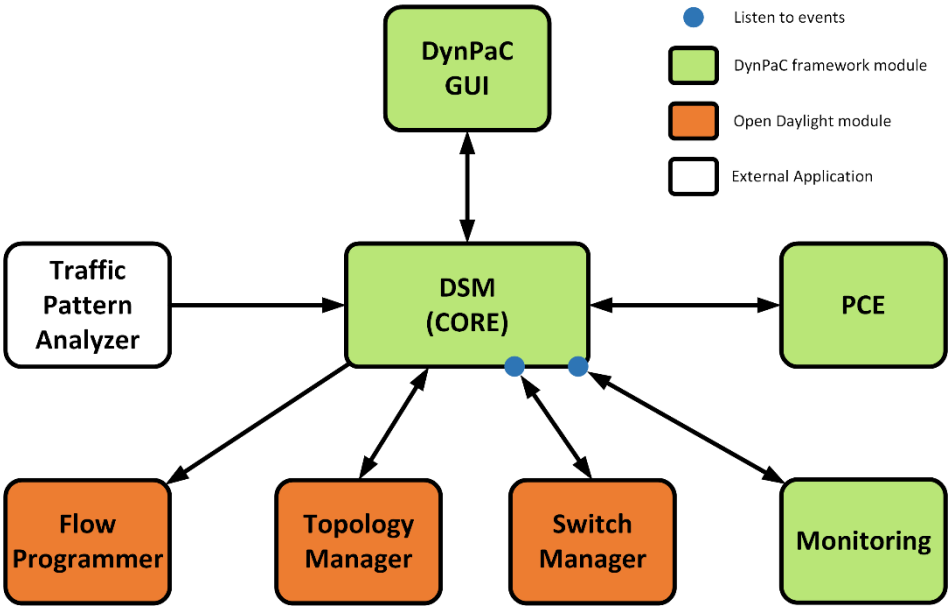

Figure 6: Architecture of the DynPaC framework

4.1.1 DynPaC Service Manager

Of uttermost importance in the DynPaC framework is the DynPaC Service Manager (DSM), which acts as the coordinator. It is in charge of the communication among all the modules that conform the framework, and establishes the workflow among each of them. In addition, it also listens to the asynchronous events sent by other modules to inform about the changing conditions in the network.

On the other hand, the DSM is also in charge of determining whether a newly requested service can be provided or not. It takes into account the available resources (bandwidth) in the network during the entire lifetime of the requested service. To be able to do that, the DSM uses snapshots of the network, a concept that will be later explained in Section 6. In addition, it also handles the resiliency.

4.1.2 DynPaC Graphic User Interface

The DynPaC GUI allows requesting, removing and obtaining information about the DynPaC services. It consists of a tab that has been added to the Open Daylight GUI. As depicted in Figure 7, the DynPaC GUI allows to request services specifying the nodes, the end and start times, the matching fields that characterize the service (in this case source and destination MAC addresses), the bandwidth and whether the service is Golden or not.

This module also provides information about the requested services. For instance, it informs about the status of the service, differentiating the *RESERVED*, *ACTIVE* and *ENDED* status. Furthermore, it also shows the details of the specific services, including all the information that was used at the time of requesting the service.

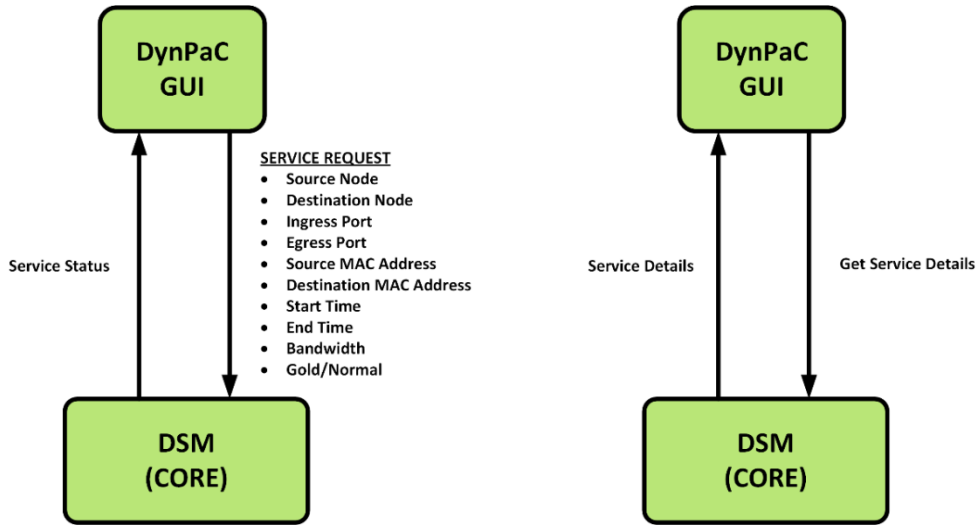

Figure 7: Communication between the DynPaC GUI and the DSM.

4.1.3 Traffic Pattern Analyzer

The DynPaC framework has also being designed taking into account an interface towards a Traffic Pattern Analyzer (TPA). This interface has been defined to receive information in the form of an XML file describing the traffic distribution of an already provisioned service. The information provided by the TPA is later

used by the PCE to perform the service disaggregation and optimize the network utilization. Figure 8 depicts how the TPA communicates with the DSM in order to provide information about a service traffic distribution.

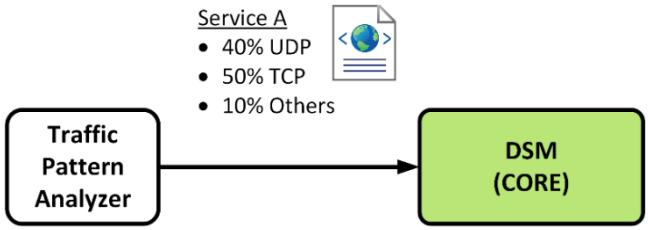

Figure 8: Information provided by the TPA to the DSM.

4.1.4 Path Computation Element

The Path Computation Element (PCE) module of the DynPaC framework is the element in charge of the computation of the paths. It implements two different algorithms. The first one provides all the possible paths between a given source and a given destination. This algorithm is used at boot up time to compute all the possible paths between all the source and destination pairs. Having all the possible paths pre-computed improves the performance of the DynPaC framework, as they are used when new services are requested by the users. Furthermore, the different possible paths for a given pair of source and destination are stored in the DSM in order to be applied if necessary when new services are requested. This approach allows rearranging the services and making room for new incoming services as a prior step to the flow disaggregation.

In addition, the PCE also implements an algorithm to disaggregate the services in more granular flows based on the information provided by the TPA. To be able to do it, it receives information about the requested service, including the source and destination nodes, so as about the status of the network during the lifetime of the service. When the PCE finds a suitable disaggregation combination that makes possible the reservation of the newly requested service, it responds with the new paths of the disaggregated subservices.

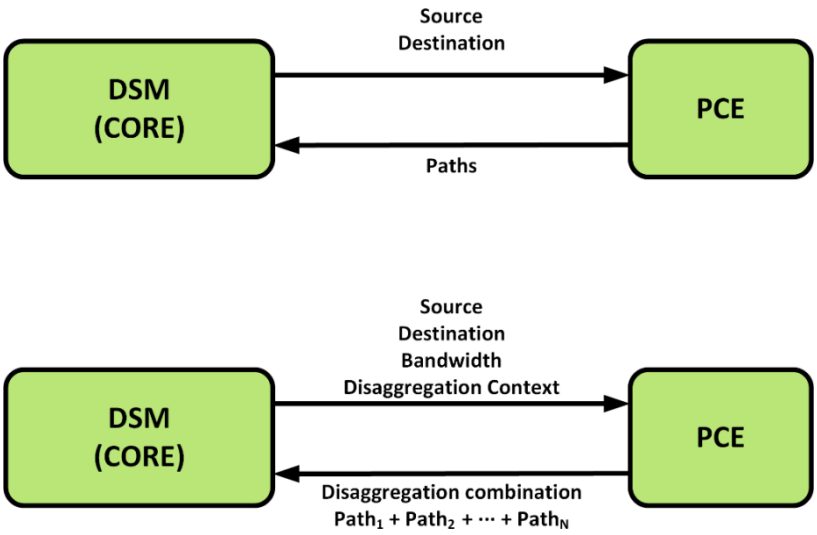

Figure 9: Communication between the PCE and the DSM.

4.1.5 Monitoring module

Finally, the DynPaC framework also includes a monitoring tool that allows to keep track of the services installed in the network. It periodically checks whether the services are compliant with their Service Level Agreements (SLA) in terms of occupied bandwidth, that is, if they are not using more than the requested bandwidth.

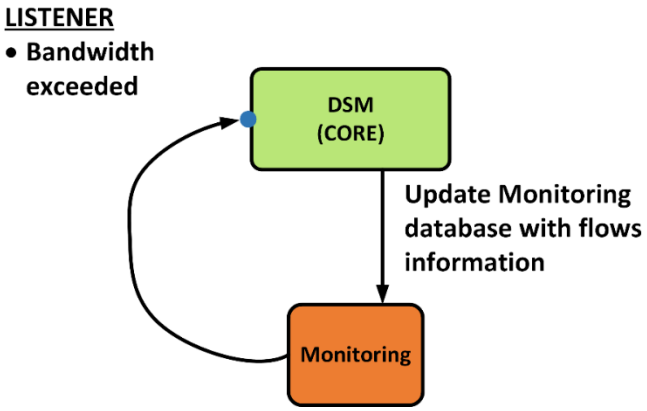

Figure 10: Communication between the monitoring module and the DSM.

4.2 Integration with the Open Daylight project

As mentioned before, the DynPaC framework uses some of the modules available at the Open Daylight Project. The reutilization of these modules has greatly simplified the development of the framework.

One of the modules being reused is the Topology Manager, which contains information about the network graph. This module is used to discover the topology of the network automatically. It also provides the topological information to the DSM when this module requests it, as it can be seen in Figure 11.

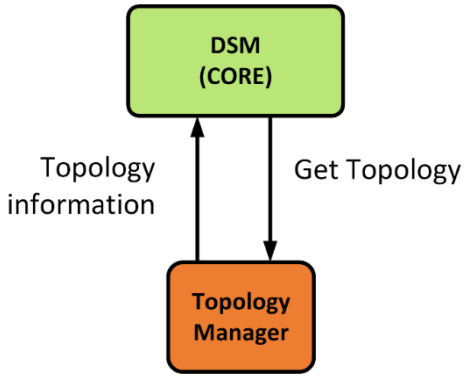

Figure 11: Communication between the Topology Manager and the DSM.

Another module that is also reused is the Flow Programmer. This module allows to program the OpenFlow nodes easily, as it already implements the necessary OpenFlow messages to insert and remove services in the devices.

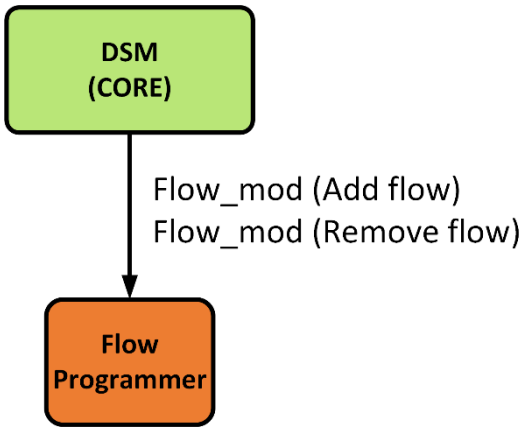

Figure 12: Communication between the Flow Programmer and the DSM.

Last but not least, the Switch Manager provides all the information to manage the elements of the network. For instance, it is in charge of retrieving the features of the nodes. Besides, it also provides information about the topology changes. For instance, it informs about port, link and node status.

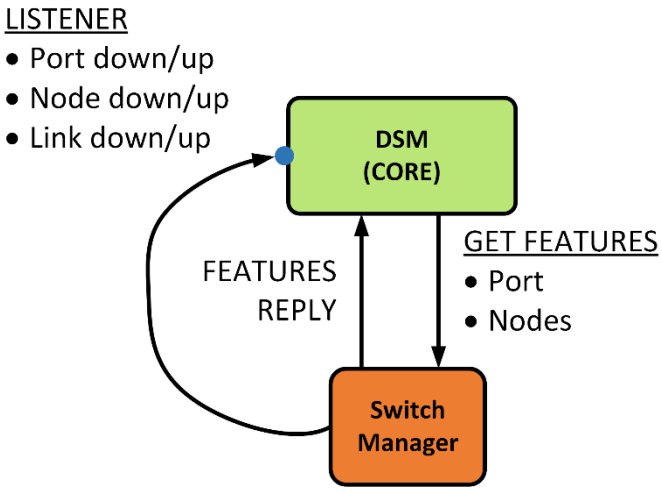

Figure 13: Communication between the Switch Manager and the DSM.

5 Constraints imposed by the GN3plus infrastructure to the DynPaC framework

This section overviews the constraints imposed by the two GN3plus infrastructures, the Géant OpenFlow Facility (GOFF) and the Géant Testbed Service (GTS), to the DynPaC project.

5.1 Géant OpenFlow Facility (GOFF)

The Géant OpenFlow Facility offers computational resources in five different locations connected in a full mesh fashion, which can be later used as hosts or as OpenFlow capable devices (Open vSwitches). The slicing of the network elements is performed by the specific purpose controller FlowVisor, which allows to share the OpenFlow devices among the different projects.

Regarding monitoring, two approaches were studied: use OpenFlow statistical messages or use standardized protocols such as SNMP, sFlow or Netflow. On the one hand, the use of OpenFlow statistical messages is completely supported by the GOFF, as the Open vSwitches (OVS) support these kind of messages and the FlowVisor does not impose additional constraints. However, this approach implies the development of a custom monitoring module. On the other hand, the use of standard monitoring protocols is not supported in this facility. The OVS do support sFlow and NetFlow, but the facility does not allow to access the OVSs in order to configure them properly. In conclusion, the GOFF imposed to perform monitoring using OpenFlow statistical messages.

Another very important constraint imposed by the GOFF is related to the Quality of Service (QoS). The QoS plays a key role in the DynPaC project, as it is designed to provide Bandwidth on Demand (BoD) services to users. Nevertheless, the GOFF facility does not allow to enforce bandwidth. Firstly, the FlowVisor assigns a queue to each slice in its later versions. As a consequence, every OFPT_FLOW_MOD and OFPT_PACKET_OUT messages are rewritten to be forwarded to that queue. Similarly, every enqueue action associated to a flow is also rewritten, which implies that it is not possible to assign flows to a queue different that the one the slice is assigned for. Secondly, even without the FlowVisor, the creation of queues in the OVSs would be impossible. The creation of queues is out of the scope of the OpenFlow protocol. OVSs have support for queues, which can be created with the OVSDB (OpenVSwitch Database) Management Protocol and manually. However, as in the case of the monitoring, the GOFF does not allow to manage the OVSs and the creation of queues is impossible.

In OpenFlow is also possible to enforce QoS with the metering messages, which allow to establish rate limiters associated to an inport. Nonetheless, this feature is only available in OpenFlow 1.3 and upper versions of the protocol. The FlowVisor is the element in charge of the slicing of the network, and it only supports OpenFlow 1.0. Thus, it impossible to use OpenFlow 1.3 and its metering messages.

Finally, one of the main interesting features of the DynPaC project is that it reacts upon topology changes. The GOFF facility did not impose any constraints regarding the discovery of the topology, but it did impose some constraints related to the introduction of changes in the topology manually. For instance, to test

that the resiliency mechanisms of the framework works properly it is necessary to force links up and down. Nevertheless, the GOFF does not provide the necessary mechanisms to do that.

5.2 Géant Testbed Service (GTS)

Due to the limitations imposed by the GOFF, the DynPaC project proceeded with the migration to the GTS. The GTS allows to reserve computational and network resources automatically, and it is a much more flexible service than the GOFF.

The only constraint encountered at the GTS is related to the monitoring. The GTS supports software and hardware OpenFlow devices, with OVS and HP switches respectively. On the one hand, it provides full access to the Open vSwitches, which implies that it is possible to configure and run standard monitoring protocols such as the previously mentioned sFlow or Netflow. On the other hand, the monitoring capabilities of the hardware switches are not that straightforward. For the moment, the virtual switch instances that are provided to the end-users of the GTS do not support these monitoring protocols. The HP switches do support SNMP, but the information obtained through this protocol would be referred to the entire switch, no only about the virtual instance provided to the end users. Therefore, it is of no use to the project goal. Besides, as in the case of the GOFF, it is still hard to simulate link failures using GTS.

6 DynPaC low level design

This section describes each of the modules implemented in the DynPac framework, giving advanced details about the implementation, flowchart and main structures and information managed by each one. At this point, it must be taken into account that the DynPaC framework has been conceived as a proof of concept to validate that the proposed flow disaggregation mechanisms allow optimizing network utilization and that the defined resiliency mechanisms are fast and effective. For this reason, the DynPaC framework has been designed and implemented prioritizing modularity, flexibility and ease of management and modification. However, such a designed is not the best alternative in terms of performance and it could be optimized.

6.1 DynPaC Service Manager

DynPaC Service Manager (DSM) represents the core of the DynPaC Framework. It processes requests coming from the web interface and network events, such as link addition or removal. It also gathers information from several modules of the Open Daylight Framework

DynPaC orchestrates network elements trying to find the best coexistence pattern considering all overlapping situations of the services running over the network when a new one needs to be installed. Furthermore, it guarantees the profile of the service (Gold or Regular) during the lifetime of the service. Not only coexistence features are considered, but also network changes.

6.1.1 Services

A service handled by DynPaC framework is treated as a connection between two edge ports inside the domain managed by ODL running DynPaC modules. This service has a limited duration. Additionally, the MAC addresses of the clients behind a port can be specified, enabling sharing the same port between several different connections, as isolation is guaranteed by setting those MACs as service properties.

6.1.1.1 *Service properties*

For each service, several parameters are gathered and stored for further processing from the user interface.

- **Name**: name provided for the service.
- **Source Node**: ingress OpenFlow node for the service.
- **Source Port**: source port ID at ingress node.

- **Destination Node**: egress OpenFlow node for the service.
- **Destination Port**: out port ID at egress node.
- **Start time**: stored in milliseconds obtained from the date and time provided from the web interface or REST API.
- **End time**: stored in milliseconds obtained from the date and time provided from the web interface or REST API.
- **Bandwidth:** total bandwidth allowed for the service.
- **Gold type:** this value is true if this service has been reserved as gold (backup guaranteed)

Apart from these parameters which are specified by the user/experimenter, more parameters are also associated with a service, and specified by the Service Manager. These parameters are for internal usage and transparent to the experimenter.

- **State**: dynamic property that indicates which is the current status of the service
    - **REQUESTED**: the service request has been successfully processed and accepted.
    - **ACTIVE:** service is currently running and pushed into the network.
    - **ENDED:** service has already finished and it has been removed from the network.
- **Cookie:** Service unique identifier. This identifier is opaque to the user and it only has meaning inside DynPaC context. This identifier is assigned when a new service is requested.
- **Alternative Flow Patterns:** a list of path patterns that are physically valid to allocate the service in the network when no traffic is running over the network. Those patterns can be pairs of paths, if a backup is required, or a single path for a regular service when backup requirements are not mandatory. Moreover, these flow patterns can also be disaggregated into several sub-flows. This feature will be further detailed in section *6.3.2.4 Disaggregation feature*.
- **Priority:** Current priority used for the service. OpenFlow priority is used during the transition between different network configurations to avoid packet losses. By using higher priority rules DynPac is able to overwrite existing forwarding configurations for the same flows.

6.1.1.2 *Service types*

DynPaC can distinguish two different types of services regarding the resiliency level guaranteed to the service. This property is set during the service request process. A **Gold service** has a totally disjoint backup path associated with it, which is guaranteed in any circumstances, considering this virtual reservation when new service requests are done. In contrast to a Gold service, when requesting a **Regular service** backup option is not assured. DynPaC tries to provide the backup, like in the case of a Gold service, as default criteria.

However, depending on the concurrent services running on a certain time frame, it might not be possible to guarantee this backup path all the time.

6.1.1.3 *Service lifetime*

All the services accepted to be installed into the DynPaC framework must have a start date later to current system time and a finite end date. Currently, no recursive service can be requested automatically. Services can be requested using DynPaC web interface or Rest API

6.1.1.4 *Service coexistence*

A service is accepted only if all the coexistence situations with the rest of the services already programmed have at least a solution where all the concurrent services for each situation can maintain all their requirements regarding resiliency and bandwidth.

To solve these interactions, a new concept is introduced: network snapshot. The concept of network snapshot is deeply explained in section *6.1.2 Network Snapshot*.

6.1.1.5 *Encoding service into OpenFlow cookie field*

OpenFlow cookie is a 64-bit field used to identify a flow in an OpenFlow enabled switch. It makes easier the task of handling flows with a quick identification way or bulk delete actions.

DynPaC makes use of this field and encodes several parameters into that cookie when pushing a flow into the network. The structure of the cookie field is depicted in Figure 14:

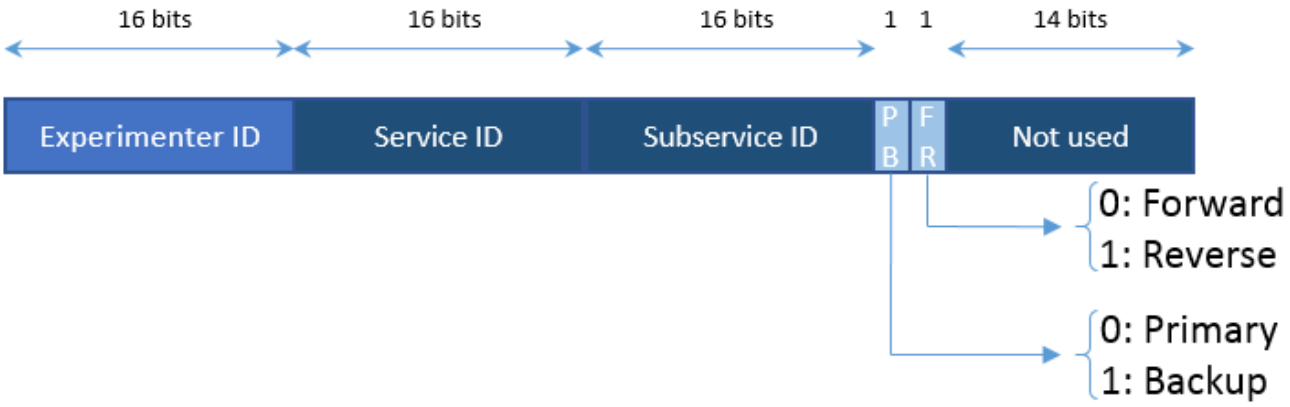

Figure 14: Communication between the Switch Manager and the DSM.

- **Experimenter ID:** implemented as a future support of multiple experimenter framework.

- **Service ID:** sequential identifier assigned to each service request.
- **Subservice ID**: Identifier assigned to each sub-flow of a certain service, when disaggregation is applied to the given service. This field is internally handled to track statistics about a flow
- **Forward / Reverse flag**: Used to distinguish the direction of rules pushed into the network for the same service.
- **Primary / Backup flag**: Intended to indicate whether the Primary path or the Backup path of a service is currently loaded into the network. Used to provide resiliency mechanisms depending on the alternative being used.

6.1.1.6 *Service Patterns*

Each service has an associated set of alternative paths suitable to provide the connectivity defined by the service. These path patterns represent all the available path configurations (considering also backup paths if available or required). Each pattern must be unique, and consequently, a unique vector identifies each pattern.

Inside a pattern structure, different information is stored about what kind of pattern it is. There are some flags to check whether the service has a backup associated or not, or to mark a certain service as disaggregated. When a service is disaggregated, the resulting disaggregated sub-flows are also referenced from this instance.

6.1.2 Network Snapshot

A Network snapshot is the object handled by the DynPaC Core to define all the different possible network configurations for the same set of concurrent services. That is, each snapshot considers all the possible alternative paths for a given set of concurrent services and for a certain time lapse.

In this way, for a new service to be admitted, all the different service interactions or snapshots during the lifetime of the newly requested service must converge to one or more solutions. This solution may not be unique to resolve all the snapshots taking place during the lifetime of a service, as service paths can be rerouted during their lifetime to obtain the best traffic pattern fits that maximize the amount of services allocable into the network at any time.

6.1.2.1 *Network Matrix View*

The entire network is modeled as a column matrix where the row index represents each link of the network unequivocally. This association is dynamically done, formed during the link discovery procedure when DynPaC boots up. Once this relationship has been established, it is consistent during the whole DynPaC execution lifetime. Both go and return links for the same physical link are grouped into the same identifier, as installed paths are considered symmetrical.

The value of each position represents the remaining bandwidth for the corresponding link. As an example, when DynPaC boots up, the topology vector will represent the maximum capacity of the links managed by DynPaC, as represented in the example provided by Figure 15, where all the links are 10GB full duplex links.

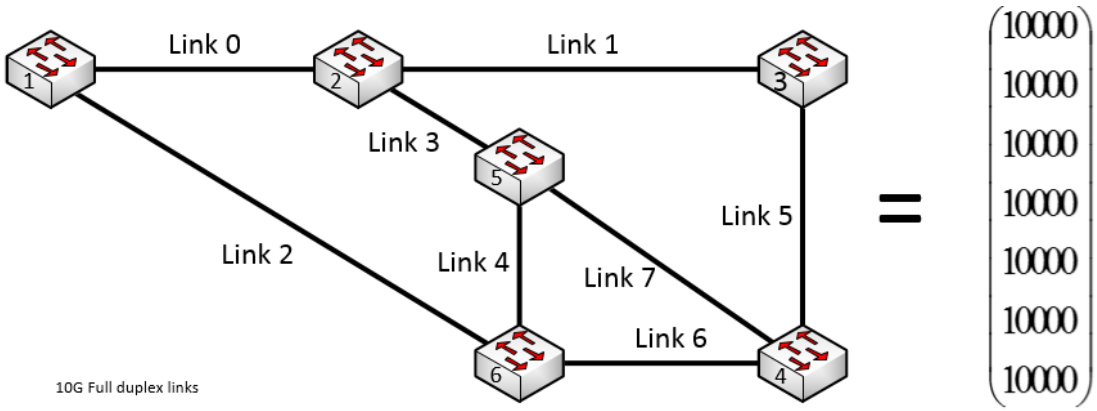

Figure 15: Topology matrix representation

These vectors are used to compute alternatives that can solve a certain network snapshot, A path can also be represented by a vector where the position of the links used are set to its bandwidth value. Then, solving a network snapshot implies that the overall addition of the concurrent paths in use must not exceed the maximum capacity vector.

