D4.5 “Implementation of sub-lambda assignment element”

Status and Version:		
Date of issue:	31.01.2012	
Distribution:	Public	
Author(s):	Name	Partner
Rahimzadeh Rofoee	Bijan	UEssex
Yixuan	Qin	UEssex
Yan	Yan	UEssex
Georgios	Zervas	UEssex

Contents

1 Executive summary

This deliverable reports the on the implementation of the SLAE tool which allocates resources of wavelength and time slices to the requests on demand. The implementation of the tool comes following our simulation studies of OBST(TSON) network conducted during D2.3, where a number of algorithms for time slice allocation had been introduced and put in to evaluations. This report includes sections to cover the software design and development. Moreover the tool placement in the TSON node is discussed. The structure of the document is as follows:

Chapter 2 gives a brief introduction on purpose and scope of this implementation. Chapter 3 starts by giving information on the resource modelling in MAINS, and explains the sub lambda assignment algorithms.

Chapter 4 explains thoroughly the developed Route Wavelength Time Assignment (RWTA) tool, including the design and implementation and interfacing with GMPLS. The RWTA tool uses Multi-Wavelength First-Fit allocation scheme introduced in D2.3. An alternative to RWTA, called Route Spectrum Assignment (RSA), which allocates frequency slices as resources instead of wavelengths and time slices, is also presented. The purpose of the RSA tool(which uses First-Fit Contiguous algorithm) is to propose ways to interconnect metro and core networks by use of Flex-Grid technologies while providing fine-granular allocation on the time domain (to support sub-wavelength flows) by use of RWTA.

Chapter 5 reports on the implementation of a java based emulation system which basically emulates the intensive metro network requests demands for various data rates and under different Erlangs, so to evaluate the performance of the path finding and resource allocation engines of RWTA and RSA on TID Metro network reference topology.

Chapter 6 explains how the developed SLAE operates in the TSON node in a more general view to the implemented engine.

Conclusion wraps up the discussions and summarise the developments. Besides, some further information on different sections is provided as appendices.

2 Introduction

2.1 Purpose and Scope

This deliverable explains the sub-lambda assignment engine (SLAE) final design and implementation details. The SLAE will be used in correspondence with the GMPLS PCE (path computation engine) to provide end-to-end sub-lambda paths over the TSON metro network. In this regard, and in addition to the SLAE, the developed software program performs routing calculation using path finding algorithms. Therefore the developed resource allocation engine is considered as a Route, Wavelength, and Time assignment tool. In other words, the developed SLAE is located inside the RWTA tool, which is implemented to perform routing and SLAE and can in turn act as PCE and SLAE.

Following this approach, the implemented RWTA tool will be used as a whole or just the SLAE part of it in integration within the control plane.

Apart from that, and to address the resource allocation in the frequency domain, the RWTA tool has been modified to represent spectral resources instead of wavelength and time, so to be used in a scenario which slices of spectrum are shared among several connections.

To assess the usability and performance of the developed tools, a set of evaluation experiment and emulation of numerous requests to the tool have been carried out.

2.2 Reference Material

2.2.1 Reference Documents

[1] http://www.jgrapht.org

[2] http://www.jgrapht.org/javadoc/

[3] MAINS deliverable D2.3_public: OBST mesh metro network desing and control plane requirements, July 2011

[4] MAINS deliverable D2.3_private: OBST mesh metro network desing and control plane requirements, July 2011

[5] Georgios S. Zervas, Joan Triay, Norberto Amaya, Yixuan Qin, Cristina Cervelló-Pastor, and Dimitra Simeonidou, "Time Shared Optical Network (TSON): a novel metro architecture for flexible multi-granular services," OSA Optics Express, vol. 19, no. 26, pp. B509-B514, 2011

[6] MAINS deliverable D3.3: Control plane extensions for GMPLS, Dec 2010

[7] MAINS deliverable D3.4: implementation of control plane extensions, Dec 2011

[8] MAINS deliverable D2.4: Design of XML#2 OBST Network Control Interface with API definitions, Sep 2011

2.2.2 Acronyms

CP	Control Plane
ERO	Explicit Route Object
FF	First Fit
FFC	First Fit Continuous
FFCT	First Fit Continuous Tuneable
FFT	First Fit Tuneable
FPGA	Field Programmable Gate Array
GE	Gigabit Ethernet
GMPLS	Generalised Multi Protocol Label Switching
JNI	Java Native Interface
KSP	K Shortest Paths
MNSI-GW	Mains Network Service Interface- Gate Way
MWFF	Multi Wavelength First Fit
OSPF-TE	Open Shortest Path First - Traffic Engineering
PCE	Path Computation Element
QoS	Quality of Service
RSA	Route Spectrum Assignment
RSVP-TE	Resource Reservation Protocol-Traffic Engineering
RWTA	Route Wavelength Time Assignment
SLAE	Sub-Lambda Assignment Engine
TNRC	Transport Network Resource Controller
TSON	Time Shared Optical Network
UML	Unified Modelling Language
VM	Virtual Machine

2.3 Document History

Version	Date	Authors	Comment
0.1	15/01/2012	Bijan R.Rofoe	First ToC draft
0.2	21/01/2012	Bijan R.Rofoe	First document draft
0.3	23/01/2012	Yixuan Qin	Review
0.4	25/01/2012	Bijan R.Rofoe	ToC updated, section 6 created, merged section 4 and 5.

0.5	27/01/2012	Yan Yan	Review
0.6	27/01/2012	Yixuan Qin	Review
0.7	29/01/2012	Bijan R.Rofoe, Yixuan Qin	Review and modifications
0.8	30/01/2012	Geroge Zervas	Final review
1.0	31/0112012	Bijan R.Rofoe	Release

3 TSON resources and the allocation algorithm design

This chapter will discuss the resource abstraction and representation defined for TSON operation in MAINS. Afterwards, the algorithms used for the sub-lambda resource assignment are explained. These algorithms carry out the core functions of resource allocation over the TSON network. Also, the routing functionality in the RWTA tool is implemented using a java library which is addressed here as well. The set of employed java libraries used for developing the tool is listed in Appendix I.

3.1 Resource abstraction in TSON

TSON network offers a hierarchy of three levels of granularity of resources: 1- connections, 2- frames, and 3- time slices, as illustrated in Figure **3-1**. By connection, it is referred to a communication session between any two end points in the TSON domain. In order to improve statistical multiplexing of data units, each connection, is split into a number of fixed size frames. The frames are considered to be of 1 msec length in the MAINS context. Each frame afterwards is divided to time slices as the smallest units of communications i.e. the actual sub-lambda resources. The frame length, and the number of time slices inside a frame, defines the minimum granularity achievable by the TSON network. TSON can establish as fine as 100Mbps connections over 10GE links, using frames of 1 msec length, and 100 time slices per frame.

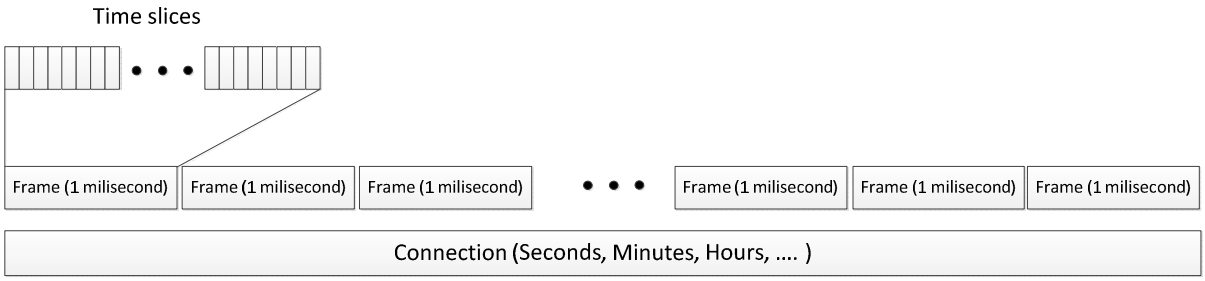


Figure 3-1: Resource abstraction in TSON networks

3.2 Resource representation

The abstracted resources information needs to be represented in a data structure for respective path computations. In this regard, the frame information per link is encoded in a matrix: rows represent the wavelengths, and columns represent the time slices. The aggregation of all the link matrices forms the TSON resource database.

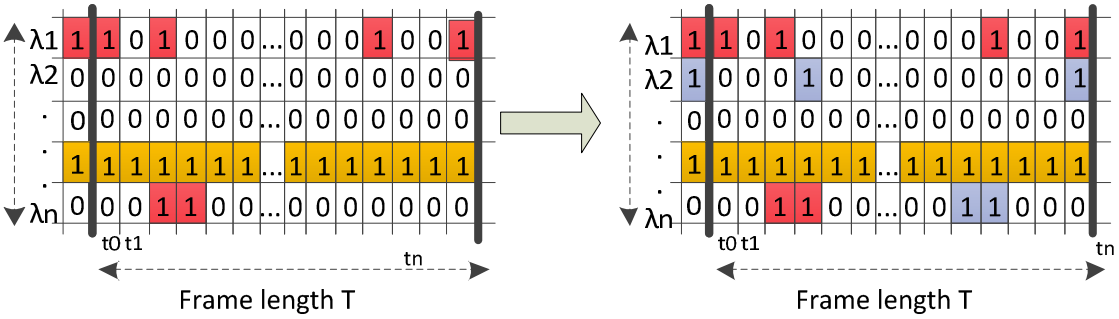


Figure 3-2: Resource (time slice) availability and occupancy are represented by 0 and 1. The left part shows a link information matrix prior to a new connection, and the right part shows how the link resource are used to accommodate new connections

Allocated time slices are filled with “1”, and the available time slices are indicated by “0”. These matrices are used by the PCE in order to accommodate the new connections on the free slices, so that the contention is avoided by time multiplexing the traffics for different requests. Figure **3-2** is showing how a link already occupied with a number of connections (colours red and orange), is used for accommodating new connections (blue) on the available slices.

