

Net2Plan 0.6.0

User's manual

JULY 2018

Pablo Pavón Mariño

Contents

1	Introduction	2
1.1	A brief overlook of Net2Plan	2
1.1.1	Organization of this document	3
1.2	Accompanying book and teaching materials	4
1.3	Videotutorials	4
1.4	Installing instructions	4
1.4.1	Directories	5
1.5	Licensing	5
1.6	Authorship	5
1.6.1	Net2Plan	5
1.6.2	Java Optimization Modeler (JOM)	6
1.7	Citing Net2Plan and JOM in research works	6
1.8	Release notes	6
2	The Net2Plan network model	9
2.1	A network - <code>NetPlan</code> object	9
2.2	Nodes - <code>Node</code> object	10
2.3	Links - <code>Link</code> object	10
2.4	Traffic demands - <code>Demand</code> object	11
2.5	Multicast traffic demands - <code>MulticastDemand</code> object	12
2.6	Multicast trees - <code>MulticastTree</code> object	13
2.7	Routing of unicast traffic: source-routing vs. hop-by-hop routing	13
2.7.1	Routing loops	14
2.7.2	Source-routing of the traffic - <code>Route</code> object	16
2.8	Resources - <code>Resource</code> object	16
2.9	Shared-risk groups - <code>SharedRiskGroup</code> object	17
2.10	Multilayer networks	18
2.10.1	Network layers - <code>NetworkLayer</code> object	19
2.11	The default failure model in Net2Plan	20
2.11.1	Default failure model in multilayer networks	20
2.12	Traffic monitoring and forecasting functionalities	20
3	The Net2Plan Graphical User Interface (GUI)	22
3.1	Menu File	22
3.1.1	<code>File</code> → Options	23
3.1.2	<code>File</code> → Classpath editor	24
3.1.3	<code>File</code> → Java error console	24
3.1.4	<code>File</code> → Java error console	25
3.2	Offline network design & online network simulation tool	25
3.2.1	Network topology panel	25
3.2.2	Focus panel	29
3.2.3	View/edit network state tab	30

3.2.4	Offline algorithms tab	33
3.2.5	What-if analysis tab	34
3.2.6	View reports tab	35
3.3	Online network simulation	36
3.3.1	The event driven simulation framework	36
3.3.2	Graphical User Interface	38
3.4	Traffic matrix design	41
3.4.1	Traffic generation: general traffic models	42
3.4.2	Traffic generation: population-distance traffic model	43
3.4.3	Manual matrix introduction/editor	44
3.4.4	Traffic normalization	45
3.4.5	Creating a set of traffic matrices from a seminal one	46
3.5	Help menu	46
4	The Net2Plan Command-Line Interface (CLI)	47
4.1	Examples	47
5	Development of algorithms and reports in Net2Plan	49
5.1	Net2Plan Library, Built-in Examples and Code Repository	50
5.2	JOM: Java Optimization Modeler	50
5.3	Preparing a Java IDE for Net2Plan programming	50
6	Technology-specific libraries	52
References		54

Chapter 1

Introduction

1.1 A brief overlook of Net2Plan

Net2Plan is an open-source and free to use Java-based software, licensed under the BSD-2 Clause license. Net2Plan has its origins in September 2011, as an accompanying resource for new network optimization courses at Telecommunications Engineering degrees in the Technical University of Cartagena (Spain). After its creation, Net2Plan has spread to other Universities, and is applied in a number of works in the academia and industry.

Net2Plan was designed with the aim to overcome the barriers imposed by existing network planning tools in two forms: (i) users are not limited to execute non-disclosed built-in algorithms, but also can integrate their own algorithms, applicable to any network instance, as Java classes implementing particular interfaces, and (ii) Net2Plan defines a network representation, so-called network plan, based on abstract and technology-agnostic concepts such as nodes, links, traffic unicast and multicast demands, routes, multicast trees, forwarding rules, resources (since Net2Plan 0.5.0), shared-risk groups and network layers.

Network instances can have an arbitrary number of layers (e.g. IP, Ethernet, OTN, WDM...), arranged in arbitrary forms. Technology-specific information can be introduced via user-defined attributes attached to nodes, links, routes, layers etc. in the network plan. The combination of a technology-agnostic substrate and technology-related attributes provides the required flexibility to model any network technology within Net2Plan, an added value from a didactic point of view. In this respect, current Net2Plan version provides specific libraries to ease the design of IP, wireless and optical networks. The novel introduction of the concept of resource permits modeling NFV (Network Function Virtualization) scenarios e.g. with service chains (see Section 2.8). In version 0.6.0 we introduce the so-called NIW library, for designing NFV over IP over WDM networks, stemming from H2020 Metro-Haul project, that greatly simplifies the development of algorithms in those contexts.

Net2Plan provides both a graphical user interface (GUI) and a command-line interface (CLI). In either mode, Net2Plan includes four tools:

- *Offline network design*: Targeted to execute offline planning algorithms, that receive a network design as an input and modify it in any form (e.g. optimize the routing, the capacities, topology etc.). Algorithms based on constrained optimization formulations (e.g. ILPs or convex formulations) use the open-source freeware Java Optimization Modeler

<http://www.net2plan.com/jom>

to interface from Java to a number of external solvers such as GPLK, MIPCL, CPLEX, XPRESS

or IPOPT, that produce a numerical solution. The modeling syntax of JOM is human-readable, and capable of handling arrays of decision variables and constraints of arbitrary dimensions, facilitating the definition and solving of complex models directly from Java in a few lines of code.

- *Online simulation:* Permits building simulations of online algorithms that code how the network *reacts* to different events generated by built-in or user-developed event generation modules. For instance, it can be used to evaluate network recovery schemes that react to failures and repairs or dynamic provisioning algorithms that allocate resources reacting to time-varying traffic demands. Several built-in algorithms exist coding e.g. how IP/OSPF networks or some types of IP over WDM multilayer networks react to traffic fluctuations and failures. Also, some distributed algorithms for congestion control, capacity allocation in wireless networks and other contexts, are implemented as online algorithms, where nodes asynchronously iterate to adapt to network conditions.
- *Automatic report generation:* Net2Plan permits the generation of built-in or user-defined reports, from any network design.
- *Traffic matrix generation:* Net2Plan assists users in the process of generating and normalizing traffic matrices, according to different models. Since version 0.6.0, some basic options exist to estimate traffic matrices from link occupation monitoring information.

1.1.1 Organization of this document

The rest of this chapter is devoted to describe some basic information about:

- Section 1.2 introduces an accompanying book published by the author with the theoretical fundamentals of network optimization, and a practical approach based on Net2Plan examples. The majority of Net2Plan built-in algorithms come from examples thoroughly described in the book. Also, teaching materials available in the website are introduced.
- Section 1.3 introduces the videotutorials available in the Net2Plan website.
- Section 1.4 comments on how to install and run Net2Plan.
- Section 1.5 comments on the license of Net2Plan and JOM.
- Section 1.6 comments on the Net2Plan origins and authorship.
- Section 1.7 informs on how we would prefer the work to be cited in research publications.
- Section 1.8 describes the release notes.

Then, Chapter 2 is devoted to describe the network representation in Net2Plan, and is an unavoidable reading for understanding the tool.

Chapter 3 describes the functionalities in the graphical user interface. Chapter 4 is focused on the command-line interface. Chapter 5 is focused on the development of new algorithms and reports in Net2Plan. Finally, Chapter 6 informs on some libraries and available algorithms in Net2Plan, specific to networks technologies like IP/OSPF, wireless or WDM (optical). Also, it includes a section on how Net2Plan can be used for modeling NFV scenarios.

1.2 Accompanying book and teaching materials

Net2Plan has been extensively used as an accompanying resource of the network optimization book:

Pablo Pavón Mariño, *Optimization of computer networks. Modeling and algorithms. A hands-on approach.* Wiley, May 2016.

The book is targeted to grow in the reader the ability to model a multitude of network optimization problems, and create algorithms for them. The book materials indexed in the repository

<http://www.net2plan.com/ocn-book>

include all the examples of models and algorithms in the book implemented as Net2Plan offline and online algorithms, and reports. They can be used to find numerical solutions to multiple real-life network problems. The book also includes exercises where the reader can develop and test their own Net2Plan algorithms, applying the techniques described.

A number of teaching materials for some courses using Net2Plan are provided in the Net2Plan website (e.g. lab work wordings).

1.3 Videotutorials

In the website, there is a video tutorial section in which users can see instructions and examples of how to use Net2Plan, and how to develop algorithms for it. Unfortunately, please note that videotutorials have not been updated to newer Net2Plan versions.

1.4 Installing instructions

For installing the software, just uncompress the `.rar` file provided in any folder. For running the software just click in the `Net2Plan.jar` file. The software does not modify any registry information in the computer. For uninstalling, just remove the folder.

Net2Plan requires Java Runtime Environment 8 or higher versions and a screen resolution of, at least, 800x600 pixels. Since it is developed in Java, it works in the most well-known operation systems (Microsoft Windows, Linux, Mac OS X).

To execute Net2Plan in Graphical User Interface (GUI) mode (see Chapter 3), just double click on `Net2Plan.jar`, or execute the following command in a terminal: `java -jar Net2Plan.jar`.

To execute Net2Plan in Command-Line Interface (CLI) mode (see Chapter 4), execute the following command in a terminal: `java -jar Net2Plan-cli.jar`.

Important: Net2Plan makes use of the Java Optimization Modeler (JOM) library for solving optimization models interfacing to external solvers. JOM is shipped with Net2Plan, and used in a number of built-in algorithms and some functionalities. Please, follow the instructions in the JOM website (<http://www.net2plan.com/jom>) to install the external solvers needed. If these solvers are not installed, Net2Plan still works correctly, but the user cannot access the subset of functionalities and algorithms using JOM.

1.4.1 Directories

The directories in the Net2Plan installation are:

- **doc/help**: includes this user's guide.
- **doc/javadoc**: the Javadoc of Net2Plan (needed by algorithm developers).
- **lib**: Includes auxiliary libraries needed by Net2Plan.
- **src**: Includes the Net2Plan code, and some examples.
- **plugins**: Includes the Java classes and source code of some of the plugins in which Net2Plan is organized.
- **workspace**: includes example code and data.
- **workspace/data**: includes example topologies and traffic matrices.

1.5 Licensing

Net2Plan is free and open-source. It is licensed under the BSD-2 Clause license.

1.6 Authorship

1.6.1 Net2Plan

Net2Plan tool has its origins in 2011, during the preparation of the teaching materials for two new courses at Universidad Politécnica de Cartagena (Spain) taught by Prof. Pablo Pavón Mariño, in Telecommunications Engineering degrees:

- *Telecommunication networks theory* (2nd year, 2nd quarter).
- *Network planning and management* (3rd year, 2nd quarter).

Pablo Pavón Mariño and José Luis Izquierdo Zaragoza were the developers of Net2Plan up to version 0.3.1¹, and Pablo Pavón continued with the project from version 0.4.0 onwards.

Since July 2016 (version 0.4.1), Net2Plan is in GitHub (<https://github.com/girtel/Net2Plan/>), and some other people from GIRTEL research group (<http://girtel.upct.es>) have been contributing, as can be seen in the GitHub tracks. I would like to thank all of them, as well as the many students, practitioners and researchers using Net2Plan, for their fruitful feedback.

¹Up to version 0.3.1, Net2Plan was part of the Ph.D. work of José Luis Izquierdo Zaragoza, supervised by Prof. Pablo Pavón Mariño.

1.6.2 Java Optimization Modeler (JOM)

Prof. Pablo Pavón Mariño is the author of JOM (Java Optimization Modeler), an open-source Java library for modeling and solving optimization problems in a simple MATLAB-like syntax. JOM is a library extensively used in Net2Plan algorithms that numerically solve network problems by means of optimization solvers.

The JOM website is:

<http://www.net2plan.com/jom>

1.7 Citing Net2Plan and JOM in research works

In research works, Net2Plan can be cited using the publication:

P. Pavón-Marino, J.L. Izquierdo-Zaragoza, “Net2plan: an open source network planning tool for bridging the gap between academia and industry”, IEEE Network, vol. 29, no 5, p. 90-96, October/November 2015.

To cite JOM, please use the web site link:

<http://www.net2plan.com/jom>

Many built-in algorithms and reports in Net2Plan (many of them using JOM) are thoroughly described in the book:

Pablo Pavón Mariño, *Optimization of computer networks. Modeling and algorithms. A hands-on approach.* Wiley, May 2016.

1.8 Release notes

- Net2Plan 0.6.0 (July, 2018)

- Possibility to attach monitoring traces of carried traffic (series of pairs date-traffic) to the links, and also traces of offered traffic to demands and multicast demands. Methods to synthetically create, import and export from Excel, and manipulate those traces are included in the GUI. See Section 2.12.
- Basic forecasting functionalities, also accessible from the GUI, to predict traffic in links and demands from the monitoring traces. See Section 2.12.
- Basic traffic matrix estimation functionalities, also accessible from the GUI, to predict demands and multicast demands traffic, from link occupations. See Section 2.12.
- QoS functionalities. Demands and multicast demands can be tagged with a user-defined QoS. Then, the user can define scheduling policies, that assign link bandwidth to the traversing demands and multicast demands with user-defined priorities, according to their QoS.
- Resources can now be detached from the nodes, and attached later to other nodes.
- Nodes now can have different (X,Y) positions at different user-defined layouts.
- Links can now be coupled to demands in the same layer, as long as no coupling cycles occur.

- Demands can now be of the type source routing or hop-by-hop routing. Previously, all the demands in the same layer were forced to be of the same type.
- Multicast trees can now be trees that reach just a subset of the egress nodes of the associated demand.
- Shared risk groups can now also be *dynamic*, meaning that the associated links and nodes are defined by a function of the network state, not statically set.
- Internally, most of the methods work with `SortedSets` instead of `Set`, so the ordering when iterating those sets is platform independent (in its turn, the order in `Set` is not defined, and can change from one from one program execution to the next).
- Multiple improvements in the GUI:
 - * Possibility to navigate through the tables, e.g. picking the origin node of a link in the links table, picks that node in the nodes table. It is possible to move back and forward the picked elements.
 - * Easier selection of the current active layer in a left panel.
 - * The tabbed panel at the right, now shows ALL the layers information, not just the active layer.
- Net2Plan 0.5.0 (February 15, 2017)
 - Major visualization improvements:
 - * Possibility to plot the network over maps, using OpenStreet Map (provided under ODbL license <http://www.openstreetmap.org/copyright>).
 - * 3-D perspective of the network plot, to be able to see multiple layers simultaneously, seeing e.g. how the traffic flows between layers, or the failures in a layer propagate to others.
 - * Nodes and resources can have icons associated.
 - * Undo/redo functionality.
 - * Addition of a *what-if analysis* panel that eases the fast visualization of the network reaction under link or node failures, and/or to traffic anomalies.
 - * Addition of a focus panel to provide more detailed information of the selected network elements, eases the navigation.
 - * Introduction of visualization filters, that permit e.g. focusing on the links, demands, etc. in other layers that put traffic to/receive traffic from a link.
 - New modeling possibilities, with the introduction of the concept of *Resource*, e.g. to model NFV scenarios. In this context, demands represent service chain requests and routes are service chains that can traverse links and resources, and can occupy a different amount of capacity in each link/resource traversed.
 - New algorithms have been introduced to produce designs handling service chains (e.g. in NFV scenarios).
 - The concept of *ProtectionSegment* has been removed. In its turn, the possibility of tagging a route as a *backup route* of other has been added.
- Net2Plan 0.4.0 (February 15, 2016)
 - Major changes in the form in which the network model is programmed. Now `NetPlan` object gives access to a number of other elements in their own classes: `Node`, `Link`, `Demand`, `MulticastDemand`, ... Also, all the elements have both an identifier (`long`, maybe non-consecutive, never changes along time) and an index (`0,1,...`) (0-indexed and consecutive, removing an element renames the rest). Link, node, demand etc. indexes are amenable as array indexes. Then, the laborious `Map`-based organization of the elements in the network model is not longer needed. In general, the algorithms' code get much simpler and clearer.

