034115

PHOSPHORUS

Lambda User Controlled Infrastructure for European Research

Integrated Project

Strategic objective:
Research Networking Testbeds

Deliverable reference D.5.9

Extended Simulation Environment

Due date of deliverable: 2009-03-31
Actual submission date: 2009-03-31
<Phosphorus-WP5-D.5.9>

Start date of project:
October 1, 2006

Duration:
30 Months

Organisation name of lead contractor for this deliverable: IBBT

Project co-funded by the European Commission within the Sixth Framework Programme (2002-2006)		
Dissemination Level		
PU	Public	X
PP	Restricted to other programme participants (including the Commission Services)	
RE	Restricted to a group specified by the consortium (including the Commission Services)	
CO	Confidential, only for members of the consortium (including the Commission Services)	

Abstract

This deliverable presents four software tools for planning and evaluating the performance of optical Grids. Initially, a module extending the NS-2 network simulator is presented that evaluates the performance of data consolidation algorithms (in terms of resulting job delays, success rate, network load, etc.) for given job characteristics (arrival rates, number of datasets required per job, etc.). Differentiated resilience for anycast services is the focus of another tool. In particular, it targets at comparing routing and server selection algorithms for resilient networks (e.g., the effect on blocking probabilities for given job arrival and failure rates). The Optical Grid Simulator provides a more generic simulation framework for optical Grids, allowing the definition of a variety of network and application models and providing high modularity. Its aim is to evaluate job scheduling and routing algorithms as well as switching paradigms (OCS, OBS and burst-over-circuit switching). Last, a network design and dimensioning tool is presented, that unlike the rest of the tools is targeted for use during the network planning/upgrade phase, so as to create optimal designs that minimize the capacity allocated to the optical network while taking into account optical characteristics (e.g., with impairment-aware regenerator placement).

List of contributors

RACTI	• Emmanouel Varvarigos
	• Kostas Christodoulopoulos
	• Panagiotis Kokkinos
IBBT	• Jens Buysse
	• Mard De Leenheer
	• Chris Develder
ULEEDS	• Taisir El-Gorshi
	• Jaafar Elmirghani
AIT	• Kostas Katrinis
	• Anna Tzanakaki

Table of Contents

Table of Figures

List of tables

0 Executive Summary

This deliverable describes the tools which have been used in the studies conducted in WP5.

First, a tool extending the NS-2 network simulator for the evaluation of new data consolidation techniques is presented to assess the performance of data consolidation algorithms (in terms of resulting task delays, success rate, etc.) for given task characteristics (e.g. arrival rates, number of datasets per task, etc.).

Differentiated resilience for anycast services is the focus of another tool. In particular, it targets at comparing routing and server selection algorithms for resilient networks (e.g., the effect on blocking probabilities for given job arrival and failure rates). The Optical Grid Simulator provides a more generic simulation framework for optical Grids, allowing the definition of a variety of network and application models and providing high modularity. Its aim is to evaluate job scheduling and routing algorithms as well as switching paradigms (OCS, OBS and burst-over-circuit switching).

Lastly, a network design and dimensioning tool is presented, that unlike the rest of the tools, is targeted for use during the network planning/growth phase, creating optimal designs and minimizing the capacity allocated to the optical network, while taking into account optical characteristics (e.g. with impairment-aware regenerator placement).

1 Introduction

Simulators are widely used in the software community to mimic the behaviour of theoretically complex models, which are too difficult to grasp on a conceptual level. By this, we can receive practical feedback when designing real world systems by studying the considered problem at different levels of abstraction. In the past WP5 deliverables we have used several simulator software distributions, which were briefly described in each deliverable. In this report we give a comprehensive overview of all of them, and outline their main goals and characteristics.

Section 2 discusses an extension of the Network Simulator (NS-2), which can be used for the study of data consolidation techniques that are applied when a job requires more than one datasets for its execution.

In Section 3, a simulator is developed to examine a differentiated resilience scheme that allows anycast flows to survive any link or server failure.

The next section describes the changes that have been made in the optical grid simulator which was initially developed in [1].

Finally Section 5 introduces a network planning tool based on MATLAB, built to optimize the design of the Phosphorus optical network infrastructure with particular attention for the optical characteristics (e.g. taking into account physical layer impairments).

2 Dimensioning and Fault Tolerance Simulation Studies for Data-Intensive Applications

The Phosphorus project is in large part dedicated to studying network-related aspects of Grid computing, and thus its focus is on data-intensive applications that are heavily dependent on network resources. In this context, we identified the crucial concept of Data Consolidation (DC) that applies to applications that need several pieces of data scattered in a number of Grid resources. The DC problem consists of three interrelated sub-problems: (i) the selection of the replica of each dataset (i.e., the data repository site from which to obtain the dataset) that will be used by the job, (ii) the selection of the site where these pieces of data will be gathered and the job will be executed and (iii) the selection of the paths the datasets will follow to arrive at the data consolidating site.

We examined the Data Consolidation (DC) operation in a number of Phosphorus deliverables, namely [2] and [3], by considering network design, resource placement and fault tolerance issues. In [2] we proposed a number of simple Data Consolidation techniques and examined how they are affected by the network design. In particular, we investigated the effect of the total number of storage resources and the effect of the placement of storage resources and datasets in the Grid Network. In [3] we added resilience features to the proposed DC techniques in order to provide fault-tolerance to the DC operation. Moreover, we proposed new DC techniques to cope with the increased network load that is generated by introducing the additional resilience features. We showed that it is possible to perform DC with increased fault-tolerance in an efficient manner.

Data Consolidation (DC) techniques along with the various design, placement and resiliency issues were studied through simulations, by using and extending the Network Simulator (NS-2) [4]. We decided to use NS-2 because it implements and simulates a large number of network-related parameters and characteristics that are very essential to examine the performance of data-intensive applications. On the other hand, NS-2 does not implement any grid-related characteristics (e.g., computational and storage resources) and as a result we implemented these features from scratch. Using NS-2 for our DC related studies, we were able to easily develop and evaluate the performance of a number of new DC algorithms in various scenarios. We believe that the large NS-2 user base can benefit from the work we performed and this is the reason we will make our source code available to the community, using the GPL open source license.

2.1 NS-2 basic characteristics

NS-2 is a discrete event simulator targeting networking research in general. NS-2 provides substantial support for simulation of TCP, routing and multicast protocols over wired and wireless (local and satellite) networks. The code implementing the protocol models as well as the simulation engine (for the simulation of event scheduling, etc.) is C++, while an OTcl (short for MIT Object Tcl) simulation user interface is provided. The user describes a network topology by writing OTcl scripts, and then the main NS program simulates that topology with the specified parameters. In addition to the simulation program, NS-2 provides a number of tools for creating topologies and random traffic patterns. We extended the NS-2 code base to develop new algorithms and characteristics.