3.3 Route finding

The first step in resources allocation is to find a route between the requests end points. In this regard we are using a K-shortest path finding algorithm which is based on Bellman Ford's shortest path algorithm [1]. Having K-shortest path decreases the chances a connection being denied because of lack of sufficient resource for the connection. This means less blocking, in the expense of more calculation time.

The algorithm is available from the Java jGraphT library [2]. The number of routes is fed as an argument to the algorithm which is used to find the shortest paths based on the number of hops between any two nodes. However it should be noted the calculated K-paths are not necessarily disjoint.

The use of route finding capability of the RWTA tool is optional, and it can be used in case of lacking a PCE based route calculation engine.

3.4 Heuristics for wavelength and time assignment:

In order to serve the requests over the links, we have designed a set of heuristic algorithms during D2.3 simulation studies of the TSON sublambda network, which operate on the accumulated occupancy of the link matrices. The matrices, indicating wavelength and time slice availabilities, from all the links involved on the possible communication routes are fed into the algorithms. For the tool operations, we have used the MWFF algorithm since in the data plane we were using a number of SFP+ modules per each node, so we were able to simultaneously operate over multiple wavelengths, so considering MWFF proven superiority in performance against the rest of the algorithm family it was implemented in the

1 from [2]"*The algorithm is a variant of the Bellman-Ford algorithm but instead of only storing the best path it stores the "k" best paths at each pass, yielding a complexity of O(k*n*(m^2)) where m is the number of edges and n is the number of vertices."*

RWTA tool. In the following MWFF has been explained, along with its counterpart algorithm once introduced in D2.3.

In this regard, we introduce five different algorithms for wavelength and time assignment.

1) Multi-wavelength non-contiguous First-Fit (MWFF). In this algorithm any allocation of time-slices over any of the wavelengths is possible. Considering the resource abstraction and representation explained in section 3.1 and 3.2, the algorithm operates on a matrix filled with zeros and ones. The algorithms will start from the first wavelength on top, and explore from left to the right of the row allocate the slots.

As shown in Figure 3-3*, the MWFF is the least restrictive time-slice assignment policy developed. In MWFF, the slices pertaining to a sub-wavelength lightpath can be scheduled all over the link resources matrix on different wavelengths and not necessarily within a continuous slice set, including different wavelengths (multi-wavelength) on the same slice index.*

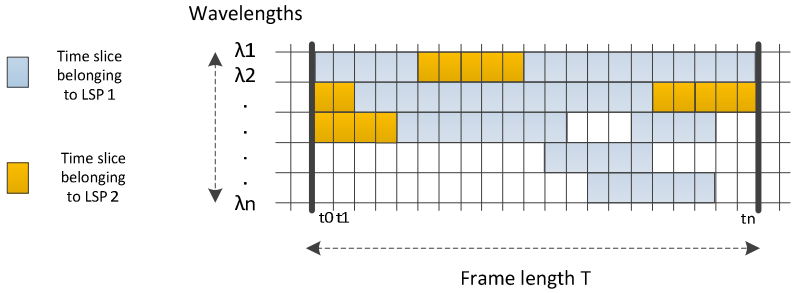


Figure 3-3: MWFF (allocation in blue and orange)

2) First-Fit (FF) solution for single wavelength non-contiguous slice/time assignment. Considering the resource abstraction and representation explained in 3.1 and 3.2, the algorithm operates on a matrix filled with zeros and ones. The algorithms will start from the first wavelength on top, and explore from left to the right of the row to accommodate the slices. However, in this algorithm, the connection can only use a single wavelength, but assigned slices do not need to be contiguous. The allocation strategy is visualised in Figure **3-4**.
Unlike MWFF, this algorithm is a single wavelength time slice allocation solution, where the new LSP time slices are allocated across an identical wavelength.

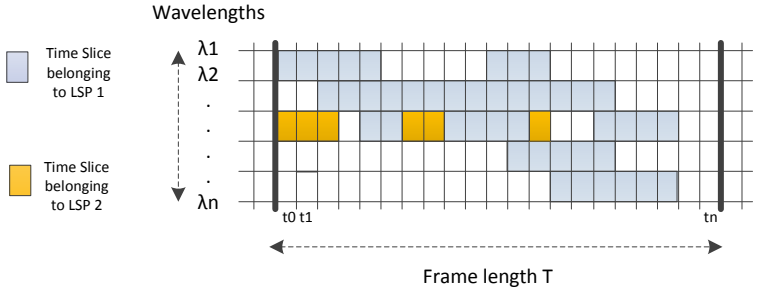


Figure 3-4: FF (allocation in blue and orange)

3) First Fit Contiguous (FFC) for single wavelength contiguous slice/time assignment. Similar as FF, but time-slices should be assigned in a contiguous manner. Considering the resource abstraction and representation explained in Section 3.1 and 3.2, the algorithm operates on a matrix filled with zeros and ones. The algorithms will start from the first

wavelength on top, and explore from left to the right of the row to accommodate the slots. The allocation strategy is visualised in Figure **3-5**,

For the FFC policy, a new requirement is given: the slice allocation for a connection needs to be contiguous; hence the sub-wavelength (requested bandwidth) cannot be split along the matrix/frame horizon.

This algorithm differs from MWFF in that the allocated slices should be contiguous. This constraint greatly reduces the efficiency levels of the algorithm since it should find the number of slices needed altogether.

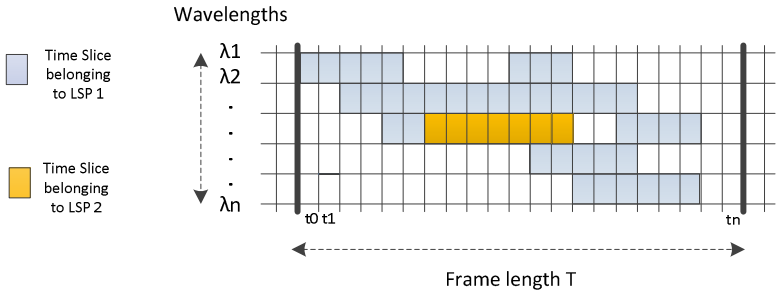


Figure 3-5: FFC (allocation in blue and orange)

4) First Fit Tuneable (FFT) for non-contiguous slice/time assignment with tuneable wavelength. It works as MWFF, but the same slice index cannot be used on different wavelengths. Considering the resource modelling and representation explained in Section 3.1 and 3.2, the algorithm operates on a matrix filled with zeros and ones. The algorithms will start from the first wavelength on top, and explore from left to the right of the row to accommodate the slots. The allocation strategy is visualised in Figure **3-6**.

The FFT policy introduces a new constraint. In this policy, connections are restricted to different slices, over a single or multiple wavelengths, i.e., two simultaneous slices on different wavelengths are not allowed. The FF policy is similar to FFT but with the added constraint to schedule the slices for a sub-wavelength connection on the same wavelength. To this end, a new variable and a couple of constraints are added to the formulation.

FFT is different from MWFF since although it can operate over multiple wavelengths, however overlapping of time slices over multiple wavelength would not be possible. A use case scenario for this algorithm is when one tunable transmission system is used.

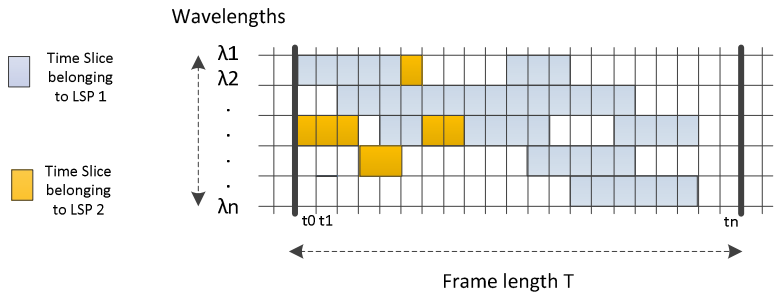


Figure 3-6: FFT (allocation in blue and orange)

5) First-Fit Contiguous Tuneable (FFCT), for assigning contiguous time slices with tuneable wavelength selection. It applies the FFT but assigned slices have to be contiguous in time. Considering the resource abstraction and representation explained in Section 3.1 and 3.2, the algorithm operates on a matrix filled with zeros and ones. The algorithms will start from the first wavelength on top, and explore from left to the right of the row to accommodate the slices. The allocation strategy is visualised in Figure **3**-**7**.

The latest policy restricts the assignment along a contiguous set of time-slices, but allows switching to another wavelength from slice to slice.

This algorithm shows more constraints in comparison to MWFF, since first of all the allocated slices over each wavelength should be continuous, and also, the transmission over multiple wavelength but identical time slices is not possible.

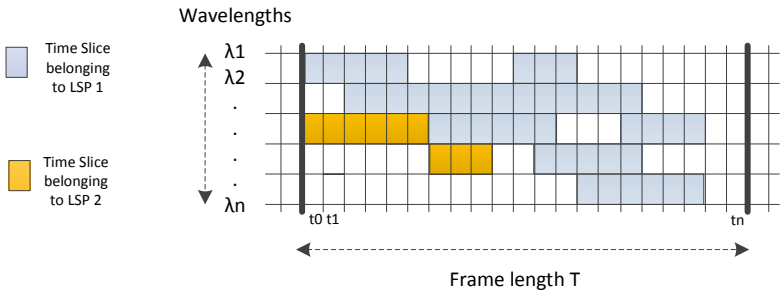


Figure 3-7: FFCT (allocation in blue and orange)

Each of these algorithms represents a specific way of allocating the resource, following different constraints. The works in [3], [4] and [5] give the detailed explanations and mathematical modelling of the introduced algorithms.

4 Route wavelength time assignment tool:

As of now, a RWTA engine is developed comprising of routing and sub-lambda assignment algorithms to perform functionalities of the PCE and SLAE. This tool executes route and sub-lambda assignment in sequence. Therefore the routes found in the first stage are used for feeding the algorithms in the second stage. It takes advantage of KSP route finding algorithm for setting up paths, and it uses MWFF for wavelength and time assignment engine to allocate time slices. The use of MWFF is justified by using multiple tuneable or fixed transmitters in the data plane.