- Full support for multicast traffic.
- Some new options in the graphical user interface.
- Full re-elaboration and reorganization of the built-in examples and reports, including plenty of new algorithms, to make Net2Plan an accompanying resource of [1].
- Net2Plan 0.3.1 (November 23, 2015)
 - Minor changes.
 - Improved documentation.
 - New: A new plugin architecture (undocumented) has been created to support the integration of external CLI/GUI tools or I/O filters. Original plugins are detached from the kernel and are also located into plugins folder.
- Net2Plan 0.3.0 (June 29, 2015)
 - Major changes in network model:
 - * New: Complete multilayer support, including layer coupling (links at an upper layer become demands at a lower layer, or viceversa).
 - * New: Identifiers for nodes, links, demands, and so on, are now long values. `get()` methods, whose output were arrays, now return maps.
 - * New: Internal speed-up via caching of common `get()` methods.
 - New: *Online simulation* is a new tool that merges the previous simulators into a common one, and it uses the same network model than for network design.
 - Improved documentation
- Net2Plan 0.2.3 (March 7, 2014)
 - Minor changes.
 - Improved documentation.
- Net2Plan 0.2.2 (October 16, 2013)
 - Minor changes
 - Improved documentation
- Net2Plan 0.2.1 (May 23, 2013)
 - New: *Time-varying traffic* simulator.
 - New: *Network design* is now able to execute multilayer algorithms.
 - Minor changes.
- Net2Plan 0.2.0 (March 18, 2013)
 - Initial Java version, many major changes from latest MATLAB version

In its very early stage Net2Plan was designed as a MATLAB toolbox. From version 0.2.0, previous MATLAB versions were discontinued, thus backward-compatibility is not ensured at all. However, interested users can find them in the website.

Chapter 2

The Net2Plan network model

This chapter describes the network model used in Net2Plan, and the key Java classes used to represent it.

2.1 A network - NetPlan object

A network design is stored into a data structure so-called network plan, represented by the class:

```
com.net2plan.interfaces.networkDesign.NetPlan
```

in the Net2Plan library). The `NetPlan` object is used in all the Net2Plan functionalities:

- In the offline network design tool, algorithms receive a network plan and return a modified network plan. E.g. an algorithm optimizing the routing, expects to receive a `NetPlan` object with nodes, links and traffic, and adds the routing information to it.
- In the online tool, the `NetPlan` object contains the state of the network at a particular simulation moment. Online algorithms receive the current design, an event to react to (e.g. new traffic, a failure or reparation...) and produce the new network state by modifying the given `NetPlan` object.
- Reports are just built-in or user-defined that receive a design (`NetPlan` object) and produce a HTML file.

There is one and only one `NetPlan` object in each network representation, that gives access through its methods to all the network elements (nodes, links, demands, etc.), which are represented in the model by specific classes like `Node`, `Link`, `Demand`, `MulticastDemand`...

The structure of the `NetPlan` class is cornerstone to understand how Net2Plan works. Below, we describe separately each of the elements that make up a design. For each of the elements described below, and for the `NetPlan` object itself, the user can attach the following information:

- *Id* (`long`): A unique identifier or serial number assigned by the kernel, that never changes along the life of the `NetPlan` object. Elements created later receive higher numbers, but not necessarily consecutive.

- *Index (int)*: An identifier (*int*) assigned by the kernel 0,1,2,... to the elements inside a given network layer, or globally (for nodes, resources, SRGs, elements not attached to a particular layer). Indexes are renumbered when an element of its same category is removed. For instance, the links in the same layer have an index, starting from 0 and growing as consecutive numbers, so elements created before have a lower index. If link with index 2 is removed, links with index 3, 4,... will automatically decrease its index in one unit.
- A network name (an arbitrary *String*).
- A network description (an arbitrary *String*).
- User-defined name-value attributes, where each attribute has a key (*String*) associated to a value (*String*).
- User-defined tags, where each tag is a *String*.

2.2 Nodes - Node object

Nodes are the basic entity of a network design. They can be the end points of the links, the sources or destinations of traffic, and also forward traffic not targeted to them.

Each node in a network is represented by a *Node* object, contained inside the *NetPlan* object. Specific node information contained in this object is:

- *(X, Y) coordinates*: The coordinates of the node in a bidimensional Cartesian plane. Serves for visualization and can be optionally used to automatically compute the length of the links between the nodes. If the visualization over a map is activated, (X,Y) node coordinates are interpreted as the node longitude/latitude in degrees. It is possible to define different layouts, so (X,Y) node positions are different at different layouts.
- *Up/down state*: A node can be *up* (working correctly) or *down* (failed). In the latter case, Net2Plan assumes that it is not able to forward traffic. The carried traffic and occupied link capacity of all the traversing routes or multicast trees is then set to zero, and is set back to its previous value when the node gets up again.
- *Population*: A number representing a node population, in any units.
- *Site*: A node can be assigned to a user-defined site (identified by a *String*).
- *Icon URL (URL)*: An optional URL indicating where to obtain an icon for representing the node in the graphical interface. Different icons can be indicated for displaying the nodes at different layers.

2.3 Links - Link object

Links are elements connecting the nodes, with the capability of carrying traffic between them. Links are always unidirectional. A link starts in one node, and ends in a *different* node (self-links are not allowed). Two nodes can be connected by zero, one or more links.

Each link in a network is represented by a *Link* object. Each link is characterized by:

- *Origin node (Node)*: The node where the link starts.

- *Destination node (Node)*: The node where the link ends.
- *Network layer (NetworkLayer)*: (Of interest in multilayer designs) The layer where the link belongs to. A link belongs to one and only one network layer. Note that in multilayer networks, nodes are not attached to any particular layer, while links are. Then, nodes are the elements in charge of moving traffic from one layer to other (traffic entering a node through a link at a given layer, and leaving it through other link at other layer).
- *Capacity (double)*: The capacity of the link, measured in the link's layer capacity units (the capacity of all the links in the same layer is measured in the same units).
- *Length (double)*: The length in km of the link.
- *Bidirectional pair*. Links can be assigned a bidirectional pair, which is a link in the same layer with the same end nodes, in opposite direction.
- *Propagation speed (double)*: The propagation speed of the signal along the link. Typically 200,000 km/s in wired networks, and 300,000 km/s in wireless. This is used in delay calculations.
- *Up/down state*: A link can be *up* (working correctly) or *down* (failed). In the latter case, Net2Plan assumes that it is not able to forward traffic. The carried traffic and occupied link capacity of all the traversing routes or multicast trees is then set to zero, and is set back to its previous value when the link gets up again.
- *QoS scheduling information*. This takes the form of a map (`Map<String,Pair<Integer,Double>`), associating a QoS identifier, with two values: (i) a priority order (lower better), (ii) maximum fraction of link occupation assignable. The link assigns bandwidth to incoming demands in order, according to its priority, and up to a maximum, according to its maximum utilization. Not assigned traffic is still carried, but tagged as violating the link QoS. Algorithms can read such QoS violations and act accordingly.
- *Link monitoring information*. A traffic series object of traffic carried monitoring information along time. A trace is represented by a `TrafficSeries` object, containing a list of pairs (date, traffic).
- *Coupled demand (Demand)*: (Of interest in multilayer designs) In multilayer networks, a link in an upper layer can be coupled to a unicast demand in a lower layer with the same end nodes, or a multicast demand in a lower layer with common initial node, and being the link end node one of the multicast demand egress nodes. Coupling reflects that the lower layer demand is realizing the link, and that the traffic in the link, propagates down to the lower layer. In these cases, the link capacity is no longer defined by the user, but automatically made equal to the coupled demand carried traffic. See Section 2.10 for further information.

2.4 Traffic demands - Demand object

Unicast traffic is modeled through a set of demands (or *commodities*). Each demand represents an *offered* end-to-end unidirectional traffic flow to the network, between two different particular nodes (self-demands are not allowed). Two nodes can have zero, one or more demands between them.

Each demand in a network is represented by a `Demand` object. Each demand is characterized by:

- *Ingress node (Node)*: The node where the demand starts.
- *Egress node (Node)*: The node where the demand ends.

- *Network layer* (`NetworkLayer`): (Of interest in multilayer designs) The layer where the demand belongs to. A demand belongs to one and only one network layer.
- *Offered traffic* (`double`): The amount of offered traffic, measured in the demands' layer traffic units (the offered traffic of all the demands in the same layer is measured in the same units). The traffic that is actually carried depends on how the demand traffic is routed (see Section 2.7).
- *Routing type*. Demands can be of the type source routing (and then its traffic is carried via routes), or hop-by-hop routing (and thus its traffic is carried via forwarding rules).
- *Bidirectional pair*. Demands can be assigned a bidirectional pair, which is a demand in the same layer with the same end nodes, in opposite direction.
- *Maximum acceptable latency* in ms. The user-defined limit to the end-to-end latency of the traffic of this demands. Violations are highlighted in the GUI.
- *QoS type*. A user-defined `String` representing the QoS type of the demand traffic.
- *Demand monitoring information*. A traffic series object of offered traffic information along time. A trace is represented by a `TrafficSeries` object, containing a list of pairs (date, traffic).
- *Coupled link* (`Link`): (Of interest in multilayer designs) In multilayer networks, a demand in the lower layer can be coupled to a link in an upper layer with the same end nodes, to reflect that the lower layer demand is realizing the link. In these cases, the link capacity is no longer defined by the user, but automatically made equal to the coupled demand carried traffic. See Section 2.10 for further information.
- Since Net2Plan 0.5.0, a demand can be assigned a sequence of *resource types*. If so, all the routes in the demand must traverse in its path as many resources as the number of types designated, and the types of the traversed resources must correspond to the ones indicated, in the same order. See Section 2.8 for a deeper explanation.

2.5 Multicast traffic demands - `MulticastDemand` object

Multicast traffic is modeled through a set of multicast demands. Each demand represents an *offered* multicast traffic flow, starting in a particular ingress node, and ending in a particular set of egress nodes (different to the ingress node). The number of multicast demands affecting a node is arbitrary.

Each multicast demand in a network is represented by a `MulticastDemand` object. Each multicast demand is characterized by:

- *Ingress node* (`Node`): The node where the multicast demand starts.
- *Egress nodes* (`Set<Node>`): The set of nodes where the multicast demand ends.
- *Network layer* (`NetworkLayer`): (Of interest in multilayer designs) The layer where the multicast demand belongs to. A multicast demand belongs to one and only one network layer.
- *Offered traffic* (`double`): The amount of offered traffic, measured in the demands' layer traffic units (the offered traffic of all the demands in the same layer is measured in the same units). The traffic that is actually carried depends on how the multicast demand traffic is routed through its associated multicast trees.
- *Maximum acceptable latency* in ms. The user-defined limit to the end-to-end latency of the traffic of this demands. Violations are highlighted in the GUI.

- *QoS type*. A user-defined **String** representing the QoS type of the demand traffic.
- *Demand monitoring information*. A traffic series object of offered traffic information along time. A trace is represented by a **TrafficSeries** object, containing a list of pairs (date, traffic).
- *Coupled links (**Set<Link>**)*: (Of interest in multilayer designs) In multilayer networks, a multicast demand in the lower layer can be coupled to a set of links in the upper layer, all of them starting in the demand ingress node, and ending in each of the multicas demand egress nodes. The coupling reflects that the lower layer multicast demand is realizing the set of links. In these cases, the link capacities are no longer defined by the user, but automatically made equal to the coupled multicast demand carried traffic. See Section 2.10 for further information.

2.6 Multicast trees - **MulticastTree** object

A multicast tree is an element carrying the traffic of a multicast demand. It is composed of a set of links comprising a unidirectional tree, starting in the associated multicast demand ingress node, and ending at a subset (maybe empty) of the demand egress nodes. Multicast trees must connect the ingress node and each of the egress nodes without loops. This results in that the number of links in the tree will be equal to the number of nodes minus one.

A multicast tree can be assigned to only one multicast demand, but a multicast demand can be carried by zero, one or more trees. When the number of trees is two or more, we say that the multicast routing is bifurcated.

Each multicast tree in a network is represented by a **MulticastTree** object. Each multicast tree is characterized by:

- *Associated multicast demand (**MulticastDemand**)*: The multicast demand of which this tree is carrying traffic. The ingress and egress nodes of the demand must be the ones of the tree.
- *Traversed links*. The set of links (maybe empty), composing a tree initiated in the demand ingress node, and ending in a subset of the demand egress nodes.
- *Carried traffic (**double**)*: The amount of traffic that this tree is carrying, measured in the tree layer traffic units.
- *Occupied link capacity (**double**)*: The amount of capacity in the traversed links that this multicast tree occupies, measured in the tree layer link capacity units. Typically, the demand traffic and link capacity are measured in the same units (e.g. Gbps), and the tree carried traffic equals its occupied link capacity.

2.7 Routing of unicast traffic: source-routing vs. hop-by-hop routing

The routing of a demand is the form in which its offered traffic is carried on the layer links. In Net2Plan there is only one form of routing the multicast traffic: using multicast trees. However, Net2Plan permits defining *two* different forms of routing the *unicast* traffic of a demand:

- *Source-routing*: In source routing, the traffic demand is assigned a set of routes, from its ingress to its egress node. A route defines a sequence of traversed links, the amount of traffic of the demand that it carries (in traffic units) and the amount of capacity that consumes in each link (in link capacity units). Source routing is characteristic of connection-oriented technologies like MPLS,

ATM, OTN or SONET/SDH, where a flow completes a connection establishment stage before sending any data. During this stage, the network decides and preconfigures the flow routing in the traversed nodes.