6.1.2.2 *Service pattern matrix view*

As the network bandwidth available is represented by a vector, service patterns are also modelled as vectors representing bandwidth used per each link. Both patterns containing primary and backup paths or single path patterns have the same behaviour. Figures 16 and 17 provide examples of the representation of regular services and gold services as vectors respectively.

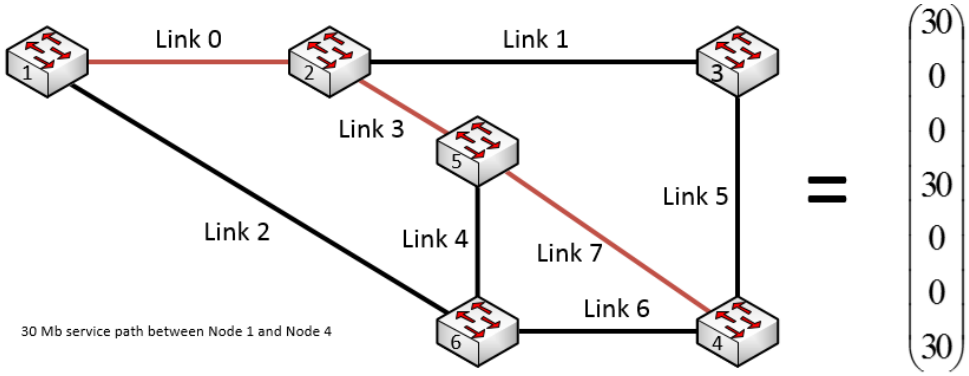

Figure 16: Regular service matrix representation

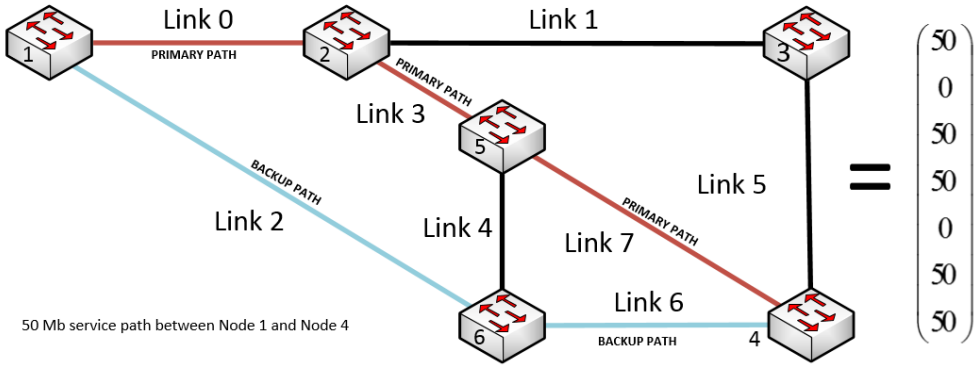

Figure 17: Gold service matrix representation

6.1.2.3 *Network Snapshot generation*

Any service registered at DynPaC framework generates, at least, a new snapshot, depending on the number of interactions with the already allocated services.

When just one service is programmed in DynPaC, the snapshot duration is the same as the duration of the service that creates the snapshot. This is the situation depicted in Figure 18

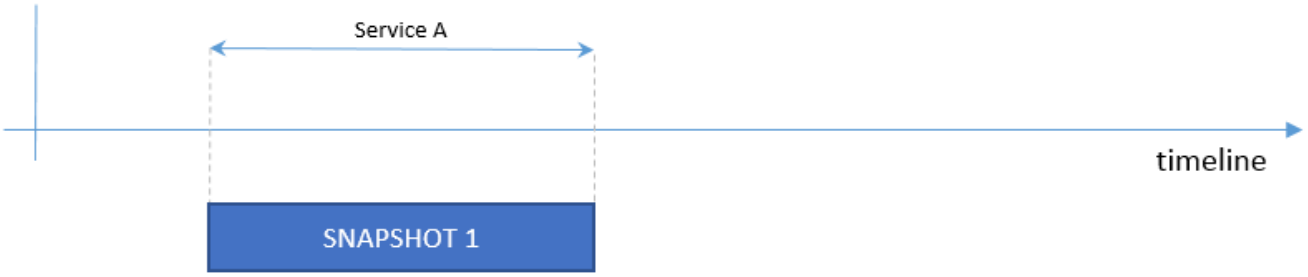

Figure 18 : Simple snapshot generation from a service request

However, when there is a collision between services, one or more snapshots can be generated. In Figure 19 a more complex interaction is represented.

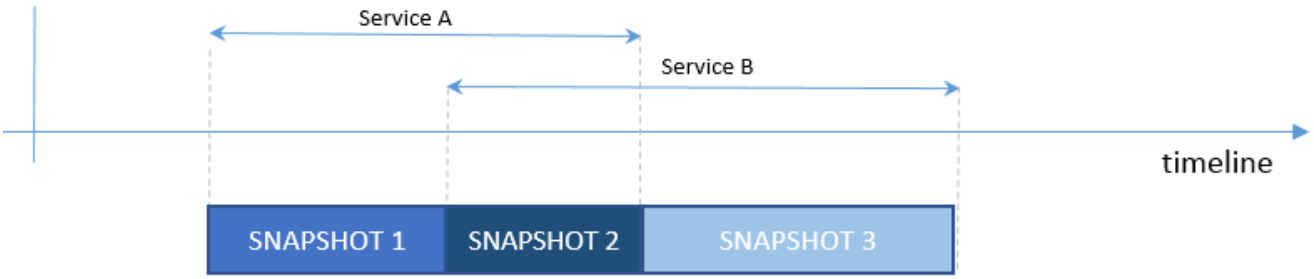

Figure 19: Interaction between two services.

The snapshot 1 is already solved, as all the possible network configurations were already computed at the time of providing service A. However, its end time has been modified. A new snapshot is created from interaction between services A and B, and a second one is also generated where only service B is active. All these snapshots must have at least one possible resolution in order to be able to provide service B when service A already exists.

Snapshot generation patterns

- **Simple pattern**: when no interactions with other services are present, allocation can be done straight forward, as shown in Figure 20.

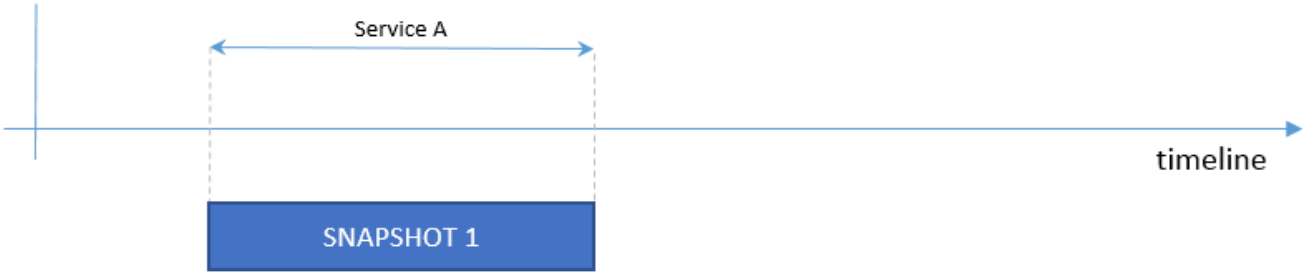

Figure 20: Snapshot generation for a single pattern

- **Partial overlap**: from this type of interaction several actions have to be solved. Considering that service A was the first service pushed into the network, snapshot 1 was already resolved, but it has to be resized, its end time is shortened up to the service B start time. Snapshots 2 and 3 are new and have to be resolved. Snapshot 2 has to take into account the interaction between the two services, whereas snapshot 3 is a simple one, considering just the existence of service B. The explained situation is graphically represented in Figure 21.

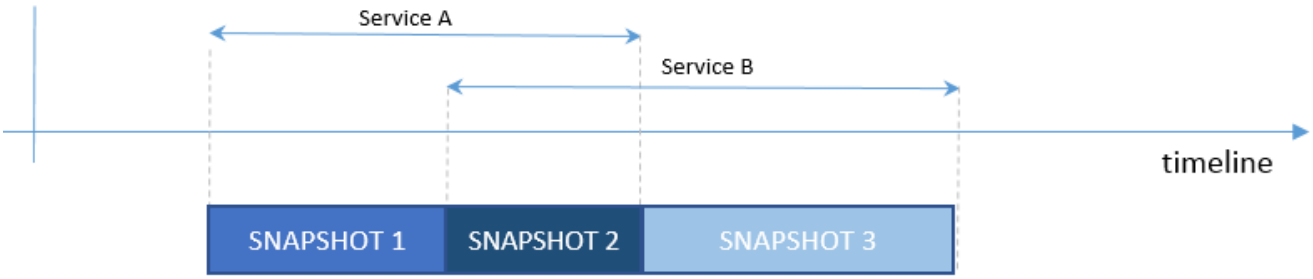

Figure 21: Snapshot generation for partial overlap between 2 services

- **Full overlap:** in such a scenario, the new service fully overlaps an existing one or the new service is fully contained inside an existing one. Two of the three snapshots created have the same solution, already computed, but different start and end time. Snapshot 2 always would need to be calculated. Such a situation is represented in Figure 22.

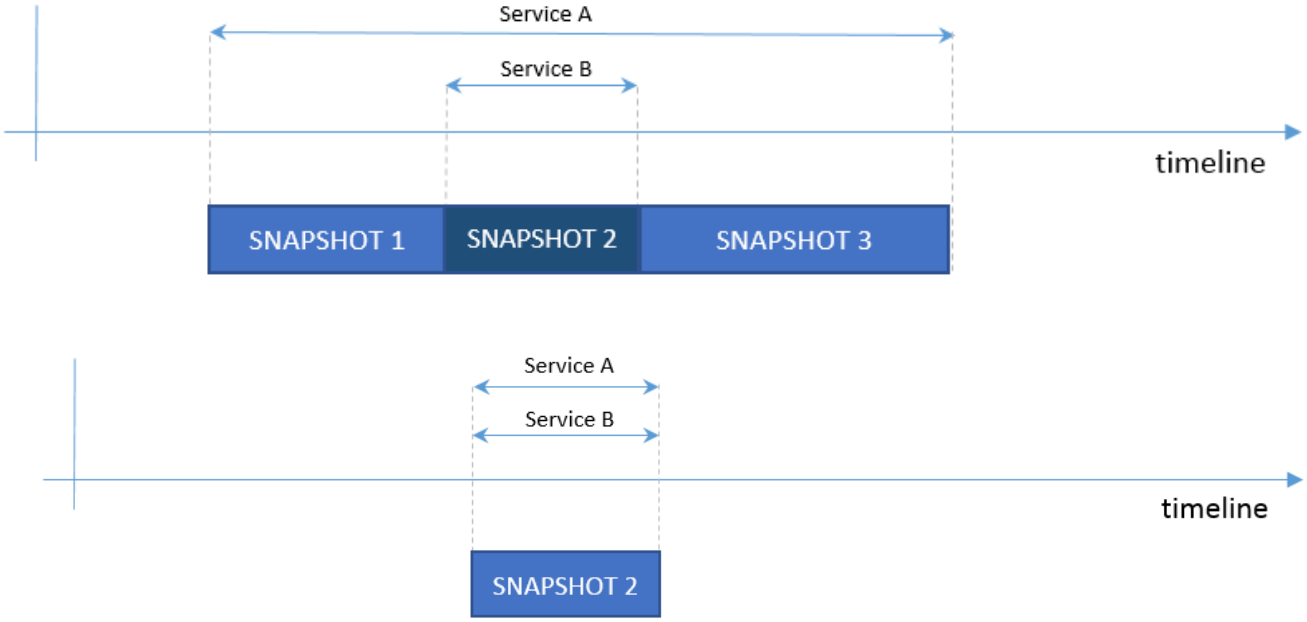

Figure 22: snapshot patterns generated by a full overlap

Building potential service alternatives

All the potential alternatives to solve a given snapshot are considered in the first phase of the resolution, as shown in Figure 23. If no solution can be found, DynPaC tries to disaggregate one or more of the existent services (only if it has already been profiled by the Traffic Pattern Analyzer) and reallocate the resulting sub-flows along different paths. Then, new alternatives can be found and applied to solve each snapshot

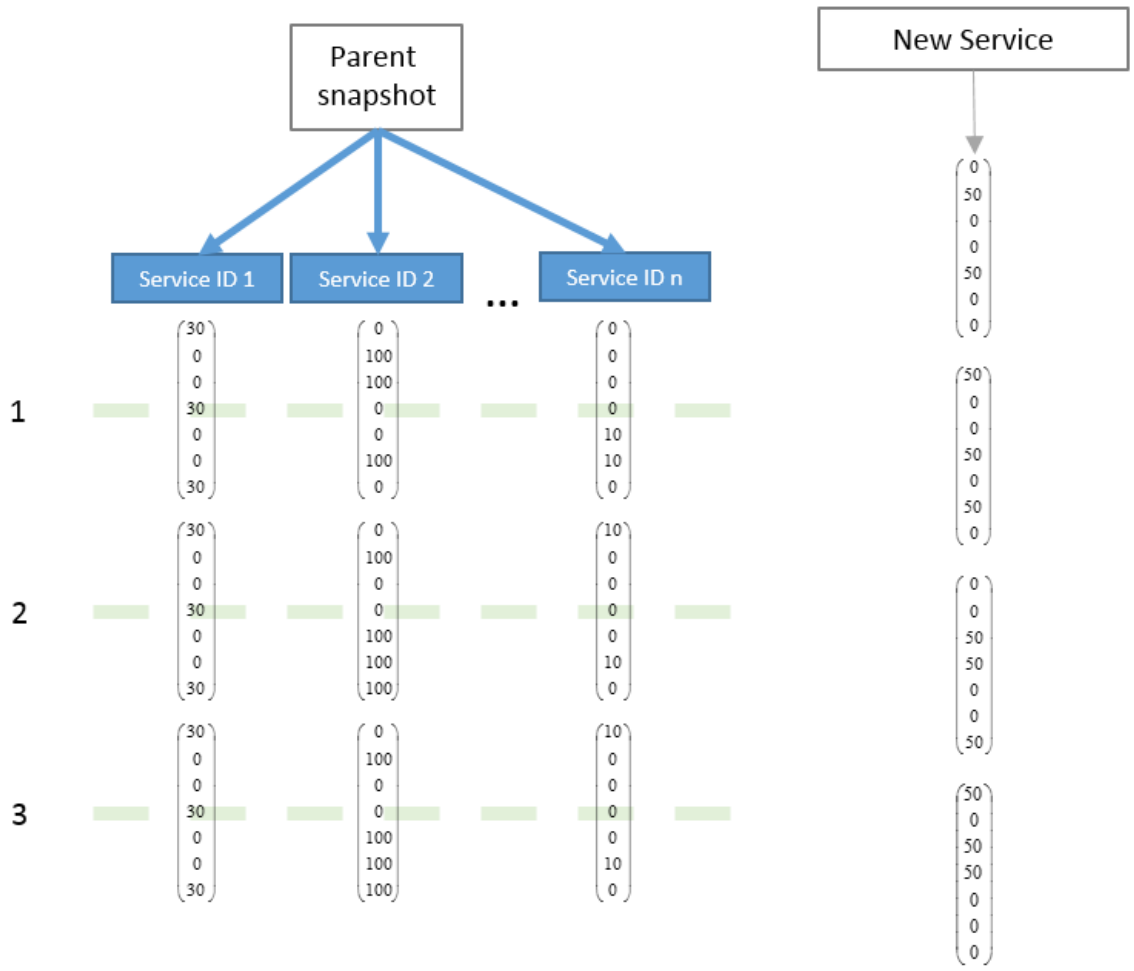

Figure 23: Potential initial alternatives for a snapshot resolution

6.1.2.4 *Network Snapshot resolution*

To resolve a snapshot, DynPaC core stores some information about it. Firstly, a snapshot has a pointer to the parent snapshot from which it has been created. It inherits solutions from previous snapshots and starting from these solutions, tries to resolve each previous resolution pattern with alternatives provided to allocate the new service in the network.

In order to obtain all the possibilities, DynPaC framework tries to combine alternatives which were valid to solve parent snapshot with flow patterns of the new service. This way, all the alternatives not exceeding the maximum bandwidth available vector are considered as valid. This combination process is graphically represented in Figure 24.

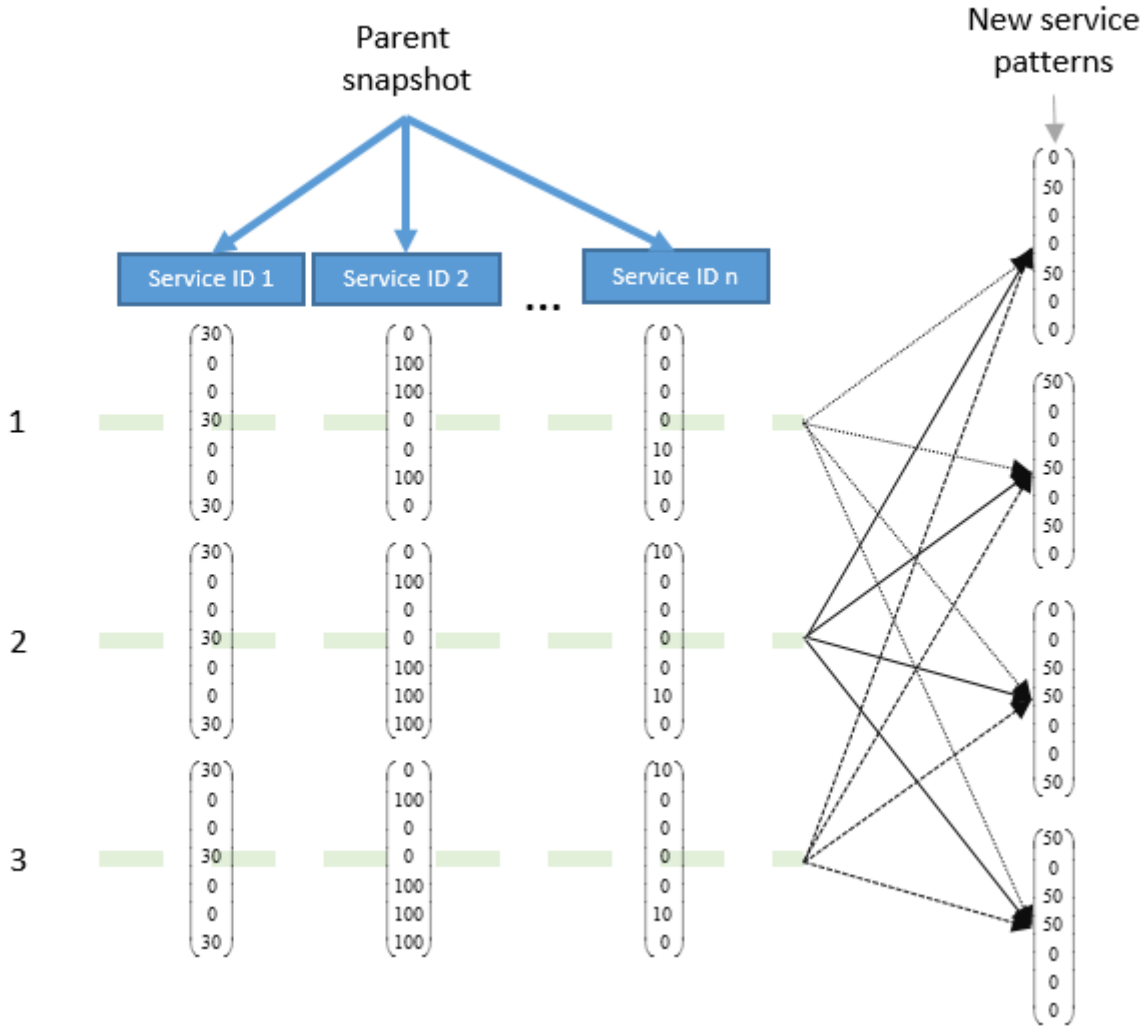

Figure 24: trial of finding the potential solutions to the snapshot

Only successful trials are stored as valid solution patterns for each snapshot. All the resulting snapshots can be resolved at the same time, as they are completely independent.

6.1.2.5 *Transition between snapshots*

When there is no gap between two snapshots, and one or more services of the ending snapshots continue in the next one, DynPaC core tries to maintain as many used traffic patterns as it is possible. This way, the minimum required flows are only moved during the transition.

The example depicted in the Figure 25, describes a smooth transition between 2 consecutive snapshots. The difference between both snapshots is that service 3 is substituted by service 4. Service 1 and service 2, which are common to both snapshots, are not moved to a different path when the snapshot change occurs, as the paths used in the first snapshot are still valid during the second snapshot.

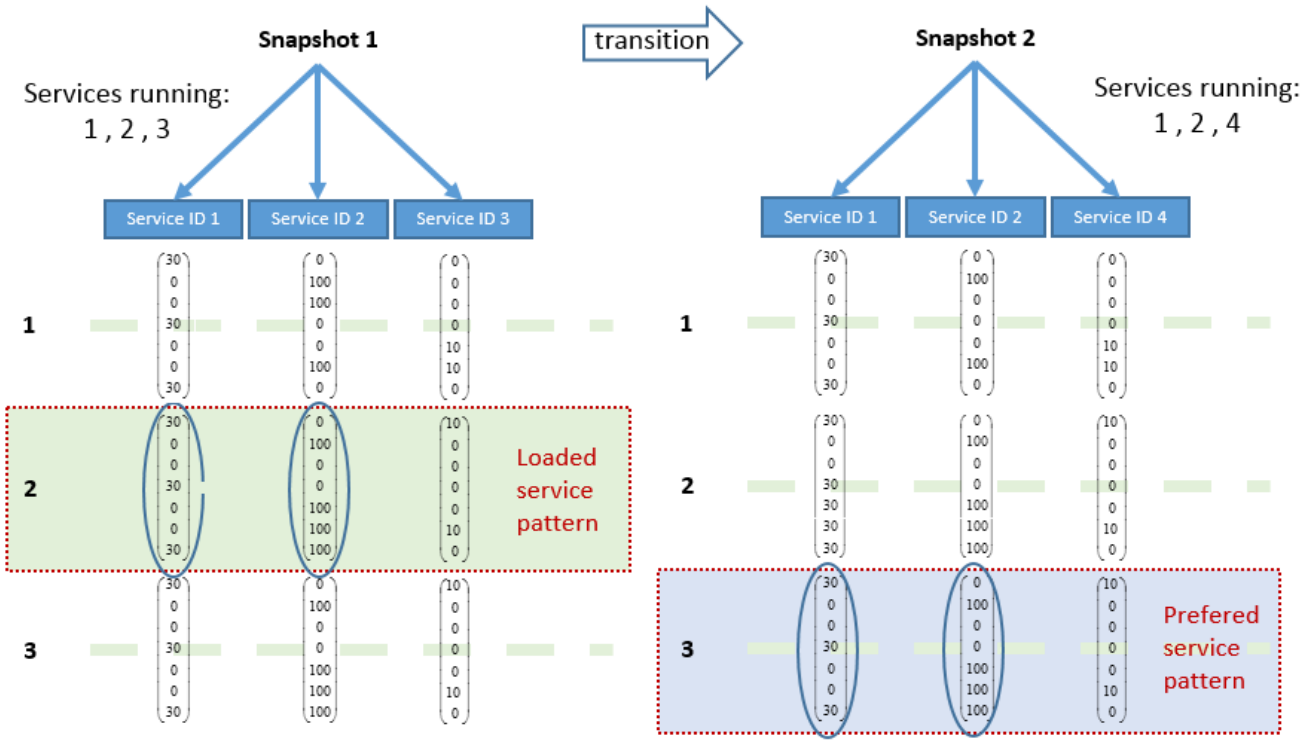

Figure 25: smooth transition mechanism between two contiguous snapshots

6.1.3 Functional charts

6.1.3.1 *Link detection at DynPaC start up*

Link detection is performed when DynPaC boots up. It delays up 1 minute (adjustable) to gather topology information, considering this would be enough time for the discovery process to consider topology as stable. The procedure is graphically described in Figure 26.

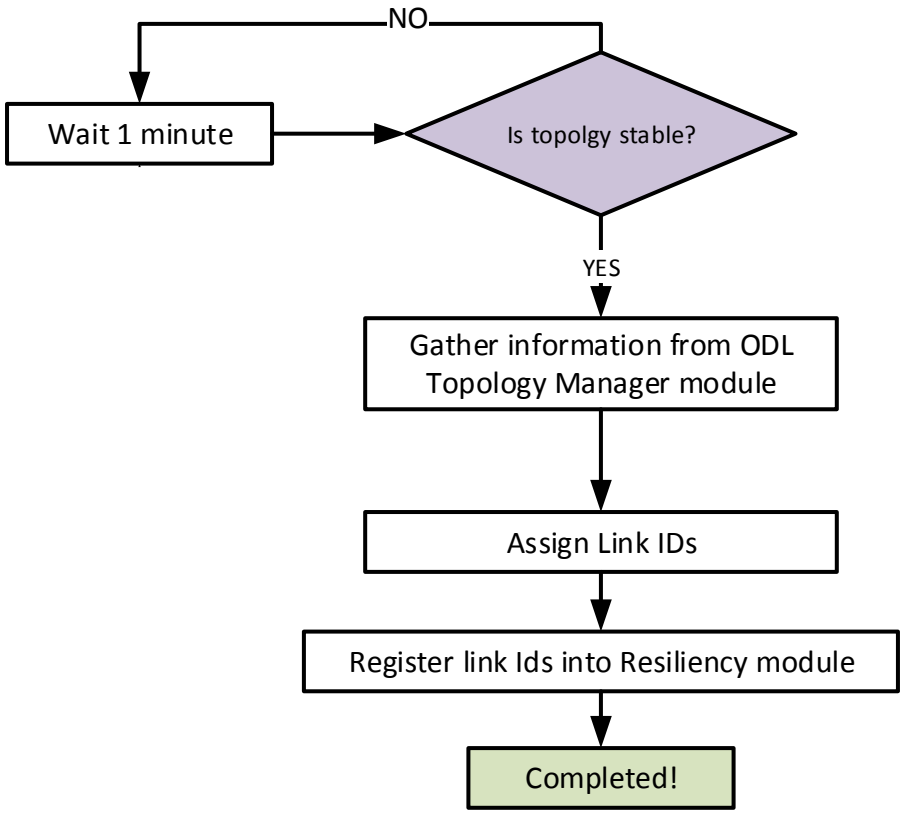

Figure 26: Link detection procedure at DynPaC start up

6.1.3.2 *New service request*

Next, Figure 27 represents the processing flow performed when a new service is requested.