2.2 GridNs module: Grid and Data Consolidation related extensions to NS-2

In order to add Grid related characteristics to the NS-2 simulator, we implemented the gridNs module. In this module we also implemented the various Data Consolidation and fault tolerance techniques/algorithms previously investigated in [2] and [3]. In particular the gridNs module consists of the following files:

- **cpp\DC_agent.cc** and **cpp\DC_agent.h**: This C++ class implements a computational and storage Grid resource.
- **cpp\DC_replication_table.cc** and **cpp\DC_replication_table.h**: This C++ class implements the structure of a resource where datasets are stored.
- **cpp\DC_scheduler.cc** and **cpp\DC_scheduler.h**: This C++ class implements the Grid scheduler and the DC algorithms.
- **cpp\task_struct.cc** and **cpp\task_struct.h**: This C++ class implements a Grid job that has both communication and computation requirements.
- **cpp\DC_hdr.h**: This C++ class implements the DC specific data packet header.
- **cpp\DC_graph.cc** and **cpp\DC_graph.h**: This C++ class implements the topology graph. It is used by a number of DC algorithms in order to select the DC and the data replicas sites. In this class we used the Boost library's [Boost] Minimum Spanning Tree algorithms.
- **cpp\stat_collector_dc.cc** and **cpp\stat_collector_dc.h**: This C++ class is used for collecting statistics.
- **tcl\lib\ns-dc-defaults.tcl**: This tcl file contains the default values for a number of variables.
- **tcl\lib\ns-dc-lib.tcl**: This tcl file contains a number of functions used by the tcl interface in order to create a Grid topology.
- **tcl\example\experiment.tcl**: Through this tcl file the parameters and characteristics of an experiment are set up, such as the Grid topology, the characteristics of the links and the resources, the requirements (computational and communicational) of the jobs generated and other.

Figure 1 shows the workflow of the gridNs module, both in OTcl and in C++ level.

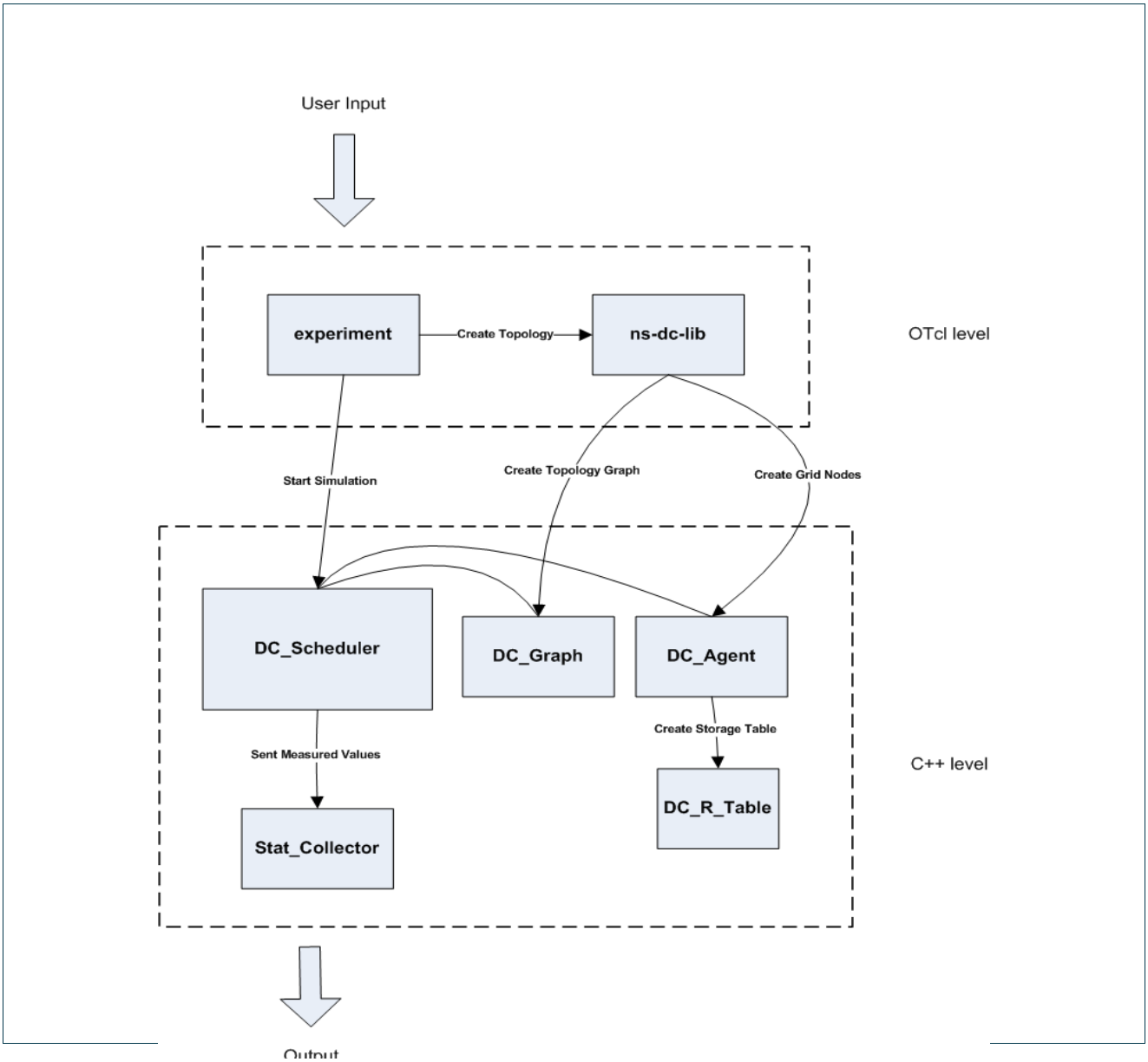

Figure 1 GridNs module workflow.

2.3 Implementation details and metrics used

In our implementation we assume a point-to-point (opaque) optical network; the delay for transmitting between two nodes includes the propagation, queuing and transmission delays at intermediate nodes. Only one transmission is possible at a time over an optical link, so a queue exists at every node to hold the data waiting for transmission.

Moreover, at the beginning of the simulation a given number of datasets are generated and two copies of each dataset are distributed in the network; the first is distributed among the Tier 1 sites and the second is placed at Tier 0 site (we assumed a hierarchical data resource architecture, which consists of two Tiers). Since the storage capacity is bounded, there are cases where a site does not have the free storage capacity required to store a needed dataset. In such a case, one or more of the oldest and unused datasets are deleted until the new dataset can be stored at the resource.

In all our experiments we keep constant the average total data size S that each job requires:

$$S = L \cdot V_I, \quad (1)$$

where L is the number of datasets a job requests and V_I is the average size of each dataset. The job workload W correlates with the average total data size S, through parameter a, as:

$$W = a \cdot S. \quad (2)$$

As a increases, the jobs become more CPU-intensive, while as a decreases the jobs have less computation demands. Also, when a job completes its execution we assume that there is no output data returned to the originating user.

We have implemented the following metrics to measure the performance of the DC algorithms examined:

- The *average job delay*, which is the time that elapses between the creation of a job and the time its execution is completed at a site.
- The *average load per job* imposed to the network, which is the product of the size of datasets transferred and the number of hops these datasets traverse.
- *The job success ratio*: This is the ratio of the number of jobs that where successfully scheduled, over the total number of jobs generated. When a large number of jobs are queued or under execution, it may be impossible for the scheduler to find a resource with sufficient free storage space, where a new job's datasets can consolidate. In this case the job cannot be scheduled and is counted as a failure.
- The *Data Consolidation (DC) probability*, which is the probability that the selected DC site will not have all the datasets required by a job and as a results Data Consolidation will be applied.

The first metric characterizes the performance of the DC strategy in executing a single job, while the second and the third express the overhead the DC strategy induces to the network. The fourth metric gives information on the way the DC site is selected, with respect to the datasets that are located (or not) at this DC site.