4.1 Challenges and strategies

The central sub-lambda assignment engine is in correspondence with GMPLS PCE. Whether using the RWTA routing mechanism, or exploiting an external PCE, different combinations of SLAE and PCE are possible as it has been specified between NXW and UEssex (different combinations in Appendix II). In any case, the route finding and the sub-lambda assignment engine will be either in the same virtual machine, or on the separate ones. Regarding the first solution, NXW explains the communication between the PCE and SLAE being in the same virtual machine:

- Requirements:
  - A user defined computation code wrapper
    - To let the TSON SLAE tool be considered as a “common” user defined computation code
    - To offer the dynamic loading functions towards the PCE core
    - To implement the two computation phases (Load+Request) of a user defined computation code
    - To translate and bind the Load+Request procedure in the native TSON SLAE computation procedure
  - A Java Native Interface (JNI)
    - To translate the internal PCE core topology data structures in the correspondent TSON SLAE tool topology data structures (i.e. nodes, links, wavelengths, etc.) and vice versa
    - To translate the TSON SLAE tool explicit route object (ERO) format in the correspondent internal PCE ERO format (i.e. sequence of links, wavelengths, time-slices) and vice versa

Alternatively and as the second architecture, the GMPLS PCE and SLAE may be deployed on two separate virtual machines. The communication between the PCE and the SLAE takes place through the use of one Transport Network Resource Controller (TNRC) Specific Part (i.e. TNRC SP) for GMPLS CP interactions with the TSON controller through the XML#2. In this trend, two way communications are to be deployed using clients and servers on the two virtual machines.

The implementation needs and the possible issues are explained in the following:

- Requirements:
  - A user defined computation code wrapper

- To let the TSON SLAE tool be considered as a “common” user defined computation code
- To offer the dynamic loading functions towards the PCE core
- To implement the two computation phases (Load+Request) of a user defined computation code
- To translate and bind the Load+Request procedure in the native TSON SLAE computation procedure

– A dedicated interface to let MAINS PCE and TSON SLAE interact and cooperate

- E.g. Socket based interface implemented by the wrapper
- Serialization/De-serialization of data structures
- Translation of the internal PCE topology and ERO data structures

Following the interfacing requirements between the SLAE tool, and the GMPLS stack, UEssex and NXW have parameterised the communication, which is provided in Appendix III.

The developed tool uses the heuristics explain in Section 3.4 to provide connectivity over the links in the TSON architecture. However, two constraints of wavelength continuity and time-slice continuity in resource allocation over the paths should be taken in to account to address the SLAE and routing in practice.

Wavelength continuity: since in the TSON data plane there are no wavelength converters envisaged on the intermediate nodes, the light paths established between any source and destination should stay on the provisioned wavelengths.

Time-slice continuity: in regards to maintain minimum implementation complexity and avoid optical buffering, nodes need to transparently switch the traffic on a very timely basis. Nodes should get synchronised by compensating the propagation and process delays between them. Assuming the nodes are synchronised the allocated time-slices will be the same on all the links for one connection.

4.2 RWTA Design

The work flow of the tool is demonstrated in the Figure 4-1. The RWTA tool has to interact with the GMPLS/PCE stack. This interaction is explained in the following.

- **Request from GMPLS:** RWTA tool is being invoked by GMPLS through the XML interface. A number of arguments are needed for the operation of the tool, which are:
  - Network topology matrix
  - Number of wavelengths per link
  - Number of time slices per each wavelength
  - Source node
  - Destination node
  - Bit-rate request(in form of number of time slots)
  - The number of path for KSP

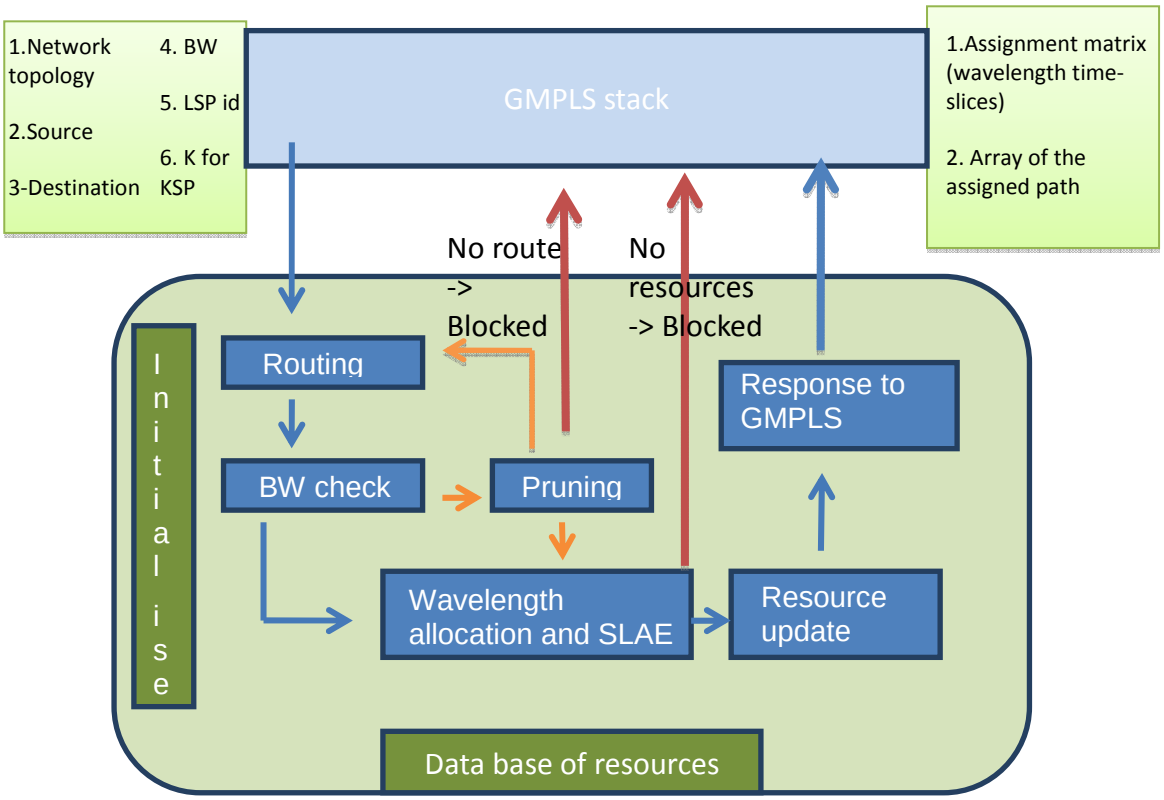


Figure 4-1: SLAE+PCE tool in interaction with GMPLS stack

- **Routing**: The route finding engine, exploiting the KSP algorithm, will provide the requested number of routes between the source and destination nodes using the imported topology. Routes are calculated and sorted based on their number of hops.
- **Pruning:** These routes however, might use links which have already run out of the availabilities due to the prior allocations. To address this issue and to avoid a congestion, a mechanism has been deployed which checks the route's links aggregated availabilities, and if the route meets the minimum required number of time slices, then it will be used in the algorithm, otherwise the path calculator will prune the topology, and recalculate the paths. This mechanism however, depends highly on the network offered load and topology, and because of so, the time for recalculations can cause unacceptable delays. To oversee this issue, a watchdog timer has been also deployed to stop the recalculation mechanism, and to invoke the algorithms in case there were any routes available.
- **Sub-lambda assignment:** In the next stage, the obtained routes will then be fed into the algorithms for allocating the time slices. In this stage, based on which algorithm is chosen to be used in the tool, wavelengths and time slices will be allocated.
- MWFF allocation algorithm has the most flexibility to accommodate the request over the wavelengths. In this regard, this algorithm will search through the intermediate links on the path, and assign the resources.
- **Updating the Database:** The allocation of the resources needs to reflect on the data base. In this regard if the allocation was successful, prior to communicating the results back with the GMPLS, each links table involved will be updated

- **Response to GMPLS:** The information of routing and the allocated resources are sent back to the GMPLS stack so it can setup the path.

4.3 RWTA Implementation

In order to achieve the logical performance explained in the previous section, several Java classes and methods have been developed. The Figure 4-2 displays a UML diagram made of some of the essential classes defined inside the tool.

Basically, the tool is programmed in following the categories listed below:

- Main class
- Utilities classes
- Network classes
    - Link class
    - Node class
- Routing classes
- Resource classes
- Algorithms and sub-lambda assignment classes

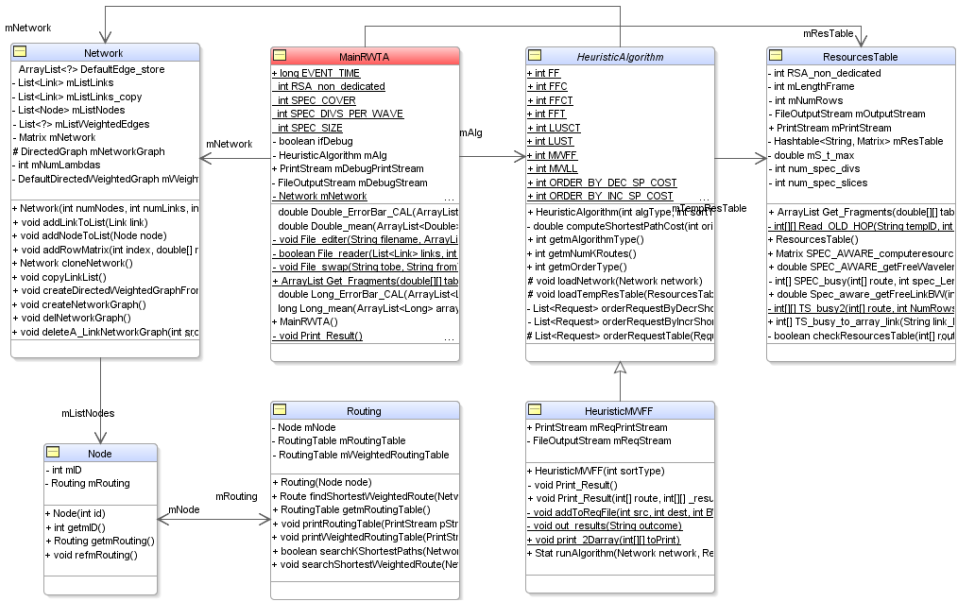


Figure 4-2: RWTA tool UML diagram. The main class called MainRWTA, with the supplementary classes for the RTWA functioning and their interconnections is shown.