Since Net2Plan 0.5.0:

- The capacity occupied by a route can be different in different traversed links. This can model e.g. situations where the traffic gets *thinner* along the route, since some part of it is blocked in intermediate nodes. In its turn, the carried traffic in a route is unique, and represents the amount of traffic that successfully reaches the destination.
- A route path is composed by an arbitrary sequence of links *and resources*, to extend the concept of route, to that of a *service chain*. See Section 2.8 for details.
- *Hop-by-hop routing*: In hop-by-hop routing, nodes in the network define the routing using so-called forwarding rules. Forwarding rules are triples (d, e, f) , where d is a demand, e a link and f a number between 0 and 1. f represents the fraction of traffic that appears in the origin node of link e (either is generated by it if it is the ingress of d , or enters it through the input links), that is forwarded through link e .

Typically, the forwarding rules of the output links of a node are configured in structures called *routing tables* or *forwarding tables* in it. Given a node n , and a demand d , the forwarding rules of d associated to the node output links must sum at most 100%:

- If they sum 100%, the node forwards all the traffic of the demand. This is the typical case when the node is not the egress node of the demand.
- If n is the end node of the demand d , the output forwarding rules typically sum zero. Net2Plan assumes that the non-forwarded traffic was successfully received.
- If they sum less than 100% (e.g. 0%), and the node is *not* the end node of the demand, Net2Plan assumes that non-forwarded traffic is dropped in the node.

Forwarding rules reflect better the behavior of connectionless routing, where a source can inject traffic without a previous connection establishment. The two common cases are IP and Ethernet networks. These networks are based on forwarding rules defined in the nodes, with the particular aspect that forwarding decisions typically depend solely on the destination of the traffic. To model this with Net2Plan, all the forwarding rules in a node, for the demands which have the same egress node (e.g. whatever its ingress node is) should be the same.

A note about translations between different routing types:

- Source routing can be always translated into hop-by-hop routing, except when service chains are defined (a demand has a non-empty list of resource types to traverse). The translation assumes that the occupied traffic in the links equals the carried traffic.
- Translating from hop-by-hop routing to source routing, may be not possible. If forwarding rules are defined such that traffic gets trapped into an open or close loop, their equivalent routes cannot be found.

2.7.1 Routing loops

In both source-routing and hop-by-hop routing looping situations can occur.

In source routing, a route is defined by specifically determining the sequence of links to traverse. Net2Plan allows routes which traverse a node and/or a link more than once, and thus routings with arbitrary loops can be defined, as long as they are of finite length.

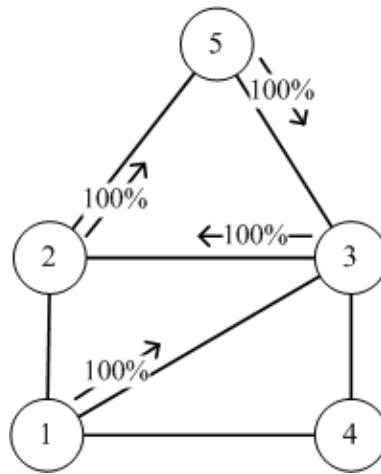


Figure 2.1: Example of a closed loop (3-2-5), for a demand from node 1 to node 4.

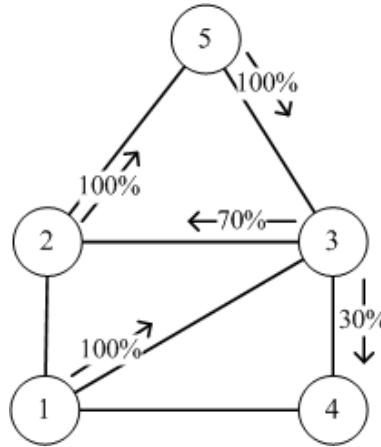


Figure 2.2: Example of an open loop (3-2-5), for a demand from node 1 to node 4.

The definition of the forwarding rules can also create loops, as occurs in reality. However, the routing loops in hop-by-hop networks have a defining aspect: the traffic can potentially make an infinite number of hops when it enters a routing loop. We distinguish two different situations:

- Closed loops where the traffic entering in them never reaches the destination (e.g. Fig. 2.1). In this case, the traffic only enters the loop and never leaves it, and thus accumulates and saturates the link capacity.
- Open loops where the traffic entering a loop can make a number of cycles, and eventually leave it reaching the destination- Fig. 2.2 shows an example. In this case, a traffic unit to node 4 reaching node 2, could enter the cycle between nodes 2-5 or nodes 2-3-5. If the forwarding decision in each node is randomly taken, the probability of staying in the loop after k hops is positive for any k .

Naturally, looping situations in hop-by-hop networks would reflect a wrong design of the routing tables. Net2Plan is equipped with the appropriate functions to detect them, and to determine the traffic in the links from the offered traffic and forwarding rules also in this cases. Further details on the techniques to do that can be consulted in Chapter 4 of [1].

2.7.2 Source-routing of the traffic - Route object

Demands with a source-routing routing type can be assigned an arbitrary set of routes, which determine how the demand traffic is carried. Each route is defined by a sequence of *links* of the same layer where the demand belongs to, that make up a path from the demand ingress node to the demand egress node.

A route can be assigned to only one demand, but a demand can be carried by zero, one or more routes. When the number of routes is two or more, we say that the unicast routing is bifurcated.

Since Net2Plan 0.5.0:

- A route path can contain not only links, but also *Resource* objects hosted in the traversed nodes. Then, the route concept is able to model *service chains*. The types of the traversed resources must match the sequence of resource types defined in the demand. The occupied capacity can be different in every link and/or resource.
- A route can be tagged as a *backup route* of one or more routes of the same demand. A route r which is backup of a route p , cannot have itself other backup routes. Designating routes as backup can be of interest for some algorithms that e.g. 1+1 protection systems can be modeled by adding to each original route, other route with a disjoint path, designating it as its backup.

Each route or service chain in a network is represented by a `Route` object. Each route is characterized by:

- *Associated demand (Demand)*: The demand of which this route is carrying traffic. The ingress and egress nodes of the demand must be the ones of the route. The types of the traversed resources of the route must be the same as the sequence of resource types defined in the demand.
- *Bidirectional pair*. When the assigned demand has a bidirectional pair, routes can be assigned a bidirectional pair, which is a route of the bidirectional demand pair.
- *Sequence of links (and potentially resources) (List<NetworkElement>)*: The sequence of links/resources of the route. The sequence should start at the demand ingress node, end at the demand egress node, and be contiguous: a link ends in the same node as the next link starts, and a resource traversed after a link should be hosted in the link end node.
- *Carried traffic (double)*: The amount of traffic that this route is carrying (the amount that successfully reaches the destination), measured in the route layer traffic units.
- *Occupied link/resource capacities (List<Double>)*: The amount of capacity in the traversed links/resources that this route occupies, in the same order as the sequence of links/resources. The amount associated to the links is assumed to be measured in the route layer link capacity units. The amounts of the resources are measured in each resource designated capacity units. Typically, the demand traffic, link and resource capacities are measured in the same units (e.g. Gbps), and the route carried traffic equals its occupied link capacity.

2.8 Resources - Resource object

A *resource* is an element associated to a particular node, with the following characteristics:

- A resource has a particular *type*. For instance, “Virtual Firewall”, “NAT engine”, “Web server” or “RAM”, “CPU”, “HardDisk” are typical types in NFV contexts.

- A resource has a capacity, measured in the user-designated resource capacity units.
- A resource can be traversed by routes (and then these routes are typically called service chains), and each traversing route occupies an arbitrary amount of resource capacity.
- A resource can have an arbitrary set of *base resources* associated to it, which must be hosted in the same node, and occupy in each of the base resources an arbitrary amount of capacity. For instance, a resource “Virtual firewall” can have as base resources “RAM”, “HD” and “CPU” and occupy in each 4 GBytes, 8 GBytes and 1 CPU respectively. Base resource relations permit model situations where resources are *instantiated* (like in NFV scenarios), consuming other *base resources* (e.g. raw IT resources like CPU, RAM or HD).

Note: The resource type cannot change after resource creation. The base resources occupied can change, as long as the resource has no upper resources. The capacity that a resource occupies in its base resource can always change: for instance, a “virtual firewall” resource can variate the amount of CPU/RAM/HD consumed depending on its traversed traffic.

Each resource in a network is represented by a `Resource` object, contained inside the `NetPlan` object. Specific node information contained in this object is:

- *Type (String)*: An arbitrary `String` with the type of the resource. The resource type defines the routes that can traverse it, and how: recall that demands can establish a sequence of resource types to traverse.
- *Host node (Node)*: The node where this resource exists. If the node fails, or the resources in it also fail. Resources can be not attached to any node. In that case they cannot have base resources, nor be traversed by any route. Resources hosted by a node can be detached from it, and later attachde to other node.
- *Capacity (double)*: The capacity of the resource, a non-negative number.
- *Capacity units (String)*: An arbitrary string indicating the units in which the resource capacity is measured.
- *Processing time (ms) (double)*: An arbitrary non-negative quantity, so that traffic traversing the resource is assumed to be delayed by this amount.
- *Icon URL (URL)*. An optional URL indicating where to obtain an icon for representing the resource in the graphical interface.

2.9 Shared-risk groups - `SharedRiskGroup` object

A shared-risk group (SRG) represents a particular risk of failure for the network that, if happens, creates a simultaneous failure in a particular set of links and/or nodes. For instance, a SRG can be associated to the risk of accidentally cutting a particular duct that holds the links between two nodes (e.g. one in each direction). If this cut occurs, the two links would fail simultaneously. Then, they would remain unavailable until a reparation of the damage is completed.

Each SRG in a network is represented by a `SharedRiskGroup` object. Each SRG is characterized by:

- *Mean Time To Fail (MTTF) (double)*: The average time between two consecutive failures.

- *Mean Time To Repair (MTTR)* (`double`): The average time between the moment the failure occurs, until it is repaired.
- *Associated set of nodes* (`Set<Node>`): The nodes that simultaneously fail when the risk associated to the SRG occurs.
- *Associated set of links* (`Set<Link>`): The links that simultaneously fail when the risk associated to the SRG occurs.

SRGs are used to model the failure risks that threaten the network, and eases the design and evaluation of the network recovery mechanisms. As an example, SRG information is used by some built-in online algorithms that create failure and reparation events in the network according to the statistical information in the SRGs, and send these events to the recovery algorithms that must react to them. Also, some built-in Net2Plan reports can estimate analytically the availability of the network for arbitrary network recovery mechanisms (protection or restoration based), from the SRG information.

2.10 Multilayer networks

Communication networks are organized into layers, governed by different protocols and potentially managed by different companies or institutions, such that the links in an upper layer appear as traffic demands carried by typically a lower layer in an underlying topology. For instance, in IP over WDM optical networks:

- The upper layer is composed of a set of IP routers, connected through optical connections of fixed capacity (e.g. 10 Gbps, 40 Gbps, 100 Gbps) called lightpaths. The IP routers see each lightpath as a direct link or pipe to other router, and the traffic is routed on top of the lightpaths according to the IP nodes routing tables.
- In the upper layer, each lightpath is a demand to carry traversing a path of optical fibers in the underlying topology of optical fibers. Each lightpath is assigned a wavelength, that cannot be changed along its route, unless wavelength conversion devices are available. The optical switching nodes forwarding the lightpaths are called Optical Add/Drop Multiplexers (OADM)s.

Thus, in the previous example, a lightpath appears to the IP layer as a direct link between two routers of a fixed capacity, irrespective of the actual route of the lightpath across the fibers. The topology of IP links (each corresponding to a lightpath) is usually referred to as the *virtual topology*, since each link is not backed by an actual wire, but by a lightpath that follows an arbitrary optical path across the fiber topology.

Multiple other examples exist of multilayer networks. For instance, a common three-layer structure is that of IP routers connected through a topology of OTN circuits, that are routed on top of a topology of lightpaths, that are routed on top of a topology of optical fibers.

Since Net2Plan 0.6.0 it is possible also to couple a link in a layer with a demand in the same layer. This is useful for instance in IP/MPLS situations: an MPLS circuit is a route traversing IP links, but such MPLS circuit can be coupled to a link in the same layer, and make the MPLS circuits appear as other IP links.

2.10.1 Network layers - NetworkLayer object

To be able to represent arbitrary multilayer designs, in Net2Plan, a network is composed of a number of layers, at least one. One out of them is defined as the so-called default layer. In many methods indicating the layer of a link, demand, route etc. is optional, and if not specified, the default layer is assumed. Thanks to this, common users not interested in multilayer designs can work without really knowing the possible complexities of multilayer networks. Also, it is possible to use (without any change) algorithms or reports for single layer networks in a selected layer of a multilayer design, by just setting our layer of interest as the default (e.g. selecting it in the graphical user interface).

According to the Net2Plan multilayer model, the following elements are associated to one and only one network layer:

- Link
- Demand
- Route
- Forwarding rules
- Multicast demand
- Multicast tree

That is, the offered traffic, the links, and how the traffic is routed on them can be defined differently for each network layer.

The following elements are not associated to a particular network layer:

- Nodes: a node can have input/output links at different layers.
- Resources: a resource can be traversed by routes belonging to different layers.
- Shared risk groups: Represent a risk of failure that can affect e.g. links at different layers.

Each layer in a network is represented by a `NetworkLayer` object. Each network layer is characterized by:

- *Traffic units (String)*: A string defining the units in which the offered and carried traffics in the layer are measured (e.g. Mbps).
- *Link capacity units (String)*: A string defining the units in which the capacity of all the links in the layer is measured (e.g. Mbps). Typically, traffic and link capacity units are the same. However, in some occasions it may be interesting to have different units (e.g. traffics in Mbps and link capacities in “number of channels”, or in “MHz”).
- *Icon URL (URL)*. An optional URL indicating where to obtain an icon used as the default icon for representing the nodes in the graphical interface, when they are displayed at a particular layer.

2.11 The default failure model in Net2Plan

In order to evaluate network recovery mechanisms, the network model allows setting *up* or *down* nodes and links. The *default* behavior of Net2Plan corresponds to a network that makes nothing to adapt to the failures, and just the affected traffic is dropped.

The default reaction of Net2Plan to link and node failures is described below.

- When a link sets its state to *down* (failed), then:
 - All the traversed routes set their carried traffic and occupied capacity in the traversed links to zero. Calling the methods `getCarriedTraffic` or `getOccupiedLinkCapacity` for these routes would return a zero. However, calling the methods `getCarriedTrafficInNoFailureState` and `getOccupiedCapacityInNoFailureState` would return the nominal carried traffic and occupied link capacities, the one that would exist if no failed links/nodes were traversed.
 - All the forwarding rules of the link are set to zero, for all the demands, and the routing for all those demands is recomputed.
- When a link sets its state to *up* (is repaired), then:
 - All the traversed routes are checked. If thanks to this repair, now they traverse only up nodes and links, its carried traffic ad occupied link capacities get their nominal values (the ones returned by `getCarriedTrafficInNoFailureState` and `getOccupiedCapacityInNoFailureState` methods of the route).
 - All the forwarding rules of the link take the nominal values, for all the demands, and the routing in the is recomputed.

The failure and repair of a node is equivalent to simultaneous failure and repair of all the in/out links of the node.

2.11.1 Default failure model in multilayer networks

Recall that in the multilayer representation of Net2Plan, upper layer links are implemented as traffic demands in the lower layer, and the upper layer link capacity is made equal to the coupled demand carried traffic.