2.4 Installing and using the gridNs module

In order to use the gridNS module one must follow the following steps:

1. Install the *ns-allinone* software package: http://www.isi.edu/nsnam/ . Our module was tested with version 2.31.

2. Install the *boost* library: http://www.boost.org/ . Our module was tested with version 1.34.
3. Unzip the gridNs package in the *ns-allinone/ns/* folder.
4. In the file *ns-allinone/ns/common/packet.h* add the '*PT_DC'* packet type to the packet_t enumerator.
5. In the file *ns-allinone/ns/common/packet.h* add the '*name_[PT_DC]="DC";'* assignment command, in the constructor of the p_info (packet information) class.
6. Change the *ns-allinone/ns/Makefile*, so as to include the gridNs files.
7. Compile the NS-2 code, using the commands: '*make clean'*, '*make all'*
8. Define the desired Grid topology in the *tcl/example/rules.tcl* file.
9. Run the experiment using the command:
   ns gridNs/examples/experiments.tcl [total_number_of_jobs] [data_size] [data_requested] [a_value] [l_arr] [scheduling_str] [given_seed] [number_of_nodes_with_cpus]
   where:
   - *total_number_of_jobs*: the number of jobs to be executed
   - *data_size*: the average size of a dataset (exponential random variable)
   - *data_requested*: the data requested per job
   - *a_value*: this parameter defines whether a job is computation or data-intensive *l_arr*: the job average creation interarrival (exponential random variable)
   - *scheduling_str*: the Data Consolidation policy used
   - *given_seed*: the seed used
   - *number_of_nodes_with_cpus:* the number of sites having computational and storage resources.
10. A result file will be created containing the following data: "number_submitted_jobs", "datasets_requested", "computational_complexity", "data_output_size", "searched_agents", "datasets_not_found_in_same", "datasets_transfers", "datasets_transfer_hops","data_transfer_size", "data_transfer_multpl", "datasets_deleted", "executed_jobs", "total_time", "consolidation_time", "including_queue_time", "including_exec_time", "larger_total_time", " jobs_not_scheduled", " jobs_larger_than_maxqueue", "transfer_cost".

3 A Simulator for Examining Differentiated Resilience for Anycast Flows

In this section a simulator is developed to examine a differentiated resilience scheme that allows anycast flows to survive link and server failures. The simulator is developed as a discrete event simulator. The simulator is written in C programming language as it is extremely flexible, reliable and quick, and moreover, the C language is designed so that separate modules can be developed independently and afterwards combined altogether which facilitates subdividing complicated development into smaller and less complex modules. The main features of the simulator include:

- The simulator is simple and easily comprehensible as it is developed to be as modular as possible by defining a number of different structures.
- Flexibility in examining different routing algorithms and introducing new traffic classes with different resilience requirements.

The resilience approach implemented is based on the assumption that an anycasting request can be served by any suitable replica server. Under this approach, referred to as backup server, routes of both upstream and downstream connections to a backup server(s) are calculated; if the working server fails, then the backup server takes responsibility. Differentiation is created by introducing three classes of resilience. Class 1 and Class 2 connections are recovered through dedicated and shared protection to an alternative server, respectively. A Class 3 connection assures restoration by using the spare capacity left after recovery of the other two classes. In the simulation scenario, we compare the performance of the different classes under different combinations of routing and server selection algorithms. Explicit routing is used for the Label Switched Paths (LSPs) selection. Dynamic routing is implemented to calculate routes (upstream and downstream connections) to both the primary and the backup servers. Two routing algorithms are examined: the Constraint Shortest Path First (CSPF) algorithm [4] and the Least Interference Optimization Algorithm (LIOA) [4]. Three server selection algorithms are presented: Hop Number Server (HNS), Residual Capacity Server (RCS) and Hop Number Widest Server (HNSW) [3].

3.1 Simulator Implementation

In order to design a modular simulator, a number of different structures and functions have been defined. Each structure is defined in a different C module, but some of them have access to the properties of other structures to which they are linked. In this section we present the main structures and functions defined in the simulator and their main parameters.

3.1.1 Node

A node is a structure characterized by its name, ID and its nodal degree and the links connected to it. Each node is assumed to be used as an ingress and egress node.

3.1.2 Server

Server is another defined structure characterized by its location. A given number of nodes will be randomly selected (using a given random seed to allow for reproducible experiments) to function as servers. A Server is assumed to have unlimited resources (e.g. storage, processor etc) which implies that link bandwidth is the only potential bottleneck for the anycast flows.

3.1.3 Link

A Link is a structure characterised by its ID, length, propagation delay, link capacity, total link flow, residual capacity and nodes connected to it.

3.1.4 Link state information

Link state information required for server selection and routing includes link capacity, total link flow and the location of the replica servers. For simplicity, in our simulation scenario it is assumed that all the above information is known administratively: we do not explicitly model link state information distribution protocols.

3.1.5 Anycasting Request

An anycasting request is a defined structure characterised by its origin node, upstream/downstream bandwidth requirements and resilience requirements (traffic class). To establish an anycast flow to one of multiple destinations, a destination is selected according to one of three server selection algorithms mentioned above.

3.1.6 Residual Network

The residual network for a new anycasting request is built with nodes with sufficient resources (e.g. memory, processor, etc.) and links with a residual capacity exceeding the request bandwidth requirement. Bandwidth requirements of both upstream and downstream routes are considered, and as such only reachable servers with enough resources are taken into account in the selection. Such servers are called available servers.

3.1.7 Results

The connection blocking probability is used as the comparison metric. The blocking probability is calculated for each class individually as the ratio of the number of blocked requests of that class to the total number of requests of that class. An anycast request is considered to be blocked if no available servers are found. Also Class 1 and Class 2 anycasting requests are blocked if either the upstream or downstream connections of the paths to the primary or the backup servers cannot be established. An anycast request of Class 3 is blocked if either the upstream or downstream connections of the path to the primary server cannot be established. The obtained results are averaged out over various cases of server locations. Results are collected in a specially defined structure and stored in an easily processed form.

3.1.8 Traffic Generator

The traffic of each node is simply generated prior to the start of the simulation and stored in a file. When a simulation is started, a function called *read_anycasting_request()* is used to fetch the traffic from the files into the nodes.

For the simulation scenario, we assume the total traffic consists of 10% Class 1, 30% Class 2, and 60% Class 3. As the anycast flow is asymmetrical, bandwidth requirements are assumed to be as follows: The upstream connections of all traffic classes are assumed to be 1 bandwidth unit. The downstream connections requirements are assumed to be uniformly distributed in the range of (4-10) units for Class 1 and Class 2. Requirements of Class 3 are assumed to be uniformly distributed in the range of (0-4) units. The values of these parameters can be easily changed to run the simulator under a different traffic scenario.

3.1.9 Routing and Server Selection Algorithms

The user can choose to run the simulator under any combination of the server selection algorithms (HNS, RCS and HNSW) and the routing algorithms (CSPF and LIOA) by executing the *select_server()* and *routing_algorithm()* functions.

3.2 Simulation Cycle

Figure 2 illustrates the simulation cycle of a node. The node fetches an anycasting request from the node traffic file using the *read_anycasting_request()* function. According to the network link state information and the anycasting request bandwidth requirements, the node executes the *build_residual_network()* function to build residual network. If no servers are available, the anycasting request is blocked. Otherwise, the node checks the resilience requirements of the request to provide the required protection level. For Class 1 and Class 2, a primary and a backup server are selected according to the implemented server selection algorithm using the *select_server()* function. Then link disjoint paths are provisioned to the primary and backup servers according to the implemented routing algorithm by executing the *routing_algorithm()* function. If the request cannot be provisioned according to the current network state, existing Class 3 connections can be rerouted to allow the otherwise blocked requests of Class 1 and Class 2 to be routed. If still the request cannot be rerouted, it is blocked. Rerouting of existing Class 3 connections is done by executing the *build_residual_network()*, *select_server()* and *routing_algorithms()* functions. For class 3 only a primary server is selected and a path is provisioned to it. Failing to provision a path to the primary server results in blocking the connection. Finally the request blocking probability results are collected.