The **Main** class which is named MainRWTA.java contains the main method of the RWTA for start and ending of the program. It also manages the inputs and outputs to the program. The inputs are arguments passed to the program before the run time, as explained in Section 4.2. The main class retrieves the information from the arguments, and then invokes the essential classes to generate the required software objects for setting up the scenario. In order to carry out some of the required methods for communication of the program with the external database, and also to define some necessary functions, a set of classes are defined as the utilities to facilitate the work on different stages of the program run time.(Figure 4-3).

Apart from running resource allocation algorithms for accommodating different connectivity requests, the main class has been extended with options for ending a connection, restarting the database (which is basically is setting all the allocation in the database to zero, and removing all the records) or providing network utilisation. These functionalities are invoked by command arguments from GMPLS control plane. The RWTA tool holds a local record of all the allocations made, per each link. All the recorded allocation events are labled with distinct IDs, which come from GMPLS. So upon GMPLS request on removing a connection with a corresponding ID, the RWTA clears the allocations for that specific record. Deliverable of D3.5 discusses this from GMPLS point of view.

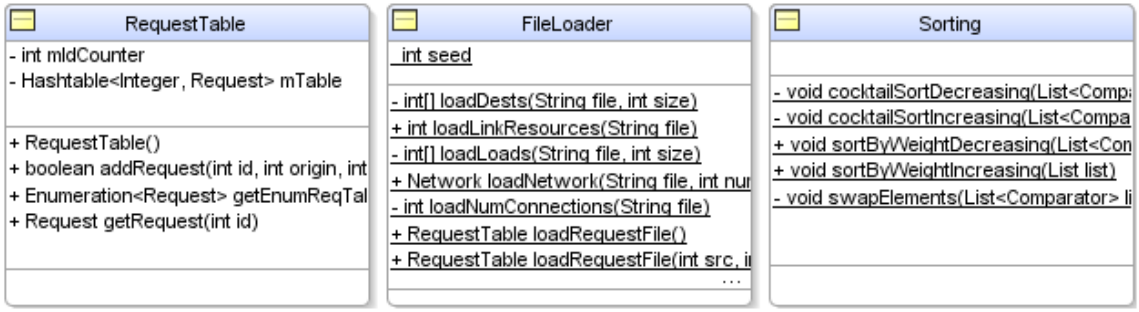


Figure 4-3: Utilities classes and methods

Network class is one of the first classes, which is called by the main class to create a network instance based on the topological information passed to the node. The network class will generate links and nodes using the abstract classes defined for this purpose. The links and nodes classes will then be identified and will contain the required information for handling the traffic. Network class is illustrated in Figure **4-4**.

In order to keep track of the resource availabilities on the links and to consume this information each time the allocation engine is being invoked a set of text files are considered for keeping the information externally to the program. Besides the advantage of sustaining the information between each call to the RWTA program, other engines such as the PCE can have access to the utilisation information to perform resource-aware path computation. The resource representation on the data base is based on the availability matrices per each link. As explained in Section 4.2 the resources matrices of the database are being updated each time an allocation or deletion of the request happens. To handle the corresponding tasks a class called ResourcesTable has been implemented. This class upon instantiation refers to the database to create an updated resource table.

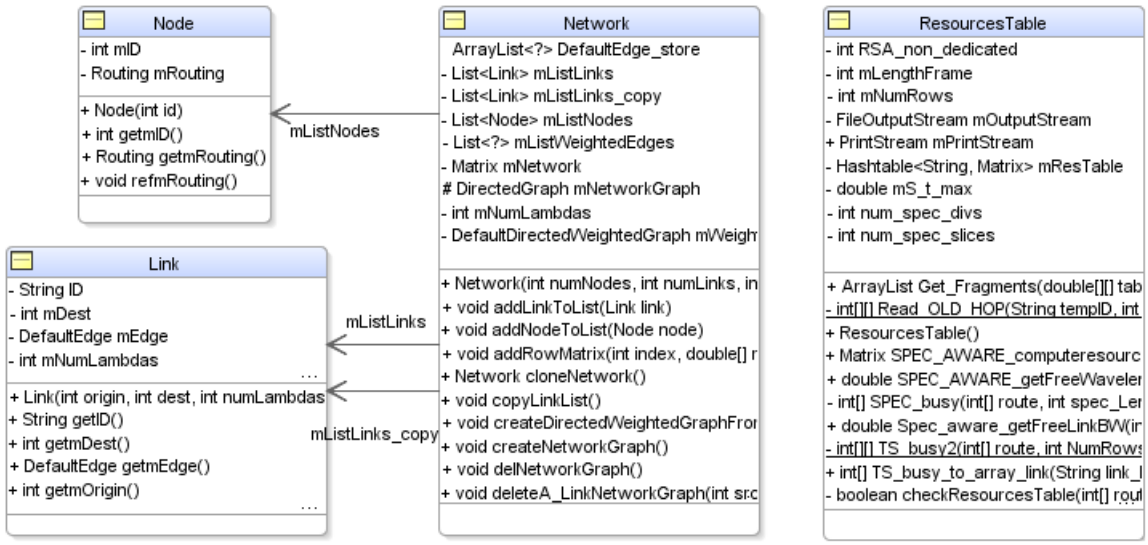


Figure 4-4: The figure shows the network class in the middle, which creates objects of nodes and links after the initialisation, using the topological information. ResourcesTable is another object class which is built using the resource availability information of the database.

After the network instance is initialised and network elements are generated, the main class invokes the routing class to access the routing algorithms. A set of routing algorithms such as shortest path methods of Dijkstra and Bellman Ford have been implemented in this class. However, for the purpose of the MAINS project and to address the need of having multiple paths selected, a K-Shortest Path algorithm is used to find a number of paths between the end nodes. Having multiple paths help to reduce the blocking in the network since the alternative route may exist. This however will come in a cost of longer calculation time. The routing mechanism requires a set of classes such as routing tables, which is implemented as well Figure **4**-**5**.

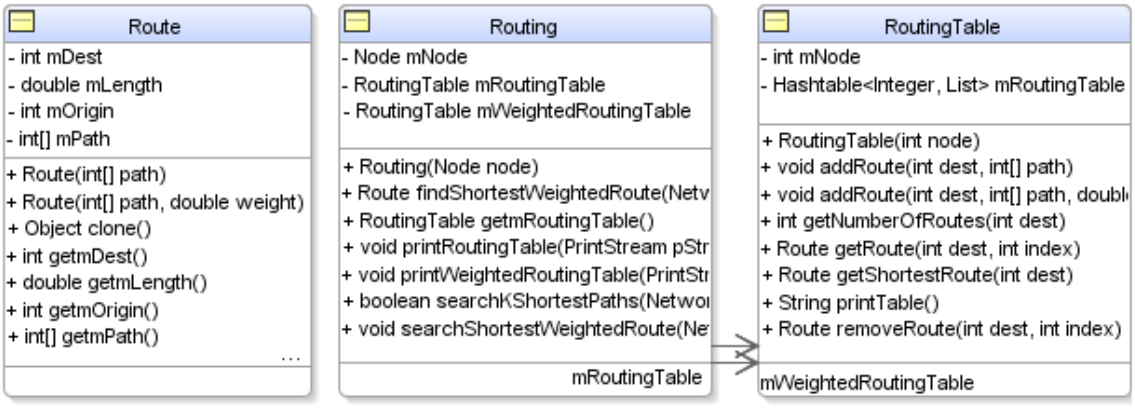


Figure 4-5: Routing related classes and methods

After the routing between the source and the destination is completed and the information is stored, the main class will start the sub-lambda assignment procedure. The routing information in addition to the resources utilisation, are passed to the SLAE. This engine is

consisted of a number of classes for incorporating the set of defined algorithms (explained in 3.4), to execute wavelength and time slice assignments. (Figure 4-6)

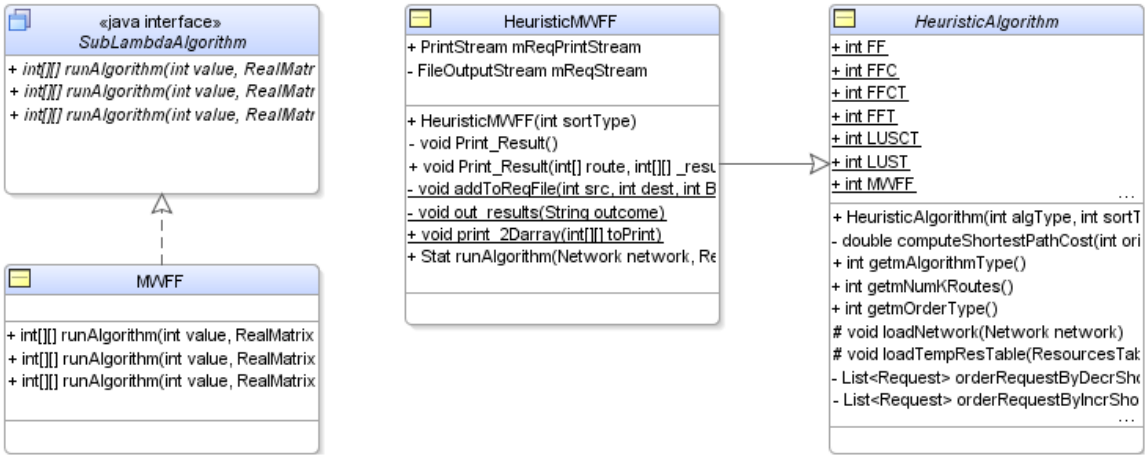


Figure 4-6: Heuristic algorithm class and the corresponding classes which enable the RWTA to run the desired algorithm.