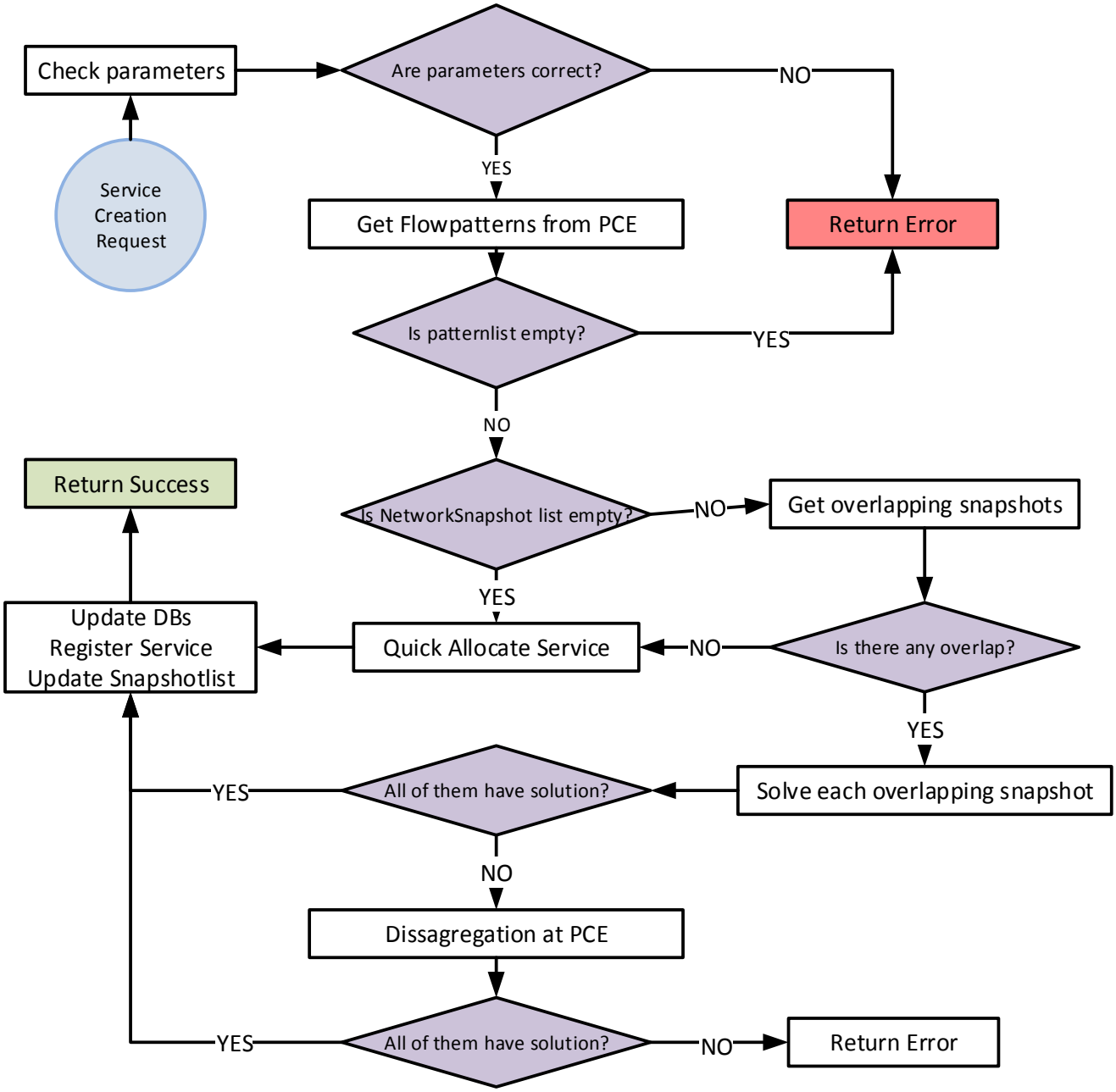

Figure 27: New service request processing flow chart

6.1.3.3 *Existing service deletion request*

Figure 28 shows the procedure followed in order to delete a service in the DynPaC framework.

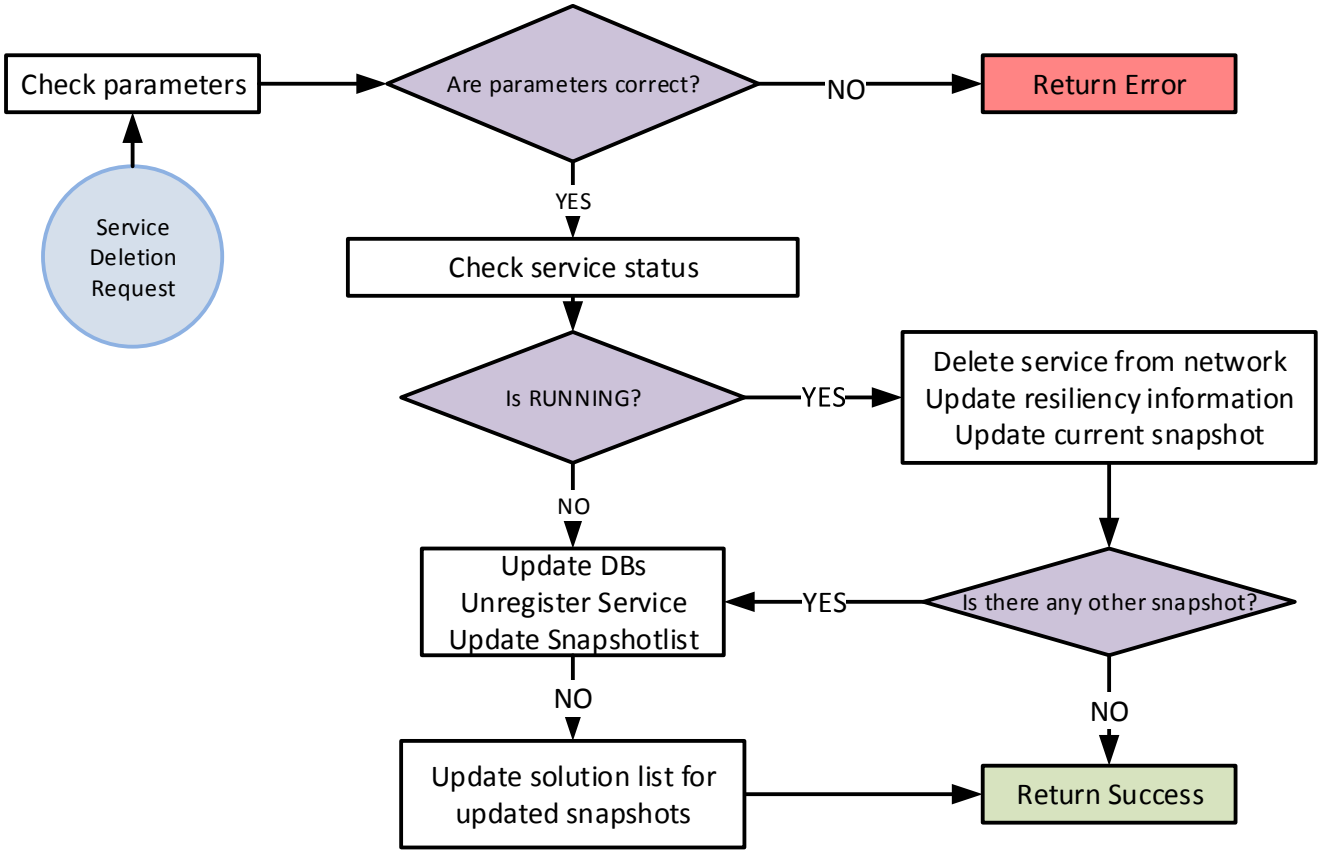

Figure 28: Service deletion request processing flow chart

6.1.3.4 *Snapshot transition*

As described in 6.1.2.5, when a snapshot finishes, DynPaC framework tries to maintain the path of as many of the services that are continuing in the next snapshot as possible, without moving them from the current used paths. The process of unloading and loading of the next snapshot is depicted in Figure 29.

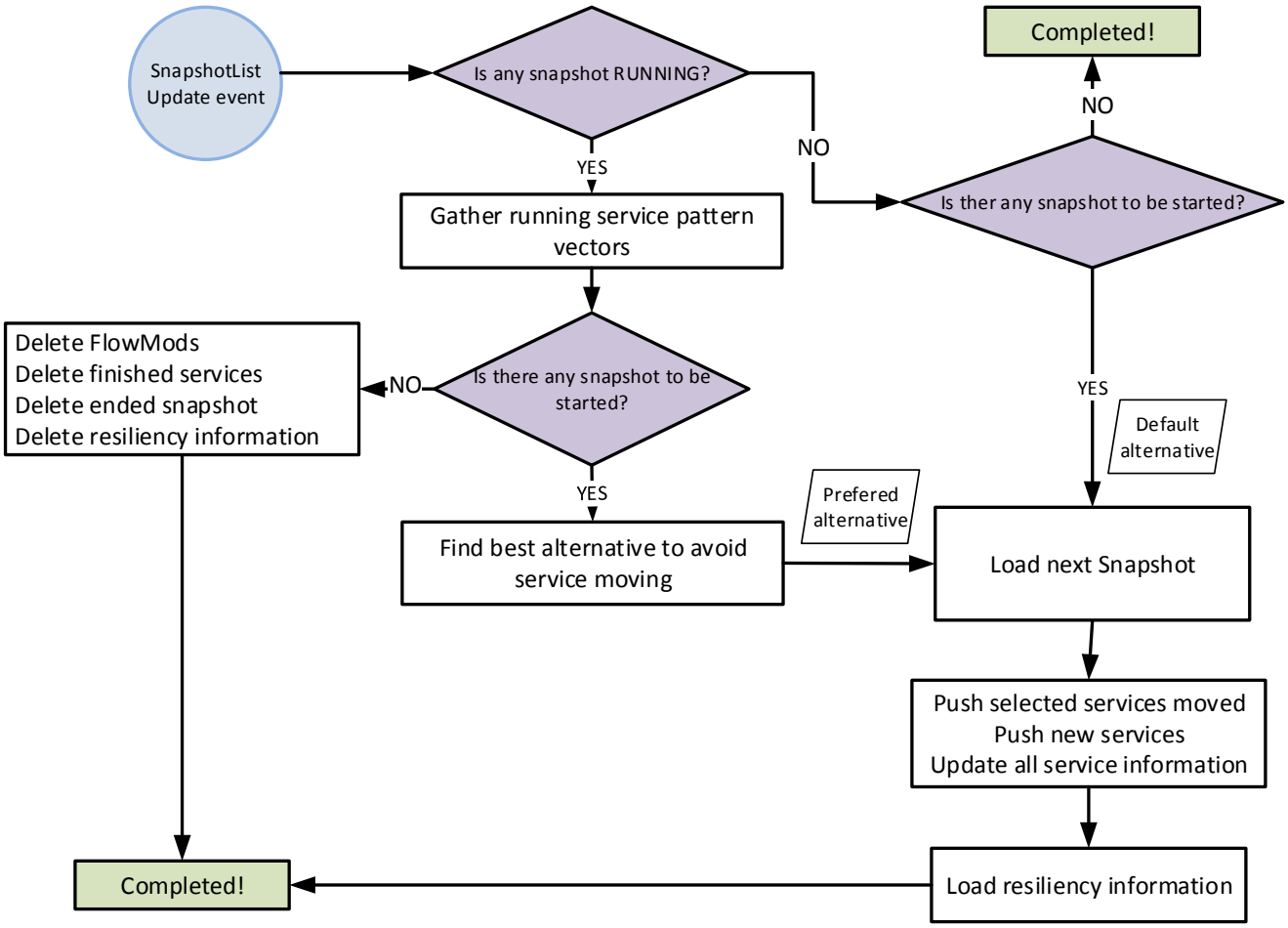

Figure 29: snapshot update process flow chart

6.2 Resiliency

DynPaC Service Manager implements resiliency mechanisms which allow making a fast changeover for services affected by a link failure.

When a service is pushed into the network, DynPaC registers which are the links used by the path selected for that service. In order to achieve this aim, using the cookie of the service as identifier, the cookie is registered in a resiliency database for each used link.

In case of link failure (notification received from switch manager module of ODL), this resiliency database is consulted looking for the failed link identifier. Once gathered affected cookies for that link, recovery process starts pushing backup paths into the network, deleting broken path from network and updating the service database with the current path use. As already mentioned, gold services are always protected against link failures, whereas in regular services, this feature cannot be always guaranteed, and is dependent on the current network usage and the number of concurrent services running for each snapshot.

6.2.1 Loading resiliency feature

When a new snapshot is loaded into the network, all the previous information is deleted and, immediately, all the services loaded with it are registered into the resiliency database, registering used links per each path.

6.2.2 Recovery process

Figure 30 shows the procedure used to implement resiliency features in DynPaC.

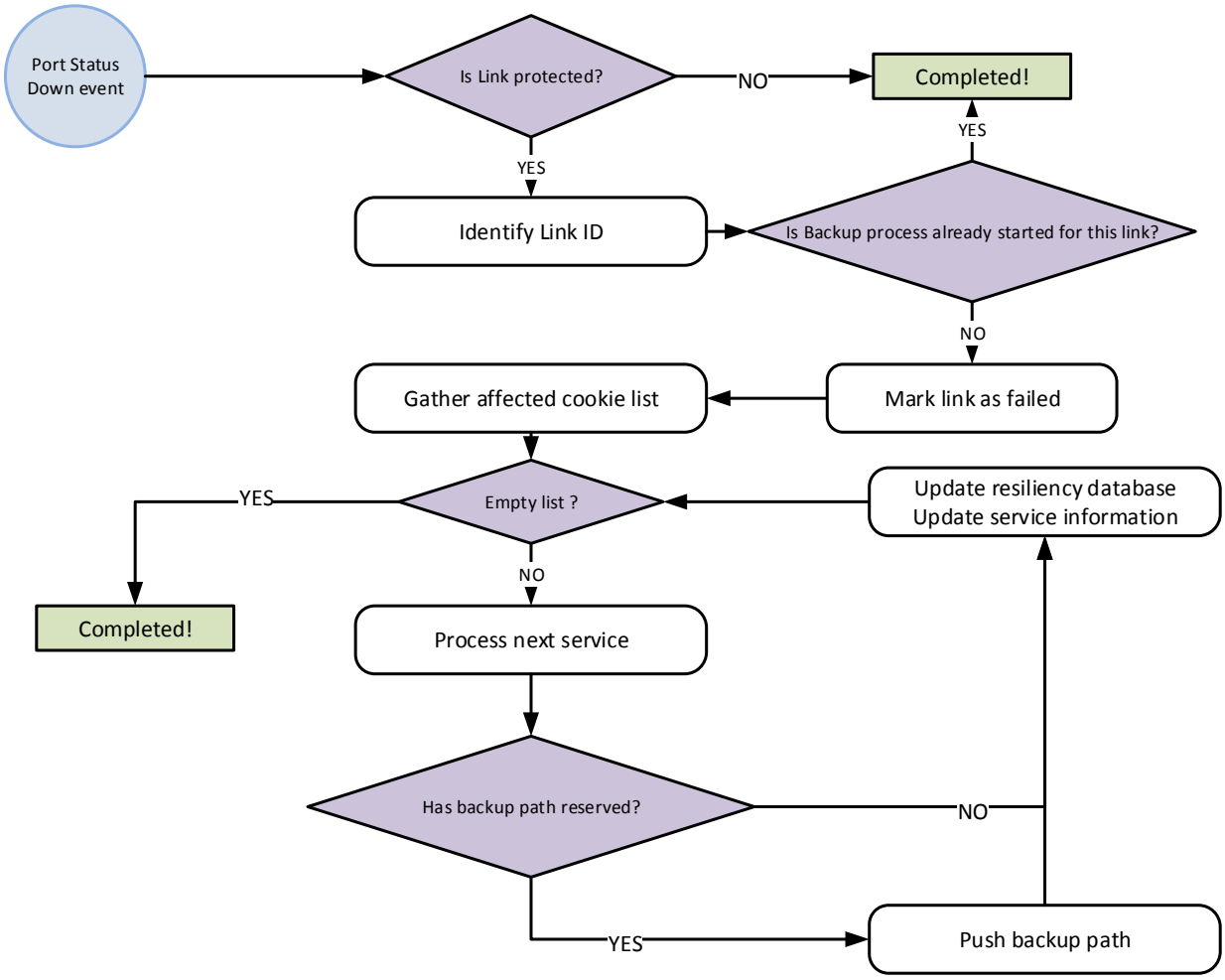

Figure 30: Recovery process flow chart

6.3 Path Computation Element

The PCE module is in charge of most of the functionalities related to the path computation. It has three main functionalities:

- It builds a set of all the possible paths for each ingress/egress node pair.
- Taking into account the network resources' utilization at a given time, it offers the shortest path between a given ingress/egress node pair for the requested bandwidth.
- It computes a disaggregation context provided by the DynPaC core, trying to find a solution by disaggregating one or more of the flows already allocated.

Figure 31 represents the PCE architecture:

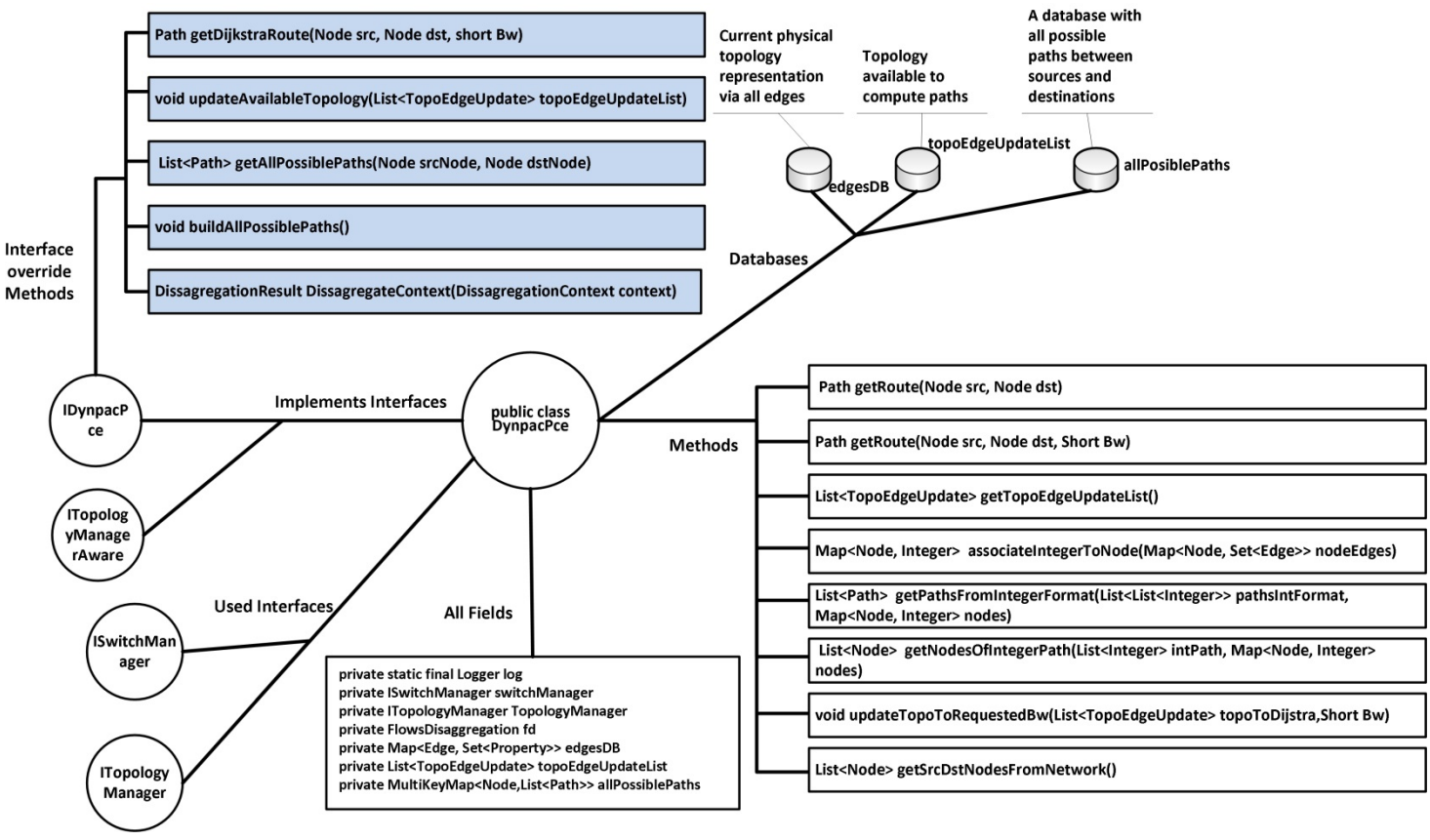

Figure 31 DynPac PCE module design

In order to gather the information required by this module, the PCE interacts with the Topology Manager and switch manager from ODL. Making use of the *ITopologyManager* interface, the DynPaC PCE can obtain topology information, whereas using *ISwitchManager* interface, the PCE can collect some information about nodes and links.

Figure 32 represents this interaction between modules.

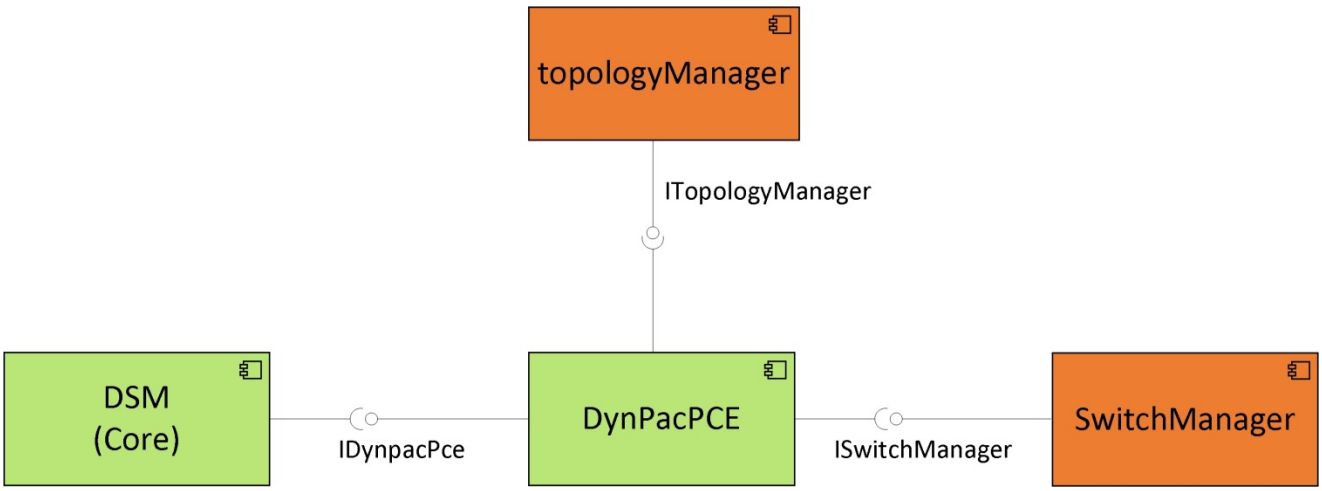

Figure 32: Interfaces between the DynPaC PCE module and other modules

6.3.1 IDynpacPce interface

IDynpacPce is the interface exposed by the PCE module to allow other modules to interact with it. The methods exposed through this interface are graphically represented in Figure 33 and detailed below.

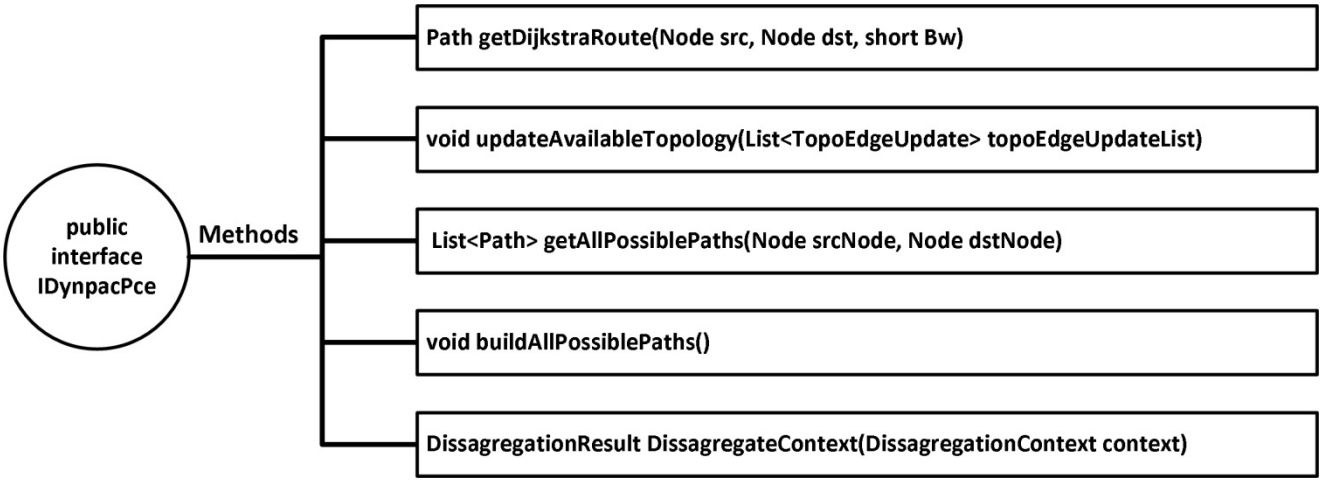

Figure 33: DynpacPce Interface

- ***getDijkstraRoute***: given a pair of nodes, this method returns the shortest path in terms of hop count, applying the Dijkstra algorithm from the ODL routing module.
- ***updateAvailableTopology*****:** this method updates PCE topology database for a certain context.
- ***getAllPossiblePaths*****:** given a pair of nodes, this method returns a list of potential paths available connecting those nodes.
- ***buildAllPossiblePaths*****:** invoking this method, the database containing all the possible paths is refreshed and updated, regenerating all the potential paths between each pair of nodes present in the network.
- ***DissagregationResult*****:** for a given disaggregation context, this method tries to obtain a solution considering the disaggregation of any of the services already present in the context.

6.3.2 Functional overview

6.3.2.1 *Current topology database update*

The PCE topology database must always be kept updated in order to obtain the correct information when gathering information required to perform some path computation. This database is updated from the *IDynpacPce* interface.

Figure 34 depicts the flow chart corresponding to the process of current topology database update.