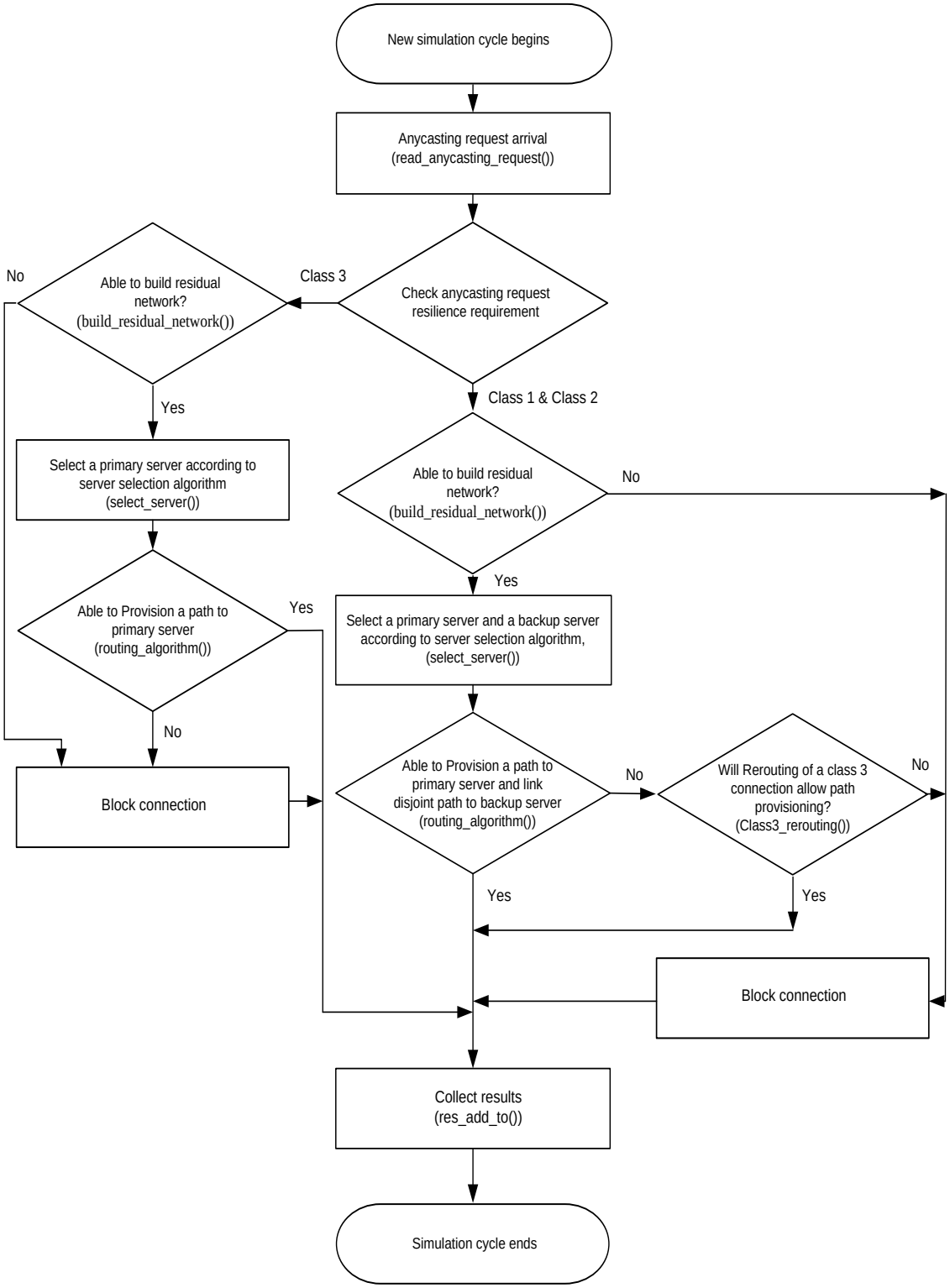
New simulation cycle begins
Anycasting request arrival
(read_anycasting_request())
Check anycasting request
resilience requirement
Class 3
Able to build residual
network?
(build_residual_network())
No
Yes
Select a primary server according to
server selection algorithm
(select_server())
Able to Provision a path to
primary server
(routing_algorithm())
Yes
No
Block connection
Class 1 & Class 2
Able to build residual
network?
(build_residual_network())
No
Yes
Select a primary server and a backup server
according to server selection algorithm,
(select_server())
Able to Provision a path to
primary server and link
disjoint path to backup server
(routing_algorithm())
No
Will Rerouting of a class 3
connection allow path
provisioning?
(Class3_rerouting())
No
Yes
Yes
Block connection
Collect results
(res_add_to())
Simulation cycle ends

Figure 2 Node simulation cycle.

4 Optical Grid simulator

4.1 Introduction

The Phosphorus project addresses several key technical issues to enable on-demand, end to end network services in an optical grid environment. In WP5 innovative architectures and algorithms have been proposed, which are not, however, easy to test in a real life test bed. Therefore, in [1] we developed a simulator that implements an optical grid network and a detailed job model that corresponds to a wide range of applications. In the context of the present deliverable, we have extended the simulator, by implementing an additional key feature and by optimizing important operations of it, so as to reach a certain performance level.

4.2 Overview

The simulator is written in Java and its base classes make up a discrete-event simulator, which models a chronological sequence of events, each marking a change in the state of the system. The basic flow of the simulator is as follows:

1. The various entities (switches, resources, clients …) are initialized.
2. System variables and the main clock are initialized.
3. An initial event is scheduled, i.e. an initial event is put into the event list from which a chain reaction of other events will occur.
4. While the simulator does not reach its ending condition
    a. Set clock to next event time.
    b. Execute the next event and remove it from the event queue.

c. Update the entity's statistics.

4.2.1 Class diagram

There are no big changes in the overall design of the simulator as outlined in deliverable [1]. Each real world object in the grid setup (such as a link, a switch, a server) is represented in the simulation framework as an Entity. Entities in the simulation framework exchange messages (e.g., representing jobs) through so-called in/out ports. These messages are wrapped into a *SimbaseEvent* and are passed via the *GridInPort* of the receiving entity. The *SimbaseSimulator* class is responsible of keeping track of every event and executes each event chronologically.