A failure in the links of a lower layer, can make the carried traffic of a demand in the same lower layer drop, even become zero. This capacity update is automatically seen by the coupled *upper layer* link. However, the upper layer link *does not become automatically down*. Users developing network recovery algorithms for multilayer networks should take this into account (e.g. if they want to propagate the failure to the upper layer, by setting the upper layer links as down, they have to program this behavior themselves).

2.12 Traffic monitoring and forecasting functionalities

Net2Plan 0.6.0 includes new functionalities related to traffic monitoring and forecasting. These functionalities are available as right-click options in the links, demands and multicast demands tables.

- *Monitoring traces.* It is possible to attach to links, demands and multicast demands, *traffic traces*. A traffic trace is a time series, a sequence of pairs (date , traffic). They are assumed to correspond to carried traffic for the links, and offered traffic for the demands. Users can load these traffic traces e.g. from network monitoring systems, and import them into Net2Plan via an Excel importing option. They can also create synthetic traces of such monitoring information for simple models (e.g. linear or exponential traffic growths with added random noises).
- *Traffic forecasts from traces.* For any link, demand or multicast demand, it is possible to create a prediction or forecast of the traffic for a given date, computed from regression models applied to the monitored information for that link or demand. Several methods are offered, some of them offering statistical confidence guarantees.
- *Traffic demand estimation from link traffics.* This functionality offers simple models for estimating the offered traffic of the demands for a particular date, based on (i) the link monitoring information for that date, (ii) optionally merged with predictions of the demand traffic coming each demand monitoring traces if available.

Chapter 3

The Net2Plan Graphical User Interface (GUI)

The graphical user interface of Net2Plan is launched by double clicking in the `Net2Plan.jar` file after decompressing the file with the Net2Plan distribution. It is also possible to launch it with the command `java -jar Net2Plan-jar` from a console.

Fig. 3.1 shows the initial user interface, where a welcome message is printed. From it, the user can choose among the following menus:

- **File**, to access some general configuration options. They will be described in Section 3.1.
- **Tools**, gives access to the main tools within Net2Plan: *offline network design & online network simulation* and *traffic matrix manipulation*. They will be described in Section 3.2 and 3.4 respectively.
- **Help**, gives access to Net2Plan documentation and the welcome(about) screen. This menu is described in Section 3.5.

3.1 Menu File

This menu has four options: *Options*, *Classpath Editor*, *Show Java console*, and *Exit*.

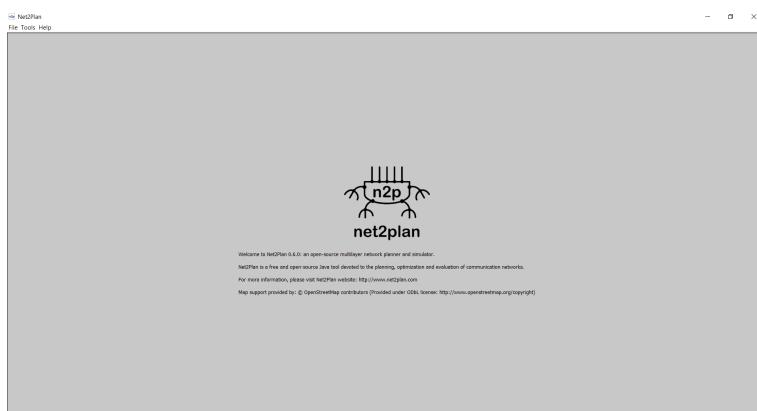


Figure 3.1: Net2Plan welcome screen.

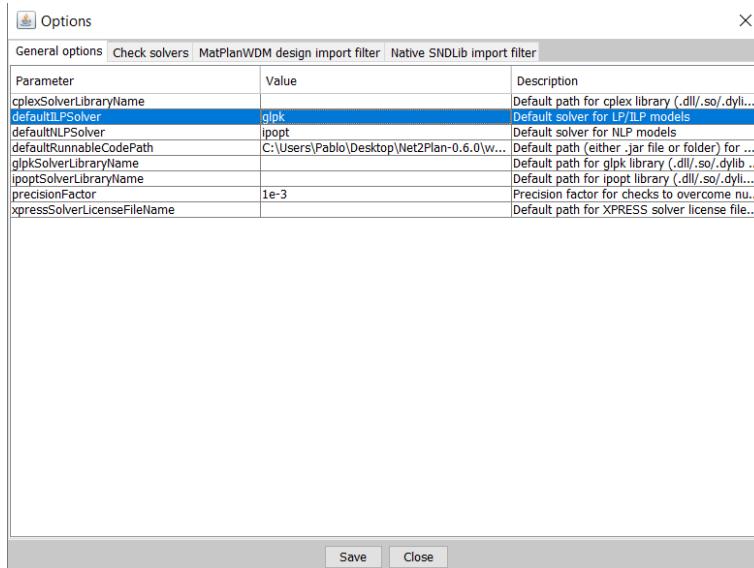


Figure 3.2: Configurable options in the `Options` menu.

3.1.1 File → Options

Use `Options` to set Net2Plan-wide parameters. These options have a global scope to all Net2Plan modules: are used within the kernel, and, for instance, to compute delay metrics in built-in reports. The description of each parameter is included next to its name. Users implementing their own algorithms/reports have read access to these parameters, as a map that links the parameter name and its current value.

In this version the general configurable options (Fig. 3.2) are:

- **precisionFactor**: Precision factor for checks to overcome numeric errors. This parameter allows considering in the kernel small tolerances in the sanity-checks of the network designs. It avoids situations in which numerical inaccuracies (e.g. caused by finite precision of the solvers) would be interpreted as errors. For instance, if an algorithm returns a design where the traffic carried by a link is 10.0000001 and link capacity is 10, the kernel may show a warning. The precision factor applies since the actual check performed has a margin given by the precision factor. Default value of `precisionFactor` is 10^{-3} , and its value is constrained to be in range (0,1).
- **defaultRunnableCodePath**: Default path that will be used by tools in the GUI as the first option to load external code (e.g. algorithms). It can be either a `.jar` file or a folder. Default value is the `BuiltInExamples.jar` included within Net2Plan.
- **defaultILPSolver**: Default solver to be used for solving Linear Programs (LP) or Mixed Integer Linear Programs (MILP). Default: `glpk`.
- **defaultNLPsolver**: Default solver to be used for solving Non-Linear Programs (NLP). Default: `ipopt`.
- **cplexSolverLibraryName**: Default path for cplex library (.dll/.so file). Default: None
- **glpkSolverLibraryName**: Default path for glpk library (.dll/.so file). Default: None
- **ipoptSolverLibraryName**: Default path for ipopt library (.dll/.so file). Default: None
- **xpressSolverLicenseFileName**: Default path for XPRESS solver license file (if installed). Typically `xpauth.xpr`. Default: None



Figure 3.3: Java error console.

Default solvers are used for a few internal operations requiring a solver (e.g. the option of multi-cast tree automatic creation, or some traffic normalization features in the traffic design tool). External algorithms from users, or even built-in examples may have their own solver-related parameters. Regarding the solver library names, they are used if, and only if, an algorithm specifies two solver-related parameters, solverName (i.e. cplex) and solverLibraryName, but the solver library name is left empty. Otherwise, the non-empty default value for the algorithm (e.g. *cplex.dll*) will be used by default.

The tabs *MatPlanWDM design import filter* and *Native SNDLib import filter* are specific options of two plugins of Net2Plan, that permit reading network files in the old MatPlanWDM format (MatPlanWDM is a MATLAB-based planning tool developed by the author, now discontinued), and in the SNDLib format (files in the repository of network topologies <http://sndlib.zib.de/>).

3.1.2 File → Classpath editor

Although a moderate library set is provided within Net2Plan, users may require extra Java libraries (*.jar* files) to develop their own algorithms or reports (e.g. mathematical or graph theory libraries). So, the classpath editor avoids the tedious task of including Java libraries in environment variables (i.e. **CLASSPATH** in Windows).

Important: In the current version of Net2Plan Java libraries can be included in run-time, but unfortunately it is not possible to do the same to remove libraries. In this case, user is forced to restart Net2Plan.

3.1.3 File → Java error console

This feature centralizes the error handling within Net2Plan. When an error is thrown, for example, due to invalid input parameters in an algorithm, the error and the stack trace is shown there. Moreover, **System.out/System.err** is redirected there also, allowing users to debug their Java code. The console can be accessed also using the combination **ALT+F12**.

Important: Due to limitations in Java Virtual Machines, when JNI/JNA for native library access is used, the native output (i.e. `stdout` in C/C++) is not equivalent to the Java output, thus such information will not appear in the Java error console. A workaround is to start the GUI from the command-line.

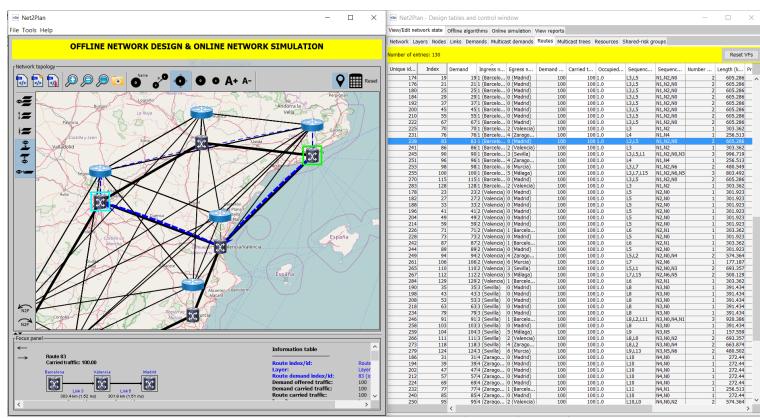


Figure 3.4: Network design window.

3.1.4 File → Java error console

Quits Net2Plan.

3.2 Offline network design & online network simulation tool

Selecting the submenu **Tools** → **Offline network design & online network simulation tool** (or **ALT+1**) opens the offline & online network design tool. This tool is targeted to operate in different forms with network designs, internally represented by **NetPlan** object, containing the elements: network nodes, links, unicast and multicast traffic demands, routes, multicast trees, network layers, resources and SRGs, as described in Chapter 2.

The graphical interface give access to the following features, described later in different subsections:

- Network visualization, optionally over background maps, in the topology panel (Section 3.2.1).
 - View/edit the network in the inspection tables (Section 3.2.3).
 - Navigate through detailed information on different network elements in the focus panel (Section 3.2.2).
 - Run offline network design algorithms (Section 3.2.4).
 - Run online network simulations (described in a separated section, Section 3.3).
 - Configure automatic reactions of the network for what-if analysisRun online network simulations (Section 3.2.5).
 - Produce automatic reports on network design (Section 3.2.6).

3.2.1 Network topology panel

This panel shows graphically the current network design, and permits modifying some parts of it. Users are able to add or remove nodes and links, zoom out and zoom in (and reset zoom), save a screenshot of the currently shown topology (in PNG format), and other options to tune the display.

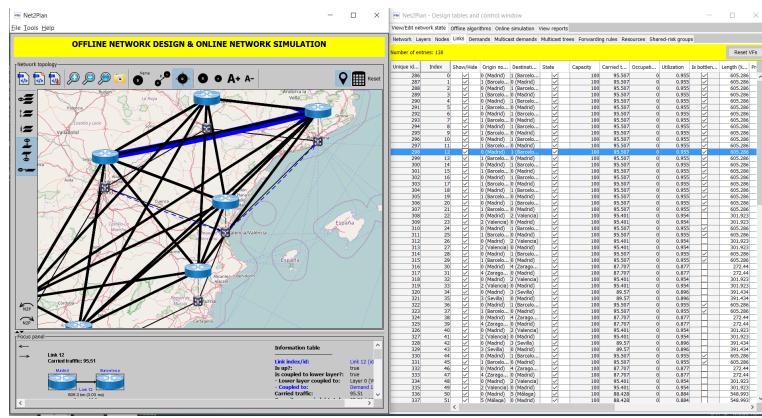


Figure 3.5: Clicking an IP link.

Multilayer display

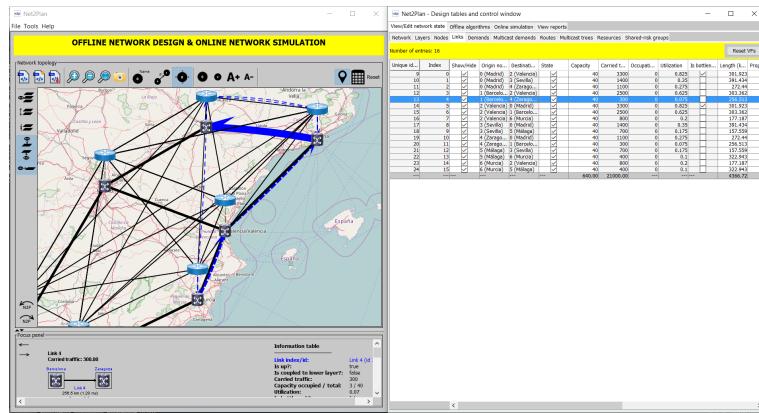
Net2Plan 0.5.0 introduced the multilayer graphical view of the network topology. This consists in:

- Network layers in multilayer networks are printed at different heights.
 - While nodes are network elements that do not belong to any layer, they can be printed differently (e.g. different icons) at different layers (e.g. see Fig. 3.4, with an IP over WDM design). The different node icons at different layers are connected by virtual interlayer links with dashed lines.
 - Links and nodes at the default (or active) layer are printed thicker and bigger.

Enhanced visualization functionalities have been added to the multilayer display. They permit visualizing how the traffic propagates between the network layers. These functionalities are described below, using as an example an IP over WDM design like the one shown in Fig. 3.4, where IP traffic is realized through WDM lightpaths. Nodes at the IP layer are routers, and nodes at the WDM layer are OADMs. Dashed lines between the IP router and OADM in the same node represent how the routers injects IP traffic in the lightpaths initiated in the node, or receives it from lightpaths ending in the node.

The enhanced visualization functionalities mean that selecting some elements like links, demands, routes, SRGs etc. in the table, in the focus panel or in the topology panel, other network elements are highlighted to provide insight on how the traffic flows in the different layers. In particular:

- Selecting a link we can highlight (i) the other links and demands in the same layer that carry/produce traffic that appears in the link, (ii) the links in upper layers that produce the traffic that appears in the link, (iii) the links and demands in lower layers that carry the traffic of this link. For instance, clicking an IP link (Fig. 3.5), we can see the lightpaths that carry the link traffic, clicking a WDM fiber link we can see (Fig. 3.6) the IP links and IP demands that put traffic in it.
 - Selecting a demand or multicast demand in a layer highlights the links in that layer that carry its traffic, the links in upper layers that put traffic in that demand, and the links in lower layers that carry that traffic. For instance, clicking an IP demand (Fig. 3.7), we see the IP links carrying its traffic, and for each the lightpaths that realize the IP links. Clicking a WDM demand (lightpath) we see (Fig. 3.8) the traversed fibers, and the IP link coupled to it.