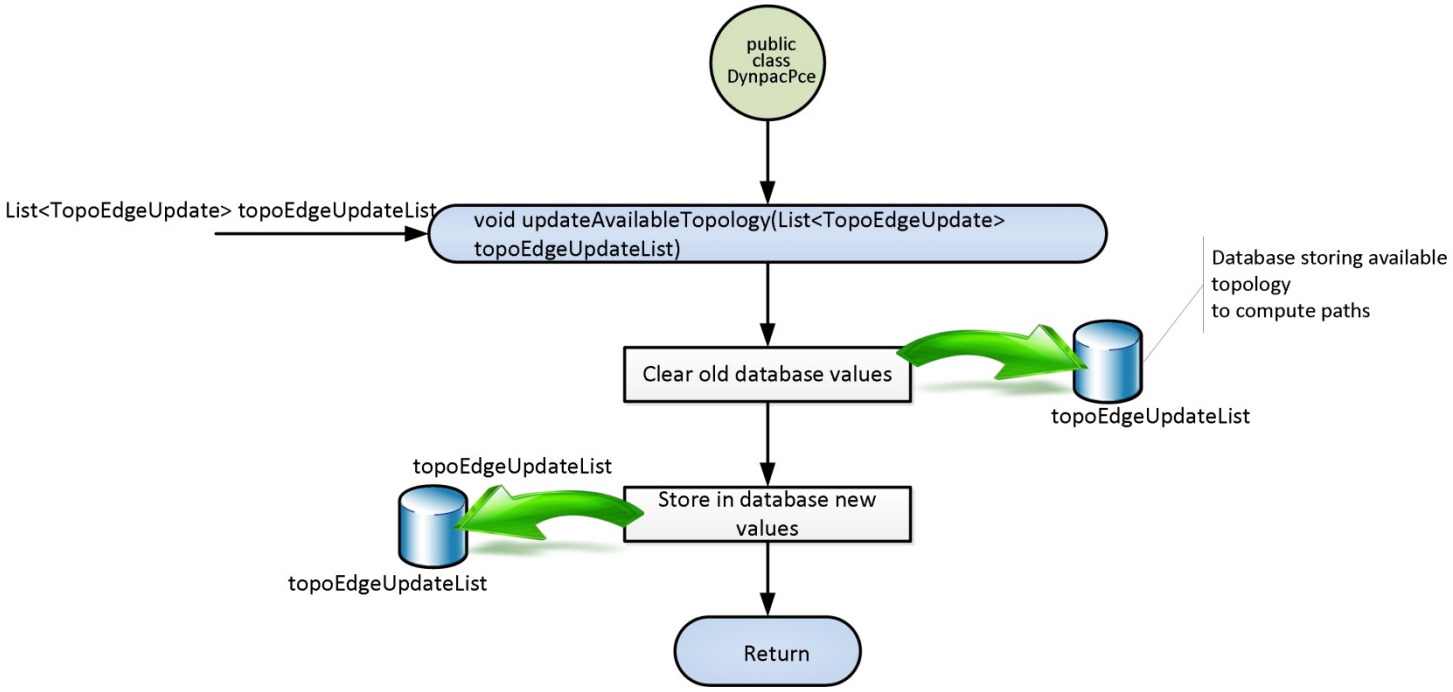

Figure 34: Flow diagram of update process of available topology database

6.3.2.2 *Obtaining all potential paths between 2 different nodes*

The *GetAllPossiblePaths* method is used to obtain a list of all the possible paths between two different nodes, if any path exists. In cases of no path available, an empty list will be returned.

The request is performed according to the flow chart depicted in Figure 35.

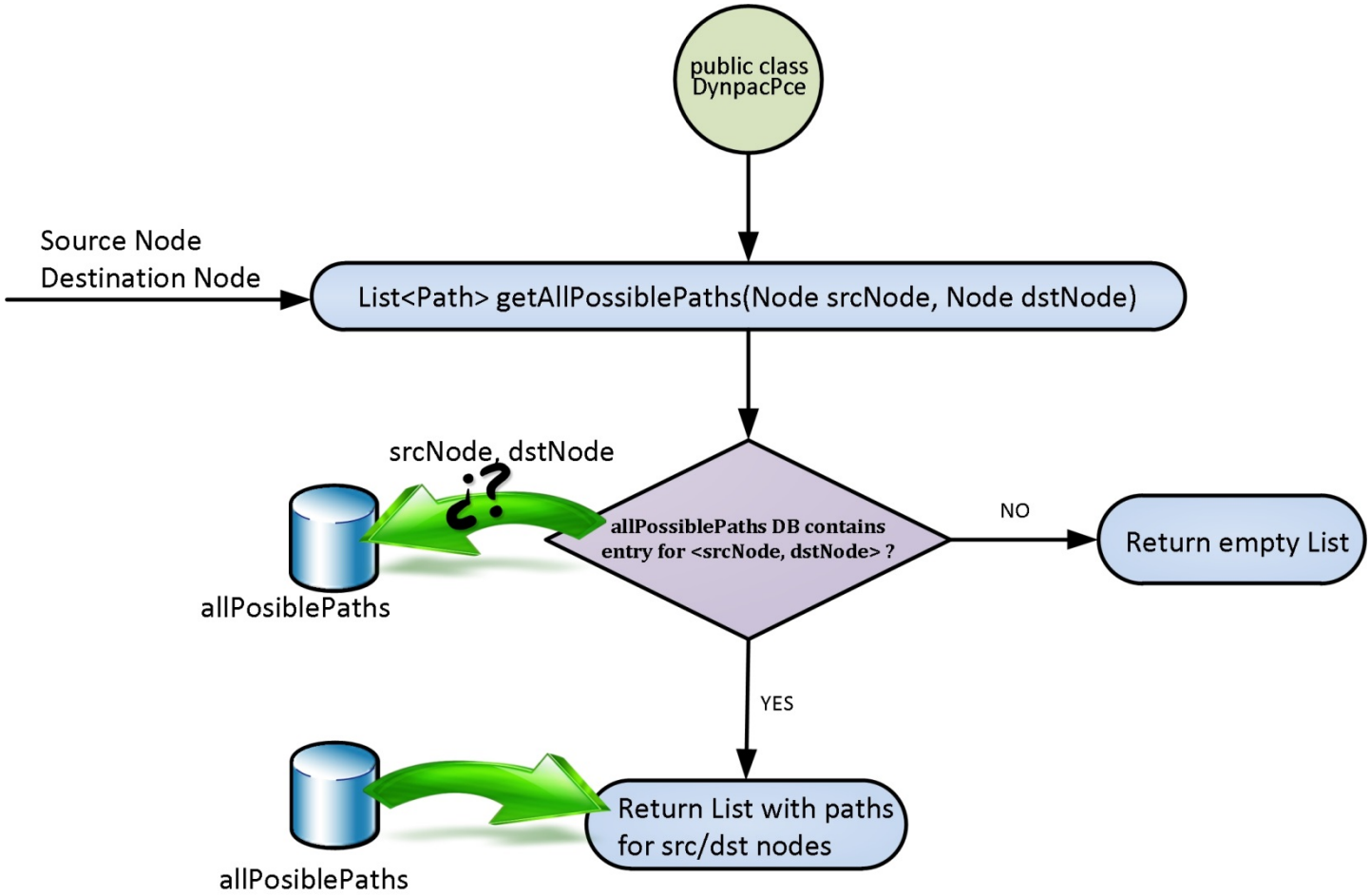

Figure 35 Flow diagram of request all possible paths between two nodes

The algorithm implemented to get all the possible paths is described in Figure 36:

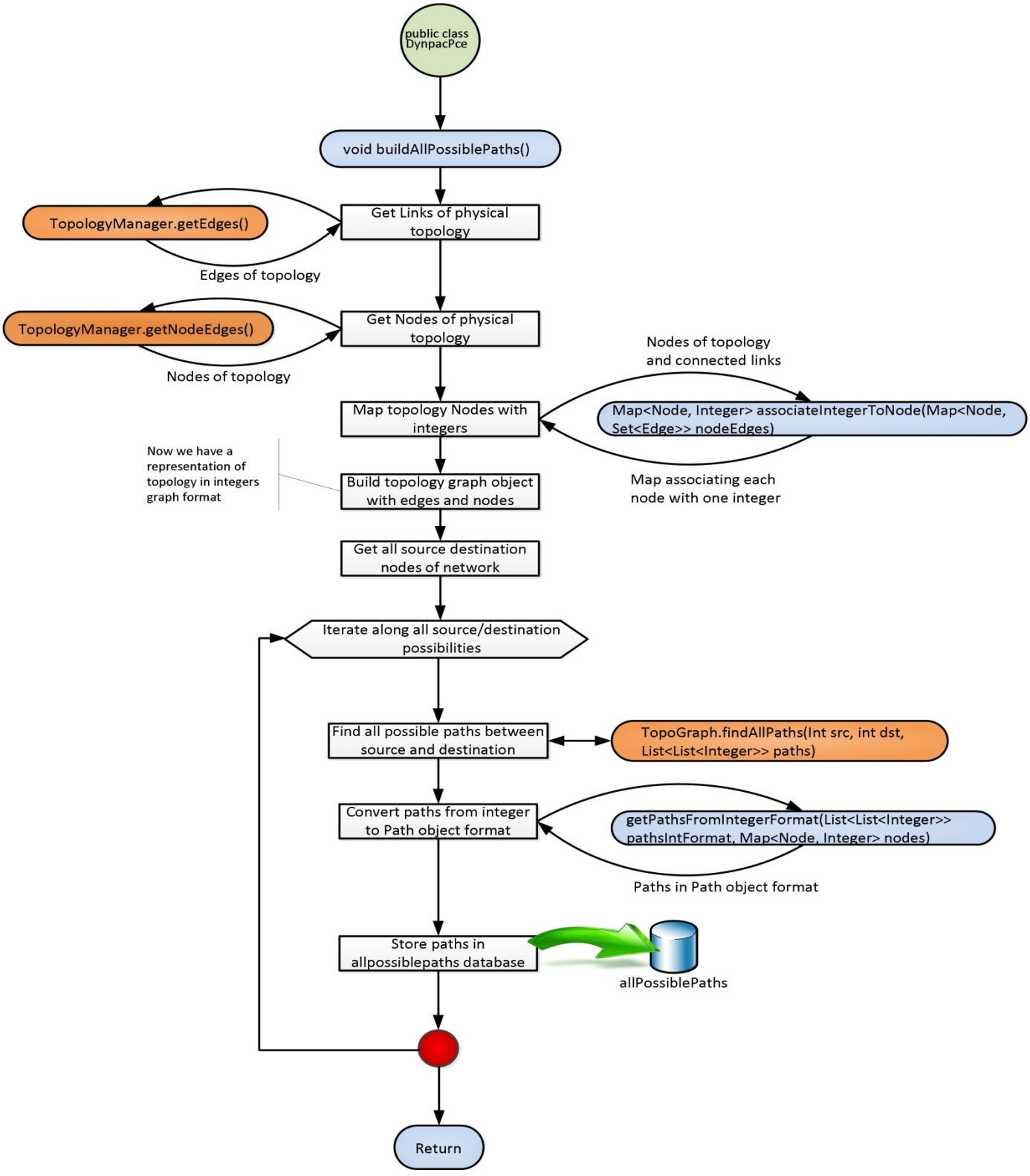

Figure 36: Flow diagram of the algorithm implemented to compute all the possible paths

As described in Figure 36, firstly the complete network information is collected from the Topology Manager of ODL. After that, each node is assigned a numerical identifier to simplify the handling of the nodes by the PCE, easing the calculus performed by the PCE. Once the topology has been processed, all information about node ports is processed specifying if they are internal or external ports.

Then, all the node pairs that can be formed between network available nodes are computed and for each of these pairs all potential paths are computed. These paths are stored in a database containing all possible paths. This computation is done making use of *Topograph* class. To work with this class, information is adapted to it. The algorithm used is a modified version of *DepthFirstSearch* algorithm.

6.3.2.3 *A route computation with Dijkstra algorithm*

The PCE can compute a route between 2 different nodes considering available bandwidth in the network and also, the amount of bandwidth requested. The PCE builds an available topology map with the current occupation conditions of the network, according to the lifetime of the requested service. Then, applying Dijkstra implementation from ODL, the PCE is able to provide the shortest path route, also considering the requested bandwidth. This process is depicted in the Figures 37 and 38.

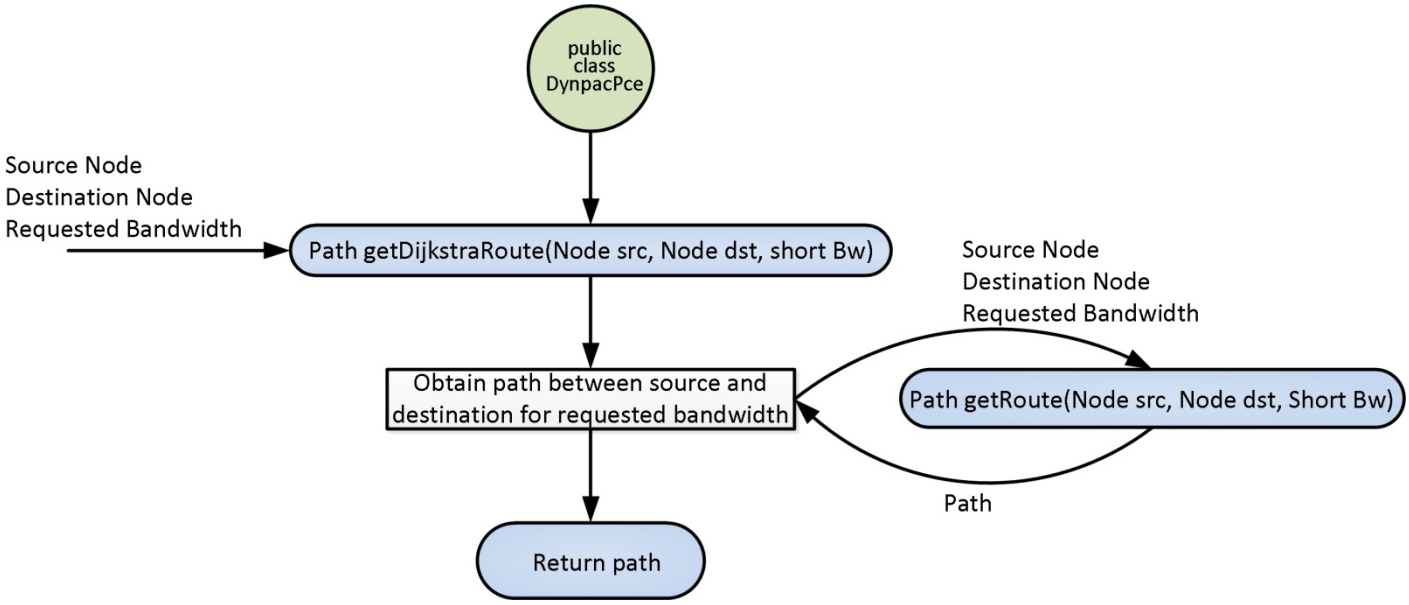

Figure 37: Flow diagram of path computation using Dijkstra

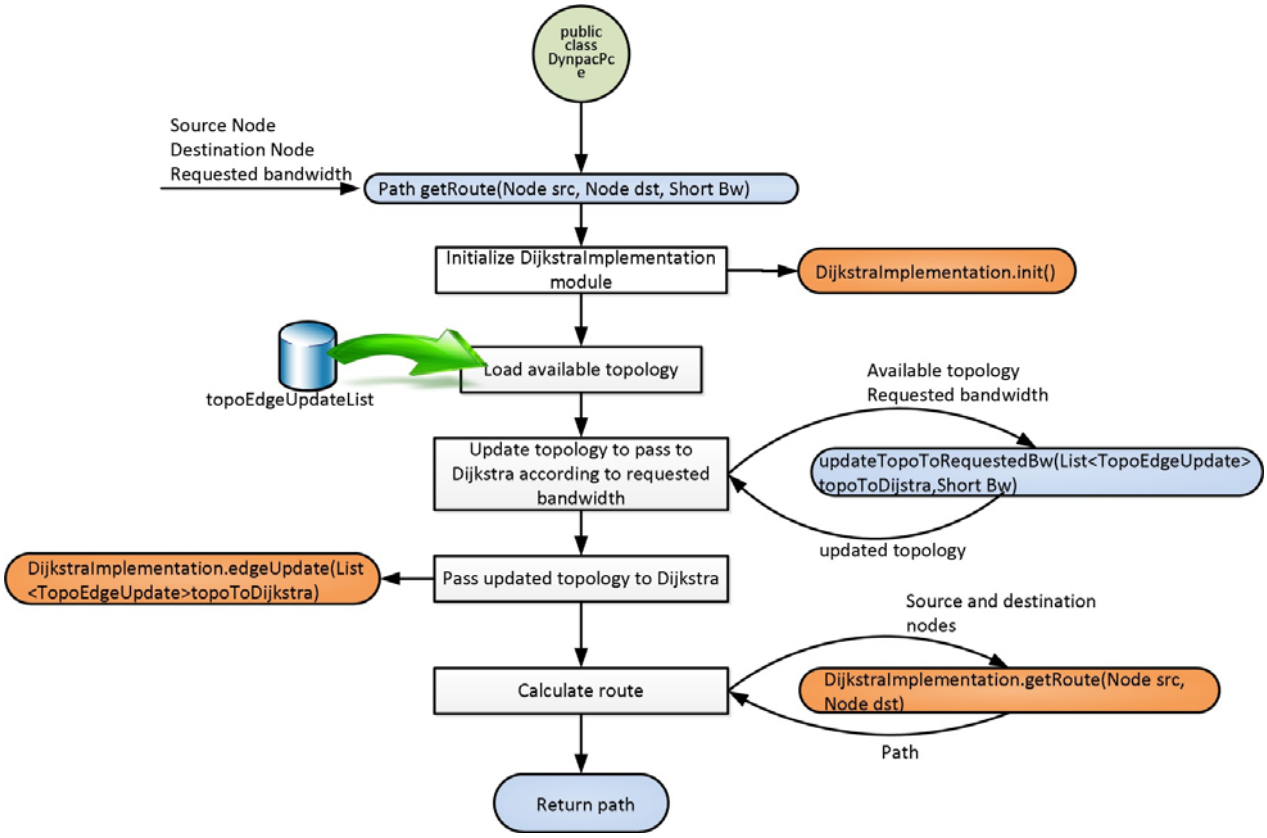

Figure 38: Flow diagram of route computation

As described in the previous figures, the network occupation map is obtained from the *topoEdgeUpdateList* database containing the available topology. Due to this fact, it is important to maintain this database permanently updated with the context to be computed. Considering the information obtained from this database and available bandwidth information, the PCE builds a suitable topology to provide to the Dijkstra module. This topology is used by the Dijkstra module to perform the path computation. Finally, if a solution is found to the provided context, a path is returned as a result of the computation.

6.3.2.4 *Disaggregation feature*

When the DynPac Service Manager is not able to find a solution for a certain snapshot, as no enough free resources are available, a disaggregation context is built by the DSM and given to the PCE, looking for an advanced resolution mechanism based on sub flow splitting of the original services running. The PCE uses the *FlowsDisaggregation* class to perform disaggregation computation. Further details about the implementation of the disaggregation functionality are given in the next subsections.

6.3.2.5.1 FlowsDisaggregation class

This class represents the core of the disaggregation algorithm. *TryDisagreggation* method starts the processing of the flow reallocation. Figure 39 represents the schema of FlowsDisaggregation class.

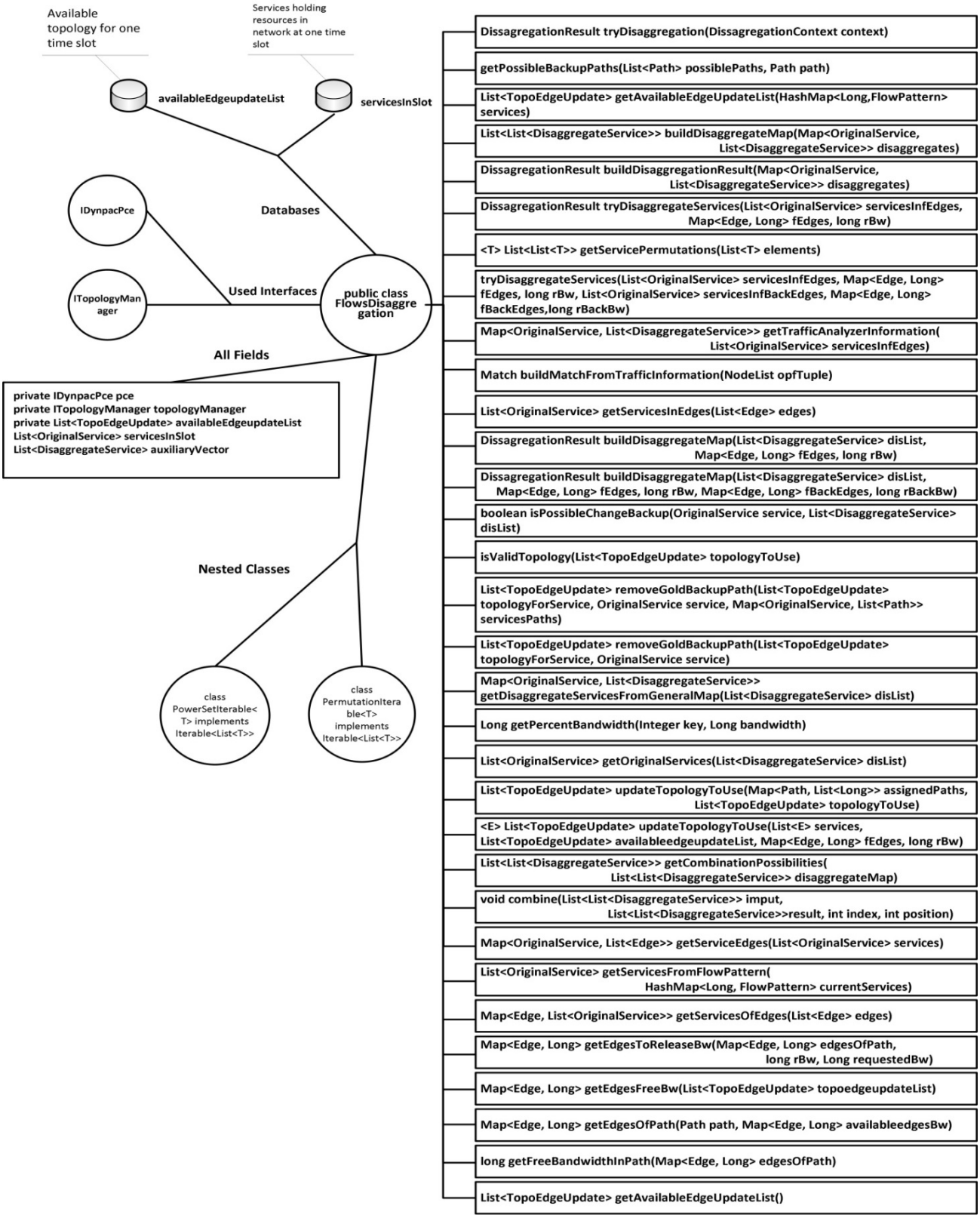

Figure 39: FlowsDisaggregation class schema

6.3.2.5.2 Service disaggregation

The procedure followed by the PCE to solve a disaggregation context is depicted in Figure 40.

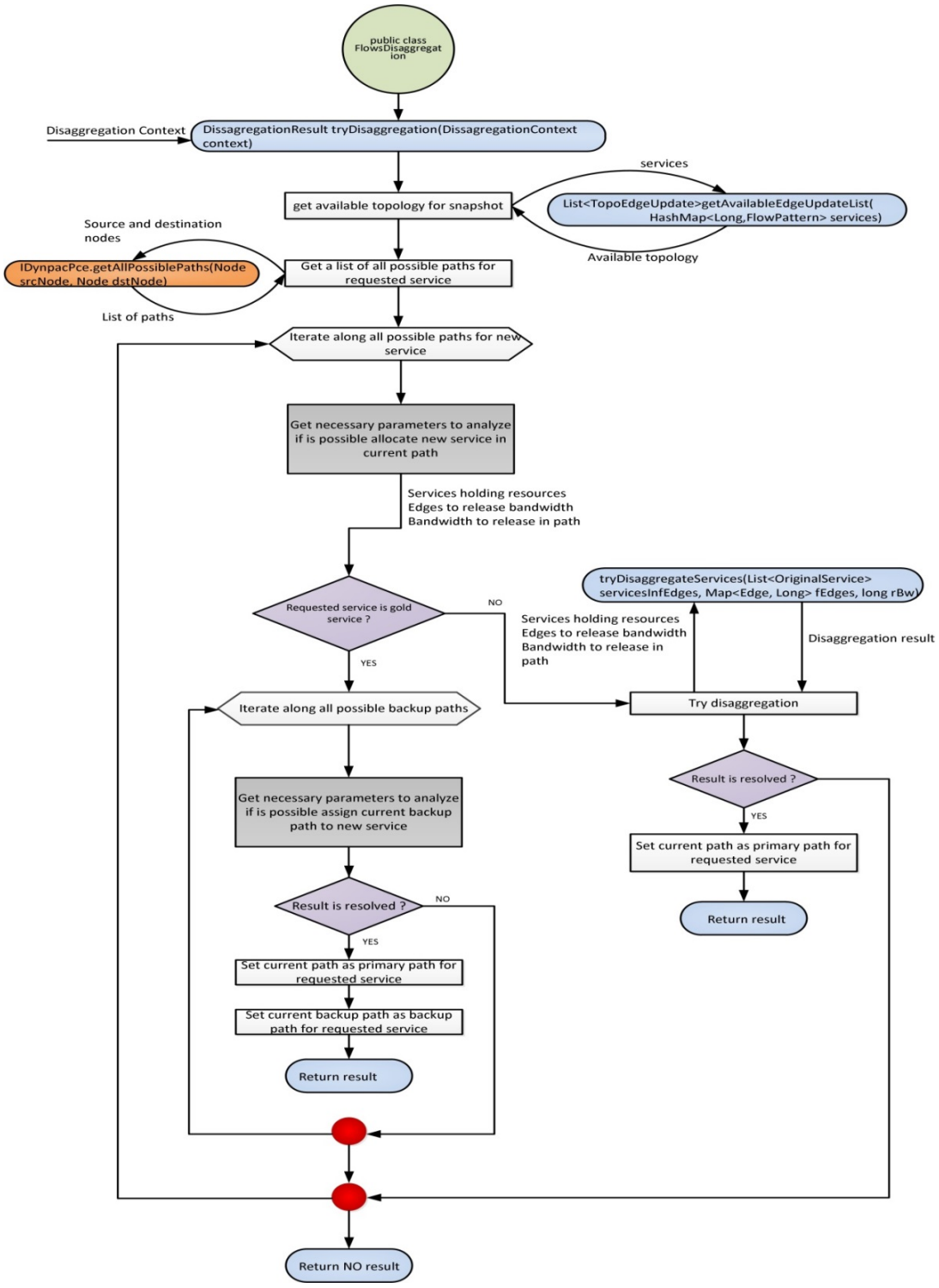

Figure 40: Flow diagram of the disaggregation process

The procedure used to obtain all the necessary parameters to evaluate if it is possible to allocate the new service in the path under evaluation is given in Figure 41.