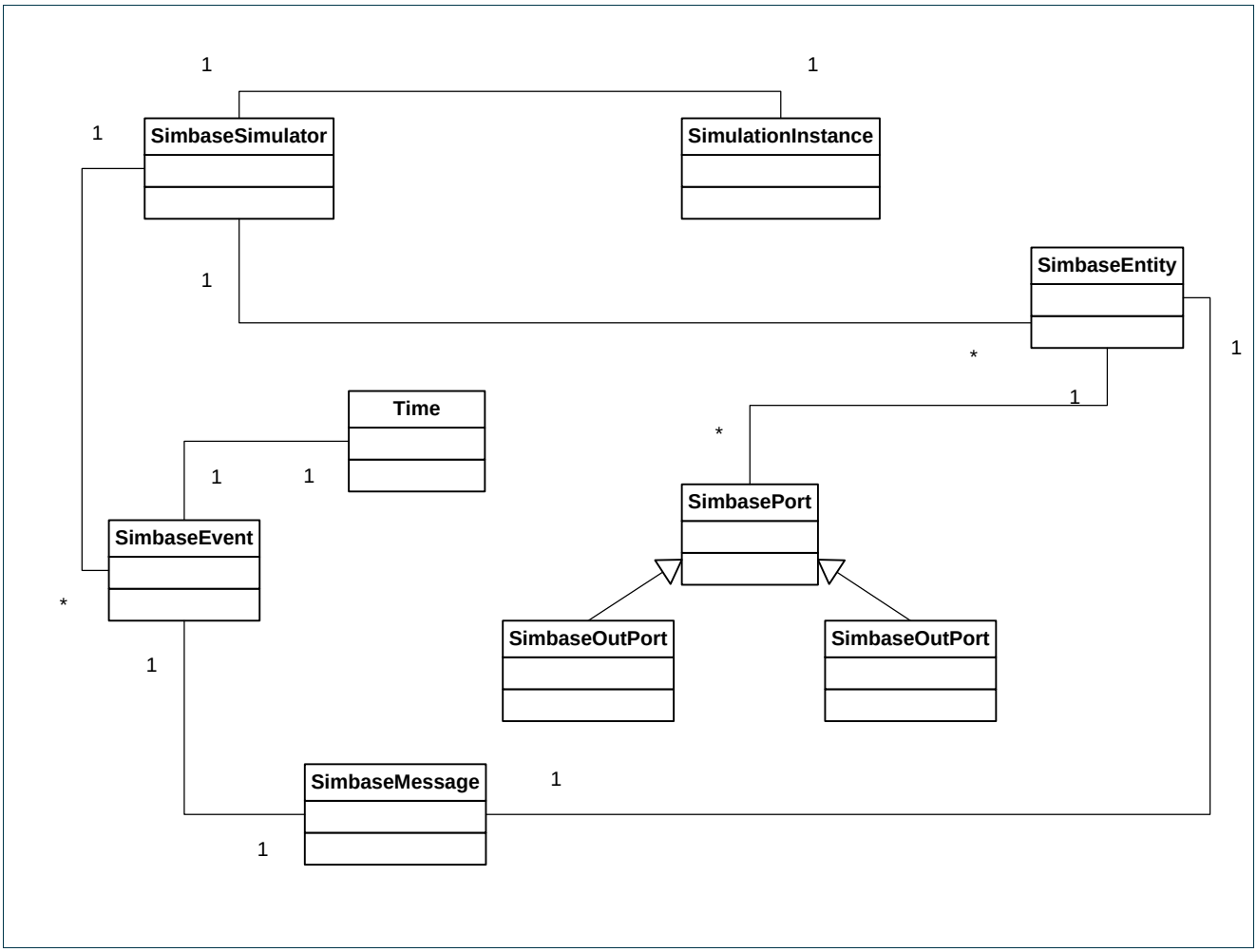

Figure 3 Class diagram of the discrete event simulator.

To implement the grid specific functionality, several classes are implemented which are derived from the base classes described above, such as *ClientNode*, *ResourceNode*, *Switch*. For more information regarding the architecture of the simulator we refer to [1].

4.3 Optimization and additional feature

4.3.1 Jung library

In the first version of the simulator we implemented our own graph representation structures and graph related algorithms. For example, we implemented from scratch the Dijkstra algorithm, so as to find the shortest path from node A to node B. However, this requires a significant programming effort, while several libraries are available that already implemented such functionality. As a result, in the context of the present deliverable, we decided to integrate one of these libraries, namely the JUNG [4], in our simulator.

JUNG (Java Universal Network/Graph Framework) is a software library that provides a common and extensible language for the modelling, analysis, and visualization of data that can be represented as a graph or network. This library is incorporated in our software in such a way that the user of the simulator does not need to know how to use it. JUNG also provides a visualization framework that makes it easy to construct tools for the interactive exploration of network data.

4.3.2 Hybrid switching – Burst over Circuit Switching

The main update from the previous version of the simulator is that it is now able to model the Burst-Over-Circuit switching technique[1] [5]. This became possible by the introduction of the *HybridSender* class, which takes care of the switching technique used by an entity. This sender is a composition of two other senders: namely an *OCSSender* and an *OBSSender*. First, the *HybridSender* tries to send the data using the *OCSSender*, in order to take advantage of an already established *OCSCircuit*. If this is not possible, there are two remaining possibilities, namely:

1. We have reached the beginning of an *OCScircuit* and the data must be conveyed on this circuit.

2. There are no circuits available and the OBS switching technique should be used to forward the data.

This execution flow can bee seen in Figure 4.

[1] In the Burst-Over-Circuit-Switching model, the circuit switched network technology works as a server layer providing a virtual topology with light paths to a burst switching client layer. Optical bursts are only switched in the client layer nodes and transparently flows in light paths through the server layer nodes.

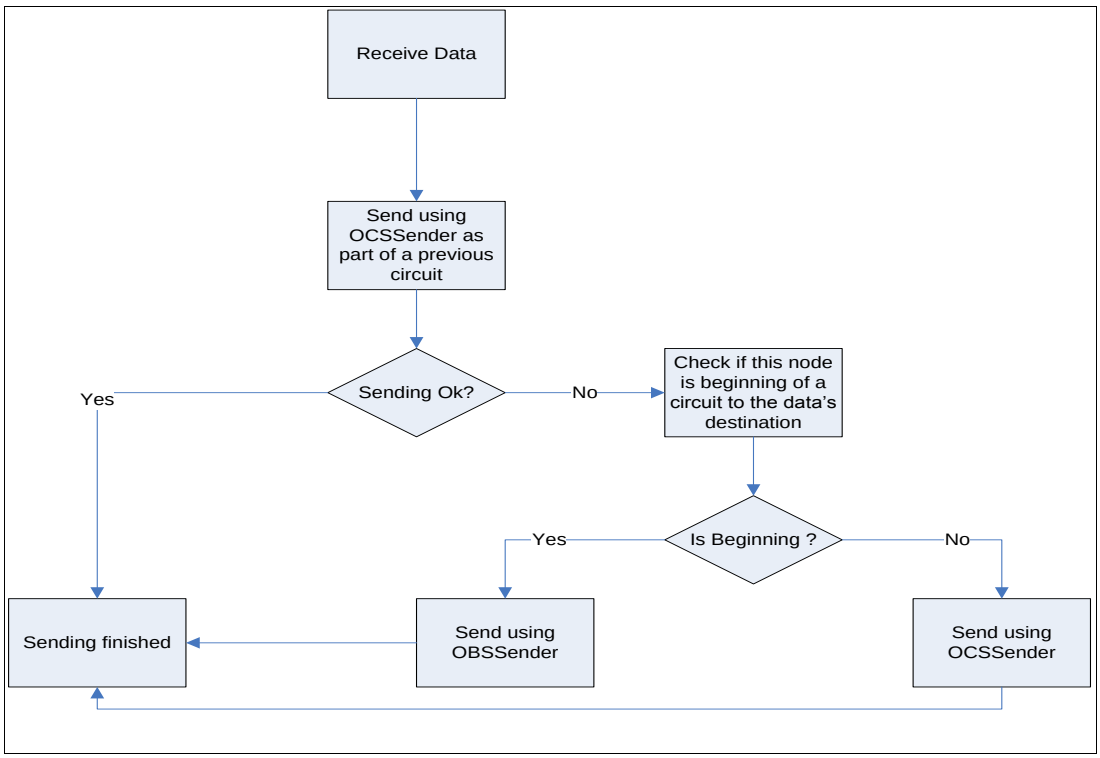

Figure 4 Execution flow of a HybridSender.

4.4 Running a simulation

Defining a simulation scenario and running it, it is done by writing a Java program, using the classes and interfaces implemented by our framework.