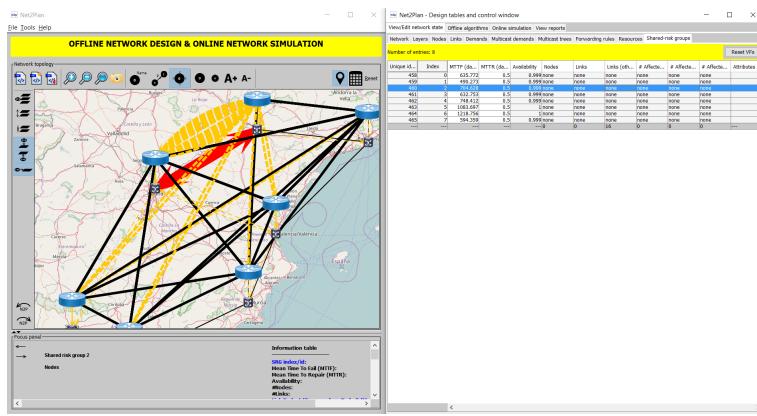


Figure 3.9: Clicking a SRG.

- Selecting a forwarding rule, route or a multicast tree we can see its path in the same layer, the links in upper layers that put traffic in the route/tree/forwarding rule, and the links in lower layers that carry the traffic of the route/tree/forwarding rule links.
- Selecting a SRG, we can see highlighted in thick red lines the nodes/links of the SRG, and in thinner red lines, the links in all the layers that carry traffic that is affected by the SRG. For instance, Fig. 3.9 shows how clicking an SRG consisting in the two WDM fibers between two nodes, we see the links carrying some traffic affected by it. Note that being affected by a SRG does not mean being *lost* if the SRG risk happens: it would only be lost if the network makes no recovery action.

Basic topology manipulations

Some basic topology manipulations are possible in the topology panel:

- *Add nodes.* Nodes are inserted by right clicking into the canvas and using the option *Add node here*.
- *Remove nodes.* Nodes can be removed by right clicking on them and using the option *Remove node*.
- *Move nodes.* It is possible to move nodes by dragging them, while pressing the **CTRL** key.
- *Add link.* Links are inserted by clicking first in the origin node and then in the destination node. It is possible to insert unidirectional links or bidirectional ones (in this latter case, the user must press **SHIFT** key during that process). Note that in Net2Plan all the links are unidirectional, and with *bidirectional link* we mean the automatic creation of two unidirectional links of opposite directions. Finally, links can be also inserted by right clicking over the origin node and selecting the destination node in the popup menu.
- *Change default layer.* Double-clicking a node in a particular layer automatically makes that layer the default or active one.

Tuning the visualization

It is possible to tune the visualization using different controls in icons and buttons in the panel. They are described below:

1. Button  loads a network design from a **.n2p** file. Loaded design becomes the current network design, previous design is lost.
2. Button  loads a **.n2p** file, but only extracting the set of offered unicast and multicast demands from it (ignoring any other information). The loaded traffic demands replace the set of demands in the current network design, and leaves unchanged the rest of the current network design. Typically, the loaded file was generated using the *Traffic matrix design* functionality. If the number of nodes in the loaded file and the current network design are different, the operation is not completed and an error message is shown.
3. Button  saves current design into a **.n2p** file.
4. Buttons ,  and  make a zoom-in, zoom-out and zoom-all respectively of the shown design.
5. Button  allows to take a snapshot of the canvas and save it to a **.png** file.
6. Button  toggles between showing or not node names next to them in the canvas.
7. Button  toggles between showing or not the link utilizations next to the links (measured as the total traffic divided the total capacity).
8. Button  toggles between showing or not non-connected nodes (those without input nor output links at the shown layer).
9. Buttons  and  increase/decrease the node sizes respectively.
10. Buttons  and  increase/decrease the font sizes respectively.
11. Button  toggles between showing or not the topology over a map brought from OpenStreet map. This option requires Internet connection, since the maps are retrieved on the fly. In addition, when this happens (X,Y) positions of the nodes are interpreted as the longitude/latitude of the nodes in degrees. If any node is outside the range [-90,90], the map is not shown.
12. Button  shows and puts focus on the right panel table.
13. Button *Reset* erases the current network design, which becomes an empty design.
14. Buttons  and  permit increasing or decreasing the vertical separation between the layers in the visualization.
15. Button  activates or deactivates the functionality of showing how the traffic in a layer link, demand etc. propagates *to lower layers*.
16. Button  activates or deactivates the functionality of showing how the traffic in a layer link, demand etc. propagates *from upper layers*.
17. Button  activates or deactivates the functionality of showing how the traffic in a layer link, demand etc. propagates *in the same layer*.
18. In the left side, there is panel that shows the current layer names, and permits selecting the active layer, show/hide layers, and changing the vertical ordering of the layers in the visualization.

3.2.2 Focus panel

The Focus Panel appears at the bottom of the topology panel, as shown in Fig 3.4.

In this panel you can see some figures and detailed information associated to the selected items (links, nodes, routes, resources, etc.) in the tables or in the topology panel. The focus panel permits

also *navigating* among different elements. For instance, when a route information is shown, it is possible to click in the links or nodes, to select them, and thus make them appear in the focus panel.

It is possible to control the navigation among selected elements using the buttons \leftarrow and \rightarrow for undoing/redoing the visualization. Undoing a visualization action, focuses in the last element selected, redoing reverts this. Everytime that a change occurs in the current NetPlan object, the visualization undo list is cleared.

3.2.3 View/edit network state tab

The **View/edit network state** tab shows complete information about the current network design, including some basic statistics and warnings that permit to visually fast-check the design and its performances.

Also, this tab permits completing some simple modifications in the design, like adding/removing any element layers, nodes, links, unicast and multicast demands, routes, multicast trees, forwarding rules, resources and SRGs, setting the capacity of links, the offered traffic of demands, or the carried traffic of the routes.

The *View/edit network state* tab is organized into sub-tabs. Left-most tab *Network* provides summary statistics of the network. Then, one subtab exists for each network layer, which itself contains a number of subtabs to give access to the per-layer information. Each subtab inside each layer corresponds to each of the elements in the Net2Plan network representation: **Nodes**, **Links**, **Demands**, **Multicast demands**, **Routes**, **Forwarding rules**, **Resources** and **Shared-risk groups**. Some general statements applicable to all the sub-tabs:

- In the tables shown, fields coloured in gray are not editable, since they show information calculated from other base fields.
- Right-clicking in the sub-tabs provide fast-access to popup menus with some element-related specific actions.
- Placing the mouse in a column or a field shows a help message with detailed information of its content.
- Clicking on a column name reorders the table rows according to it (more clicks toggle between ascending and descending ordering).
- (**New in Net2Plan 0.5.0**) The last row of the tables contain aggregated information. Sometimes this aggregated information is the column sum (e.g. in the link tabs, the total capacity summing all the link). In other occasions it shows the maximum value shown in the column (e.g. the maximum worst case end-to-end propagation delay in demands).
- Columns in the table can be hidden or shown, and its order can change. Also, the column associated to element attributes can now be expanded into one column per attribute, to ease the attribute manipulation.
- For those columns showing a real number the user can increase or decrease the number of decimal positions shown.
- (**New in Net2Plan 0.6.0**) Cells showing a network element, act as hyperlinks. For instance, in the links table, a column shows the column ingress node. The, clicking in a cell, makes the visualization show the node table, picking the selected node. The navigation path is stored, and it is possible to navigate back and forward the picked elements using ALT+LEFT or ALT+RIGHT.

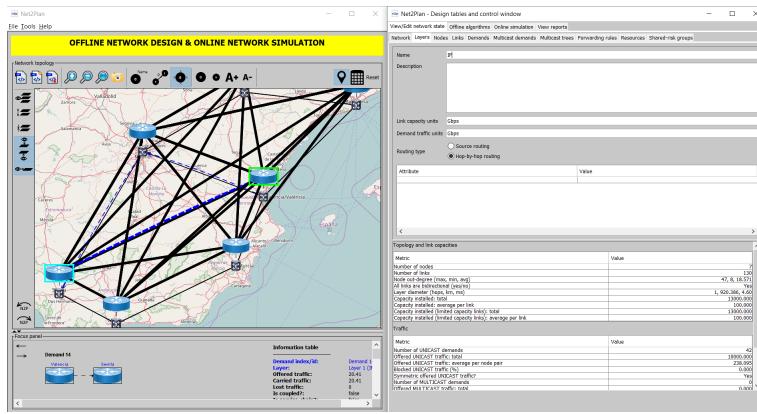


Figure 3.10: Network design window. Layer tab.

- By right-clicking in a row, you gain access to visualization *Filters*. For instance, right clicking in a link, we can filter-out from the tables all the elements (demands, links, etc.) but those in this layer or other layers that carry traffic that appears in the link. Filters can concatenated, so whenever a filter is applied, the already not-shown elements because of previous filters are not shown. By clicking the *Reset VFs* button in the upper part of the table, the filters are removed, and all the elements are shown.

A more detailed description of the information in each sub-tab follows.

View/edit network state → Network tab

This tab shows statistical information describing the current network design at a network level (e.g. see Fig. 3.4).

- *Name*, *Description* and *Attributes*: Shows the network name, description message, and set of user-defined key-value parameters associated to the `NetPlan` object in the design. Right-clicking in the **Parameters** panel permits adding/removing/editing attributes.
- *Number of layers*, *Number of nodes*, and *Number of SRGs* are read-only fields displaying this information.
- **Layer information**: This panel shows basic information about all the layers in the network such as name, description, link and demand units, attributes, and number of items for each layer-dependent element (links, demands, and so on). The user can add/remove layers by right clicking on the table and using the corresponding option.

View/edit network state → Layer tab

This tab shows statistical information describing the current network design at a layer level (see Fig. 3.10).

- Upper part of this tab shows and permits editing the layer name, a layer description message, link and capacity units, and the set of user-defined key-value parameters associated to the `NetworkLayer` element in the design. Right-clicking in the **Parameters** panel permits adding/removing/editing attributes.

- Four tables showing network performance metrics associated to the layer are provided. By placing the mouse over a metric name, a full description is provided. The tables are:
 - *Topology and link capacities* table contain statistics related to the network nodes and links.
 - *Traffic* table provides statistics regarding the unicast and multicast offered and blocked traffic.
 - *Routing* table provides information regarding the routing, like the average number of hops, symmetry, bifurcation, or existence of loops.
 - *Resilience information* table is only active when the routing is of the *source-routing* type. It provides information regarding the backup routes defined.

View/edit network state → Nodes tab

This sub-tab shows the information related to network nodes. Clicking on each node highlights it in the left panel.

It is possible to show/hide a node and setting them as up/down. The up/down option allows users to play with the network to see the effect, in terms of traffic losses, of a failure. We would like to remark, that when setting up/down a node the standard Net2Plan reaction described in Section 2.11 is applied (and not any user-defined recovery mechanism).

View/edit network state → Links tab

This sub-tab shows the information related to network links. Clicking on each link highlights it in the left panel.

Similarly to the nodes, it is possible to show/hide a link and setting it as up/down. The behavior of Net2Plan in both cases is analogous to that with the nodes.

View/edit network state → Demands tab

This sub-tab shows the information related to traffic demands. Clicking on each demand highlights its associated ingress and egress nodes in the left panel, and all the links in the network carrying traffic of the demand. When the routing is of the source routing type, the links associated to backup routes are drawn in yellow.

View/edit network state → Multicast demands tab

This sub-tab shows the information related to multicast traffic demands. Clicking on each demand highlights its associated ingress and set of egress nodes in the left panel, and all the links in the network carrying traffic of the demand.

View/edit network state → Routes tab

This sub-tab appears only if the layer routing is of the source-routing type. It shows the information related to the routes defined.

View/edit network state → Multicast trees tab

This sub-tab shows the information related to the multicast trees defined. Clicking on each tree highlights its traversed links in blue.

View/edit network state → Forwarding rules tab

This sub-tab appears only if the layer routing is of the hop-by-hop type. It shows all the forwarding rules defined in the network layer. Clicking on each forwarding rule highlights the associated demand ingress and egress nodes, and the link.

View/edit network state → Resources tab

This sub-tab shows the information related to the resources defined. Clicking on each resource highlights its associated node, if any.

View/edit network state → Shared-risk groups tab

This sub-tab shows the information related to the SRGs defined. Clicking on each SRG highlights its associated links and/or nodes (the ones that simultaneously fail when the SRG fail).

3.2.4 Offline algorithms tab

This tab permits the user running built-in or user-defined algorithms, that take the current network design (shown in the tables and in the panel), and modify it in any form e.g. routing algorithms that receive a design with nodes, links and offered traffic, and add the routing information to it). The word offline here means that all the variables in the network plan are supposed to be static (do not change along time). For instance, offered traffics are assumed to be constants representing the average traffic volumes, although in reality the traffic can fluctuate around this average according to statistical patterns.

The *Offline algorithms* permits the users configuring the algorithm execution. The input parameters of the algorithm are: (i) the current network design as stated before, (ii) a set of algorithm-defined parameters, and (iii) Net2Plan-wide parameters. Algorithms are expected to modify the passed network design (`NetPlan` object), producing a new network design that becomes the current one. Aside of modifying the passed `NetPlan` object, algorithms can return an output message string. Fig. 3.11 shows an example.

To execute an algorithm, users should specify the Java class (implementing the `IAlgorithm` interface) containing the algorithm implementation. A `.class` file can be selected using the `Load` button. In addition, a `.jar` file can be also selected. In that case, the pull-down menu below permits selecting one among the `.class` files in the `.jar`, that implement the `IAlgorithm` interface.

Once an algorithm is selected, the `Description` text field shows the algorithm description as returned by the `getDescription()` method of the algorithm. The `Parameters` panel shows the set of input parameters of the algorithm. Net2Plan invokes the algorithm `getParameters()` method to obtain the list of input parameters, with a name, a default value and a description message for each. This information is displayed in the `Parameters` panel. Then, the graphical interface allows the user modifying the value of any parameter before running the algorithm.

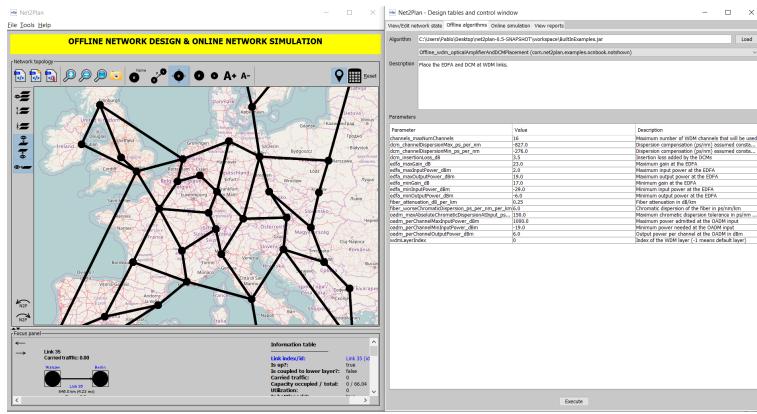


Figure 3.11: *Algorithm execution* tab.