4.4 Alternative to RWTA: routing and spectrum assignment (RSA)

In order to address the emerging sub-lambda/super-lambda switching approaches in the spectrum domain, we have developed an alternative to RWTA, and that's a route, spectrum assignment (RSA) tool.

This tool follows the same architecture as the RWTA: routing based on K-shortest path algorithm to find multiple routes for routing the traffic, and the calling one of the heuristics to allocate the spectrum resources.

4.4.1 Resource representation

Regarding the resources however, RSA uses the spectrum slices and not time slices over the wavelengths. Therefore the database of the RSA uses vectors per each links which represents spectrum availabilities. Based on the spectrum range and the resolution for dividing that range to the spectrum slices, different granularities of band width for communications are obtained. For example, if we consider the C band of the spectrum between 1530nm and 1570 nm, with 5THz available spectrum, 200 slices of 25 GHz, or 400 slices of 12.5 GHz are achievable.

4.4.2 Spectrum assignment

Figure **4-7: RSA workflow**

The work flow of the tool is demonstrated in the. It follows almost the same work flow as RWTA as shown in Figure 4-1, but replacing SLAE module with spectrum slice assignment module. For spectrum assignment, the FFCT algorithm is considered. FFCT as explained in Section 3.4 allocates contiguous slices over the available spectrum as shown in Figure 4-8. It should be noted that since there is only one row, FFC and FFCT will have the same

functionality. Any of the other algorithms can also be applied in the tool, with respect to the technology in hand.

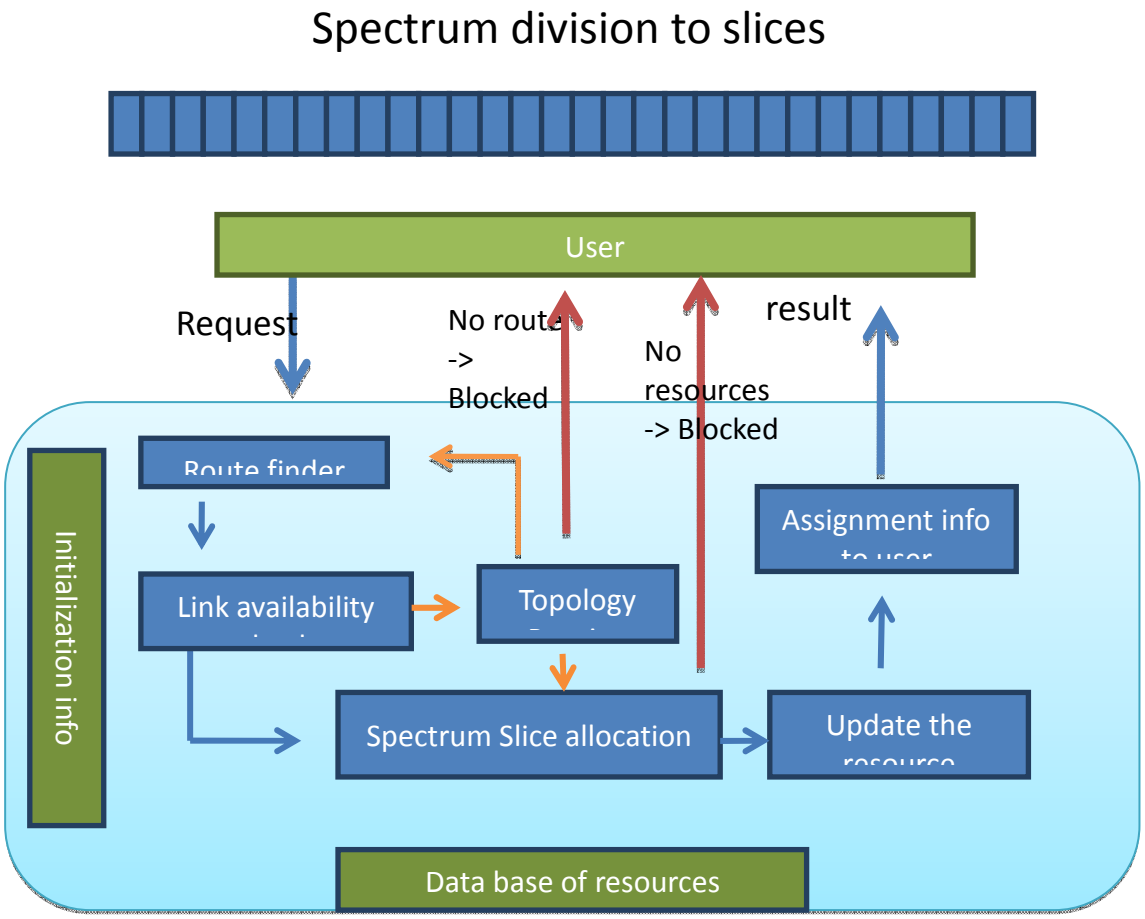


Figure 4-7: RSA workflow

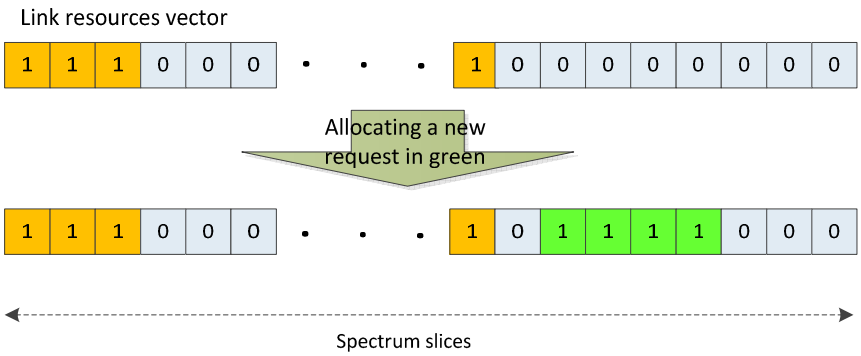


Figure 4-8: instead of having table of resources, will have a vector per each link, representing the spectrum slices on each.

4.5 Release and accessibility

The tool is written in Java JDK 1.6, and is released as a jar file. The jar file can be incorporated into other java based programs, by exploiting the runtime classes available in JDK1.0 onwards. This class includes a variety of methods supporting the execution of Java jar files with different option.

Considering the MAINS context, the developed Java SLAE tool will interface with the developed GMPLS PCE stack. Different options for integrating the RWTA jar file with the GMPLS software stack have been considered; and importing the jar file in the GMPLS stack seems to be the most practicable one. The GMPLS stack is written in C++ and hence, to be able to access java libraries and executables from C++ Java Native Interface seems to be the solution. This solution and the other possible ones are under study by NXW and UEssex.

5 Algorithms, scalability and performance evaluation

In order to evaluate the performance of the path finding and resource allocation engines of RWTA and RSA, a request-maker emulator has been developed using Java language, so to test the performance and integrity of the tool for extensive number of requests, under different Erlangs. The request-maker emulator contains a platform for running the tools for slice allocation, deleting an allocated request, providing utilisation info, and son on. It receives the information to it as arguments. The basic arguments are the one defined for the tools operations. In addition to that, the time of the emulation, and the duration of the communication are passed as well to have their operation synchronised.

5.1 Event based and Real Time based

The request-maker emulator has been implemented with two different approaches.

5.1.1 Event based emulation of multitude of requests to the resource allocation tool for performance evaluation

In this type of request-maker emulator, a counter based clock is considered in the request-maker emulator, to be used for coordination of the events. Based on the clock, events are scheduled and executed. The use of the event based approach helps to understand how the network performs for allocating a considerable numbers of the bandwidth request, using the tools.

Using some distribution functions like Poisson we are able to emulate the arrival of the requests distributed in time, their duration, and termination.

Using the event based request-maker emulator, we have evaluated the performance of the tool in handling 3000 requests over the Madrid metro network topology in Figure 5-1. This topology is composed of 15 nodes, 23 bidirectional links, with a nodal degree connectivity of deg(G) = 3:07, an average link length L = 56:87 km and length standard deviation L = 41:70 (i.e. some links are much longer than others).

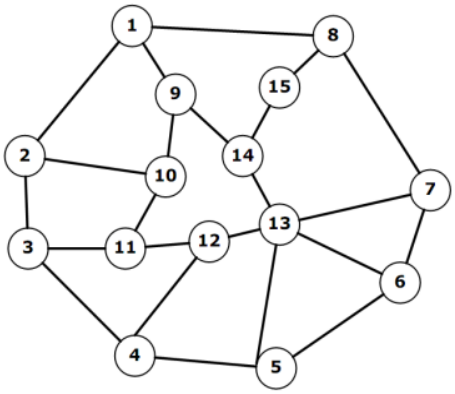


Figure 5-1: Madrid metro network topology

The requests were applied to the network using Poisson model for inter-arrival of the requests of mean 2, and using variable holding time. The results of emulating multitude of requests for RWTA and RSA are shown below:

RWTA tool was tested by applying the requests for 1Gbps, 10 Gbps and 20 Gbps, over the 10G links, with 21 wavelengths per each link. Each wavelength is divided into 100 time slices, each time slice representing 100Mbps. 1Gbps requests were served with 10 time slices over each wavelength, and the 10G and 20G requests were responded by 1 and 2 whole wavelengths if available. Besides, of the all the request, about 89 % were targeting 1Gbps, 8 % were for 10 Gbps, and 2-3 % for 20 Gbps.