- First the Simulator and *SimulatorInstance* objects should be created. The *SimulatorInstance* receives as input parameter the location of the configuration file which contains the standard parameters for the simulation, such as the number of wavelengths per link, the processing power of a resource core, etc. Individual parameters can be altered by invoking the corresponding setter method on that entity for that parameter.

```
simInstance = new GridSimulation("configFiles\\loadCircuits.cfg");
simulator = new GridSimulator();
```

- Next, the entities of the simulation have to be constructed.

```
switch1 = Grid.Utilities.Util.createHybridSwitch("SWITCH1", simulator);
broker = Grid.Utilities.Util.createHybridServiceNode("BROKER", simulator);
resource1 = Grid.Utilities.Util.createHyridResourceNode("RESOURCE1", simulator);
client1 = Grid.Utilities.Util.createHybridClient("CLIENT1", simulator, broker);
```

- Each resource is assigned the broker/scheduler.

```
resource1.addServiceNode(broker);
```

- The links between the entities have to be created, defining this way the topology.

```
Grid.Utilities.Util.createBiDirectionalLink(client1, switch1);
Grid.Utilities.Util.createBiDirectionalLink(switch1, client1);
Grid.Utilities.Util.createBiDirectionalLink(switch1, broker);
Grid.Utilities.Util.createBiDirectionalLink(switch1, resource1);
```

- Insert a circuit between two entities (e.g., a client and a resource).

```
Grid.Utilities.Util.createOCSCircuitInHybridNetwork(client1, resource1, simulator);
```

- Finally, the simulation is started and its output is printed out.

```
simInstance.run();
output.printClient(client1);
output.printResource(resource1);
output.printSwitch(switch1);
```

5 NeDeTo (Network Design Tool)

Deploying lambda Grids in a cost-efficient and yet effective manner entails among others prior careful planning of the target optical network infrastructure. The latter is realized through the application of known optimization methods to the network design/dimensioning problem, with the ultimate goal of deriving a minimum cost network that satisfies a set of desired constraints, taking into account the physical layer characteristics (impairments).

Towards optimizing the design of the Phosphorus optical network infrastructure, we have implemented the Network Design Tool (abbreviated to NeDeTo). Although it is implemented using the MATLAB programming framework, the tool can be run in the form of a standalone executable in any commodity operating system (OS), given that a related compiler compatible with the MATLAB framework is used to compile the code with a specific OS as target. Currently, we have provided for an executable targeted at machines running Microsoft Windows. In addition, a working MATLAB installation is needed to create *.mat* input parameter files, as well as to process the files containing the optimization output results. NeDeTo can be efficiently run on any contemporary commodity workstation, without stringent or exotic hardware requirements in terms of CPU frequency, processor architecture, main memory size or hard disk space. Still, the optimization core will greatly benefit from high-end CPU and memory capabilities, resulting in execution speedup. This is reasonable when designing medium- to large-scale networks, mainly due to faster processing of the branch-and-bound tree and due to lower I/O activity between main memory and hard disk for fetching parts of the tree to the working memory.

5.1 Application Workflow

NeDeTo implements the network design/dimensioning approach presented in [2]. As such, it employs the respective network cost model and implements the physical impairments model and the specific dimensioning methods specified in [2].

Figure 5**Błąd! Nie można odnaleźć źródła odwołania.** illustrates the workflow of the design process as perceived by the application user, which follows naturally from the theoretical specification of the network design approach presented in [2]. The user has first to specify all information that forms the input to the network design problem that is to be solved by NeDeto. More precisely, following input is required:

1. Physical network topology, which is specified by loading a properly formatted (see Appendix for format specification of the topology file) topology file that among others contains number of switching nodes and the set of candidate fibre links with per link physical distance. The tool distribution made available includes topology files modelling the Phosphorus European and Phosphorus extended [2] topologies, as well as topology files modelling toy networks to be used for correctness testing.

2. Assigning values to optical parameters e.g. bit rate. The application expects a file with optical parameter value assignments bundled in a MATLAB data file, named after “p1.mat” and located in the same directory as the main executable. The tool distribution supplied includes a “p1.mat” file that is per default used to assign values to the optical parameters of the network that is to be planned.

3. Specifying a set of connection requests between node pairs of the input topology. The designed network, among other constraints, will be capable of fulfilling these connection requests. The input traffic matrix, contained in a properly formatted file (see Appendix for format specification of the traffic matrix file), is loaded by the user through the GUI interface of the tool. The tool distribution made available includes traffic files for the Phosphorus European and Phosphorus extended topologies, as well as traffic files modelling that are to be used with the toy network topologies provided.

4. Specifying technology and routing parameters, namely number of wavelengths/fibre, maximum number of fibres allowed per physical link and number of alternative paths to be considered by the k-shortest path algorithm.

5. Specify the cost parameter values, as mandated by the cost model extensively presented in [2].

After all input data have been loaded/specified, the design method to be used to optimize the total cost in the optical WDM network needs to be selected among three alternatives (all presented in [2]):

- Network design/dimensioning without use of regenerators (“Basic” method)
- Network design/dimensioning with length-based regenerator placement
- Network design/dimensioning with impairment-aware regenerator placement

The completion of all the above steps yields a well-defined instance of the network design/dimensioning problem that is solved by the NeDeTo tool to optimality, as soon as the user initiates the optimization process. The results of the optimization process are written to *.mat* MATLAB data files, whose structure is extensively specified later in this section. In addition, utilization of resources in the designed network is depicted using three graphs:

1. Graph illustrating the utilization of alternative paths by each connection request, among the *k* alternative paths that can be taken by the k-shortest path algorithm used.

2. Graph illustrating number of fibres installed per physical link in the designed network.

3. Graph illustrating number of activated wavelengths per physical link in the designed network.

While the illustration of these data does not convey a lot about the efficacy of the design method used, we found these graphs very useful in understanding the behaviour of the various design approaches, depending on the particularities of the problem input. In any case, the availability of various evaluation metrics data contained in the results .mat files allows the user to create custom code to create custom graphs that will serve the purpose of illustrating the results of interest to serve the research goal that is to be met through the use of our Network Design Tool.

