



GÉANT Testbed Service (GTS) User and Resource Guide



Version 3.1

June 2016

Document Code: <GN4-P1-YY-nnn-nnn>

Authors:

© GEANT Limited on behalf of the GN4-1 project.

The research leading to these results has received funding from the European Union's Horizon 2020 research and innovation programme under Grant Agreement No. 691567 (GN4-1).

Contents

| | | |
|-------|--|----|
| 1 | Introduction | 4 |
| 2 | Testbed topology and resources | 5 |
| 3 | How to get started | 6 |
| 3.1 | Access to GTS | 6 |
| 3.2 | Login to the GUI | 6 |
| 4 | Working in the testbed environment | 10 |
| 4.1 | Defining a testbed with DSL | 11 |
| 4.2 | Reserving testbed resources | 14 |
| 4.3 | Activating testbed resources | 16 |
| 4.4 | Setting reservation windows | 18 |
| 4.5 | Deactivating testbed resources | 19 |
| 4.6 | Releasing testbed resources | 20 |
| 4.7 | Querying details | 21 |
| 4.8 | Internet Access Gateway (IAGW) | 23 |
| 4.8.1 | Using IAGW with Windows | 23 |
| 4.8.2 | Using IAGW with Debian / Ubuntu | 25 |
| 4.9 | Persistent Shared Project Folder | 30 |
| 4.10 | Example: Working with GTS OpenFlow switches (OFXs) | 31 |
| 5 | Help and Support | 39 |
| 6 | Introduction to Domain Specific Language | 40 |
| 6.1 | DSL in GTS | 40 |
| 6.2 | Quick Start | 40 |
| | Appendix I: Resource Guide | 45 |
| I. | Composite types | 46 |
| II. | Host | 47 |
| III. | Link | 49 |
| IV. | OFX | 50 |
| V. | BNF Grammar | 55 |
| VI. | Examples: One host | 56 |
| VII. | Example: Two hosts linked together | 57 |
| VIII. | Example: Triangle between three locations | 59 |

| | | |
|-----|--|----|
| IX. | Example: Three triangles built using DSL code iterations | 61 |
| X. | Example: Two OpenFlow switches, one controller and three hosts | 63 |

1 Introduction

Welcome to the new GÉANT Testbeds Service (GTS)! This new production service gives you the opportunity to setup your own customized experimental platform in order to be able to test novel concepts in networking and telecommunications.

This new service is made available in several releases and will grow further over time. This user guide describes features available in GTS Version 3.1.

The service allows you to set up an isolated customized packet-based testbed environment that is completely isolated from other testbed domains and enables you to simulate your particular experiment over resources in a real network without having to worry about impacting other testbeds or production services. After a short registration process a first time user has access to a pool of virtualized resources that can be programmed and reserved using a Domain Specific Language (DSL) description or selected via a graphical user interface. This reservation of resources also allows a user to specify start and end times during when these resources should be activated and made available for experimentation. New features available in this version include an Internet Access Gateway (IAGW) for experimenters so that they have the possibility to load their own software and also an external interface that can be selected as a resource whenever a testbed needs to be linked to external nodes.

The GTS facility is intended for use by researchers who are part of the GÉANT Community and its associated OpenCall projects [FAR-2014, SZE-2014]. Other researchers will be provided access on a space available basis.

2 Testbed topology and resources

Currently, GTS allows you to select resources from the following geographical locations: Amsterdam, Bratislava, Hamburg, Ljubljana, London, Madrid, Milan, Paris and Prague (see Figure 1). All these locations are sitting on the GÉANT backbone network point of presences. Later on, it is expected that new resource locations will be available outside of GÉANT, in the NRENs or other peer networks' domains.

Available resources in Version 3.1 of GTS include:

- Hosts (VMs with data ports; implemented using OpenStack)
- Virtual circuits (Ethernet pipe with data ports; implemented using Network Service Interface (NSI))
- OFXs (OpenFlow switch instances (OpenFlow Specification 1.3) with data ports)
- External domain.

The detailed description of the resources, their parameters and default attributes can be found in the Appendix I.



Figure 1: GTS physical infrastructure topology.

3 How to get started

This section describes the steps you need to complete until your testbed setup is ready for use.