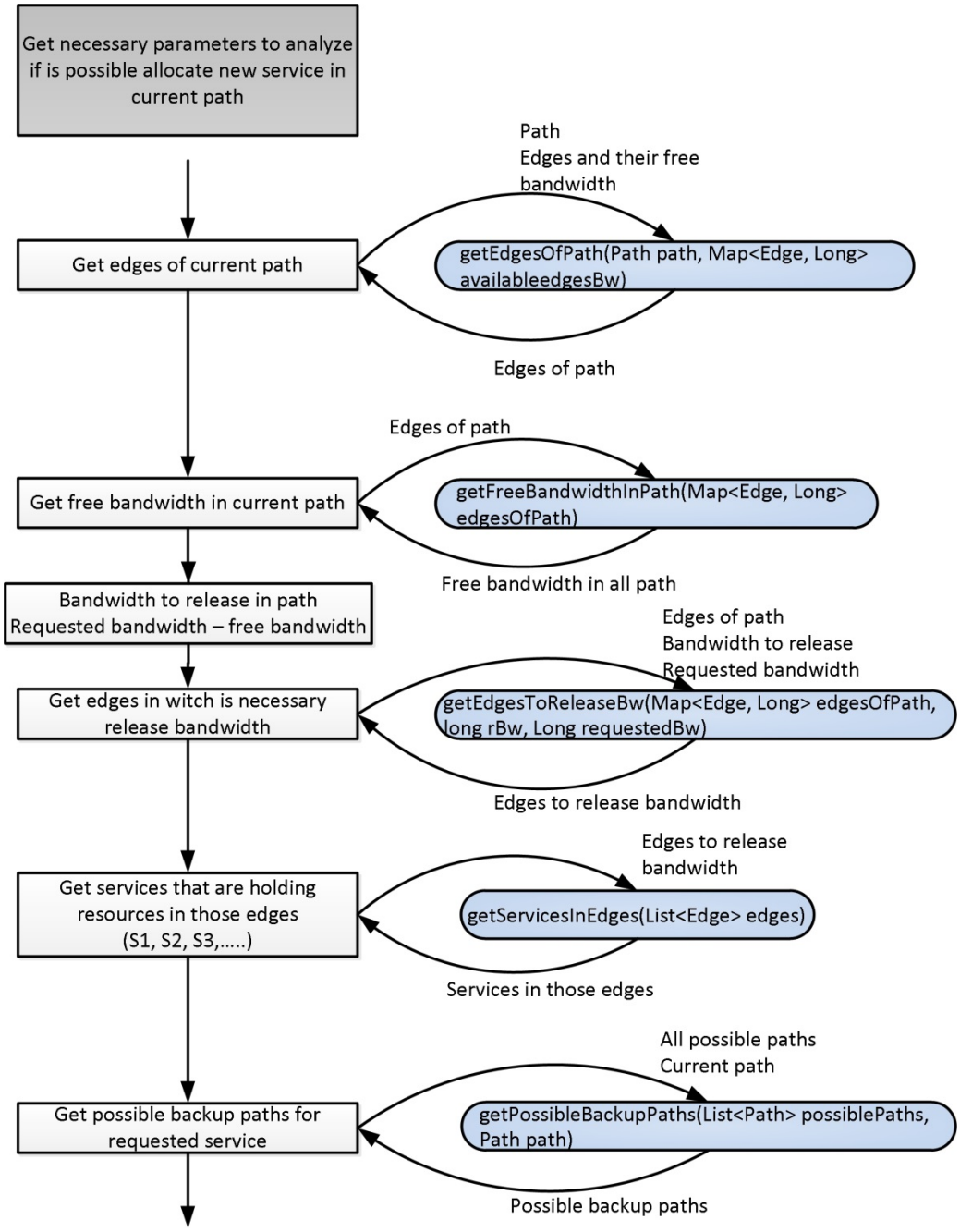

Figure 41: Flow diagram of the procedure used to get all the necessary parameters for path evaluation

In a similar way, the flow diagram corresponding to the algorithm implemented to get all the necessary parameters to analyze if it is possible to assign current backup path under evaluation to the new service is described in Figure 42.

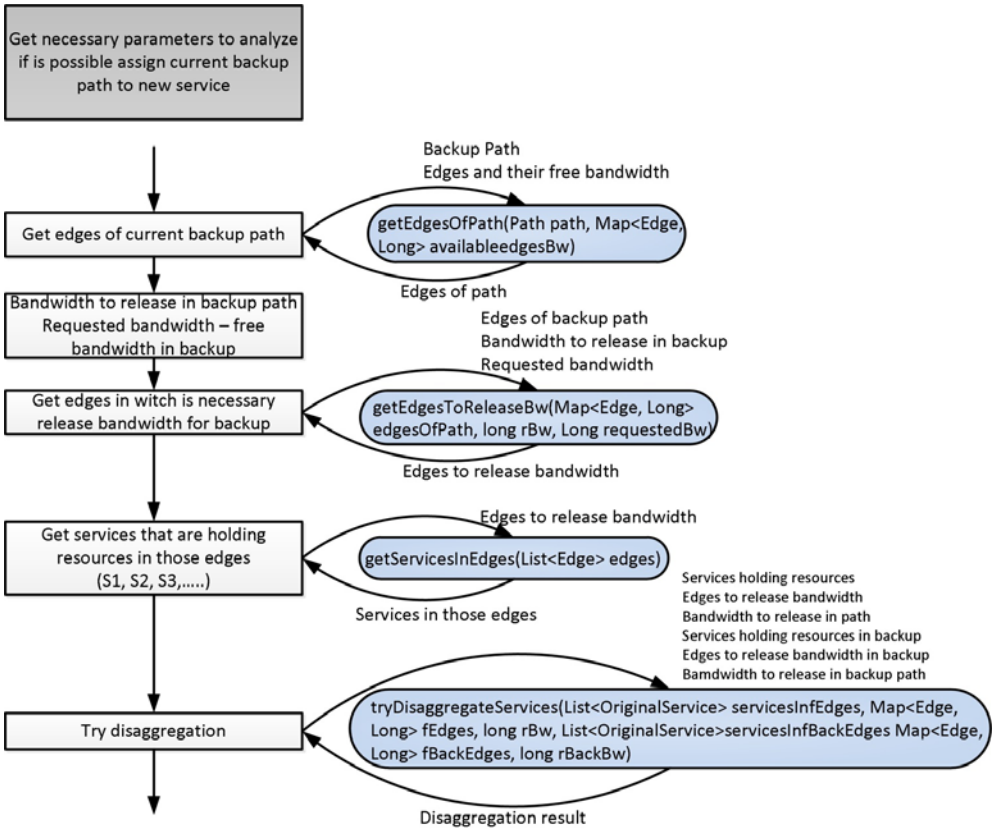

Figure 42: Flow diagram of the procedure used to get all the necessary parameters for backup path evaluation

As graphically shown in the previous figures, firstly, all potentials paths are obtained for the new service, and then, all of them are checked to filter which are suitable to allocate the new service and which are not. For each path, considering network occupation and the requested bandwidth, the bandwidth to be freed in each link is determined. Then, the services that are occupying resources in the affected links are identified. Once the required resources are identified for the path chosen by the algorithm, a new potential path map is built for the new service to be allocated. In case of a gold service, the same computation is made for the backup path computation.

Once all the required results have been gathered, the disaggregation function is called to obtain a solution. To this function, information about services occupying bandwidth over the selected path and the amount of bandwidth to be freed are passed as parameters. If a solution is found, this is returned, otherwise, the same algorithm is executed for the next path in the list.

6.3.2.5.3 Disaggregation function

When a path, or a pair of paths in the case of gold services, is considered as suitable for disaggregation, information about them is requested to the Traffic Pattern Analyzer module. The interface with this external module is described in section *6.6 Traffic Pattern Analyzer*. The Traffic Pattern Analyzer returns disaggregation options available for the services about which it is queried.

The Traffic Pattern Analyzer checks whether it has information about any of the services to be disaggregated, returning for each service for which it has information, the profiles generated for it. Taking as a starting point this information, the algorithm builds all the possible existing combinations among the different

disaggregation patterns of the different services, and for each possible combination, all the possible permutations among the elements of that combination are computed. Each of these permutations is in fact a possible disaggregation map. Finally, considering all the potential disaggregation maps, the first that is capable of allocating the new service is chosen. Figure 43 graphically describes the disaggregation process.

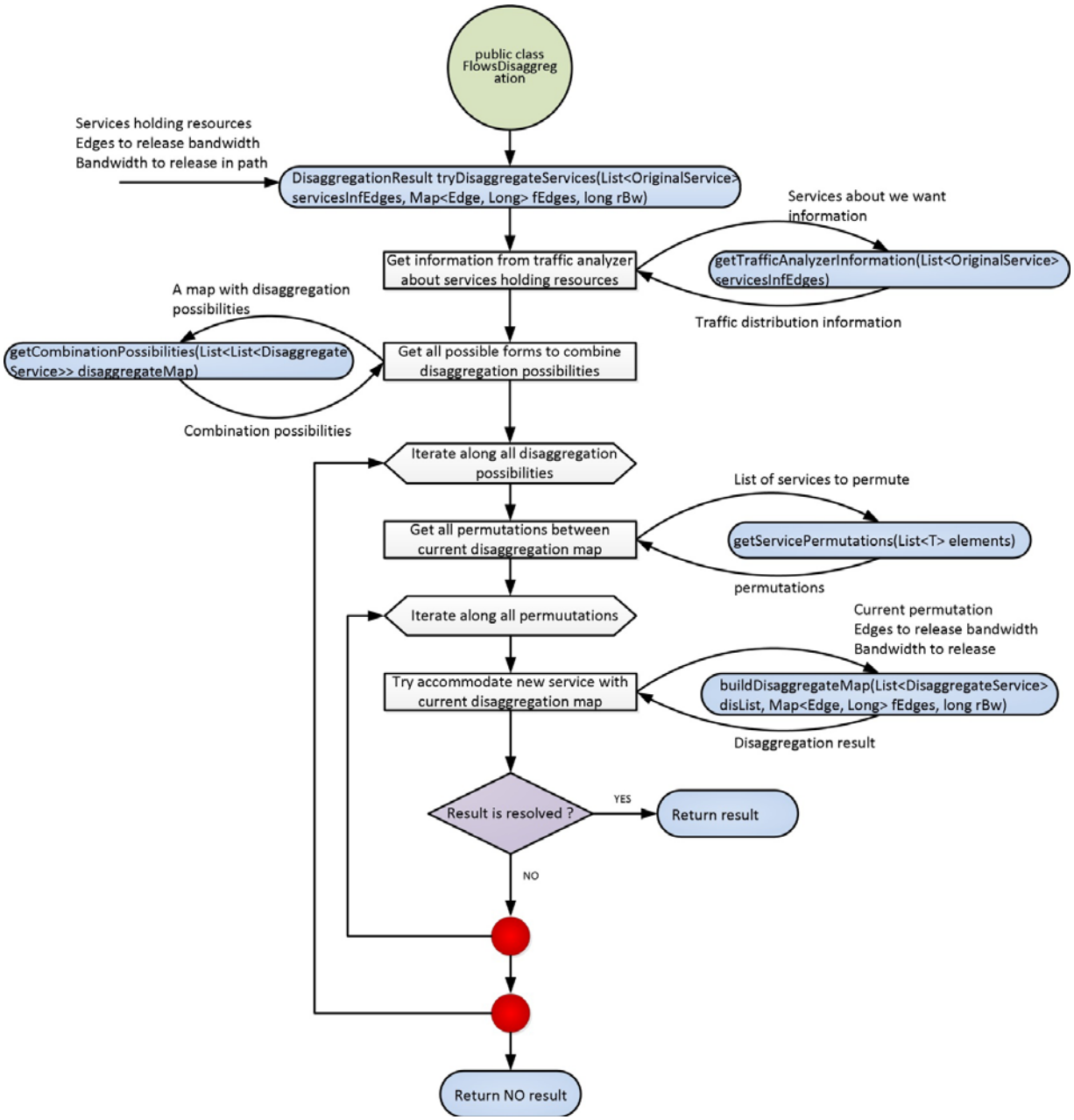

Figure 43: Flow diagram of try disaggregate process

6.3.2.5.4 Adding new service to the resolved context

In order to effectively add a new service, the DynPaC framework attempts to accommodate the newly requested service in each disaggregation map described in the previous section. With this aim, the DynPaC framework takes as a starting point a topology where the newly requested service is already accommodated in the network (taking up bandwidth in the links corresponding to the selected path) and the services included in the current disaggregation map have been removed. Then, it tries to accommodate the remaining services in

the previous topology according to the disaggregation map. Figure 44 graphically represents the described procedure.

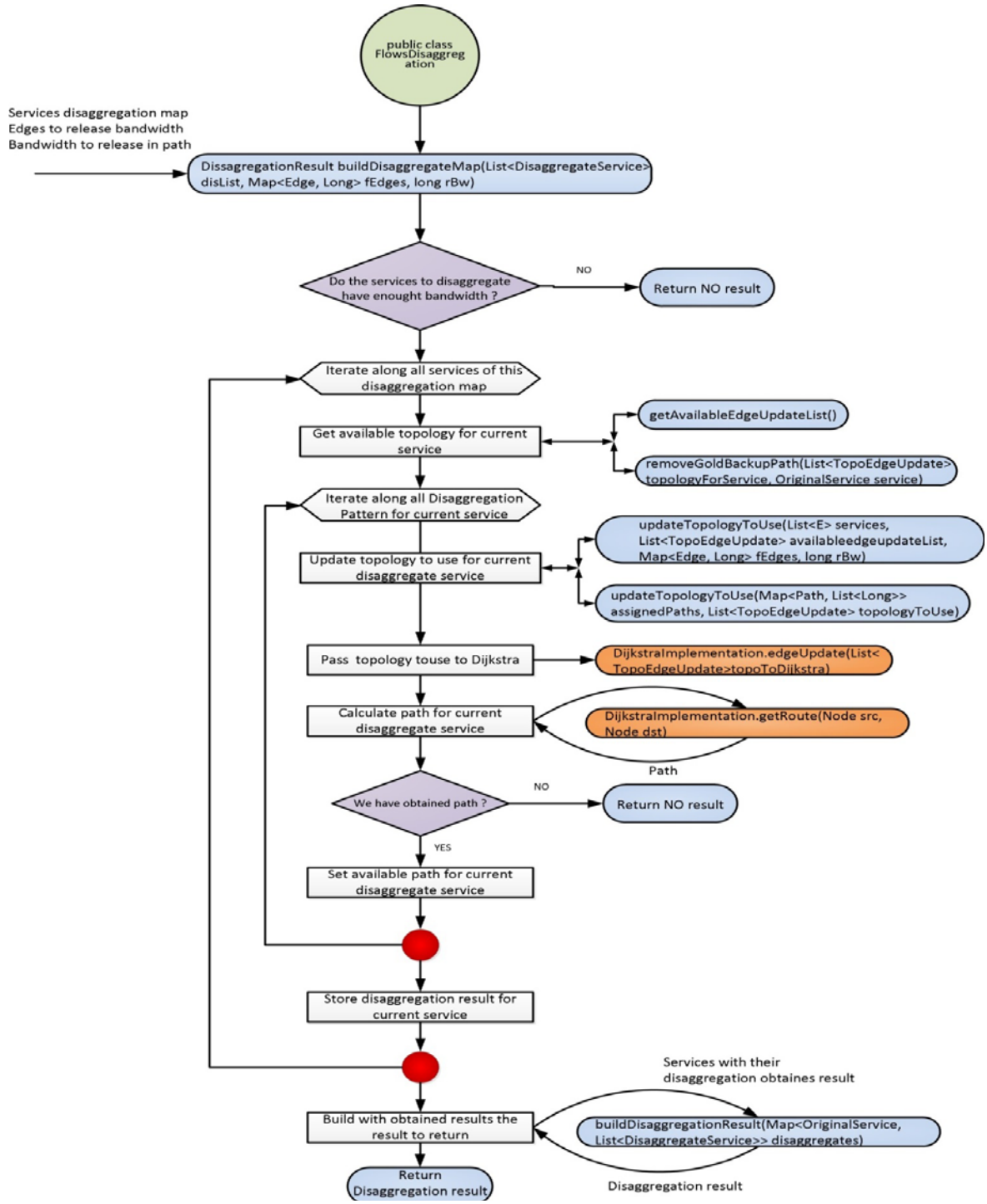

Figure 44: Flow diagram of build disaggregate map

6.4 Monitoring

6.4.1 General description

The DynPaC monitoring module is in charge of monitoring the bandwidth occupied by the DynPaC services. This module keeps track of the flows installed on the network nodes and allows to identify the services that are exceeding their maximum allowed bandwidth.

The module uses OpenFlow statistics to compute the bandwidth occupied by the DynPaC services. When the bandwidth occupied by a service exceeds a configured threshold, it communicates with the DynPaC Service Manager to inform about the event. The DynPaC monitoring module uses two databases: *trackedFlowsDB* and *riskyFlowsDB*. The first one is used to monitor the flows that are not considered problematic, whereas the second one is used to monitor problematic flows, that is, those flows that are near the maximum allowed bandwidth.

Firstly, services are installed in the *trackedFlowsDB* and they are moved to the *riskyFlowsDB* when the measured bandwidth exceeds a certain percentage of the maximum reserved bandwidth, which is specified by the RISK_BW_THRESHOLD. Similarly, when the measured bandwidth reaches a percentage specified by the MAXIMUM_BW_THRESHOLD, the DynPaC monitoring module informs the DynPaC Service Manager about it.

The main difference between the two databases is the refreshing frequency, which in the case of the *riskyFlowDB*, is 10 times higher. This decision has been taken in order to obtain more accurate bandwidth measurements just when the bandwidth is reaching its maximum. This approach allows minimizing the necessary control traffic to obtain the statistics. Figure 45 depicts the design of the monitoring module.

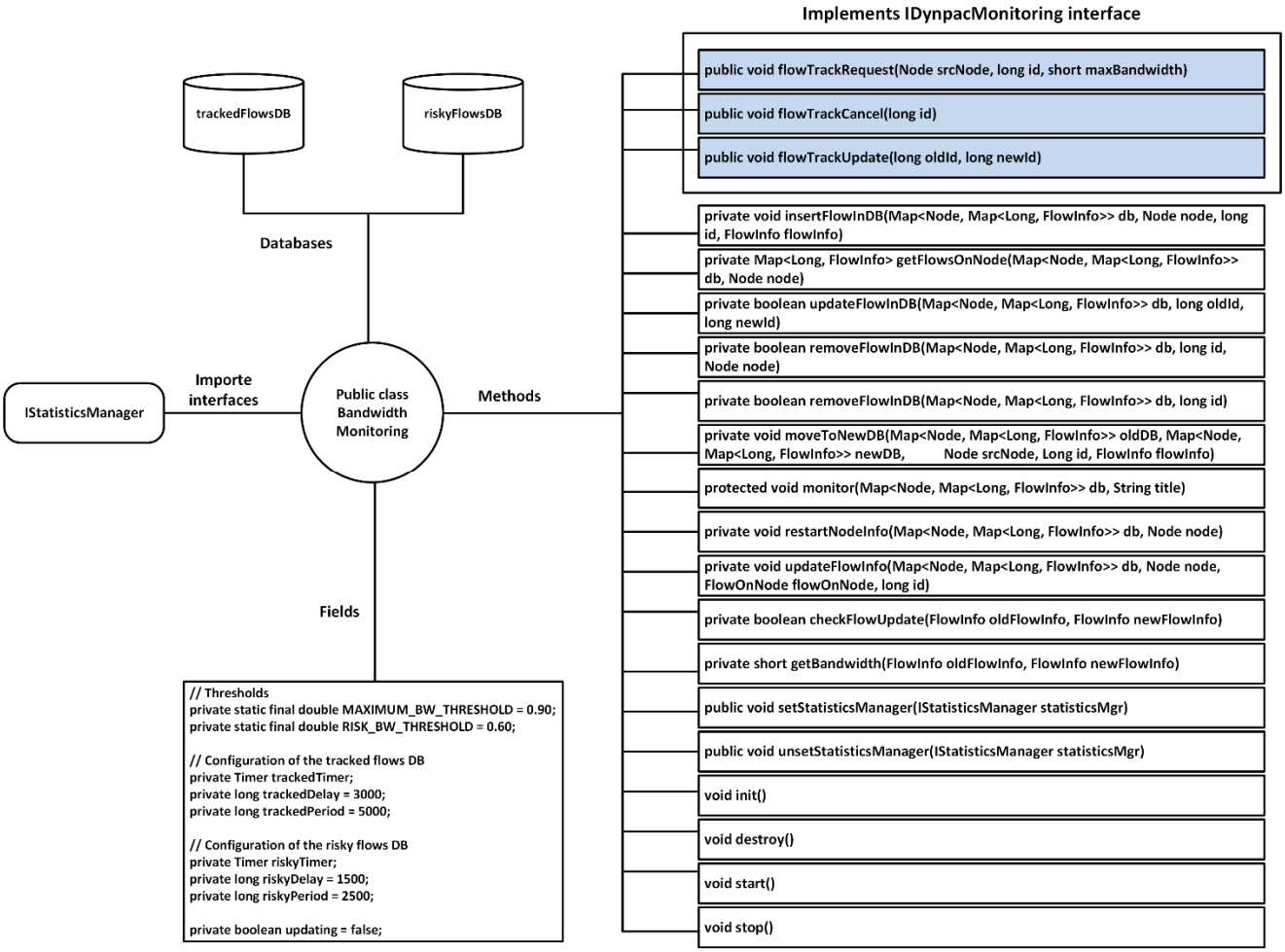

Figure 45: DynPaC monitoring module design

6.4.2 Interfaces with other modules

On the one hand, the DynPaC monitoring module uses OpenFlow statistical information to compute the bandwidth occupied by the services. As a consequence, the DynPaC monitoring module must communicate with the Statistics Manager, which is part of the Open Daylight Project.

The Statistics Manager exposes the *IStatisticsManager* interface, which is consumed by the DynPaC monitoring module. This interface exposes a set of methods that allow requesting statistical information about the installed flows, the node connectors or the flow tables in a given node. In the context of the DynPaC framework, it has been used to gather the following statistical information per flow installed in a node:

- Duration in seconds.
- Duration in nanoseconds.
- Packet count.
- Byte count.

On the other hand, the DynPaC monitoring module has been designed to be easily extended with new monitoring capabilities. It exposes an interface that the DynPaC Service Manager uses to add or remove services from the monitoring databases. For the moment, the only parameter that it is being measured is the bandwidth, but in the future this module could be extended to support the measurement of different parameters.

Through the *IDynpacMonitoring* interface the following methods are exposed:

- *flowTrackRequest*: used to add a new service in the slow monitoring database.
- *flowTrackUpdate*: updates the cookie of a flow in the slow and fast monitoring databases.
- *flowTrackCancel*: deletes the specified service in the slow and fast monitoring databases.

Figure 46 provides a graphical representation of the interfaces exposed by the DynPaC monitoring module.

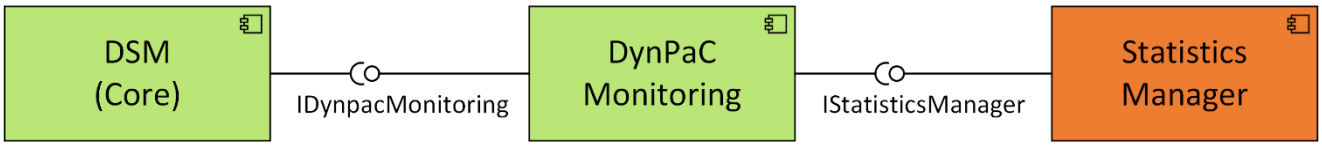

Figure 46: Interfaces between the DynPaC monitoring module and other modules.

6.4.3 Functional overview

This section provides a functional overview of the DynPaC monitoring module. It provides information about the following cases:

- Add a service to the monitoring database.
- Remove a service from the monitoring database.
- Update the ID of a service in the monitoring database.
- Flow monitoring.

6.4.3.1 *Add service to the monitoring database*

When the DynPaC Service Manager programs the network devices, it communicates with the monitoring module to add the service in the monitoring database. The flow diagram of how a new service is added in the *trackedFlowsDB* database is depicted in Figure 47.

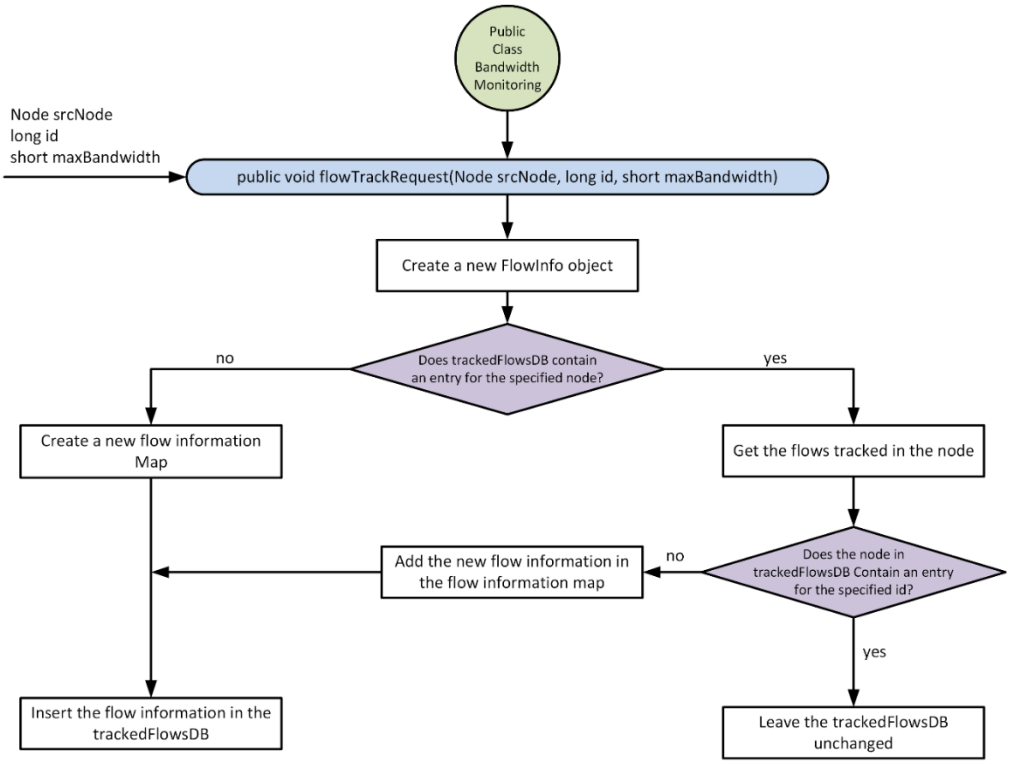

Figure 47: Flow diagram of adding a new service to the monitoring database

6.4.3.2 *Remove service from the monitoring database*

Similarly, when a service expires, the DynPaC service manager removes it from all the network devices and communicates with the monitoring module to remove the service from the monitoring databases. As the monitoring information is kept in two different databases, it is necessary to check in both of them, as shown in Figure 48.