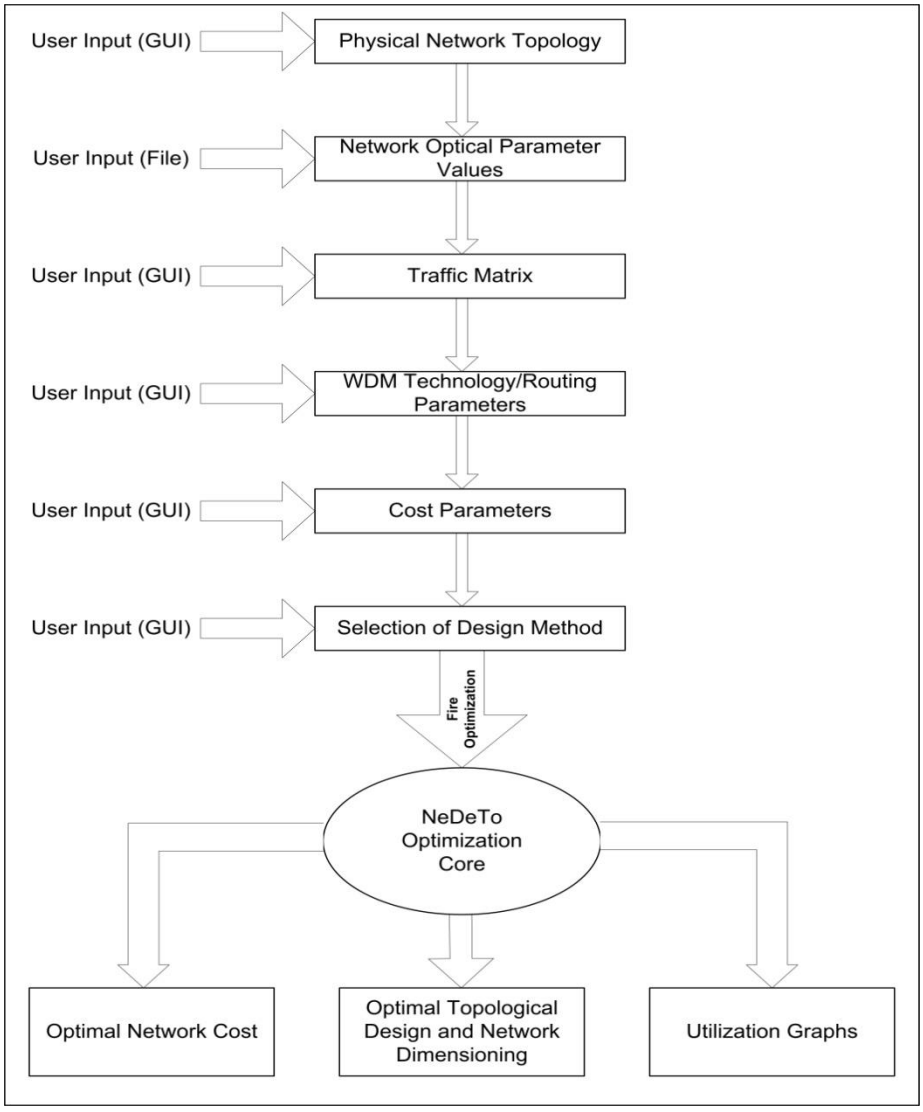

Figure 5 Workflow diagram of the Network Design Tool.

Figure 6 depicts a snapshot of the NeDeTo GUI presented to the user after initialization of the tool executable. Based on the above specification of the application workflow, the semantics of the user interface should be straightforward, with the sole exception of the following widgets that increase functionality:

- "Use Fix Fibre Costs Only" checkbox: the user may set this flag to true (i.e. tick the respective box), if the installation costs that are proportional to link distance are to be ignored. This functionality is provided for solving network planning problems, whereby fibre has already been laid down (e.g. dark fibre) and as such only fibre termination costs (fixed costs) have to be considered.

- "Use Ranges" checkbox: the user may set this flag to do a parametric study of network planning (with or without joint regeneration placement) with regards to ranging cost factors, instead of using a single value for each cost factor used. Checking the "Use Ranges" enables the "* Samples" textboxes ("#Trenching Cost Samples", etc.), as well as the second textbox for specifying the cost value of each cost factor. The value range to be used for each cost factor, e.g. for the cost of fibre/distance, is specified as follows: starting at the cost value specified by the left cost textbox (0,125 for fibre/km in Figure 6), advance the respective cost value up to the maximum value as specified by the right textbox (0,5 for fibre/km in Figure 6) in equidistant steps such that the total number of tested cost values for the this cost factor to equal the number of samples specified in the respective "* Samples" textbox (2 for fibre/km in Figure 6). There is also the option of using a range for a subset of the cost factors, while using a single cost value for the rest of the cost factors by setting the minimum and maximum cost to the same value (in the latter case the value of samples is ignored). The total number of ILP problems that will be used, if the range option is used, will be equal to the product of the number of samples per cost factor.

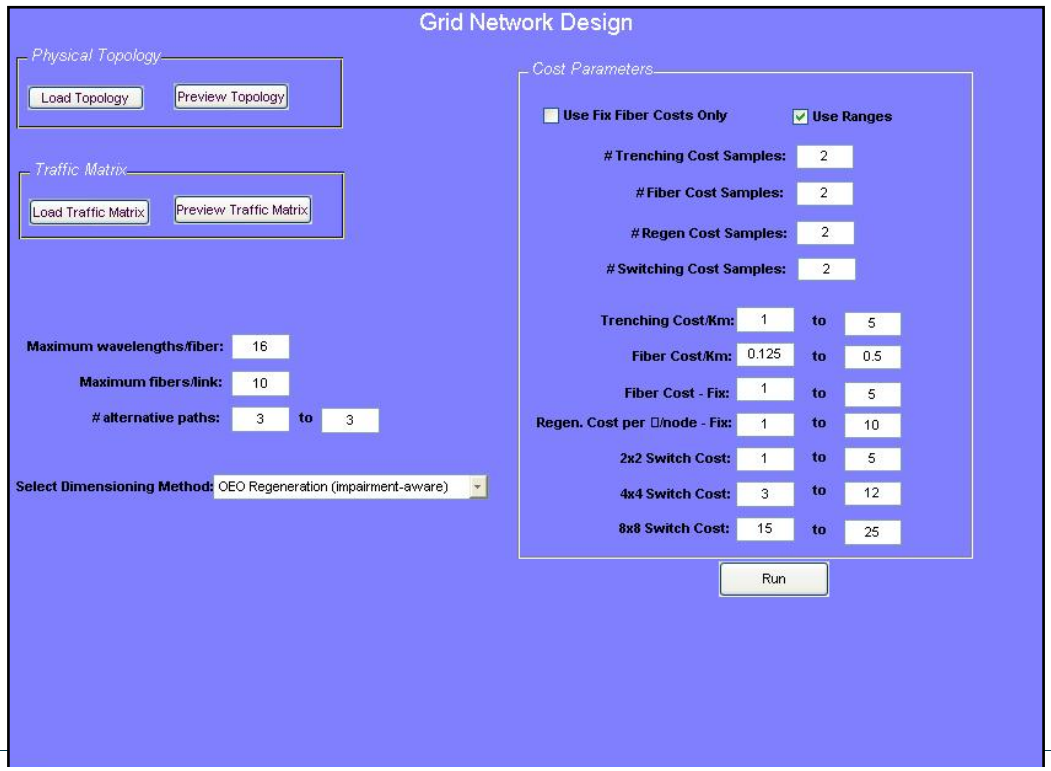

Figure 6 Snapshot of the Network Design Tool graphical user interface.

5.2 Optimization Core

As already stated in the previous subsection, the Network Design Tool implements the Integer Linear Programming (ILP) methods presented in [2]. The optimization core part of the tool uses first all problem input supplied by the user to provide for a formulation of the network design problem that is to be solved, in the form of a mathematical program. Depending on the design method selected, the tool may include the pre-processing step of adding regenerators in the process of creating the ILP formulation. As soon as the vectors and arrays forming the (linear) objective function's coefficients and the constraints of the problem are formed, the integer program is solved by calling the GNU Linear Programming Kit (GLPK [8]) solver. The latter derives first a solution to the LP relaxation of the input ILP and then applies branch-and-cut techniques to specify the integral optimal solution, which is passed back to the NeDeTo optimization core. The optimal integral solution is processed to extract the network design semantics of the optimization result. Among others, the topology of the optimal-cost network is derived, the paths and wavelengths (light paths) used to route the input traffic requests, the switching nodes and ports where OEO regeneration occurs, etc. The optimization core finishes by writing the network design results into data files (more on their format specification in the next subsection) and returns control to the main application.

5.3 Network Design Results Data Structures

The results of the network design process are returned in a *.mat* MATLAB data file, in a structured format instead of raw data representation. The results file is written in the same folder with the Network Design Tool executable. Depending on the specific network design/dimensioning method selected, the name of the results file is:

- "results.mat", if dimensioning without regeneration is used
- "results_oeo_exact.mat", if dimensioning with impairment-aware regenerator placement is used and
- "results_oeo_length.mat", if dimensioning with length-based regenerator placement is used and

Table 1 lists the data structures used to report the results of the network design process, together with a description of the type and semantics of each data structure. These structures can be further processed by custom written code to derive further statistics that stem from the fundamental results, such as number of regenerators, percentage of impaired paths, etc.