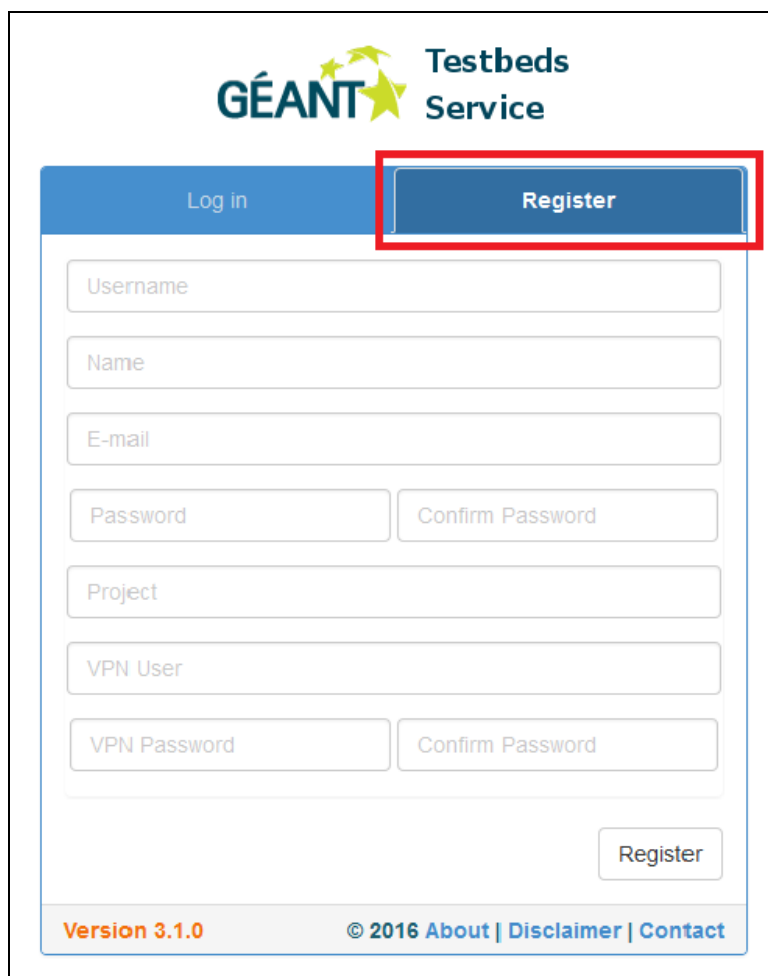
3.1 Access to GTS

The GTS front-end web Graphical User Interface (GUI) is available at: <http://gts.geant.net>. NOTE: Whenever you get the message “The service is currently undergoing maintenance, may not be fully functional. Check back later!” it practically means that we are upgrading and verifying the software/hardware functions of the production environment. The system may not be fully functional while the notice is displayed.

The GUI is the primary user interface of GTS but there are other means of accessing the testbed services. The GUI is built on a restful API in front of the GTS core software. In later versions of the software, the API will also be open for application designers and system integrators.

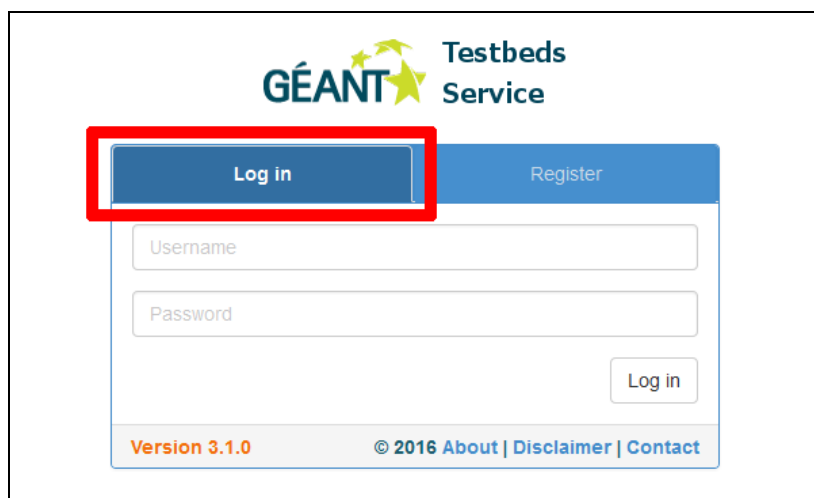
3.2 Login to the GUI

First time users have to register and define VPN credentials (Figure 2); users that are already registered can login with their user names and passwords as shown in Figure 3. During the initial registration process you have to define your username and provide your real name, e-mail address, password, password confirmation and the name of your project. You also have to define a VPN username and password that you will later use to VPN into your project.