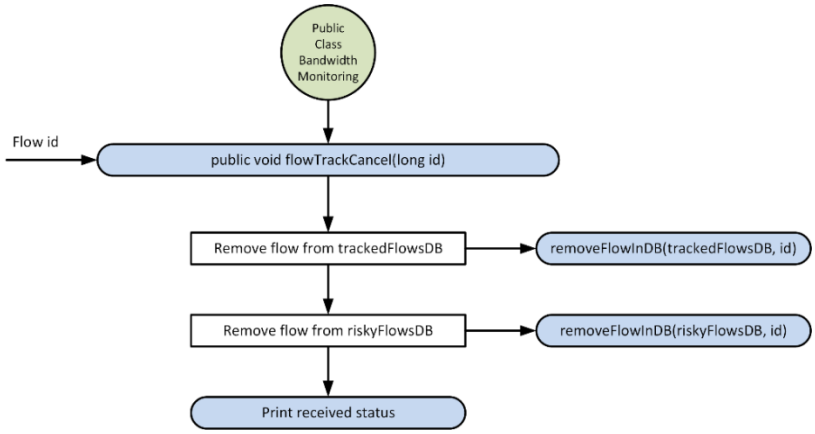

Figure 48: Flow diagram of removing a service from the monitoring databases.

In each database, the procedure to remove the information associated to a service is the one described in the flow diagram of Figure 49.

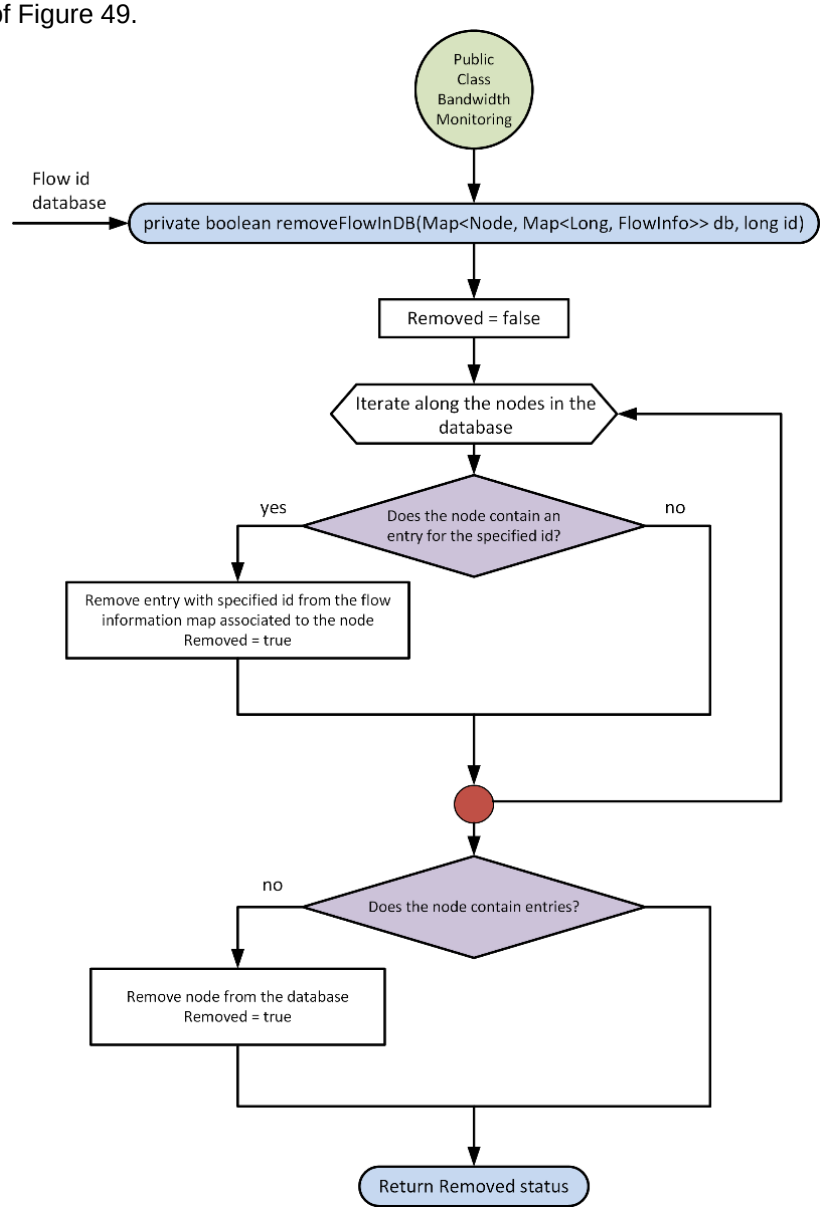

Figure 49: Flow diagram of removing the information associated to a given service in a given database.

6.4.3.3 *Update the id of a service in the monitoring database*

The *ID* of the flows used in the nodes may change when a service changes from its primary path to its backup path. As a consequence, it is necessary to modify the ID of the affected flows in the monitoring databases. As the monitoring information is held in two different databases, it is necessary to check in both of them as depicted in Figure 50. Additionally, Figure 51 shows the procedure used to update an ID in a given database.

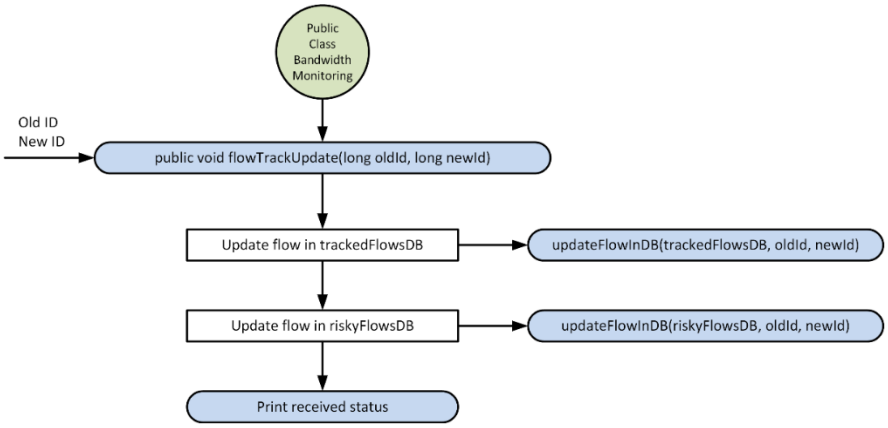

Figure 50: Flow diagram of updating the ID of a given service in both databases.

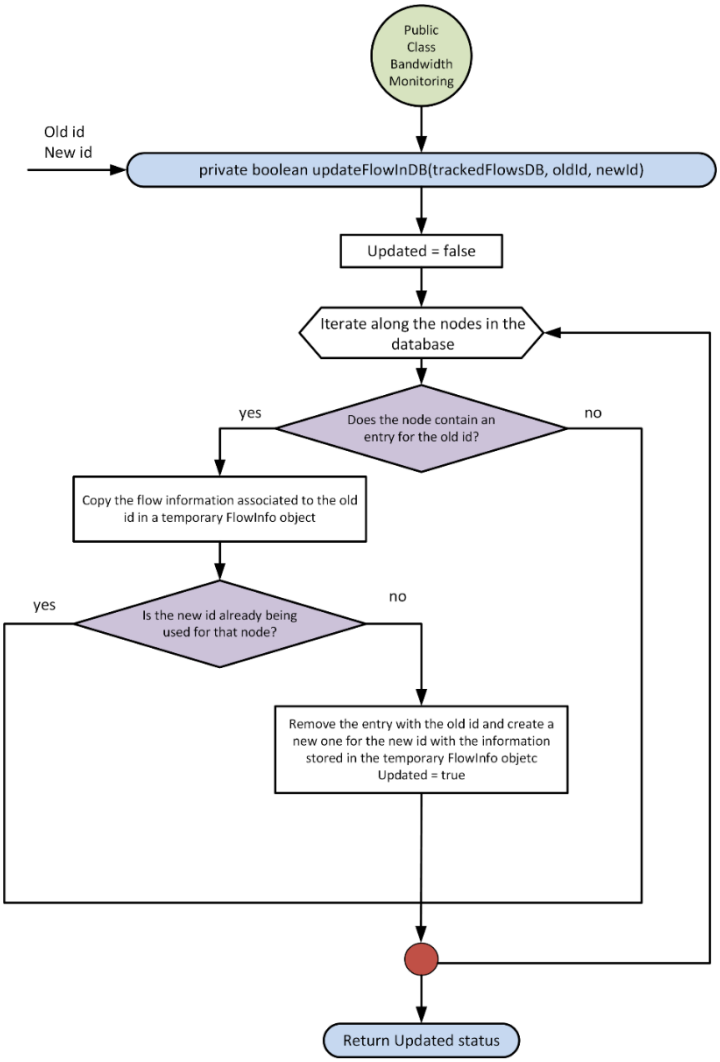

Figure 51: Flow diagram of updating an ID in a given database.

As shown in Figure 51, in each database, it is necessary to check whether the service must be updated in all the nodes, as the ID can exist in multiple nodes. Furthermore, before updating the ID, it is necessary to check if the new ID is already being used or not, to avoid duplicates.

6.4.3.4 *Flow monitoring*

The DynPaC monitoring method implements the methods typical to bundles that act like servers: init(), start(), destroy() and stop(). Next, the implementation of the init(), start() and destroy() methods is presented. The stop() stage is not described as it is the default one.

During the initialization phase, the bundle creates all the necessary databases and timers to keep track of the monitored flows. Figure 52 depicts the flow diagram of the init() method.

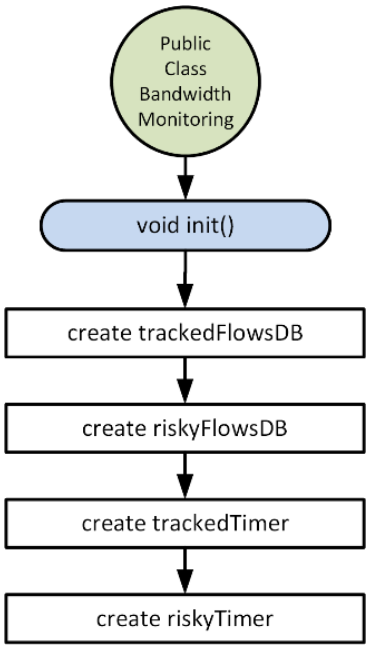

Figure 52: Flow diagram of the init() phase.

Once the necessary elements have been initialized, the start() method is used to activate the service. In that method the schedulers that allow to periodically monitor the services stored in the corresponding databases are initialized. Each scheduler is defined by a different configuration set (offset and period), and calls the monitor() method passing the database that must be monitored in that specific time as an argument. Figure 53 shows how the two schedulers are created and how the monitoring of the databases is performed periodically.

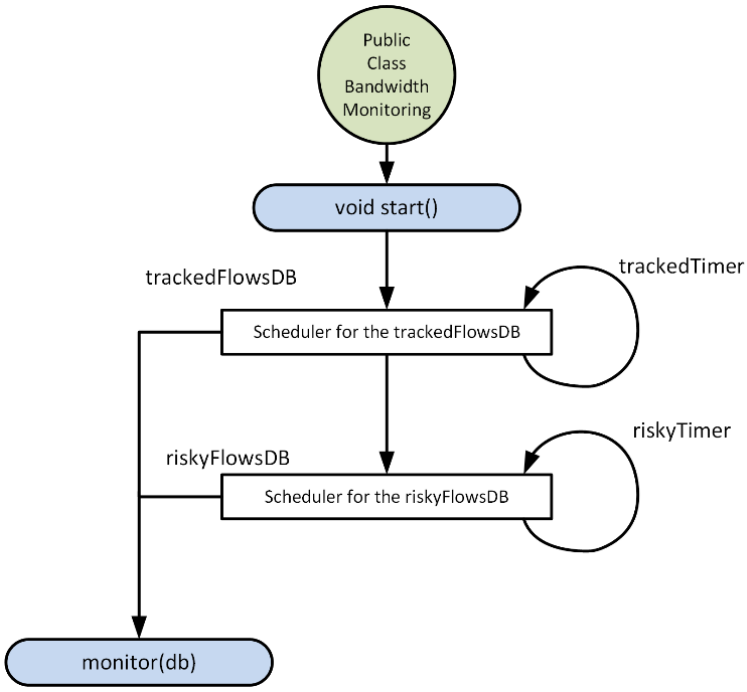

Figure 53: Flow diagram of the *start*() stage.

Each time the monitor method is called, it iterates along the database that must be monitored and retrieves the statistical information regarding all the services installed in the nodes that the database stores information about. Then, it uses the newly obtained statistical information to update the entries of the services. If there are no flows installed in the node, the monitor module automatically restarts all the counters for that node. Figure 54 depicts the flow diagram of the monitoring of a given database.

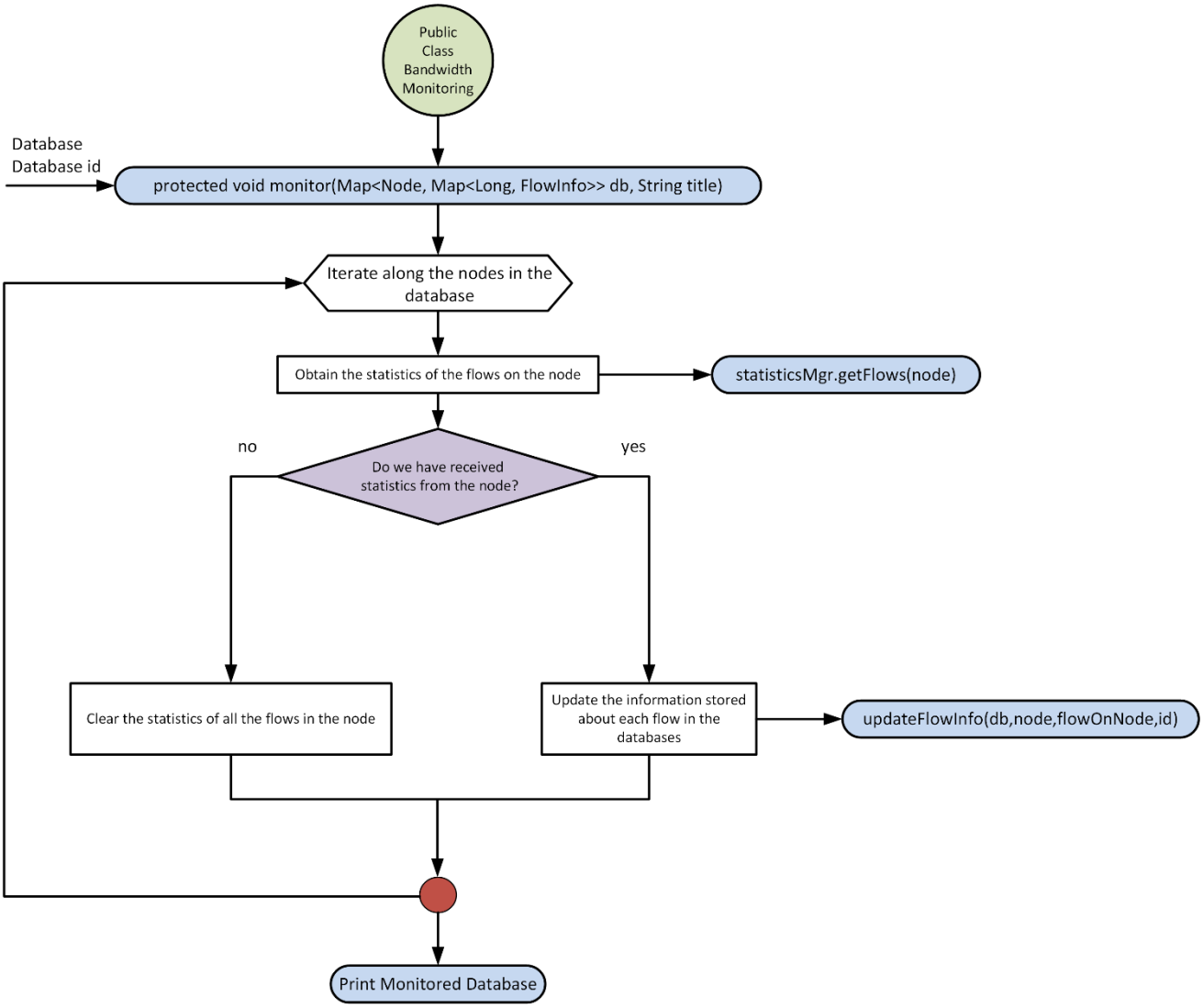

Figure 54: Flow diagram of the monitoring of a database.

Finally, the information obtained for each flow is used to update the monitored database. In this case, the bandwidth is also calculated using the following mathematical expression. Figure 55 shows how the flow information is updated in the database.

$$Bandwidth = \frac{\left(ByteCount_{new} - ByteCount_{previous}\right)\cdot 8}{\left(Duration_{new} - Duration_{previous}\right)\cdot 2^{20}}[Mbps]$$

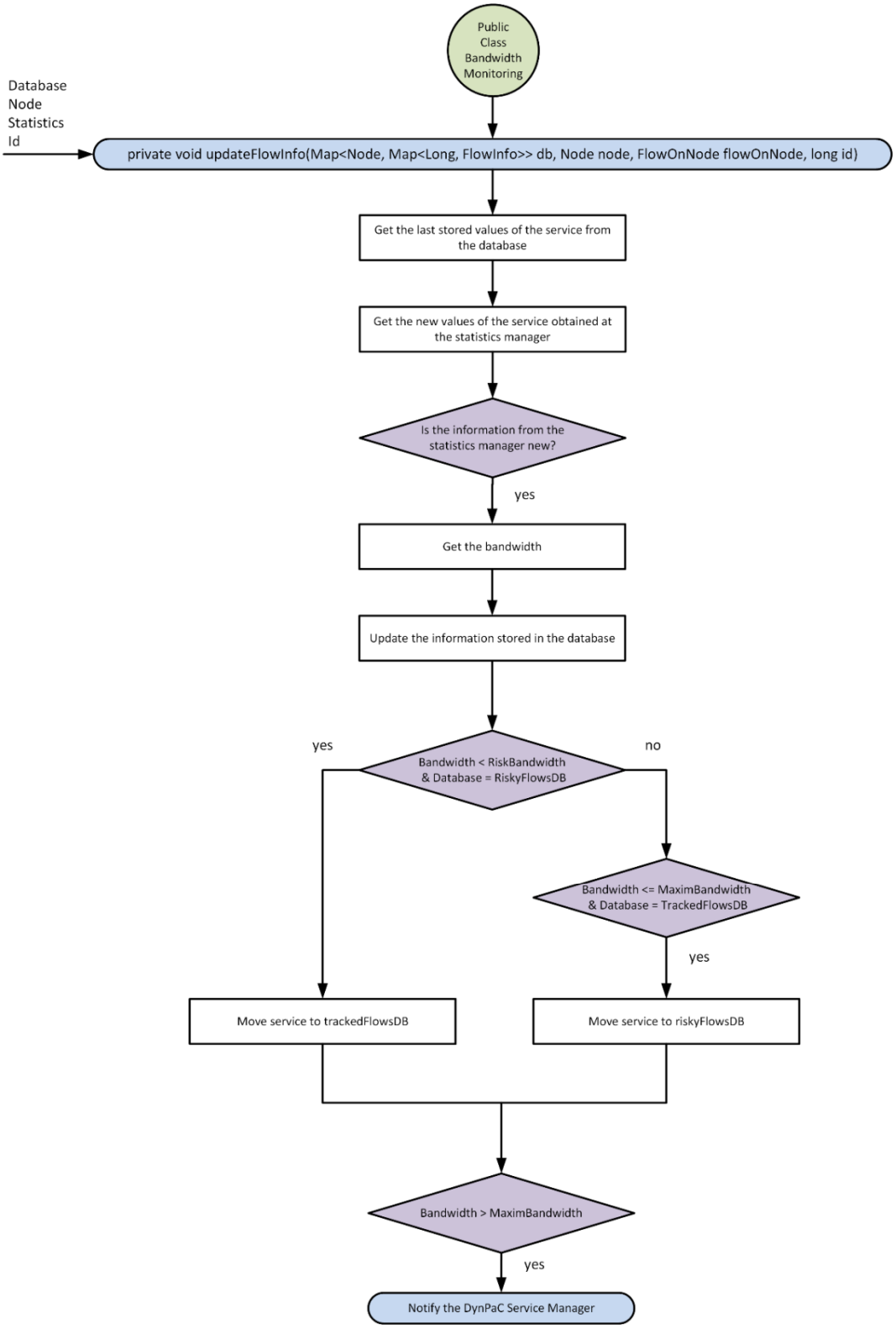

Figure 55: Flow diagram of a flow update.

6.5 DynPaC GUI

The DynPaC GUI is based on the main OpenDaylight Web. Its main features are:

- Service request.
- Cancel a service reservation.
- Expose information about services programmed.

DynPaC exposes a REST interface to interact with it. This allows interacting with DynPAC directly from the web or the REST interface. This module is also in charge of invocating the corresponding methods over the DynPaC service manager and parsing the responses coming from it.

Next, Figure 56 shows the interaction of the user with the DynPaC framework.

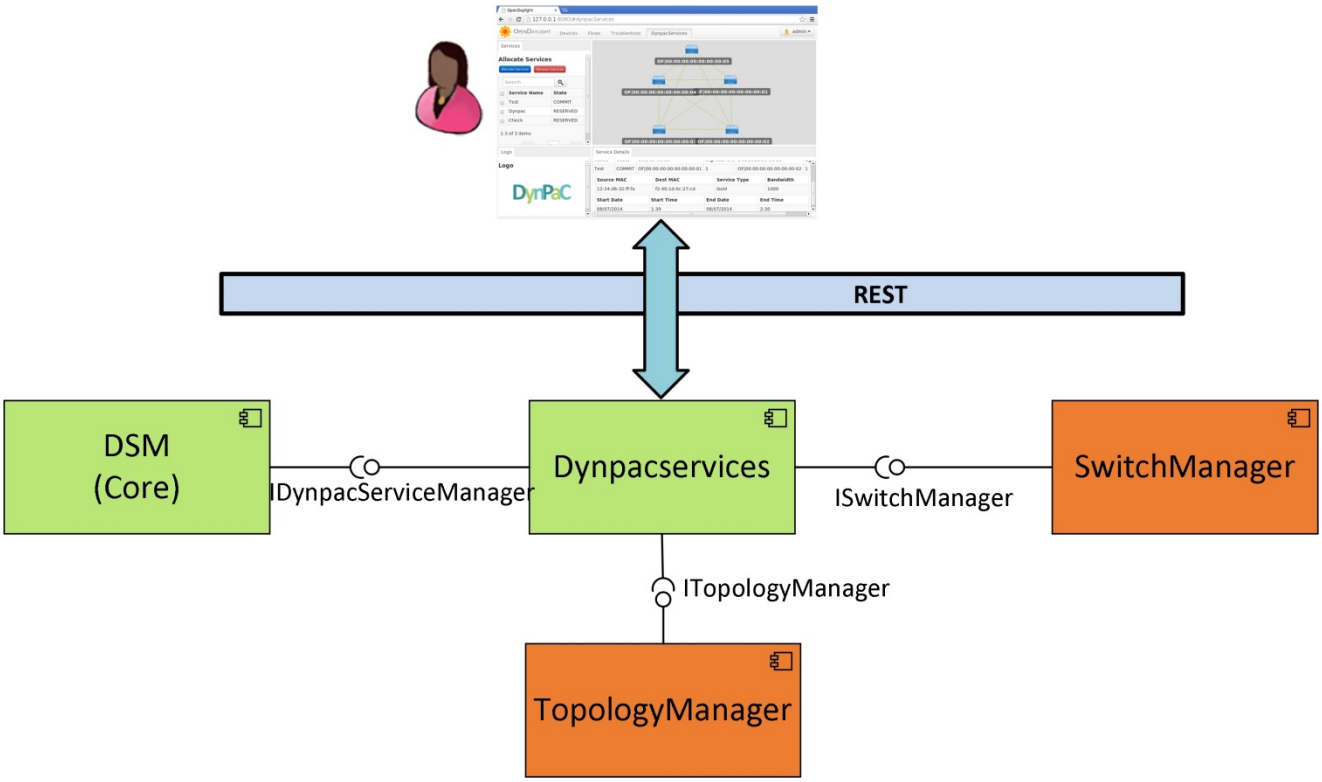

Figure 56: User interaction with the DynPaC framework

6.5.1 DynpacServices class

The DynpacServices class is the bundle that, integrated inside OpenDaylight web functionality, allows the communication between the DynPaC user and the DynPaC core.

Figure 57 shows the schema corresponding to this class.