Table 1 – Specification of data structures containing the results of the network design/dimensioning process

Data Structure Name	Data Structure Type	Semantic
acceptable_paths	2-dimensional array	Each row corresponds to a path installed to serve a traffic request. The path is represented in node format, i.e. each column corresponds to the switching node ID covered by the path. The last column of each row contains a unique integer ID of the corresponding path. Note that paths exhibiting Bit Error Rate (BER) below a given threshold appear as all-zero row entries in this array.
optimalCost	MATLAB Structure	Each structure fields holds a fraction of the total optimal cost, categorized as follows: optimalCost.ductCost: cost due to trenching optimalCost.fiberCost: cost of installed fiber and fiber-related equipment optimalCost.lambdaCost: cost of activating wavelengths (termination equipment) optimalCost.switchCost: cost of switching equipment optimalCost.regenCost: cost of deployed regenerators
paths	2-dimensional array	Same semantics with the „acceptable_paths" array, with the sole difference that no BER checking is

		applied in this array, i.e. all paths are listed independent of path BER.
paths_BER	2-dimensional array	Each row entry comprises two columns, where the first column is the analytically calculated BER of the installed path and the second column is the unique integer ID of the path
statistics	MATLAB structure	Links of the initial topology used in the designed network, as well as resource utilization statistics; specifically fibers per link installed, wavelengths per link activated and usage of alternative paths. In case a design method employing regeneration is used, the number of regenerators deployed is also reported in this structure.

6 Conclusions

We presented several software tools that were developed and used for the evaluation of algorithms and technologies proposed in previous WP5 deliverables, focusing on dimensioning and fault tolerance issues for Grids. Different approaches have been followed for the development of these tools, which were based on the focus of the simulations. For the evaluation of data consolidation techniques that are applied when a job requires more than one dataset for its execution, we extended Network Simulator (NS-2). This way it was possible to focus more on the proposed algorithms than the underlying network architecture or technology. On the other hand, in order to evaluate differentiated resilience in optical WDM networks for anycast services we created a software tool from scratch. In the present deliverable we also presented the extensions that have been made to the simulator described initially in [1]. The main extension involves the introduction of a class, which makes it possible to switch traffic using the Burst-over-Circuit-Switching paradigm. We also presented a software tool targeted for use during the network planning/growth phase. This tool was built using the MATLAB environment, offering efficient development and fast execution. This tool is used for creating optimal designs and for minimizing the capacity allocated to the optical network, while taking into account optical characteristics (e.g. with impairment-aware regenerator placement).

7 References

[1] Grid Simulation Environment. s.l. : Phosphorus, 2008.

[2] Grid Network Design. s.l. : Phosphorus, 2008.

[3] Resilient Grid Networks. s.l. : Phosphorus, 2008.

[4] NS. [Online] http://www.isi.edu/nsdam/.

[5] A Framework for QoS-based Routing in the Internet. **Crawley, R.N (n.d).** s.l. : Phosphorus, 2009, Vol. Resilient Grid Networks.

[6] Traffic engineering: the Least Interference Optimization Algorithm. **B.A. Bagula, M. B.** Paris : IEEE International Conference on Communications ICC, 2004.

[7] JUNG. [Online] http://jung.sourceforge.net/.

[8] Cost-effective Burst-Over-Circuit-Switching in a hybrid optical network. **Jens Buysse, Marc De Leenheer, Chris Develder, Bart Dhoedt, Piet Demeester.** Valencia : s.n., 2009. International Conference on Networking and Services . p. 6.

[9] GLPK: GNU Lineari Programming Kit. [Online] http://www.gnu.org/software/glpk/.

[10] Boost. [Online] http://www.boost.org.

7.1 Acronyms

BER	Bit Error Rate
GLPK	GNU Linear Programming Kit
GUI	Graphical User Interface
ILP	Integer Linear Programming
OEO	Optical-Electronic-Optical
OS	Operating System
WDM	Wavelength Division Multiplexing

Appendix A Network Design Tool Input File Formats

A.1 Network Topology Input File Format

A NeDeTo topology file starts with a line summarizing the network graph, namely number of nodes and number of links (the trailing “1” is a placeholder for future use):

```
<number of nodes><TAB><number of physical links><TAB>1<newline>
<newline>
<newline>*
```

The term “<newline>*” stands for either zero or any number of new line characters. The next N lines specify the coordinates of each node in the two dimensional space for rendering purposes, where N is the number of nodes. The format of this line is as follows (note that the topology parser expects exactly N of such lines for a topology with N nodes):

```
<X-coordinate><TAB><Y-coordinate><TAB><Node Population>1<TAB>20<TAB>20<TAB>20<TAB><newline>
<newline>
<newline>*
```

The “node population” corresponds to the population that the node serves and is reserved for future automatic creation of traffic matrices based on demographics. Currently, this parameter is unused, but it is required for correct operation of the parser. The same is true with the last four trailing constant integer values. The next E lines specify the physical links of the topology and their physical distance (in kilometres), where E is the total number of links in the topology. The format of this line is as follows (note that the topology parser expects exactly E of such lines for a topology with E links):

```
<src-node><TAB><dst-node><TAB><link-distance><TAB>20<newline>
<newline>
<newline>*
```

Again here, the last integer constant is reserved for experimental use, but still is needed for the correct operation of the parser. The same is true for the following last lines of a topology file:

```
1<newline>
<newline>
<newline>*
10
<newline>
<newline>*
```

Combining all the above line specification, the format of a complete topology file renders as follows:

```
<number of nodes><TAB><number of physical links><TAB>1<newline>
<newline>
<newline>*
<X-coordinate><TAB><Y-coordinate><TAB><Node Population>1<TAB>20<TAB>20<TAB>20<TAB><newline>
<newline>
<newline>*
<src-node><TAB><dst-node><TAB><link-distance><TAB>20<newline>
<newline>
<newline>*
1<newline>
<newline>
<newline>*
10
<newline>
<newline>*
****************END_OF_FILE_.PHYS****************
```

Examples of topology files are bundled together with the tool distribution released and can be used together with the above specification for the creation of new topologies. Note that any line that starts with the reserved character combination "//" is considered as a comment line and thus the entire line is ignored by the parser.

A.2 Traffic Matrix Input File Format

A NeDeTo traffic file starts with a line specifying the number of nodes of the corresponding network topology:

```
<number of nodes><newline>
<newline>
<newline>*
```

The next N lines, where N is the number of nodes of the network topology, correspond to a NxN matrix, where an entry X at position (i,j) specifies that X wavelengths are requested from node i to node j (it is straightforward that the main diagonal will have strictly zero entries):

```
<nr-wavelengths-1,1><TAB><nr-wavelengths-1,2><TAB>...<nr-wavelengths-1,N><newline>
<nr-wavelengths-2,1><TAB><nr-wavelengths-2,2><TAB>...<nr-wavelengths-2,N><newline>
.
.
.
<nr-wavelengths-N,1><TAB><nr-wavelengths-N,2><TAB>...<nr-wavelengths-N,N><newline>
<newline>
<newline>*
***************END_OF_FILE_.TRAFF***************
```

Examples of traffic files are bundled together with the tool distribution released and can be used together with the above specification for the creation of new input traffic matrices. Note that any line that starts with the reserved character combination "//" is considered as a comment line and thus the entire line is ignored by the parser.