The algorithm is executed pressing the **Execute** button. At this moment, Net2Plan invokes the `executeAlgorithm()` method of the algorithm, passing as inputs the current network design, the values of the input parameters (as `String` objects, any parsing should be done by the algorithm), and the current values of the Net2Plan-wide parameters (see Section 3.1.1). The `executeAlgorithm()` method returns a `NetPlan` object that becomes the current network design. If the method raises a `Net2PlanException`, it is shown in the window. If the method raises any other `Exception`, the stack trace is printed in the Java console for helping the users to debug their algorithms. In either case, if the `executeAlgorithm()` raises any exception, the current design is unchanged, whatever changes to it were made in the algorithm before the exception was raised.

To see more information about how to develop user-made offline network design algorithms see the Chapter 5.

3.2.5 What-if analysis tab

In this tab, the user can activate or deactivate the *what-if analysis mode*, by toggling the activation button (Fig. 3.12). When this mode is active, the user will be able to easily see the reaction of the network under (i) failures and reparations of links and nodes, (ii) traffic anomalies taking the form of a change in the offered traffic of one or more demands.

In the what-if analysis pane, the user should choose an event-processor algorithm that is expected to code how the network reacts to such changes.

As described in Section 3.3, event processor algorithms can be also used for event-driven simulation of the network. In the what-if analysis, the processor algorithm will receive (i) the current input design, and (ii) an event with the action to take. The events received will be one among these types:

- `SimEvent.DemandModify`, that signal that a demand has modified its offered traffic. This will occur when the user changes the demand offered traffic in the demands table.
- `SimEvent.MulticastDemandModify`, that signal that a multicast demand has modified its offered traffic. This will occur when the user changes the demand offered traffic in the multicast demands table.
- `SimEvent.NodesAndLinksChangeFailureState`, that signal that a link and/or node has changed its up/down state. This will occur when the user clicks in the checkbox in the *State* column of the links or nodes tables, that change the state of a node/link from up to down or from down to up.



Figure 3.12: *What-if analysis* tab.

3.2.6 View reports tab

In this panel users can select a report to apply to the network plan. The structure is similar to that for executing algorithms.

To run a report, users should specify the Java class (implementing the `IReport` interface) containing the report code. A `.class` file can be selected using the `Load` button. In addition, a `.jar` file can be also selected. In that case, the pull-down menu below permits selecting one among the `.class` files in the `.jar`, that implement the `IReport` interface.

Once a report is selected, the `Description` text field shows the report description as returned by the `getDescription()` method of the report. The `Parameters` panel shows the set of input parameters of the report. Net2Plan invokes the `getParameters()` method which returns the list of input parameters, with a name, a default value, and a description message for each. This information is displayed in the `Parameters` panel. Then, the graphical interface permits the user modifying the value of any parameter before running the report.

The report is executed pressing the `Show` button. At this moment, Net2Plan invokes the report `executeReport()` method, passing as inputs the current network design, the values of the input parameters (as `String` objects, any parsing should be done by the algorithm), and the current values of the Net2Plan-wide parameters (see Section 3.1.1). The `executeReport()` method returns a `String`, which is interpreted as an HTML file, and shown in the tabs in the lower part of the tab.

Users can see a report in a browser using the option `View in navigator`, or even saving it to an external HTML file.

Reports can be closed individually using the `CTRL+W` combination.

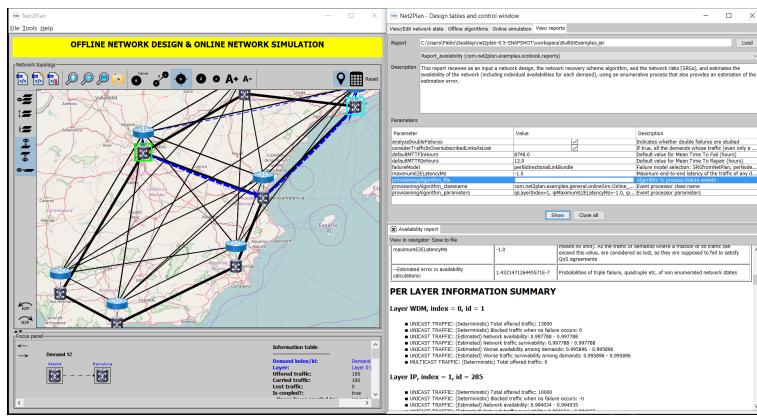


Figure 3.13: *View reports* tab.

3.3 Online network simulation

In a real-world environment, network conditions vary during its operation, according to different phenomena. Failures in nodes and links, establishment of new virtual circuits, or variation in traffic volumes are some examples. In this case, users could be interested in analyzing, using an event-driven simulation, how their networks react to those changes and how their designs are consequently adapted for them.

Net2Plan provides a post-analysis simulation tool that allows to the user the (joint) evaluation of the availability performance of protection and restoration algorithms in the network, the performance of on-line provisioning schemes that allocate resources to incoming connections (e.g. virtual circuits requests, lightpath requests, phone calls, multimedia sessions), the performance of dynamic allocation algorithms which react to variations in traffic demand volumes, or in general any allocations during network operation. Allocation is not only referred to modify traffic routing, but also it means that the network topology may change along time (e.g. adding new links, updating the link capacities, instantiating new resources...) for adapting to the new network conditions.

3.3.1 The event driven simulation framework

The architecture of the simulator is based on the well-known discrete-event simulation paradigm. The network operation is modeled as a discrete sequence of events in time. Each event occurs at a particular time instant and marks a change of state in the system. Between consecutive events, no change in the network occurs; thus the simulation can directly jump in time from one event to the next.

Event generator and event processor modules

An online simulation is governed by two objects in Net2Plan. For both, the user can develop its own algorithm, or use a built-in one:

- *Event generator.* The event generator is an object implementing the interface `IEventGenerator`. Its typical use is implementing the code that generates the external events that the network is going to react to: e.g. traffic variations, failure/repairs. Event generators can send and receive events, but *cannot* change the current state of the network, represented by a `NetPlan` object.

Example. The built-in class `Online_evGen_generalGenerator`, implements a generator which

can produce fast traffic variations mixed or not with slow (multi-hour) traffic variations and/or failure/repair events in the network.

- *Event processor.* The event processor is an object implementing the interface `IEventProcessor`. Its typical use is consuming the events produced by the event generator, implementing the particular form in which the network state will change. Then, when consuming an event, the event processor will receive from the kernel the event to react to, and the current network state as a `NetPlan` object, and should return the network reaction by modifying the given `NetPlan` object, which will become the new network state.

Example. The built-in class `Online_evProc_generalProcessor`, implements a processor which allows the user to choose among some allocation schemes to react to the events created by the `Online_evGen_generalGenerator` module.

The `SimEvent` object

Both event generator and processor can send events to themselves and to the other module. Their difference is mainly that event generators cannot modify the current network state.

An event is an object implementing the interface `SimEvent`, which at least contains the event time, event priority (for ordering the simultaneous events among them) the destination module (generator or processor), and any `Object` containing specific information of the event.

Some built-in classes extending `SimEvent` exist, that are used by the built-in algorithms provided as examples in the Net2Plan repository. These are basic events like events to add/remove traffic demands, or to signal a change in the up/down state (fail/repair) of nodes and links.

The user can implement their own `SimEvent` classes, or use these ones. The full list of classes implemented can be seen in the Javadoc, as public classes inside `SimEvent`.

The simulation cycle

The Net2Plan kernel is in charge of governing the simulation. The complete process is described below:

1. The user loads the event generator and event processor modules. The kernel prints the description of each module and their lists of input parameters by calling the methods `getDescription` and `getParameters`.
2. The user can set the values of the input parameters in the GUI. Also, can set some simulation wide parameters (not algorithm dependent) like the simulation duration, transitory time or how some internal kernel statistics should be computed.
3. The user starts the simulation pressing the *Run* button. The `NetPlan` object corresponding to the network design in the topology panel becomes the current network state. The kernel calls the `initialize` method of the event generator, and the event processor. It passes them the current network state (the event generator cannot modify it), and the parameters. Any initialization routines that these modules need should be coded there. Also, at least one of these methods should produce events using the `scheduleEvent` method. If not, there will be no events to consume, and the simulation will end.
4. The Net2Plan kernel handles the simulation loop. It keeps a list of future events to consume (FEL, *Future Event List*), ordered according to the global simulation clock (which initially started in zero), and repeats the *event scheduling loop*:

- Advance the clock to the time of the first event in the list (the one with the lowest time, using the event priority to order among simultaneous events).
 - Call the `processEvent` method of the event destination module (generator or processor), passing to it the event to consume, and the current network state. Both event generator or processor can produce new events (only with a time equal or higher than the current simulation time). However, only the event processor can call the methods in the `NetPlan` object of the current state that modify it. A `UnsupportedOperationException` is raised if they are called in the event generator.
5. In the event scheduling loop, when the clock reaches the time configured as transitory time in the simulation-wide parameters, the kernel calls the method `finishTransitory` of both event generator and event processor. Typically, these methods reset any internal variables for computing statistics that they may have. Also the kernel computed statistics are reset. The event generator or event processor can also force the kernel to call the `finishTransitory` methods and reset the kernel statistics, by invoking the method `endTransitory`.
 6. In the event scheduling loop, when the clock reaches the time configured as simulation end time in the simulation-wide parameters, the kernel calls the method `finish` of both event generator and event processor. These methods return a `String`, which is later printed by Net2Plan in the simulation report. Then, `finish` methods typically return a short report with any internal statistics computed. The event generator or event processor can also force the kernel to end the simulation and call the `finish` methods, by invoking the method `endSimulation`.

Simulation statistics

Two types of statistics are collected during the simulation, and then printed in the final report:

- Net2Plan general statistics. If the `disableStatistics` option is not set, Net2Plan computes a complete set of statistic on the simulation evolution. They are technology-agnostic statistics like the blocked traffic (observing the demands offered and carried traffics), average link occupations, etc.
- Event generator or event processor internal statistics. Typically, these modules can collect and then return in the `finish` method algorithm-specific statistics.

3.3.2 Graphical User Interface

Selecting the *Online simulation* tab gives access to the online simulation tool. Typically, to perform a simulation, a network design is loaded or comes from the execution of an offline network design algorithm. When the online tab is active, the current design becomes the initial network state.

When the simulation is paused, the current state is shown in the topology and the tables. This means that, combined with the *Step* button, it is possible to track visually how the changes occur in the network.

Simulation input parameters tab

In this tab users can set the input parameters of the simulation. In the upper part, the user can tune the global simulation parameters:

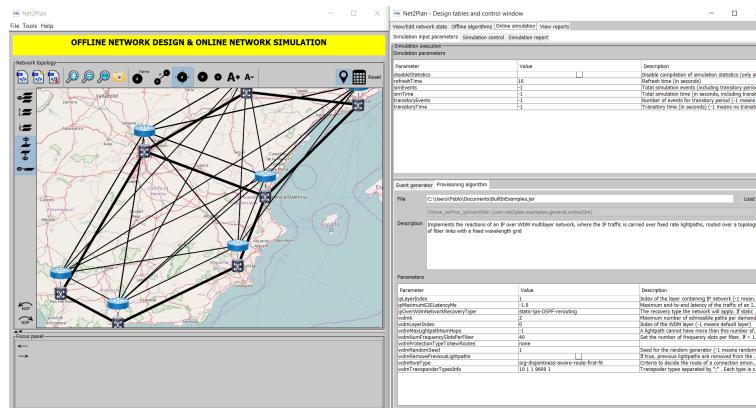


Figure 3.14: *Online simulation* tool. Execution controller panel.

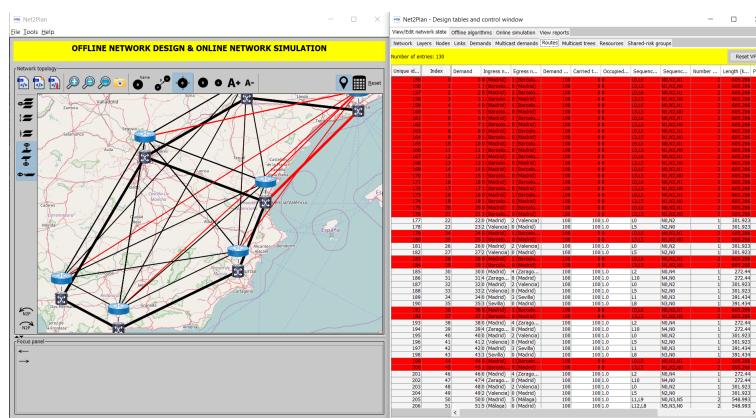


Figure 3.15: Online network design.

- **disableStatistics**. In some occasions, users might be interested in collecting only their own statistics, and would prefer avoiding the overhead that requires statistics collection by the kernel. This can be done with this parameter.
- **refreshTime**. If the option *Refresh* has been activated in the simulation controller panel, information about the current simulation, (number of processed events, simulation and CPU time...) will be shown there. This information is refreshed every number of seconds given by this parameter
- **simEvents**. Total number of events (including transitory events) to be simulated. If the parameter **simTime** is also specified, the simulation will automatically finish when the first stopping condition is met. Allowed values are integers greater than zero, or -1 for no limit (simulation must be manually stopped, or stopped by the algorithm calling the `endSimulation` method).
- **simTime**. Total simulation time (including transitory events) in seconds. If the parameter **simEvents** is also specified, the simulation will automatically finish when the first stopping condition is met. Allowed values are numbers greater than zero, or -1 for no limit (simulation must be manually stopped).
- **transitoryEvents**. Number of events for the transitory period. If **transitoryTime** is also specified, the transitory period will finish when the first condition is met. Allowed values are integers grater than 0, or -1 for no transitory period.
- **transitoryTime**. Transitory time in seconds. If **transitoryEvents** is also specified, the transitory period will finish when the condition is met. Allowed values are integers greater than 0, or -1 for no transitory period.

The lower part of the panel permits establishing the event generator and event processor algoritm that will govern the simulation:

- **Event generator**. The event generator algorithm can be specified, as a Java class extending the `IEventGenerator` class. A description of the algorithm will be shown in this panel as well as any input parameter that may be modified by the user.
- **Provisioning algorithm**. The event processor algorithm can be specified, as a Java class extending the `IEventProcessor` class. A description of the algorithm will be shown in this panel as well as any input parameter that may be modified by the user.

When the simulation is paused/stopped, users are able to save the `NetPlan` object collecting the current network state using the corresponding button in the network topology view.

Important: The simulation input parameters cannot change once the simulation is started. Users should stop and reset the simulation (clicking the `Reset` button) to perform changes in the input parameters.

Simulation control tab

In this tab users can control the execution and flow of the simulation. Several buttons can be manipulated here:

- **Run**. This button will start the simulation. Information about the current simulation will be shown in the text panel (the refresh time is defined by the `refreshTim` parameter). If a previous simulation has already been started it must be stopped and reset before starting a new one.