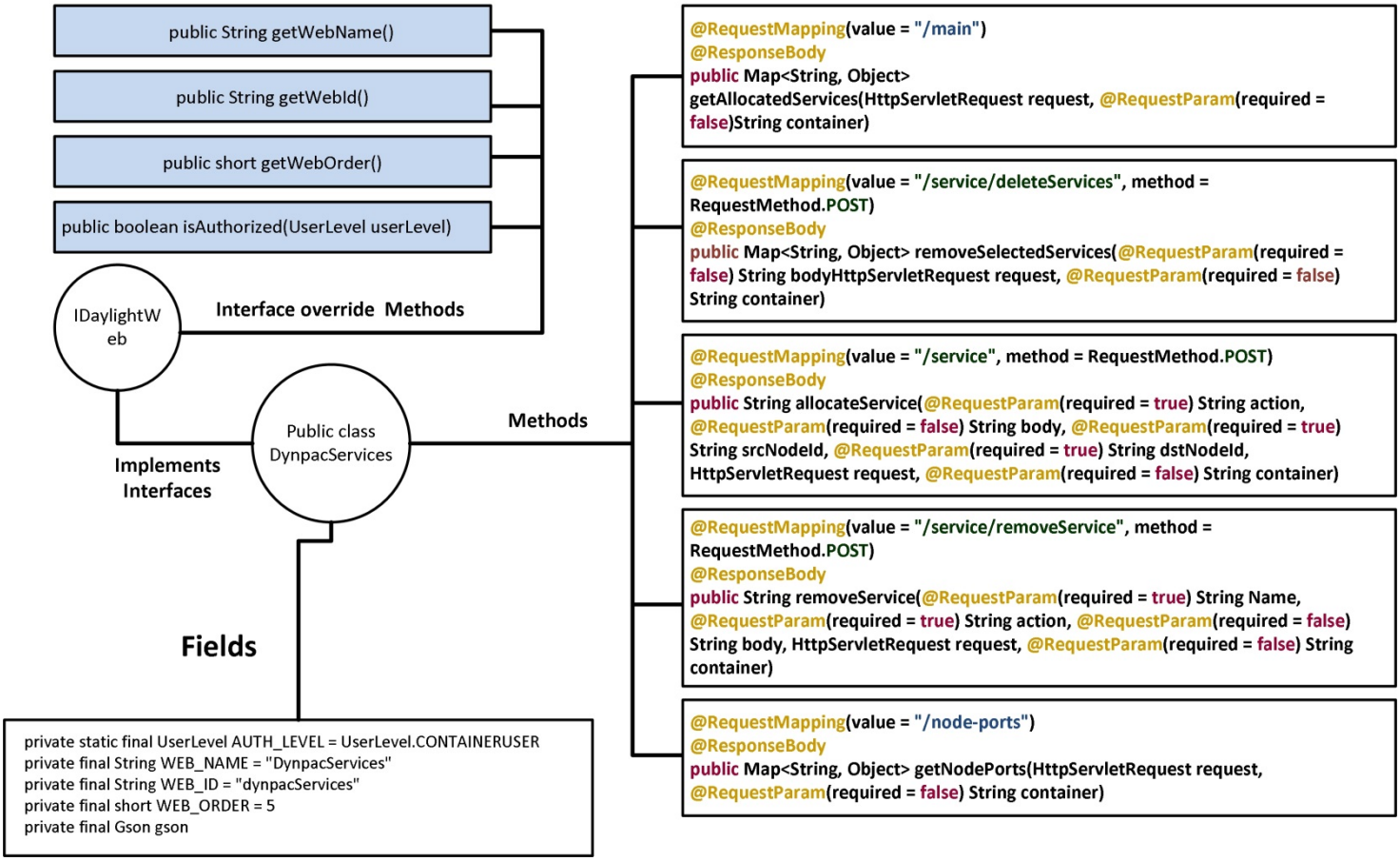

Figure 57: DynPac GUI web bundle schema

Implemented methods that can be found in the DynpacServices class are:

- ***getAllocatedServices:*** used to gather information about programmed services.
- ***RemoveSelectedServices:*** allows deleting a list services from the DynPac framework.
- ***AllocateService:*** used to reserve a service at DynPac.
- ***removeService:*** allows to delete a list services from the DynPaC framework.

- ***getNodePorts:*** used to gather information in order to expose only external ports to the user. It filters all internal ports, not allowing a user to choose these internal ports from the web interface to mark them as ingress or egress ports during the setup of the service.

6.5.2 Functional overview

This section describes the main features of the DynPaC GUI module.

6.5.2.1 Gathering service information

When selecting each service from the list of services at the web interface, information is collected from the DSM and shown to the user. This information is related to the parameters configured or the status of the service. The information gathering process is done as depicted in Figure 58.

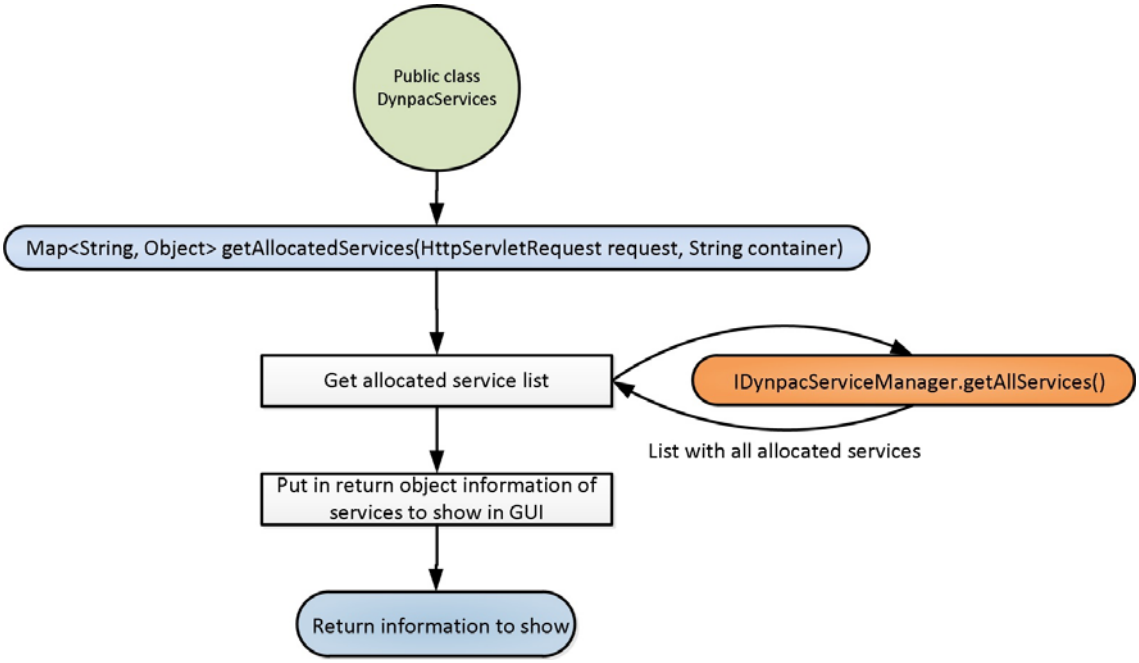

Figure 58: Flow diagram of load services in GUI

6.5.2.2 External port identification process

When exposing available ports to the user, all the internal ports, or ports not connected are hidden to the user in order to avoid configuration mistakes. Some of the features of *TopologyManager* and *SwitchManager* are used, to determine if a certain port is internal or not. The procedure followed to get the node ports is graphically depicted in Figure 59.

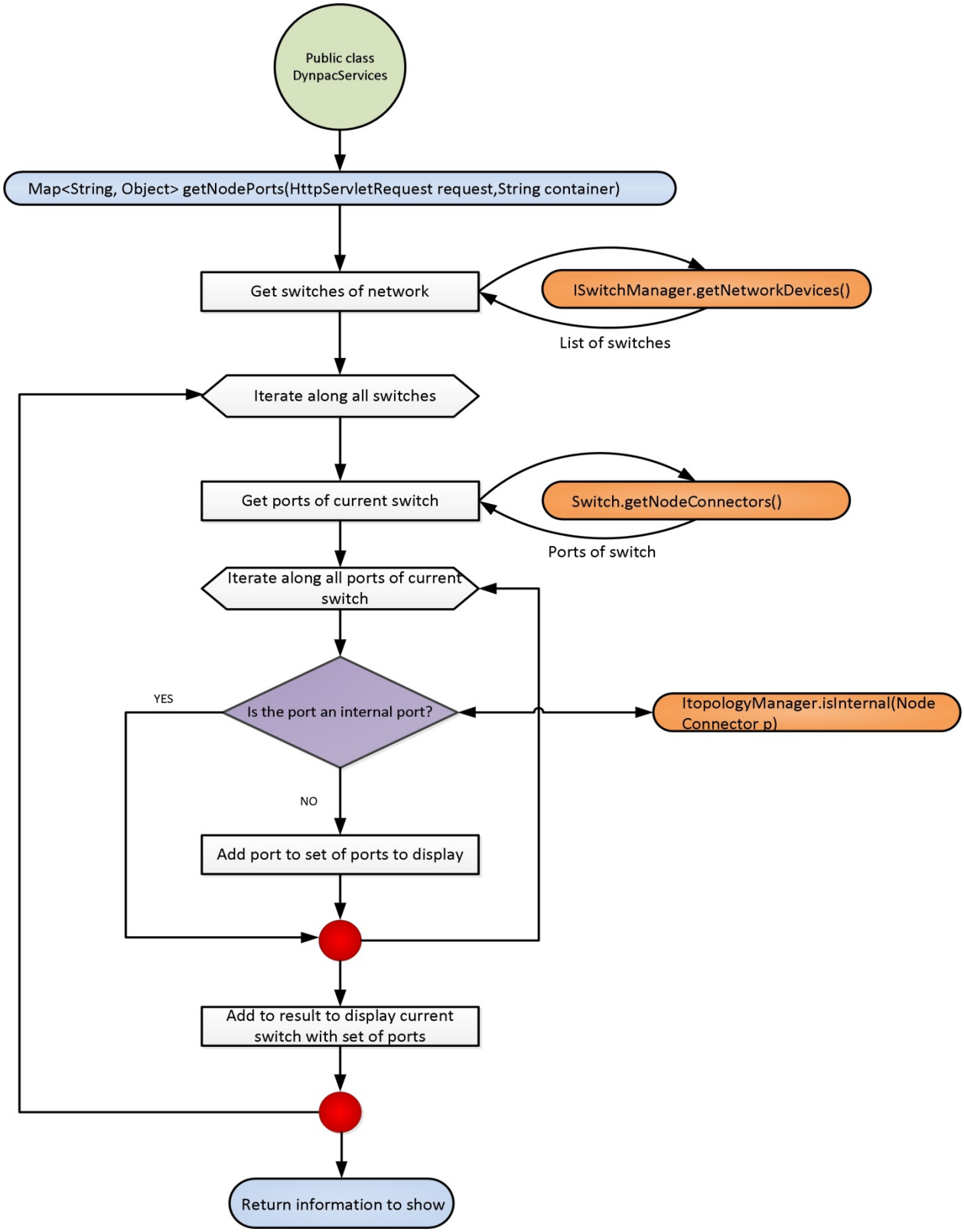

Figure 59: Flow diagram of node ports get process

6.5.2.3 New service request

When a new service is requested, information is sent to the DSM, and a response is sent back to the GUI, showing the new service in the list of programmed services if the transaction has been successfully performed, or showing an error banner with the description of the error happened during the reservation process.

Figure 60 represents the procedure used to request a new service from the web user interface.

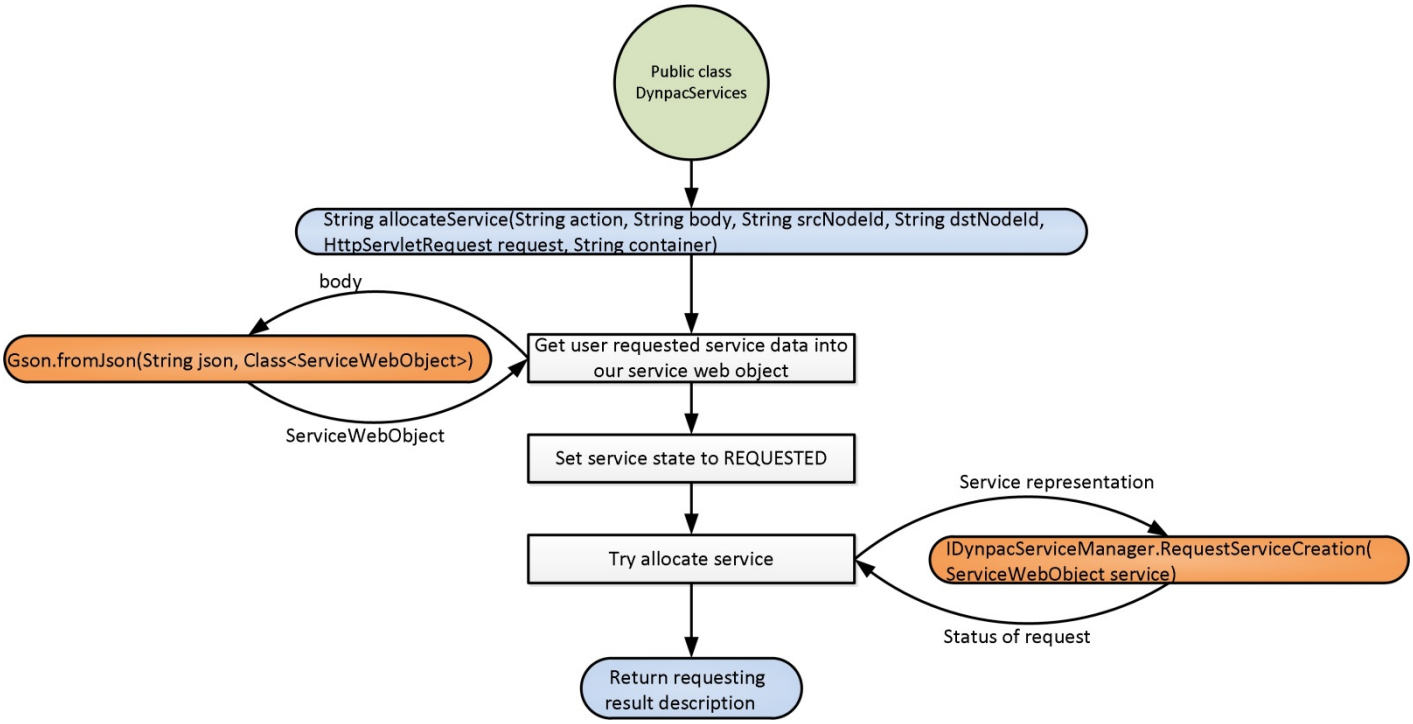

Figure 60: Flow diagram of service request

6.5.2.4 Deletion of a programmed service

Services can be deleted one by one or as a list. When this operation is done, if the result has been successfully processed, the affected service or services are deleted from the programmed service list and unloaded from the network if they were already running. If an error happens, a banner is shown with the explanation of the error.

Figure 61 represents the procedure used to delete a service from the DynPaC framework.

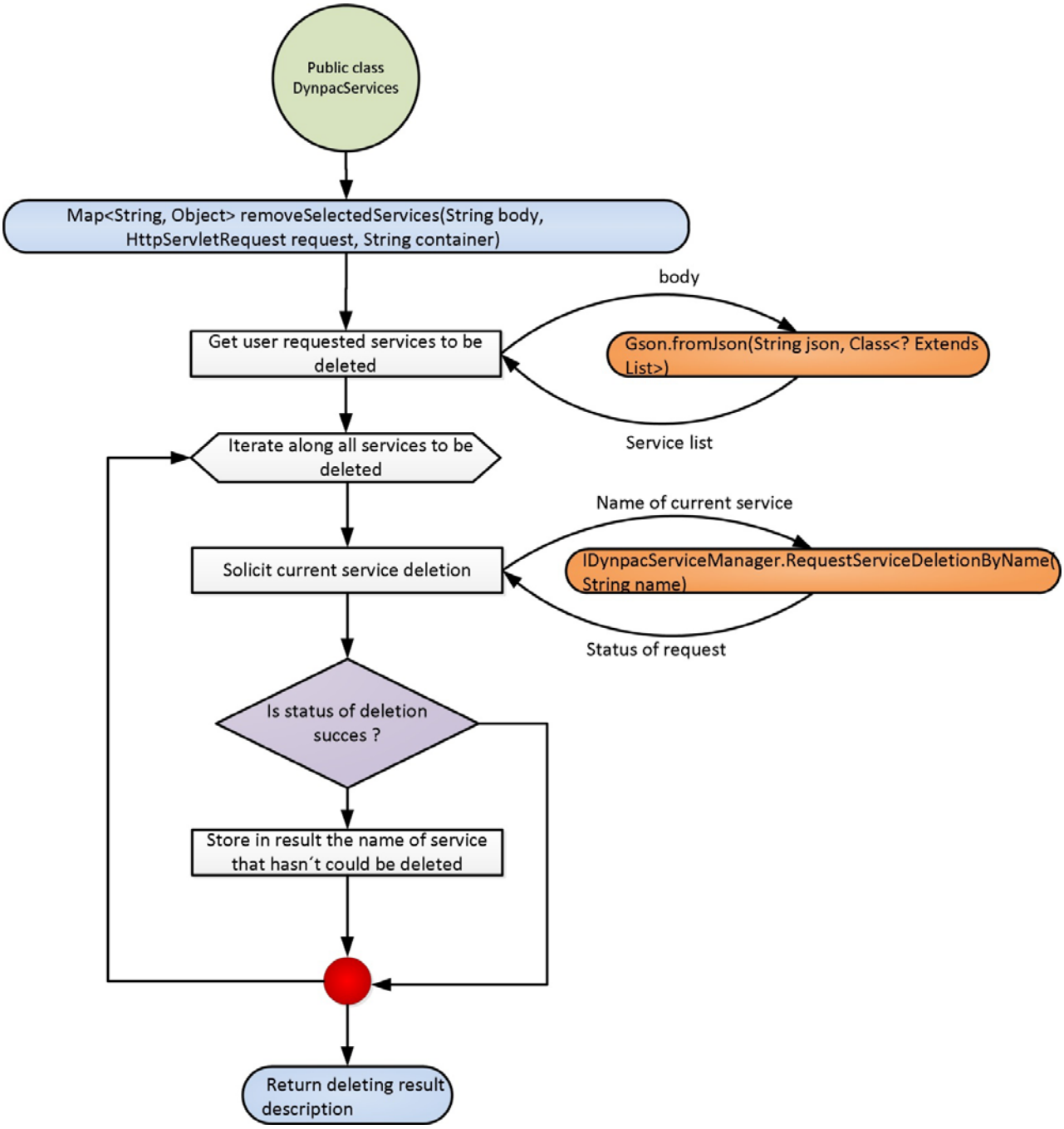

Figure 61: Flow diagram of service deletion request

6.6 Traffic Pattern Analyzer

In order to be able to suitably disaggregate services, DynPaC requires information about the distribution of the traffic of the established services. This information is currently read from a XML file that

describes the different possible traffic distributions of each of the currently established services. This XML file information emulates the information provided by an external Traffic Pattern Analyzer module.

The aim of the Traffic Pattern Analyzer is to provide characterization or pattern information associated with each active service in the network, enabling disaggregation mechanisms.

An example of the structure of the XML file is represented in Figure 62:

```
            </percent>
        </disaggregate>
</service>
<service id="S2" Bw="4">
        <disaggregate>
            <percent value="50">
                <field value="">IN_PORT</field>
                <field value="">DL_SRC</field>
                <field value="">DL_DST</field>
                <field value="0X0800">DL_TYPE</field>
                <field value="">DL_VLAN</field>
                <field value="">DL_VLAN_PR</field>
                <field value="">NW_SRC</field>
                <field value="">NW_DST</field>
                <field value="6">NW_PROTO</field>
                <field value="">NW_TOS</field>
                <field value="">TP_SRC</field>
                <field value="">TP_DST</field>
            </percent>
            <percent value="50">
                <field value="">IN_PORT</field>
                <field value="">DL_SRC</field>
                <field value="">DL_DST</field>
                <field value="">DL_TYPE</field>
                <field value="">DL_VLAN</field>
                <field value="">DL_VLAN_PR</field>
                <field value="">NW_SRC</field>
                <field value="">NW_DST</field>
                <field value="">NW_PROTO</field>
                <field value="">NW_TOS</field>
                <field value="">TP_SRC</field>
                <field value="">TP_DST</field>
            </percent>
        </disaggregate>
        <disaggregate>
            <percent value="50">
                <field value="">IN_PORT</field>
                <field value="">DL_SRC</field>
                <field value="">DL_DST</field>
                <field value="0X0800">DL_TYPE</field>
                <field value="">DL_VLAN</field>
                <field value="">DL_VLAN_PR</field>
                <field value="">NW_SRC</field>
                <field value="">NW_DST</field>
                <field value="6">NW_PROTO</field>
```

Figure 62: Structure of the XML file representing the results of the Traffic Pattern Analyzer

To read the content of the file, a DOM (Document Object Model) is used, so all the parameters can be accessed following a tree-like structure. Figure 63 describes the information reading procedure over the XML file.

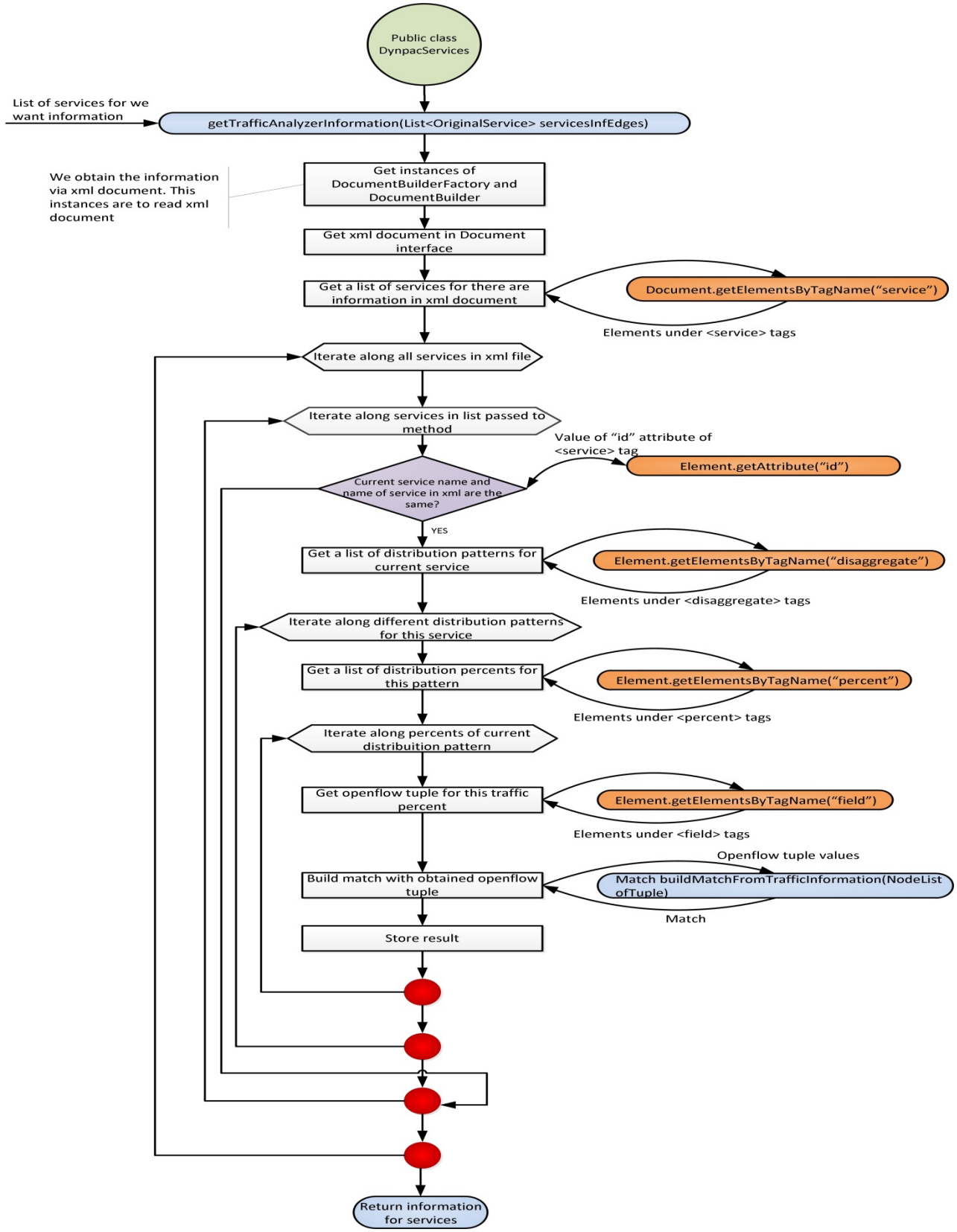

Figure 63: Flow diagram of get information from Traffic Pattern Analyzer

7 Tests performed and Results

This section describes the testing procedures defined and conducted to validate the implemented features in the DynPaC framework. The aim of this section is to check that the DynPaC framework meets the requirements defined as design objectives. As already mentioned, the design of the DynPaC framework has been carried out with ease of modification, modularity and flexibility in mind, and not prioritizing performance. For this reason, test about the performance of the framework have not been conducted.

7.1 Network topology

The topology shown in the Figure 64 has been used as the default template for these DynPaC tests. It allows showing how the developed features perform dynamic allocation of services, changing the paths or disaggregating them when necessary, with the aim of managing different conditions derived from requests for new services, broken links, and so on. In fact, this topology was requested to be deployed over GTS.

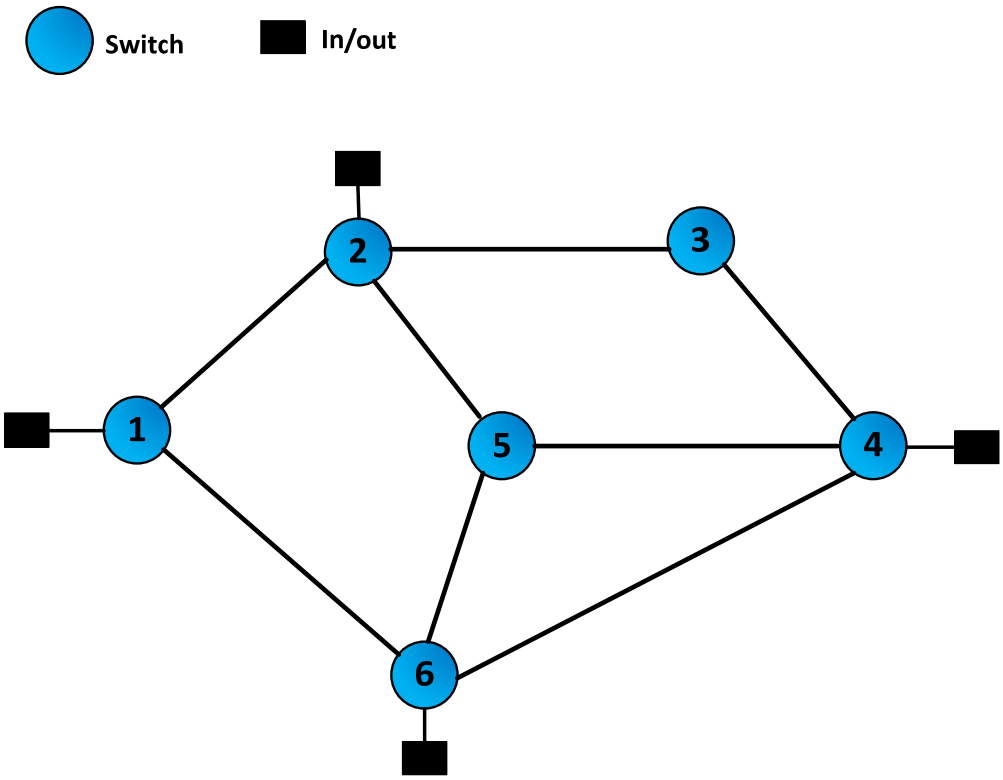

Figure 64: Default topology used for testing purposes.

7.2 Functional evaluation

This section describes the tests conducted to verify the proper functioning of each developed feature of the DynPaC framework.

7.2.1 Scheduler and snapshot creation tests

The actions performed to verify the working of the scheduler and the snapshot creation functionalities are described next.