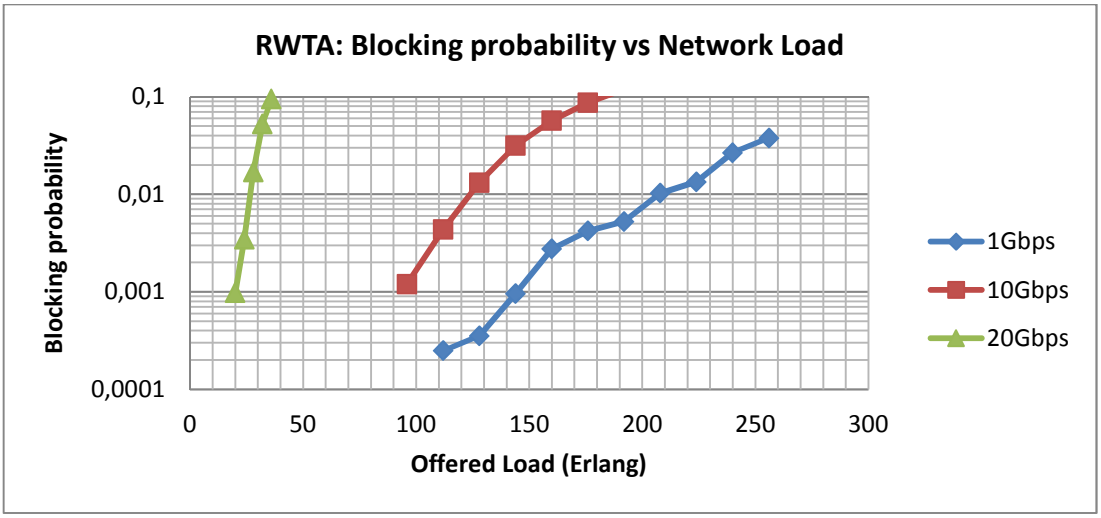


Figure 5-2: Blocking performance of the RWTA tool against different Erlangs of traffic for RWTA tool

The Figure 5-2 displays how the blocking of the request increases by the increase in the offered load. This increase for the 20 Gbps is the steepest since the chances of resource availability drops very fast after each allocation and considering that most of the resources are allocated on 1 Gbps (89%) and 10 Gbps (8%).

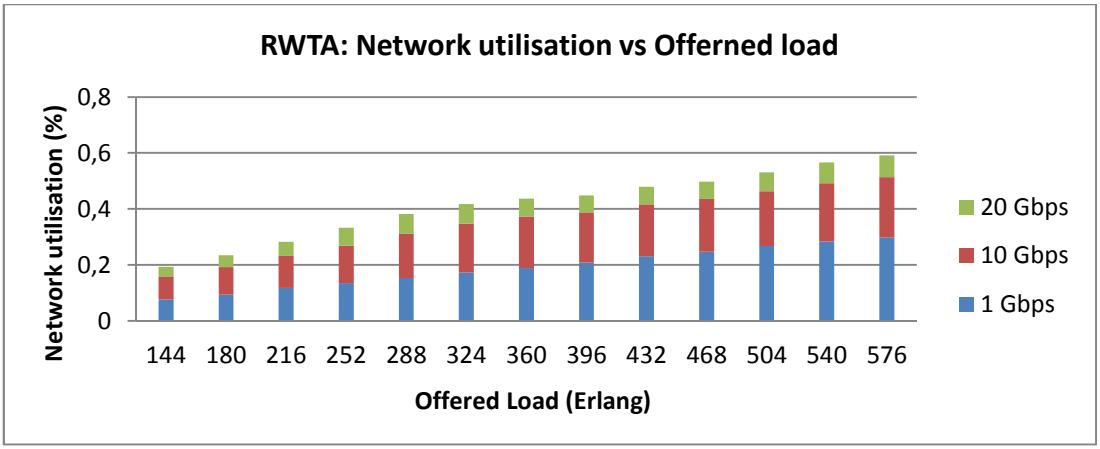


Figure 5-3: Network utilisation of the RWTA tool against different Erlangs

Figure 5-3 shows the occupancy of the network resources by each of the request types. 1Gbps request with the lowest granularity occupy the most as they comprise almost 90% of the request which were made to the network.

For RSA, the same type of results has been collected. The emulation considers 44 slices of spectrum, and the available bandwidth per each spectrum slice is considered 6.25 GHz. The characteristics were considered for the spectrum slices are summarised in Table 5-1.

Bandwidth requested	25 GHz	37.5 GHz
Allocated Spectrum slices	4 slices	6 slices
Requests percentage over all	80%	20%

Table 5-1: Spectrum characteristics

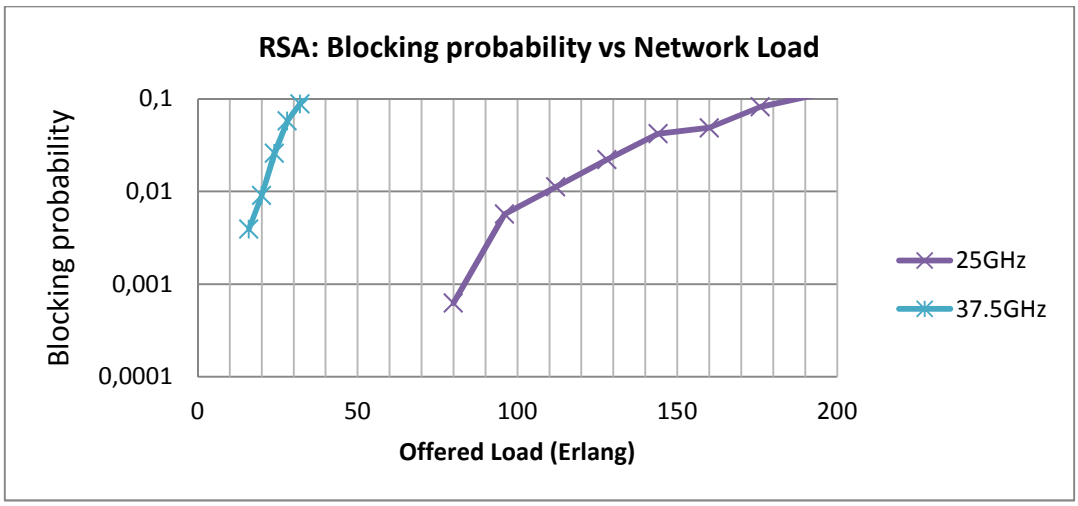


Figure 5-4: Blocking performance of the RSA tool against different Erlangs of traffic for RWTA tool

As it can be observed in Figure 5-4 the allocation of the 37.5 GHz request which requires 6 slices experiences higher blocking probability.

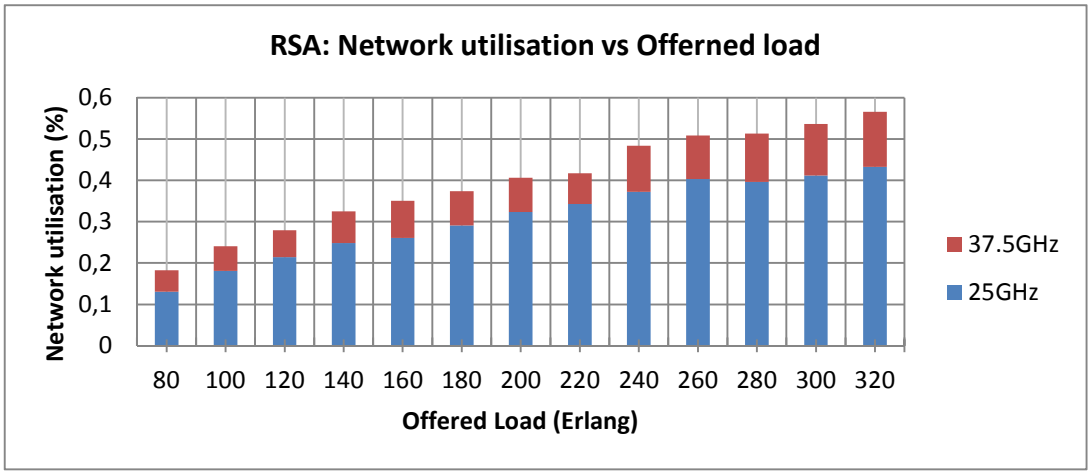


Figure 5-5: Network utilisation of the RSA tool against different Erlangs

Figure 5-5 shows the network utilisation of the applied traffics. The figure reflects applying 80% for 25GHZ requests and 20 % for 37.5 GHz requests.

5.1.2 Time Based Tool:

Since the tool is supposed to work in a real time environment, the request-maker emulator clock has been changed to work with the PCs clock, using the System API. This will enable to test the tool using the system time, which will also take into account the host server performance as well.

Java System clock gives the current time based on the UTC time reference in the mili- and nano- second scales.

5.2 Processing-time performance evaluation

In order to evaluate the RWTA performance in completing a route and sub-lambda assignment task in time, some measurements on the tool have been taken using the System clock API in java.

These measurements targeted the tool timing in whole and the algorithms separately. Also the effect of enabling pruning was also considered besides the non pruned tests.

Figure 5-6 shows the algorithms performance solely. In this figure, the MWFF and FFC algorithms are compared for different number of paths of 1 and 3. **Figure 5-7** shows how the running of the tool with all the inputs and put puts and functions inside affects the time effectiveness of the tool. In this set of experiments, the same algorithms of MWFF and FFC are used for resource allocation. Different number of paths and incorporating a pruning functionality in the tool show different impacts on the tool overall performance.

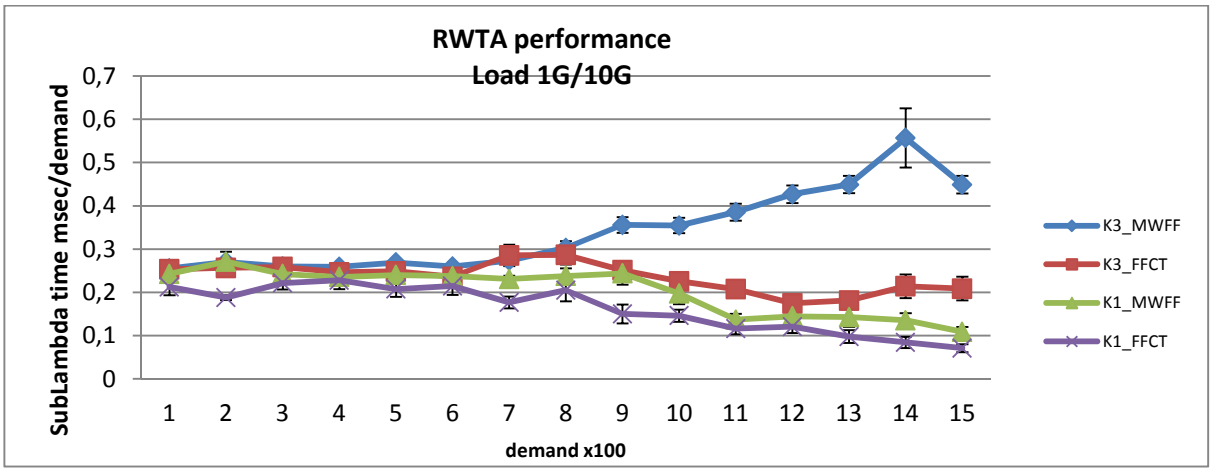


Figure 5-6: This figure shows the performance of the MWFF algorithm against FFCT algorithm.