Future event list					
Id	Time	Priority	Type	To module	Custom object
0	3 days, 6 h, 8 m, 41.732 s	0	-1-event processor	NodesAndLinksChangeFailureState [no...	
1	3 days, 6 h, 8 m, 44.730 s	0	-1-event generator	Generate repair event in SRG 663	
2	3 days, 6 h, 9 m, 32.087 s	0	-1-event generator	Generate failure in SRG 649	
3	3 days, 6 h, 10 m, 23.006 s	0	-1-event generator	Generate failure in SRG 659	
4	3 days, 6 h, 12 m, 48.011 s	0	-1-event generator	Generate failure in SRG 653	
5	3 days, 6 h, 15 m, 41.710 s	0	-1-event generator	Generate failure in SRG 651	
6	3 days, 6 h, 24 m, 41.531 s	0	-1-event generator	Generate failure in SRG 651	
7	3 days, 6 h, 26 m, 27.525 s	0	-1-event generator	Generate failure in SRG 650	
8	3 days, 6 h, 29 m, 27.540 s	0	-1-event generator	Generate failure in SRG 652	
9	3 days, 6 h, 39 m, 43.131 s	n	-1-event generator	Generate failure in SRG 648	

Figure 3.16: Online network design. Future event list

- **Step.** Instead of running continuously, the simulation can be advanced in steps. Each time the Step button is clicked the simulation will consume one event of the future event list, and then pause and refresh the text panel.
- **Pause/Continue.** The simulation will be paused or resumed each time this button is clicked. A simulation must be started in order for this button to work.
- **Stop.** Users can stop the simulation at any time, but then the simulation cannot continue.
- **Refresh.** The information in the text panel will be refreshed if this check-box is activated.
- **View FEL:** Clicking this option will show the *Future Event List*, where users can examine the future events to be processed by the provisioning algorithm (see Fig. 3.16)

Also, the text panel shows some brief simulation state information: simulation time, CPU time, last event processed... This information is updated according to the refresh time parameter.

Simulation report tab

In this tab users can obtain the statistics collected by the kernel, and also those collected by their algorithms (event generator and event processor). Users can see the report at any moment (clicking the Update button) while the simulation is running, paused or stopped, and can be opened by a web browser, as happened with the reports for network designs.

In some occasions, users might be interested in collecting only their own statistics, and they might want to eliminate the overhead that requires statistics collection. This can be done by checking the simulation parameter `disableStatistics`. In this case, only algorithm-specific statistics are shown.

Net2Plan statistics include different network-wide, per-layer, per-node, per-link and per-demand information. Fig. 3.17 shows an example.

3.4 Traffic matrix design

As stated in Chapter 2, network design contains a set of unicast traffic demands, representing the unicast offered traffic in the network. Each demand is characterized by an ingress and egress node, and a traffic volume representing an average of the demand traffic load. In occasions, the set of traffic demands is composed of one demand for each node pair. That is, there are no two traffic demands with the same ingress and egress nodes. In these cases, it is possible to represent the demand set using a compact matrix representation, so-called traffic matrix. A traffic matrix for a network of N nodes is a $N \times N$ matrix with zeros in the diagonal. The coordinate in the i -th row and j -th column of the matrix, contains the amount of traffic generated in node i that is targeted to node j . In other words, the traffic volume of the demand associated with the (i, j) node pair.

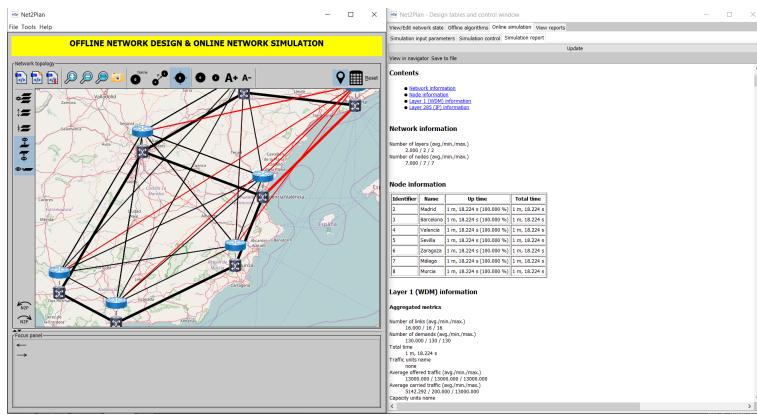


Figure 3.17: Online network design. Simulation report

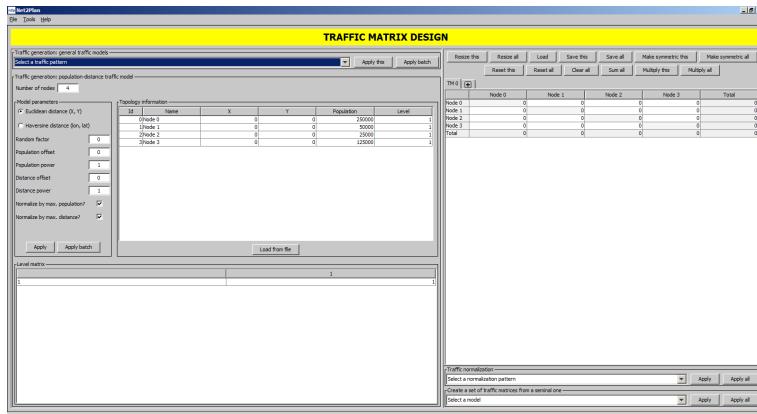


Figure 3.18: *Traffic matrix design panel tab*.

The traffic matrix design tool assists users in the process of generating user-defined traffic matrices. It permits generating new matrices manually, or following several models found in the literature (e.g. random-uniform, population-distance models...). Created matrices can be saved in .n2p format to be later applied in Net2Plan. In addition, some popular traffic matrix generation models are available in the Net2Plan library and thus can be directly integrated into Java-based design algorithms. For more information, see the class `TrafficMatrixGenerationModels` in the Javadoc.

Selecting `Traffic matrix design` under `Tools` menu (or using `ALT+2`) activates the Traffic matrix design window. Fig. 3.18 displays the workspace window for this option. The upper part of the left panel gives access to a set of general traffic generation models. Below, the user can generate matrices using one particular method: the population-distance traffic model. The right panel shows the traffic matrices generated, and permits saving (as .n2p), loading (as .n2p), resizing the matrices and some other simple modifications in the buttons above the matrices. In the lower right side, the traffic normalization panel permits applying a normalization method to one or all of the traffic matrices in the upper panel. The panel below, permits selecting a method for producing a set of traffic matrices from a seminal one, in different forms.

3.4.1 Traffic generation: general traffic models

In this panel, the user can generate one or a batch of traffic matrices, selecting one of the following traffic generation patterns:

- *Constant*. Generates a traffic matrix with a given constant value in all its coordinates.
- *Uniform (0,10)*. Generates a traffic matrix with random values in the range (0,10) .
- *Uniform (0,100)*. Generates a traffic matrix with random values in the range (0,100).
- *50% Uniform (0,10) & 50% Uniform (0,100)*. Generates a traffic matrix with 50% of its entries with random values in the range (0,100), and the rest of the entries with random values in the range (0,10) .
- *25% Uniform (0,10) & 75% Uniform (0,100)*. Generates a traffic matrix with 25% of its entries with random values in the range (0,100), and the rest of the entries with random values in the range (0,10) .
- *Gravity model*: Generates a traffic matrix according to the gravity model. The user should provide for each node n , its total traffic generated $OUT(n)$ and received $IN(n)$. Naturally, the sum of all the traffic generated by all the nodes should be equal to the sum of the traffic received by all the nodes (we denote it as H). From this information, the coordinate (i, j) of the traffic matrix is given by $OUT(i)IN(j)H$. Then, the traffic from node i to node j is proportional to the total traffic produced at i and proportional to the total traffic received at j .

In any method, diagonal values of the traffic matrix are always zero, since self-demands are not allowed. Pressing the **Apply** button, one traffic matrix is created, and shown in the right panel. In its turn, the **Apply batch** button permits creating an arbitrary number of matrices, which are also shown in the right panel.

3.4.2 Traffic generation: population-distance traffic model

In this panel, the user can generate one or a batch of traffic matrices, using the population-distance traffic matrix model described in [2]. This model receives as an input the number of nodes in the network, and for each node, its (X, Y) position, its population and a factor called node level. This information can be introduced manually in the **Topology information** panel, or loaded from a **.n2p** file, where the nodes have the population and level attribute defined.

In the population-distance model, the traffic γ_{ij} from node i to node j is calculated following the expression:

$$\gamma_{ij} = (1 - rf + 2 \times rf \times rand()) \times Level(L_i, L_j) \times \frac{\left(\frac{Pop_i Pop_j}{Pop_{max}^2} + Pop_{off} \right)^{Pop_{power}}}{\left(\frac{dist_{ij}}{dist_{max}} + dist_{off} \right)^{dist_{power}}}$$

This expression is explained in the following points:

- *Random factor* $(1 - rf + 2 \times rf \times rand())$: $rand()$, is a sample of a uniform (0,1) distribution, and $rf \in [0, 1]$ is a method parameter controlling the randomicity, since expression $(1 - rf + 2 \times rf \times rand())$ is a uniform sample in the interval $[1 - rf, 1 + rf]$. Then, if $rf = 0$, there is no randomicity (the sample is always one), if $rf = 1$, the randomicity is maximum, since the random becomes a uniform sample in the range (0, 2).
- *Node level factor* $Level(L_i, L_j)$. This factor permits multiplying the traffic between two nodes by a constant dependent on the level of each node. The $Level(L_i, L_j)$ values are defined in a $L \times L$ matrix (being L the number of levels or node types defined by the user). For instance, imagine we have a network with two types of nodes, *clients* and *servers*. We want to create a

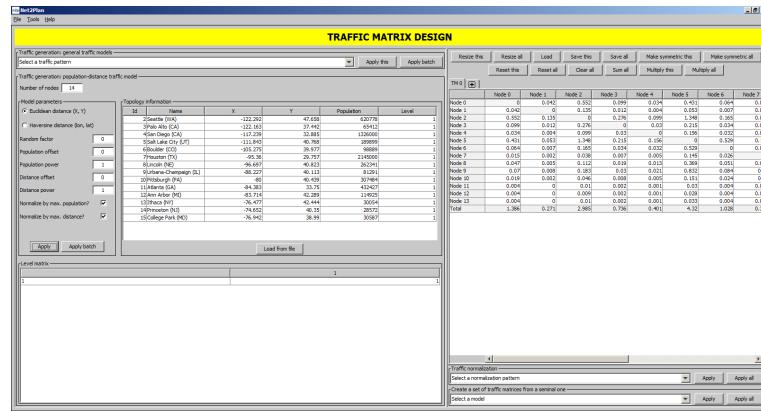


Figure 3.19: *Traffic matrix design* panel tab. Population-distance model.

traffic matrix for this network where clients do not send traffic to clients, and servers do not send traffic to servers. This can be done defining two node levels in the network for separating *client nodes* and *server nodes*. Then, we can use a level matrix with 0s in the diagonals, so the client-client traffic and server-server traffic is multiplied by 0 in the model.

- *Population factor* $\left(\frac{Pop_i Pop_j}{Pop_{max}^2} + Pop_{off}\right)^{Pop_{power}}$. The population factor makes the traffic between two nodes proportional to the product of the population of both nodes normalized by the maximum node population Pop_{max}^2 . The factor Pop_{off} is used to smooth the effects of the product (e.g. if a population is 0, the traffic is still not zero). The Pop_{power} factor controls the effect of the population in the matrices. Typical values are:
 - $Pop_{power} = 1$ in models based on so-called gravitation attraction.
 - $Pop_{power} = 0$ if site traffic is independent of the population.
- *Distance factor* $\left(\frac{dist_{ij}}{dist_{max}} + dist_{off}\right)^{dist_{power}}$. The distance factor makes the traffic between two nodes inversely proportional to the distance between them. The $dist_{off}$, $dist_{max}$ and $dist_{power}$ values have a similar function as in the population factor. Typical values are $dist_{power} = 2$ or 3 .
- Note that the population and/or distance normalization factors can be disabled. In such cases, the factors $dist_{max}$ and pop_{max} are set to one.

Pressing the **Apply** button, one traffic matrix is created, and shown in the right panel. In its turn, the **Apply batch** button permits creating an arbitrary number of matrices, which are also shown in the right panel.

3.4.3 Manual matrix introduction/edition

The right panel shows the traffic matrices generated by any of the previous models, in different tabs numbered as TM0, TM1... The user can manually modify the matrices directly typing the coordinate values. The *Resize this* button permits resizing (changing the number of nodes) of the traffic matrix in the active tab. The *Resize all* button permits resizing all of them in one step. The *Load* button permits loading a traffic matrix from a .n2p file. The buttons *Save this* and *Save all* permits saving the traffic matrix in the active tab, or all the traffic matrices in .n2p files. The buttons *Make symmetric this* and *Make symmetric all* produce symmetric matrices. The traffic between two nodes becomes the average between the traffic in both directions. The buttons *Reset this* and *Reset all* sets 0s in all the coordinates of active/all matrices. The button *Clear all* eliminates all the matrices. The button

Sum all adds a new traffic matrix to the panel, which is the sum of all the traffic matrices shown (all of them must have the same number of nodes). The buttons *Multiply this* and *Multiply all* permits multiplying all the coordinates of this/all matrices by a constant factor.

3.4.4 Traffic normalization

The traffic normalization pull-down menu in the lower-right side of the window, permits selecting a normalization method to apply to the traffic matrix in the active tab (button *Apply*) or to all the traffic matrices (button *Apply all*). Four types of normalization methods are implemented: total, row and column normalization [2]:

- *Total normalization.* The target of total normalization, is modifying a traffic matrix M , so that in the normalized matrix M' , the total amount of traffic generated equals a user-defined value S . This is done by multiplying all the elements of the original traffic matrix by a constant factor according to the expression:

$$M'_{ij} = M_{ij} \frac{S}{\sum_{ij} M_{ij}}$$

- *Row normalization.* The target of row normalization, is modifying a traffic matrix M , so that in the normalized matrix M' , the total amount of traffic generated by each node i , equals a user-defined value S_i . This is done by multiplying all the elements in the i -th row of the original traffic matrix by the same constant factor according to the expression:

$$M'_{ij} = M_{ij} \frac{S_i}{\sum_j M_{ij}}$$

- *Column normalization.* The target of column normalization, is modifying a traffic matrix M , so that in the normalized matrix M' , the total amount of traffic received by each node j , equals a user-defined value S_j . This is done by multiplying all the elements in the j -th column of the original traffic matrix by the same constant factor according to the expression:

$$M'_{ij} = M_{ij} \frac{S_j}{\sum_i M_{ij}}$$

- *Normalize to the maximum traffic that can be carried.* The target is to multiply the traffic matrix M by the maximum factor α so that the matrix αM can still be carried by a given network using the optimum routing (without oversubscribing the links). The user can choose between two forms of calculating this: an estimated method, and an exact method:

- *Estimated method.* This method actually produces a matrix which is an upper bound to the maximum traffic that an optimal routing could carry. For each input/output pair in the network (i, j) , it calculates the number of hops that the shortest path (either in hops or in km) between those nodes has SP_{ij} . For each node pair (i, j) , the quantity $\alpha M_{ij} SP_{ij}$ is the minimum possible amount of link bandwidth that the traffic between those nodes could consume in any routing. Then, the normalized matrix αM is such that the sum of this quantity along all the coordinates, equals the total amount of bandwidth summing the links in the network $\sum_e u_e$ (where u_e stands for the capacity of link e). In other words:

$$\alpha = \frac{\sum_{ij} M_{ij} SP_{ij}}{\sum_e u_e}$$

For further details, see Exercise 4.2 in [1].