7.2.1.1 *Reservation test*

First, reservation of up to 16 services has been made, and all the overlapping states have been generated and registered. The evaluated situations are graphically represented in Figure 65.

It is interesting to mention that these services have been selected so that the network has sufficient resources to allocate them.

We will consider the result of this test valid if the reservation for all services is made properly.

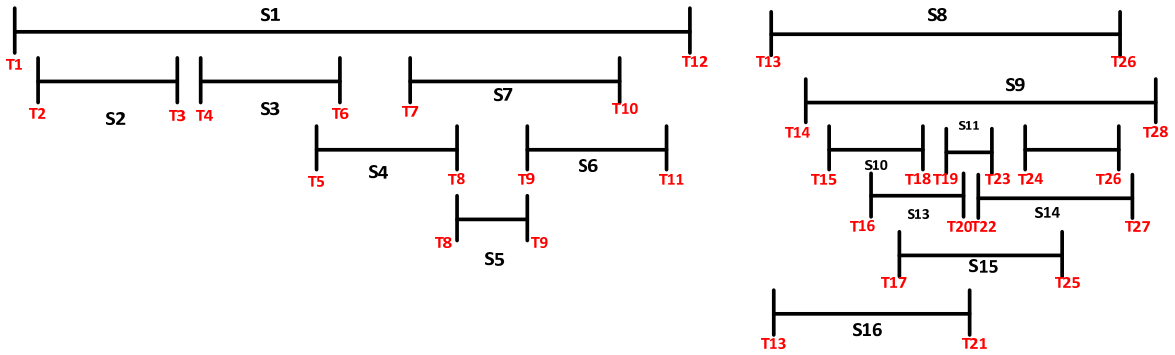

Figure 65: Services installed for testing

The following table shows the requested services together with their properties. Services have been requested in sequential order. The “ok” value included in the last column for all the rows means that the reservation process has finished in a proper way, and it has been obtained from the DynPaC core.

Service	Gold	BW	Start time	End time	Src node	Dst node	Result

S1	Yes	5 GB	T1	T12	2	6	ok
S2	Yes	5 GB	T2	T3	2	6	ok
S3	Yes	5 GB	T4	T6	2	6	ok
S4	Yes	5 GB	T5	T8	1	4	ok
S5	Yes	5 GB	T8	T9	4	6	ok
S6	Yes	5 GB	T9	T11	2	4	ok
S7	Yes	5 GB	T7	T10	1	4	ok
S8	Yes	1'5 GB	T13	T26	1	4	ok
S9	Yes	1'5 GB	T14	T28	1	6	ok
S10	Yes	1'5 GB	T15	T18	4	1	ok
S11	Yes	1'5 GB	T19	T23	2	1	ok
S12	Yes	8 GB	T24	T26	2	4	ok
S13	Yes	1'5 GB	T16	T20	1	4	ok
S14	Yes	0'5 GB	T22	T27	1	6	ok
S15	No	1'5 GB	T17	T25	1	4	ok
S16	Yes	1'5 GB	T13	T21	1	4	ok

Table 1: Service provisioning test results in the defined topology

7.2.1.2 *Service not allocable test*

The purpose of this test is to verify that when a new service is requested, and there is no available resources in the network for allocating it, considering the previously reservations made, a negative response for the reservation is obtained from the Dynpac framework. In normal conditions the disaggregation function will be requested afterwards, with the aim of checking if this request could be addressed using this feature.

For this test we used the default topology (showed in Figure 64), and the reservations for 3 services were defined, following the time schema shown below.

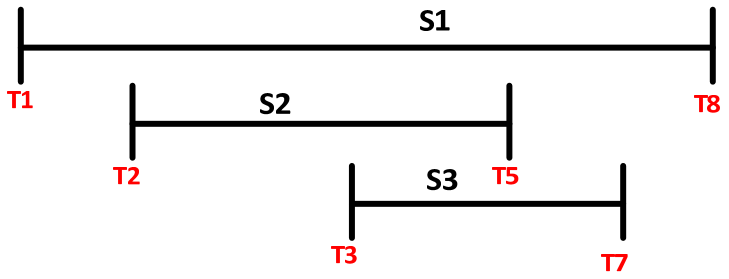

Figure 66: Service reservation schema for service not allocable testing

The following table shows the characteristics of the 3 requested services, and the obtained results for the requests.

Service	Gold service	Requested bandwidth	Start time	End time	Source node	Destination node	Result of requesting
S1	Yes	5 GB	T1	T8	1	4	ok

S2	No	1 GB	T2	T5	1	2	ok
S3	Yes	5 GB	T3	T7	1	4	X
S3	No	5 GB	T3	T7	1	4	ok

Table 2: Service not allocable test results

The results in the last column of the table shows the value “ok“ for the 2 first reservations, and a value of “X” for the third row. This value indicates that the requested service cannot be fulfilled. In this case the service cannot be addressed as a “Gold Service” as requested, but if the user accepts the provision of the service without this feature, then the service can be allocated, as the result of the fourth row shows.

7.2.1.3 *Removal of service test*

This test tries to check the right functioning of the removal of services previously requested and installed.

When this occur the snapshot database and the service database must be updated accordingly.

```
Expired timer
Current Time updated...1427123280001
Unloading SNAPSHOT...1
  Deleting Paths for TEST
Deleting from Network...
Deleting primary
Marking service as inactive...
Getting flowpattern to be activated...
Pushing FlowMods...
Building common match...
Service cookie 1234000100000000
Forward cookie 1234000100004000
Unregistering cookie 1234000100004000
FlowTrackCancel Call Received
The flow has been removed from all the monitoring DBs
DONE!
TRACKED FLOWS
```

The output of the part of the program implementing this function is included here to show the proper results obtained.

7.2.2 OpenFlow rules management tests

7.2.2.1 *Flow insertion*

The functional validation of the installation of services in network switches has been carried out by means of checking the rules pushed to the switches.

The tests conducted have shown that this function worked as expected.

Service	Gold	Requested BW	Start time	End time	Source node	Destination node	Required flows mod installed
S1	Yes	5 GB	T1	T12	2	6	ok
S2	Yes	5 GB	T2	T3	2	6	ok
S3	Yes	5 GB	T4	T6	2	6	ok
S4	Yes	5 GB	T5	T8	1	4	ok
S5	Yes	5 GB	T8	T9	4	6	ok
S6	Yes	5 GB	T9	T11	2	4	ok
S7	Yes	5 GB	T7	T10	1	4	ok

Table 3: Flow installation validation test overview

7.2.2.2 *Removal of flows*

In this test, the removal of services is made when a service expires. The carried out tests showed that this function worked as expected.

Service	Gold	Requested BW	Start time	End time	Source node	Destination node	Flows successfully removed
S1	Yes	5 GB	T1	T12	2	6	ok
S2	Yes	5 GB	T2	T3	2	6	ok
S3	Yes	5 GB	T4	T6	2	6	ok
S4	Yes	5 GB	T5	T8	1	4	ok
S5	Yes	5 GB	T8	T9	4	6	ok
S6	Yes	5 GB	T9	T11	2	4	ok
S7	Yes	5 GB	T7	T10	1	4	ok

Table 4: Flow removal validation test overview

7.2.3 Backup features

7.2.3.1 *Changeover to backup path*

With the aim of testing this feature a link failure in the network topology in mininet has to be generated. This fact will allow us to check if the affected services are re-allocated in their previously computed backup paths.

In this case, we have caused that between T5 and T6 time marks, the link between the nodes 1 and 6 is disconnected. Then, the affected flows change to move services to their secondary path.

We have checked if the flows in the corresponding switches change to reallocate services in the proper way.

The table below includes data about the reserved services before the link failure.

Service	Gold	Requested BW	Start time	End time	Source node	Destination node
S1	Yes	5 GB	T1	T12	2	6
S2	Yes	5 GB	T2	T3	2	6
S3	Yes	5 GB	T4	T6	2	6
S4	Yes	5 GB	T5	T8	1	4
S5	Yes	5 GB	T8	T9	4	6
S6	Yes	5 GB	T9	T11	2	4
S7	Yes	5 GB	T7	T10	1	4

Table 5 Gold services backup test results in simple topology

In the conducted test, in a moment between T5 and T6 we have provoked the down in the mentioned link and the affected flows changed, being reinstalled the proper rules along the secondary path of the services.

7.2.4 Disaggregation feature test

7.2.4.1 Simple disaggregation test

The aim of this test is to check if the disaggregation feature works in the expected way. In order to verify this feature we have tried to make a reservation when we know that there are no enough resources available in the network topology to allocate the new requested service in the operating snapshot, so a disaggregation is needed. This test will be passed if the new requested service is properly allocated thanks to the disaggregation of any of the previously installed services.

First, the reservation of the services made is shown in the table below.

Service	Gold	Requested BW	Start time	End time	Source node	Destination node
S1	No	8 GB	T1	T2	1	4
S2	No	5 GB	T1	T2	1	4
S3	No	7 GB	T1	T2	2	4

Table 6 Services loaded for the disaggregation test

After the “Start time” T1, the assigned paths for the reserved services are the following ones:

Service	Nodes
S1	1 – 6 – 4
S2	1 – 2 – 3 – 4
S3	2 – 5 – 4

Table 7: Paths used before service disaggregation

Once these services are allocated in the network, a 9 GB new service is requested from node 2 to node 4. From the information available about the S2 service (stored in a xml file), we can know that the S2 service profile consists in 50% TCP traffic, 40% UDP traffic and 10% other traffic. Because of that, S2 is split into three subservices and the corresponding rules are installed in the switches along the paths. The selected paths for the three subservices are shown in the following table.

Service	Nodes
S2 TCP traffic	1 – 2 – 5 – 4
S2 UDP traffic	1 – 6 – 4
S2 other traffic	1 – 2 – 3 – 4

Table 8 Services after disaggregation

The requested service has been allocated along the path 1-3-4.

8 Conclusions and future work

DynPaC is a very powerful software framework that supports the dynamic computation of paths taking into consideration time and bandwidth constraints. It is also a very powerful tool in terms of network utilization, thanks to its flow aggregation and disaggregation mechanism that allows to split the traffic into multiple sub-services in order to make room for new service requests.

As seen in Section 7 the DynPaC framework fulfils all the functional requirements outlined in Section 3. Firstly, the framework supports the aforementioned scheduling of services taking into consideration time and bandwidth constraints. Being based on network snapshots, the DynPaC framework allows to detect that the required resources are free in the specified period of time even if they are occupied in other time frames. For each new service request, all the affected snapshots are analysed, and only when the SLA can be guaranteed during the entire period of time of the new service, it is accepted.

The tests have also shown that the snapshot management, which is critical, is performed correctly. When a new service is requested, the DSM checks all the existing snapshots to determine which ones are affected by the new arrival. Furthermore, it correctly creates the resulting new snapshots and manages the smooth transition between them. Meaning that when there is a snapshot change, it minimizes the number of services being moved.

Once the service are reserved, the DynPaC framework is able to automatically program the involved network devices. When the service starts, the new flow entries for that specific flow are installed on the switches. Similarly, the flow entries are removed when the service expires. The devices are also correctly programed when a service is restored due to a link failure or when it is moved of disaggregated in order to optimize network utilization.

Regarding the resiliency and service restoration, tests show that backup paths are correctly installed when there is a link failure. It works flawlessly for both type services: Golden and Regular. In the case of the Golden services, the backup path is guaranteed, whereas in the case of the Regular services, it is provided only when there are available resources.

Finally, the services disaggregation is performed properly by the PCE. It has been validated that when there are no resources available for a new service request, the PCE first tries to move the existing services in order to make room for the new one. Furthermore, only when it is absolutely necessary the PCE disaggregates an existing service to free the necessary resources for the newly requested one. As mentioned before, the DSM handles the node programming in order to update the flow tables of the nodes with the new flow information.

Even if the DynPaC framework fulfills all the functional requirements, there are some improvements under consideration that the team would like to address in the near future.

When the DynPaC project started, the ODP was on its first stable release, named Hydrogen. In that version, the Service Abstraction Layer, which is the main element of the controller that allows to control network devices regardless of its type was an API Driven – SAL. As a consequence, being based on the Hydrogen

release, the DynPaC framework was designed to work in conjunction with that AD-SAL. However, back in September 2014 the Helium release appeared, introducing major changes. In that new release the Model Driven – SAL was introduced, which follows a completely different approach from the AD-SAL.

Due to the major differences between the two releases, the migration of the DynPaC framework to the Helium release is not straightforward. Besides, taking into consideration that the Helium release has been proven to be quite unstable, it was decided to postpone the upgrade of the DynPaC framework to the MD-SAL until the next ODP release, Lithium, planned for June 2015.

Another improvement under consideration is the result of the “Hands on the Open Daylight Project development” workshop that took place in the Géant Symposium 2015. In that workshop, people in the audience proposed to improve the GUI in order to enable the visualization of resource utilization during time. With this new functionality, users would be able to select the time frame for their service knowing in advance that resources are available. This, this is another improvement that we would like to address in the near future.

There are also plans to continue investigating on the flow aggregation and disaggregation capabilities of the framework as part of the work carried out in a PhD Thesis.

Appendix A: User guide

Getting DynPaC source code:

DynPaC source code latest version can be found at:

svn+ssh://svn@svn.geant.net/GEANT/DynPac/release1.0

DynPaC can be cloned using subversion

```
svn co svn+ssh://svn@svn.geant.net/GEANT/DynPac/release1.0
```

DynPac Requirements

- Linux system with Oracle Java or OpenJDK installed and configured
- Minimum of 2GB of RAM
- Multicore processor (recommended)
- Maven 3+ installed and configured
- 500Mb of space at HDD
- OpenFlow 1.3 compatible switching devices

To get Java JRE or OpenJDK, please visit:

Oracle JRE: http://www.oracle.com/technetwork/java/javase/downloads/index.html

OpenJDK http://openjdk.java.net/install/index.html or Linux system package repositories.

DynPac Compilation

To build DynPaC OpenDayLight version from the main directory run the following commands:

```
export MAVEN_OPTS=" –Xms1024m –XX:PermSize=1024"
cd Integration
mvn clean install –DskipTests=true –Dcheckstyle.skip=true
```

Running DynPac

After building DynPaC, this can be launched by executing ***run.sh*** located at:

```
/Integration/opendaylight/distribution/ target/distribution.opendaylight-osgipackage/opendaylight
./run.sh
```

However is recommended to set JAVA_OPT variable:

```
export JAVA_OPTS=" –Xms1024m –XX:PermSize=1024"
```

XML profile usage

XML named trafficanalyser.xml should be pasted at

```
/Integration/opendaylight/distribution/ target/distribution.opendaylight-osgipackage/opendaylight
```

XML profile edition

A new service definition can be done as follows, adding "service id" fields and defining the percent of the bandwidth used per sub flow and the OpenFlow matching.

```
<?xml version="1.0" encoding="UTF-8"?>
<services>
        <service id="B" Bw="8">
                <disaggregate>
                        <percent value="50">
                           <field value="">IN_PORT</field>
                           <field value="">DL_SRC</field>
                           <field value="">DL_DST</field>
                           <field value="0X0800">DL_TYPE</field>
                           <field value="">DL_VLAN</field>
                           <field value="">DL_VLAN_PR</field>
                           <field value="">NW_SRC</field>
                           <field value="">NW_DST</field>
                           <field value="6">NW_PROTO</field>
                           <field value="">NW_TOS</field>
                           <field value="">TP_SRC</field>
```

```
                    <field value="">TP_DST</field>
                </percent>
                <percent value="40">
                    <field value="">IN_PORT</field>
                    <field value="">DL_SRC</field>
                    <field value="">DL_DST</field>
                    <field value="0X0800">DL_TYPE</field>
                    <field value="">DL_VLAN</field>
                    <field value="">DL_VLAN_PR</field>
                    <field value="">NW_SRC</field>
                    <field value="">NW_DST</field>
                    <field value="17">NW_PROTO</field>
                    <field value="">NW_TOS</field>
                    <field value="">TP_SRC</field>
                    <field value="">TP_DST</field>
                </percent>
                <percent value="10">
                  <field value="">IN_PORT</field>
                    <field value="">DL_SRC</field>
                    <field value="">DL_DST</field>
                    <field value="">DL_TYPE</field>
                    <field value="">DL_VLAN</field>
                    <field value="">DL_VLAN_PR</field>
                    <field value="">NW_SRC</field>
                    <field value="">NW_DST</field>
                    <field value="">NW_PROTO</field>
                    <field value="">NW_TOS</field>
                    <field value="">TP_SRC</field>
                    <field value="">TP_DST</field>
                </percent>
            </disaggregate>
        </service>
    </services>
```

Appendix B: Rest API

DynPaC exposes a REST API that allows to interact with the DSM from the DynPaC GUI. This API can also be used to interact with the DSM using a different GUI, a CLI or an application like Postman [24]. The REST API is described in the following subsections.

Get Allocated Services

Type: GET
URL: http://localhost:8080/controller/web/dynpacServices//main
Description: Returns all the information of the allocated services

Example: This is the information that is obtained when a L2 service has been requested previously.


```
{
   "services": [
     {
         "srcNode": "OF|00:00:00:00:00:00:00:02",
         "srcNodeId": "OF|00:00:00:00:00:00:00:02",
         "name": "Service A",
         "state": "RESERVED",
         "service": {
            "dynamic": false,
            "status": "Success",
            "solicitService": "true",
            "name": "Service A",
            "state": "RESERVED",
            "srcNode": {
               "nodeIDString": "00:00:00:00:00:00:00:02",
               "type": "OF",
               "id": 2
            },
            "dstNode": {
               "nodeIDString": "00:00:00:00:00:00:00:03",
               "type": "OF",
               "id": 3
            },
            "ingressPort": "1",
            "egressPort": "2",
            "vlanId": null,
            "srcMac": null,
            "dstMac": null,
            "srcIp": null,
            "dstIp": null,
            "ipProto": null,
            "srcPort": null,
            "dstPort": null,
            "startDate": "30/09/2014",
            "startTime": "02:59",
            "endDate": "30/09/2014",
            "endTime": "03:00",
```

```
            "serviceType": "L2",
            "bandwidth": "200",
            "actions": null,
            "isDisaggregable": false,
            "isDisaggregated": false,
            “isGolden”:true,
            "statusSuccessful": true,
            "serviceEntry": {
               "groupName": "__StaticServices__",
               "serviceName": "Service A",
               "node": {
                  "nodeIDString": "00:00:00:00:00:00:00:02",
                  "type": "OF",
                  "id": 2
               },
               "service": {
                  "match": {
                     "matches": 0,
                     "matchesList": [],
                     "matchFields": [],
                     "ipv6": false,
                     "ipv4": true
                  },
                  "actions": [],
                  "priority": 0,
                  "idleTimeout": 0,
                  "startTime": 0,
                  "hardTimeout": 0,
                  "id": 0,
                  "ipv6": false
               },
               "internal": false
            },
            "internalService": false,
            "service": {
               "match": {
                  "matches": 0,
                  "matchesList": [],
                  "matchFields": [],
                  "ipv6": false,
                  "ipv4": true
               },
               "actions": [],
               "priority": 0,
               "idleTimeout": 0,
               "startTime": 0,
               "hardTimeout": 0,
               "id": 0,
               "ipv6": false
            }
         },
         "dstNode": "OF|00:00:00:00:00:00:00:03",
         "dstNodeId": "OF|00:00:00:00:00:00:00:03"
      }
   ],
   "privilege": "WRITE"
}
```

Get Edge Ports

Type: GET
URL: http://localhost:8080/controller/web/dynpacServices//node-ports
Description: Returns the edge nodes and their corresponding ports.

Example: This is the result with a 2 layer tree topology


```
{
   "OF|00:00:00:00:00:00:00:02": {
      "ports": {
         "1": "s2-eth1(1)",
         "2": "s2-eth2(2)"
      },
      "name": "OF|00:00:00:00:00:00:00:02"
   },
   "OF|00:00:00:00:00:00:00:03": {
      "ports": {
         "1": "s3-eth1(1)",
         "2": "s3-eth2(2)"
      },
      "name": "OF|00:00:00:00:00:00:00:03"
   }
}
```


Get nodes

Type: GET
URL: http://localhost:8080/controller/web/dynpacServices//node-services
Description: Returns all the nodes of the domain.

Example: This is the result with a 2 layer tree topology


```
{
   "OF|00:00:00:00:00:00:00:01": 0,
   "OF|00:00:00:00:00:00:00:02": 2,
   "OF|00:00:00:00:00:00:00:03": 0
}
```


Allocate a L2 service

Type: POST
URL: http://localhost:8080/controller/web/dynpacServices//service
Description: Allocate a golden service with L2 serviceType

Parameters

x-www-form-urlencoded	
x-page-url	dynpacServices
body	{"name":"service2","ingressPort":"1","egressPort":"2","serviceType":"L2","startDate":"30/09/2014","startTime":"02:59","endDate":"30/09/2014","endTime":"03:00",”isGolden”:”true”,”srcMac”:”ca:fe:ca:fe:ca:fe”, “dstMac”:”be:af:be:af:be:af”, "bandwidth":"200","solicitService":"true"}
action	add
srcNodeId	OF|00:00:00:00:00:00:00:02
dstNodeId	OF|00:00:00:00:00:00:00:03

Remove Service

Type: POST
URL: http://localhost:8080/controller/web/dynpacServices//service/deleteServices
Description: Removes the service specified with the name

Parameters

x-www-form-urlencoded	
x-page-url	dynpacServices
body	{"name":"service2"}

References

[1] https://www.opennetworking.org/sdn-resources/sdn-definition.
Software-Defined Networking, (accessed March 2015)

[2] http://www.opendaylight.org.
Open Daylight Project, (accessed March 2015)

[3] McKeown, N., Anderson, T., Balakrishnan, H., Parulkar, G., Peterson, L., Rexford, J. ... & Turner, J. (2008). OpenFlow: enabling innovation in campus networks. ACM SIGCOMM Computer Communication Review, 38(2), 69-74.

[4] http://autobahn.geant.net.
AutoBAHN, (accessed March 2015)

[5] http://www.es.net/services/virtual-circuits-oscars.
OSCARS, (accessed March 2015)

[6] https://tools.ietf.org/html/rfc4655.
RFC 4655. A Path Computation Element – based Architecture, (accessed March 2015)

[7] http://www.projectfloodlight.org/floodlight/.
FloodLight, (accessed March 2015)

[8] https://github.com/mbredel/floodlight-olimps.
OLiMPS Project, (accessed March 2015)

[9] http://www.noxrepo.org/.
NOX Controller, (accessed March 2015)

[10] https://www.opennetworking.org/images/stories/downloads/sdn-resources/onf-specifications/openflow-config/of-config-1.2.pdf.
OF-CONFIG, (accessed March 2015)

[11] https://tools.ietf.org/html/rfc7047.
RFC 4047. The Open vSwitch Database Management Protocol. RFC 4047. (accessed March 2015)

[12] http://www.cisco.com/c/en/us/products/software/one-software/index.html.
Cisco ONE, (accessed March 2015)

[13] http://www.cisco.com/c/en/us/products/routers/wan-automation-engine/index.html.
Cisco WAN automation engine, (accessed March 2015)

[14] Hassan, N., Baig, A., Qadir, J., & Din, I. (2011, December). Recovery and bandwidth sharing techniques in MPLS networks. In High Capacity Optical Networks and Enabling Technologies (HONET), 2011 (pp. 193-200). IEEE.

[15] S. Sharma, D. Staessens, D. Colle, M. Pickavet and P. Demeester, "'OpenFlow: meeting carrier-grade recovery requirements," Comput.Commun.

[16] http://www.openflow.org/documents/openflow-spec-v1.1.0.pdf.
OpenFlow switch specification v1.1.0, (accessed March 2015)

[17] A. Elwalid, C. Jin, S. Low, and I. Widjaja, "MATE: MPLS adaptive traffic engineering," in Proceedings of the twentieth Annual Joint Conference of the IEEE Computer and Communications Societies INFOCOM 2001, vol. 3, 2001, pp. 1300–1309.

[18] M. Suchara, D. Xu, R. Doverspike, D. Johnson, and J. Rexford, "Network Architecture for Joint Failure Recovery and Traffic Engineering," in Proceedings of the ACM SIGMETRICS Joint International Conference on Measurement and Modeling of Computer Systems, ser. SIGMETRICS'11. New York, NY, USA: ACM, 2011, pp. 97–108.

[19] S. Jain, A. Kumar, S. Mandal, J. Ong, L. Poutievski, A. Singh, S. Venkata, J. Wanderer, J. Zhou, M. Zhu, J. Zolla, U. H¨olzle, S. Stuart, and A. Vahdat, “B4: Experience with a Globally-deployed Software Defined Wan,” in Proceedings of the ACM SIGCOMM 2013 Conference on SIGCOMM, ser. SIGCOMM ’13, New York, NY, USA, August 2013, pp. 3–14.

[20] http://www.paloaltonetworks.com/products/features/policy-control.html.
Palo Alto Networks Secure Application Enablement, (accessed March 2015)

[21] W. Kim, P. Sharma, J. Lee, S. Banerjee, J. Tourrilhes, S. Lee and P. Yalagandula, '"Automated and scalable QoS control for network convergence," Proc.INM/WREN, vol. 10, pp. 1-1.

[22] H. Kudou, M. Shimamura, T. Ikenaga and M. Tsuru, '"Effects of routing granularity on communication performance in OpenFlow networks,", pp. 590-595.

[22] C. Yu, C. Lumezanu, Y. Zhang, V. Singh, G. Jiang and H.V. Madhyastha, '"FlowSense: Monitoring Network Utilization with Zero Measurement Cost,", pp. 31-41.

[23] C. Argyropoulos, D. Kalogeras, G. Androulidakis and V. Maglaris, '"PaFloMon--A Slice Aware Passive FlowMonitoring Framework for OpenFlow Enabled Experimental Facilities,", pp. 97-102.

[24] http://www.getpostman.com/.
Postman REST API, (accessed March 2015)

Glossary

BoD	Bandwidth on Demand
DSM	DynPaC Service Manager
DynPaC	Dynamic Path Computation
GOFF	Géant OpenFlow Facility
GTS	Géant Testbed Service
GUI	Graphic User Interface
MPLS	Multi-Protocol Label Switching
ODP	Open Daylight Project
OSCARS	On-Demand Secure Circuits and Advance Reservation System
OVS	Open vSwitch
OVSDB	Open vSwitch DataBase Management Protocol
PCE	Path Computation Element
PCEP	Path Computation Element Protocol
SDH	Synchronous Digital Hierarchy
SDN	Software-Defined Networking
SLA	Service Level Agreement
SNMP	Simple Network Management Protocol
TPA	Traffic Pattern Analyzer
WAN	Wide Area Network