Different number of K-paths of 1 and 3 has been considered. MWFF will take much higher time in allocating resources especially for the later requests since it will search through the entire available wavelength list on the links over all the paths to finally accommodate the request.

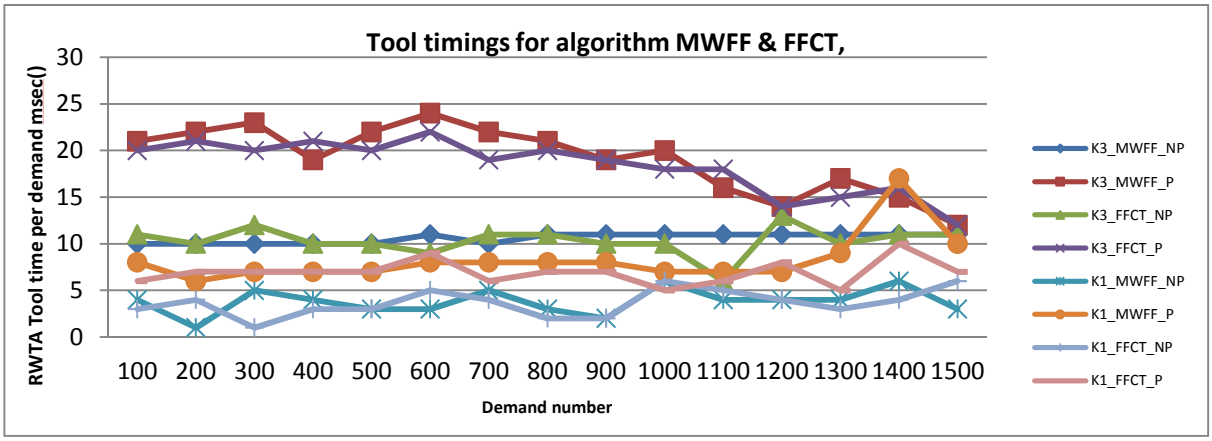


Figure 5-7: This figure shows the performance of the RWTA tool in whole using two different assignment algorithms of MWFF and FFCT, for pruning and non pruning scenarios. It can be seen how the pruning of the routes can increase the latency of the tool in doing the path finding for resources allocation.

6 SLAE in TSON/GMPLS control plane

The SLAE tool has been designed and implemented as the sub-wavelength resource allocation module. This module helps to orchestrate simultaneous bandwidth requests to the networks to gain a contention-less manner. This tool in collaboration with PCE will provide the route, wavelength, and time slices to be used for switching over the data plane.

As it can be seen Figure 6-1 on top of the TSON node, a server is located, hosting a number of virtual machine. GMPLS virtual machine hosts the extended GMPLS stack supporting sub-wavelength switching. The GMPLS required extensions and implementation have been reported in [6] and [7] respectively. A virtual machine with Windows platform has been installed as well to be used for reprogramming the FPGA connected to the server.

GMPLS is responsible to disseminate resource information and availabilities in the network among the TSON nodes. In this regard, GMPLS stack exchanges information with TSON VM. Network resources are defined in the format of XML schemas, and the communication between GMPLS and TSON VM will take place through XML based REST web-service to provide a simple and extensible structure for interoperation of the nodes. The details of the TSON xml schema definitions are provided in [8]. On the other side TSON VM using a TSON driver is in interaction with the FPGA in the lower layer which is employed for the implementing Layer 2 functionalities of TSON. TSON VM therefore bridges between the data plane and control plane.

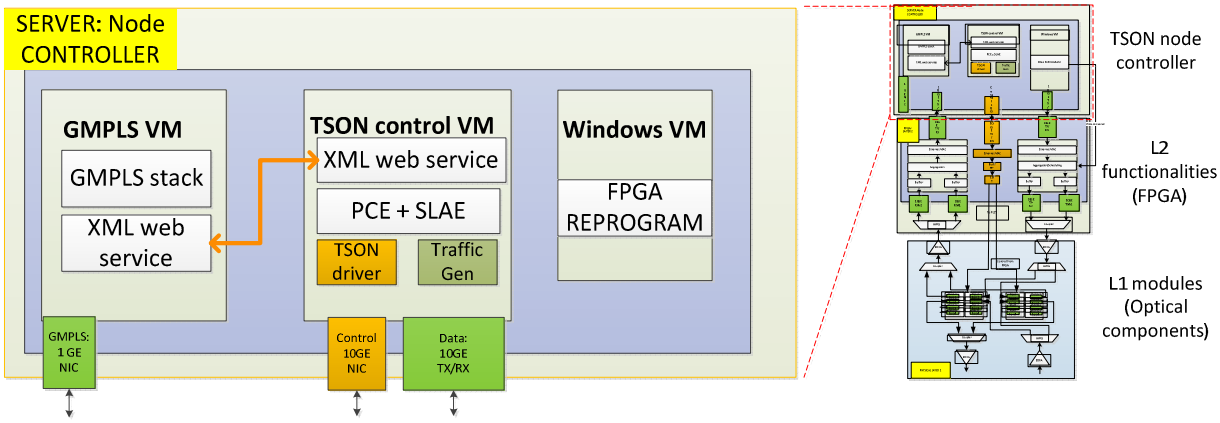


Figure 6-1: Server in the TSON node, and it contents provide the controller entity for the TSON node. Server hosts a number of Virtual Machines for different purposes.

6.1 Centralised controller

TSON network is designed to operate using a centralised controller and resource allocator, considering the limited size of the network and the expected contention less behaviour.

In the TSON node architecture chosen to be the central controller, the SLAE is located in the server on top. The SLAE should interact with the PCE (the Path Computation Element which will be placed in a virtual machine on the server) to serve the requests for connections over the TSON network.

As it can be observed in the Figure 6-1 the resource allocator tool supporting PCE + SLAE, can be located on the TSON control server as an implementation option. However, there are other viable configuration for installing an external PCE beside the SLAE are

understudy by UEssex and NXW. These possible combinations are displayed in Appendix I.

6.2 The path provisioning work flow

For the control and management plane of the network, a centralised approach has been chosen. The process of receiving a connection request until starting the data transmission is displayed in Figure 6-2 and explained in the following:

1. A new request for path set up for across the TSON cloud is received by the MAINS Network Service Interface Gateway (MNSI-GW).
2. Upon receiving the request MNSI-GW will make an enquiry to the central PCE.
3. PCE after computing a path using its database (updated by GMPLS OSPF-TE messages), calls the SLAE for sub lambda resource allocation over the specified routes to a destination node. The SLAE will use heuristics to assign time slices to the request, fulfilling the requested bandwidth. This information is then sent back to the GMPLS stack on the edge node.
4. GMPLS uses the information to initiate the route and reserve the resources on the network ahead of the data transport using the RSVP-TE protocol.
5. The intermediate nodes will use the information circulated by GMPLS to update their databases, and to reserve and schedule their resources for the expected communications.
6. Afterwards, the requesting node or entity is informed about the path setup and the data transport starts.

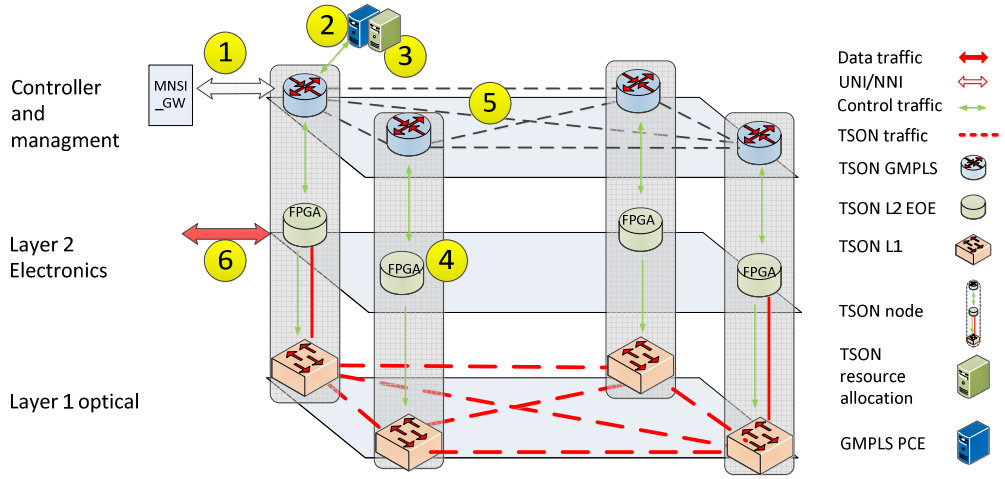


Figure 6-2: Layered TSON network architecture. The numbers in the yellow circles show the step wise flow of the TSON function.