- *Exact method.* This method solves the linear program that exactly computes the maximum factor α for which the matrix αM still has a feasible routing in the network. The formulation is solved using the JOM library. The solver used and its .DLL/.so location is obtained from the default values in the Options menu (see Section 3.1.1).

3.4.5 Creating a set of traffic matrices from a seminal one

In some occasions, network design studies require a set of traffic matrices, instead of a single one. For instance, when we need to produce a sequence of traffic matrices that reflect the forecasted traffic in following years, according to a expected traffic growth (this is called multi-period planning). Also, it may be necessary to produce random variations of a single traffic matrix, to check how the network performances vary if the traffic fluctuates. For this or other purposes, Net2Plan offers the following methods:

- *New matrices with a compound annual growth rate.* A sequence of traffic matrices is created, representing one for each of the incoming years, being the seminal matrix the traffic today. Each matrix is equal to the matrix of the previous year, multiplied by a factor $(1 + CAGR)$, where CAGR is the *Compound Annual Growth Rate*.
- *Uniform random variations.* A set of matrices is created from a seminal one. For each new matrix, each coordinate is given by a sample of a uniform random variable with average x (the coordinate value in the seminal matrix), and a user-defined maximum relative variation (in the range $[0, 1]$). For instance, a value 0.2 of the maximum relative variation means that the new coordinate is taken uniformly between $(1 - 0.2)x$ and $(1 + 0.2)x$.
- *Gaussian random variations.* A set of matrices is created from a seminal one. For each new matrix, each coordinate is given by a sample of a gaussian random variable, with average x (the coordinate value in the seminal matrix), and a user-defined coefficient of variation (quotient between standard deviation and average), and a user-defined maximum relative variation. This latter value is used to truncate the sample. For instance, a value 0.2 of the maximum relative variation means that if the sample is below $(1 - 0.2)x$ then the value $(1 - 0.2)x$ is produced, and if the sample is greater than $(1 + 0.2)x$, the value $(1 + 0.2)x$ is produced.

In any method, if negative values in the coordinates appear, they are set to 0.

3.5 Help menu

This menu has the following options:

- **About.** Gives access to the same welcome screen as the one shown in Fig. 3.1.
- **User's guide.** Shows the local copy of this document. It can be accessed also using F1.
- **Library API Javadoc.** Shows the local copy of the library API Javadoc.
- **Examples in website.** Shows the examples available in the website (requires an Internet connection).
- **Key combinations.** Shows the key combinations or shortcuts of the active tool (if any). Key combinations can also be accessed with ALT+K

Chapter 4

The Net2Plan Command-Line Interface (CLI)

Contrary to the GUI, the command-line interface allows users to make use of batch processing or large-scale simulation features, thus it is specifically devoted to in-depth research studies.

All features available for GUI mode are also available here. The execution is controlled via command-line arguments. To run Net2Plan in CLI mode, users must execute the following command in a terminal:

```
java -jar Net2Plan-cli.jar [more options]
```

Help information can be obtained through the help argument in the following ways:

- To obtain a brief information, including only the execution modes, users should execute the CLI without arguments: `java -jar Net2Plan-cli.jar`
- To obtain the complete information, including individual help for every execution mode, users should type: `java -jar Net2Plan-cli.jar -help`.
- To obtain the information about a certain execution mode, users should type: `java -jar Net2Plan-cli.jar -help mode-name`

Next, the command to execute every mode is shown:

- *Network design.* `java -jar Net2Plan-cli.jar -mode net-design [more options]`
- *Traffic matrix design.* `java -jar Net2Plan-cli.jar -mode traffic-design [more options]`
- *Online simulation.* `java -jar Net2Plan-cli.jar -mode online-sim [more options]`
- *Reporting.* `java -jar Net2Plan-cli.jar -mode report [more options]`

4.1 Examples

We show here some examples:

- To execute the `MyAlgorithm` algorithm in the Jar file `myFolder\myJar` for the NSFNet network and its reference traffic matrix, setting the parameter `myParam` to the value 3:

```
java -jar Net2Plan-cli.jar --mode net-design
--input-file workspace\data\networkTopologies\NSFNet_N14_E42.n2p
--traffic-file workspace\data\trafficMatrices\NSFNet.n2p
--output-file workspace\data\NSFNet.n2p
--class-file myFolder\myJar.jar
--class-name MyAlgorithm
--alg-param myParam=3
```

- To execute the built-in availability report over the previous design:

```
java -jar Net2Plan-cli.jar --mode report
--input-file workspace\data\NSFNet.n2p
--output-file workspace\data\report.html
--class-file workspace\BuiltInExamples.jar
--class-name Report_availability
```

- To generate a series of 4 10×10 traffic matrices using a (0, 100) random uniform model:

```
java -jar Net2Plan-cli.jar --mode traffic-design
--num-nodes 10
--traffic-pattern uniform-random-100
--output-file workspace\data\trafficMatrices\tm10nodes.n2p
--num-matrices 4
```

- To execute a simulation using `myEventGenerator` and `myEventProcessor` classes in `myFolder\myJar` file:

```
java -jar Net2Plan-cli.jar --mode online-sim
--input-file workspace\data\networkTopologies\NSFNet_N14_E42_complete.n2p
--output-file workspace\data\simReport.html
--generator-class-file myFolder.myJar.jar
--processor-class-file myFolder.myJar.jar
--generator-class-name myEventGenerator
--processor-class-name myEventProcessor
--sim-param simEvents=100000
--sim-param transitoryEvents=10000
```

Important: To avoid excessive verbosity in the CLI, package names for Java classes can be omitted. Note that in case of `.jar` files the first class matching the class name will be selected, if multiple cases in different packages share the same class name.

Important: Restrictions to the path of `.class` files must follow the guidelines in Chapter 5.

Chapter 5

Development of algorithms and reports in Net2Plan

One of the most important features of Net2Plan is that it allows users to execute their own code (algorithms, reports... in general we refer to them as runnable code). Here, we briefly describe how to integrate users' code into Net2Plan.

Runnable code is implemented as Java classes, using single `.class` files or integrated into `.jar` files, with a given signature:

- Algorithms for offline network design should implement the interface
`com.net2plan.interfaces.networkDesign.IAlgorithm`
- Reports should implement the interface `com.net2plan.interfaces.networkDesign.IReport`
- Online algorithms that process network events, used in the online simulation tool and some reports, should implement the interface `com.net2plan.interfaces.simulation.IEventProcessor`
- Modules that generate events to be consumed by online algorithms should implement the interface `com.net2plan.interfaces.simulation.IEventGenerator`.

A complete information of each interface can be found in the Library API Javadoc. Integration of runnable code simply requires saving it into any directory of the computer.

In addition, in order to improve the user experience, the kernel is able to catch any exception thrown by runnable code, and print exception messages in the Java console. Recall that any information printed to the Java console by any runnable code (e.g., exception messages, and also any messages printed on purpose into `System.out`) , can be seen in the Net2Plan Java console (see Section 3.1.3). This is a valuable resource for debugging the algorithms ran in Net2Plan. In particular, messages from exceptions include a full trace of the error (files, line number of the exception...).

When the runnable code wants to stop its execution raising an exception that needs to be informed to the user in a more clear form (not through the Java console), we recommend to throw the `Net2PlanException` class (see Library API Javadoc for more information). The message associated to this exception is printed in a pop-up dialog instead of the Java console, and thus is more visible to the user. For instance, let us assume an algorithm that receives an input parameter from the user, that should be positive. A good programming practice is starting the algorithm checking if the received parameters are within their valid ranges. If a negative number is received (or something that is not a number), it is better to raise a `Net2PlanException` that shows the information message in a pop-up in

Net2Plan, than a general Java `RuntimeException` whose message can be read only if the user checks the Java console.

Important: When runnable code is implemented as a Java `.class` files, the full path to the class must follow the package name of the class, in order to successfully load the code. For example, if we create an algorithm named `TestAlgorithm` in the package `test.myAlgorithms`, the full path to the class must be like `...any.../test/myAlgorithms/TestAlgorithm.class`. For `.jar` files there is not any restriction on the full path, they can be stored at any folder.

Important: Net2Plan allows to make *online* changes in the runnable code, that is, users can modify their runnable code, recompile and reexecute it (just clicking the `Execute` button at the graphical interface) without the need to restart Net2Plan.

5.1 Net2Plan Library, Built-in Examples and Code Repository

Net2Plan assists the task of creating and evaluating algorithms by providing built-in example algorithms and a set of libraries (e.g., k -loopless shortest paths, candidate path list creation for unicast and multicast traffic...). An exhaustive list of built-in algorithms and the Library API Javadoc can be found in Net2Plan repository and in the Javadoc. Net2Plan web site is expected to become a valuable repository for network planning algorithms. The algorithms in the repository will be open for validation and verification, improving the trustworthiness of planning results.

5.2 JOM: Java Optimization Modeler

Often, some network design problems are solved by modeling them as optimization problems (i.e. integer linear problems, linear problems, convex problems, ...), and then calling an optimization solver to obtain its numerical solution. In this context, optimization modeling tools are targeted to ease the definition of the problem decision variables, constraints and objective function, and become an interface with the (usually complex) solver libraries. AMPL and GAMS are examples of commercial modeling tools. JOM (Java Optimization Modeler) is an open-source Java library developed by Prof. Pablo Pavón Mariño, which can interface with a number of solvers using a vectorial MATLAB-like syntax, which e.g. permits the addition of sets of constraints in one line of code. Current JOM version can interface with GLPK (free), MIPCL (free), XPRESS (commercial) and CPLEX (commercial) solvers for mixed integer linear problems, and IPOPT (free) for non-linear differentiable problems. JOM directly interfaces with compiled solver libraries (.DLLs in Windows and .SOs in Linux), via Java Native Access (JNA). JOM is independent from Net2Plan and can be used for any type of optimization problem. However, Net2Plan uses JOM in all the network design algorithm examples based on solving formulations that are included in the Net2Plan distribution.

5.3 Preparing a Java IDE for Net2Plan programming

For users interested in integrating their own algorithms to Net2Plan, it is required to prepare the Java IDE to program the runnable code. Essentially, users just have to configure their preferred Java IDE to use Java 8 (or later), and to include the libraries in the `lib` subfolder of the Net2Plan folder in the Java build-path. In Eclipse, the latter can be done in the option:

```
Project => Properties => Java Build Path => Libraries => Add External JARs...
```

In Netbeans, the option can be found in:

`Run => Set Project Configuration => Customize => Libraries => Add JAR/Folder...`

Additionally, Javadoc and sources can be attached using the corresponding options in the IDEs. Once the Java IDE is configured, users can start programming their own Net2Plan code.

Chapter 6

Technology-specific libraries

As stated in previous sections, the NetPlan object contains only the base minimum member elements corresponding to a *technology-agnostic* view of the network. For instance, for the links it permits setting the length in km, the propagation speed of the signal or its capacity, which are concepts applicable to any network technology. However, if the link e.g. corresponds to an IP link in a network with a routing controlled by the OSPF protocol, we would be interested in storing in the design the link weight to apply in shortest path computations. This is an example of technology-specific information. For this, Net2Plan permits the user to add/remove/edit a `Map<String, String>` of key-value attributes per link, node, demand, etc. For instance, this link weight information could be stored in an attribute with key `ospf-linkWeight`, and value the associated weight converted to `String` (actually this is the name chosen by the `IPUtils` library in Net2Plan).

Current Net2Plan version, provides libraries with specific routines used in the following technologies:

- *IP networks*: The `IPUtils` class incorporates some routines of interest in IP networks. Among them, the computation of the traffic according to OSPF/ECMP weights. Please see the Javadoc of `IPUtils` class for the conventions used in Net2Plan when introducing IP specific information in the network model. Also, some built-in algorithms are provided for IP networks, which can be consulted in the Net2Plan code repository under keywords *IP* and *OSPF*.
- *Optical WDM networks*: The `WDMUtils` class incorporates some routines of interest in optical WDM networks. Among them, the routines to handle the occupation of the wavelengths, and simple algorithms for routing and wavelength assignment. Please see the Javadoc of `WDMUtils` class for the conventions used in Net2Plan when introducing WDM specific information in the network model. Also, some built-in algorithms are provided for WDM networks, which can be consulted in the Net2Plan code repository under keyword *WDM*.
- *Wireless networks*: The `WirelessUtils` class incorporates some routines of interest in wireless networks. Among them, the routines to compute coverage and interference matrices, according to some models of Chapter 5, and algorithms in Part II of [1]. Please see the Javadoc of `WirelessUtils` class for the conventions used in Net2Plan when introducing wireless specific information in the network model. Also, some built-in algorithms are provided for wireless networks, which can be consulted in the Net2Plan code repository under keyword *wireless*.
- *NFV scenarios*: Net2Plan 0.5.0 introduces the element *Resource*, and generalizes *Demands* and *Routes* to become service chain requests and service chains respectively. This opens the door to develop algorithms for allocating resources in NFV scenarios. These algorithms can benefit from several suitable built-in algorithms. For instance, the traditional k -minimum cost path algorithm between two nodes, has been extended to return the k -minimum cost service chain between two nodes, traversing a given sequence of resource types. The interested reader can read

the information in <http://net2plan.com/techConventions.php>, and consult the algorithms in the repository under the keyword NFV.

- *NFV over IP over WDM.* Since version 0.6.0 we include a specific library for handling such networks. This permits instantiating VNFs, create service chains over an IP network, where IP links can potentially be carried by WDM lightpaths in a WDM flexi-grid network. The library is developed mainly within H2020 MetroHaul european project (H2020-ICT-2016-2 (ID 761727)), and ONOFRE-2 Spanish project (TEC2017-84423-C3-2-P).

Remember: All key-value pairs in the attribute maps are stored as **String** values. Users are responsible to make the proper conversions. For example, you can store an **int** array as a succession of numbers separated by spaces.

References

- [1] Pablo Pavón Mariño, *Optimization of computer networks. Modeling and algorithms. A hands-on approach.* Wiley, May 2016. Accompanying website: <http://www.net2plan.com/ocn-book>.
- [2] Robert S. Cahn, *Wide area network design: concepts and tools for optimization*, Morgan Kaufmann Publishers Inc., 1998.