6.3 Host server[2]

In order to deploy virtual machines and higher layer control, DellT701 servers have been purchased with the following characteristic:

- Intel Xeon E5620 Processor (2.40GHz, 4C, 12M Cache, 5.86 GT/s QPI, 80W TDP,
- Turbo, HT), DDR3-1066MHz

[2] picture from http://www.glcomp.com/products/servers/dell-poweredge/tower-servers/poweredge-t710

- 12GB Memory for 1CPU (6x2GB Single Rank UDIMMs) 1333MHz
- 500GB SATA 7.2k 3.5" HD Hot Plug
- PERC H700 Integrated RAID Controller, 512MB Cache
- 16X DVD+/-RW Drive with SATA Cable
- Non-Redundant Power Supply (1 PSU) 1100W
- Rack Power Distribution Unit Power Cord
- Embedded Broadcom GbE LOM with TOE and iSCSI Offload HW Key, NOT
- compatible with H200/H700 on non-56xx bases
- iDRAC6 Express
- Sliding Ready Rack Rails with Cable Management Arm for Rack Configuration C2 - R1 for PERC H700, Exactly 2 Drives

Figure 6-3: Dell T710 server

7 Conclusion

This document reports on the implementation of SLAE, which is also capable of performing routing. The tool capable of operating as a PCE+ and SLAE will be incorporated in the TSON controller and will be accessed by/interfaced with GMPLS for sub-lambda resource allocation over the TSON network.

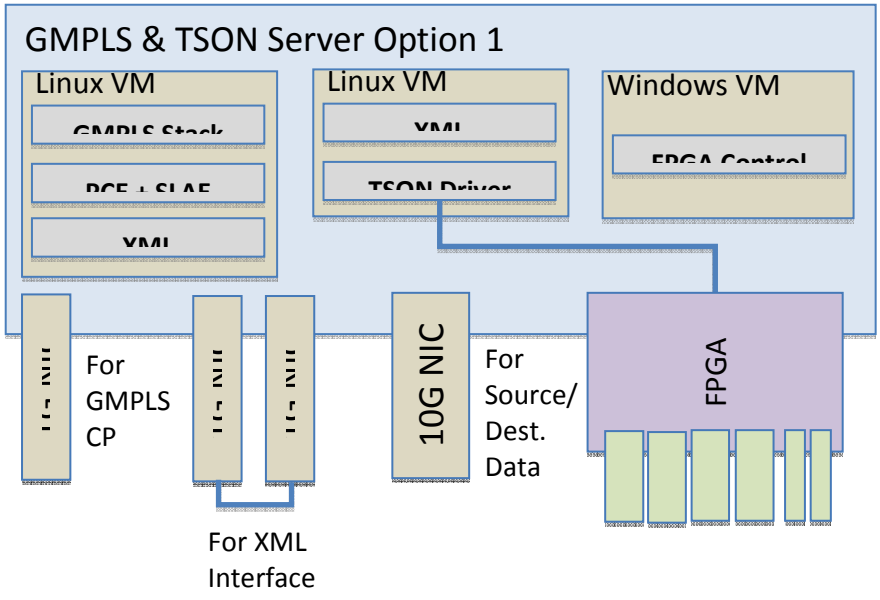
The tool logics and implementations have been explained, and some results regarding its evaluation for networking and timing are also reported. A stable version of the tool is been developed, and it conducts route and sub lambda assignment together.

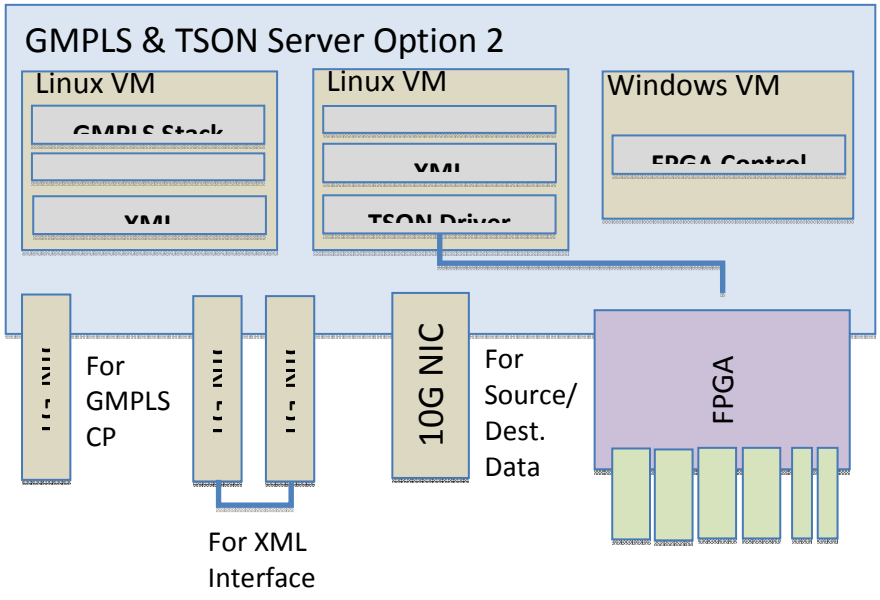
The combination/interfacing of the PCE and SLAE with other external tools, and their placement in the server are in progress and in this regards different options and possibilities are being evaluated.

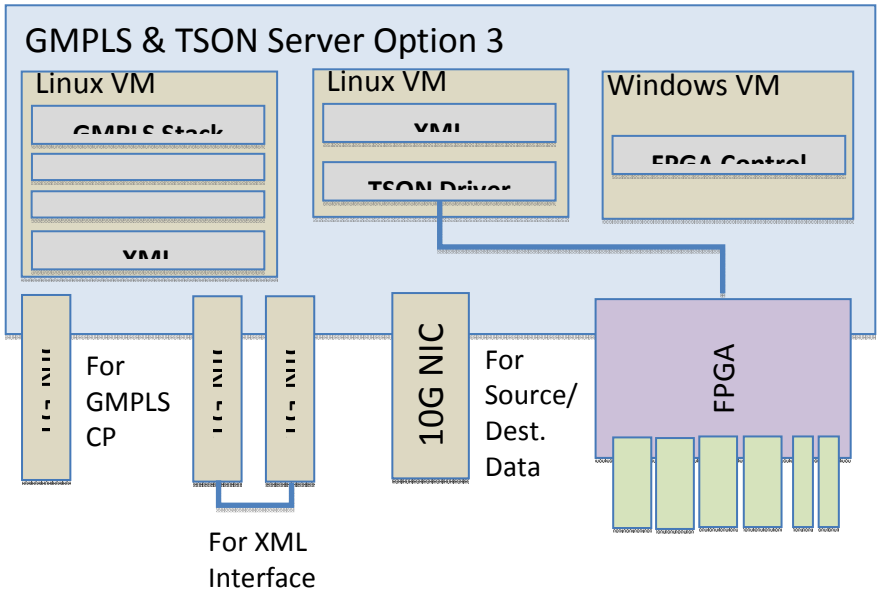
Besides the RWTA, a RSA tool has been developed which will allocate spectrum slices in the frequency domain instead of time slices over the wavelengths.

Appendix I

Different possible combination for SLAE with PCE:






GMPLS & TSON Server Option 3
Linux VM
Linux VM
Windows VM
XML
TSON Driver
FPGA Control
XML
For
GMPLS
CP
10G NIC
For
Source/
Dest.
Data
FPGA
For XML
Interface

Appendix II

Name and address of the employed java library files:

Jar file	***Web access address***
Colt-1.2.0	*http://acs.lbl.gov/software/colt/*
Commons-math-2.2	*http://commons.apache.org/math/*
Jgrapht-0.8.2	*http://www.jgrapht.org/*
DistLib_src-0.9.1	*http://en.sourceforge.jp/projects/sfnet_statdistlib/downloads/DistLib/Testy/DistLib_src-0.9.1.jar/*

Appendix III

In order incorporate tools different functionalities in to the GMSPL stack, the following categories and parameters can be considered.

Network Graph Interface			
Node	param IN	[node id]	
	param OUT	[boolean]	
Link	param IN	[local links id]	
		[local node id]	
		[remote node id]	
		[gmpls link cost]	e.g. dependent by the free bw or node load (to be used if needed)
	param OUT	[boolean]	
Graph	param IN (text)	Matrix of connectivity	nodes as in row, links in columns numbers
	paramOUT	[boolean]	

Computation Interface			
ComputeRoute	param IN	[ingress end point]	node id + link id + MAC
		[egress end point]	node id + link id + MAC
		[lsp id]	
		[lsp recovery params]	unprotected vs. protected lsp (i.e. protection type) +level of disjointness (link, node, wavelength ?, time-slice ?)
		[bw]	
		[QoS]	e.g. expressed with a value in a finite range [0,1,2..N], or with the set of delay/jitter/PLR params as for D3.3
		[algorithm]	to choose among a set of possible routing algorithms (djikstra, bhandari, etc.)
		[Start time info]	
		[Duration time info]	

		[flags]	e.g. to choose the format of the returned explicit route (i.e. node-link out vs. link in-node)
	param OUT	[lsp id]	needed if the interface is async
		[explicit route]	sequence of hops (node id, link id, time-slices) in the format specified in the request. The time-slice assignment can be expressed as described in D2.4 (i.e. list of wavelengths+time-slices) for each hop.

Time-Slice Assignment interface			
AssignTimeSlices	param IN	[lsp id]	
		[set of routes]	list of K routes computed by MAINS PCE. Each route is composed by a sequence of (node id, link id, [wavelength(s) id(s)]. Wavelength(s) are selected if MAINS PCE performs RWA. To be further investigated how to perform RWA at MAINS GMPLS when multiple wavelengths could be used in the same link (i.e. tetris).
		[lsp recovery params]	unprotected vs. protected lsp (i.e protection type) +level of disjointness (link, node, wavelength ?, time-slice ?)
		[bw]	
		[QoS]	e.g. expressed with a value in a finite range [0,1,2..N], or with the set of delay/jitter/PLR params as for D3.3
		[Start time info]	
		[Duration time info]	
		[flags]	if needed, e.g. to select some specific SLAE behaviours
	param OUT	[lsp id]	needed if the interface is async
		[route selected]	route selected by the SLAE (among the K provided as input).

			The route is filled with time-slice for each link. The output is then a sequence of hops (node id, link id, time-slices) in the format specified in the request. The time-slice assignment can be expressed as described in D2.4 (i.e. list of wavelengths+time-slices) for each hop