The screenshot shows the GÉANT Testbeds Service registration page. At the top, the GÉANT logo and 'Testbeds Service' text are displayed. Below this is a navigation bar with two buttons: 'Log in' and 'Register'. The 'Register' button is highlighted with a red rectangular box. The main form area contains several input fields: 'Username', 'Name', 'E-mail', 'Password', 'Confirm Password', 'Project', 'VPN User', 'VPN Password', and 'Confirm Password'. A 'Register' button is located at the bottom right of the form. At the very bottom, a footer bar contains the text 'Version 3.1.0' and links for '© 2016 About | Disclaimer | Contact'.

Figure 2: First-time registration



The screenshot shows the GÉANT Testbeds Service login page. At the top, the GÉANT logo and 'Testbeds Service' text are displayed. Below this is a navigation bar with two buttons: 'Log in' and 'Register'. The 'Log in' button is highlighted with a red rectangular box. The main form area contains two input fields: 'Username' and 'Password'. A 'Log in' button is located at the bottom right of the form. At the very bottom, a footer bar contains the text 'Version 3.1.0' and links for '© 2016 About | Disclaimer | Contact'.

Figure 3: Login as registered user

Once you are logged in you have access to the VPN information by clicking on the blue information icon next to your project's name (Figure 4). Logout of your project using the dropdown menu below your project name (Figure 5). This dropdown menu also gives you access to your user profile and allows updates if needed (Figure 6).

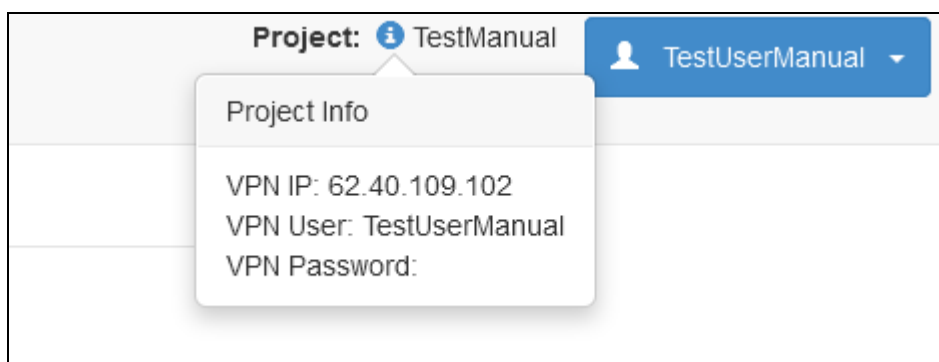


Figure 4: Project and VPN related information

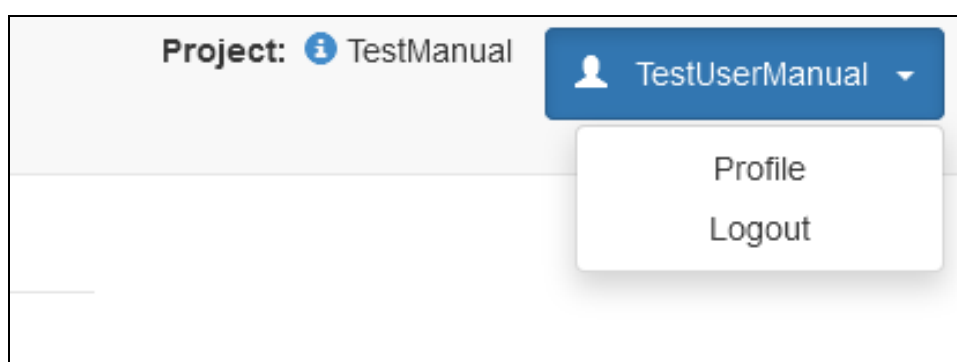


Figure 5: Log out of the project via the dropdown menu next to the project's name

In the current implementation of Version 3.1, new user registrations always end up in new projects. If you wish to add multiple users to your single (already existing) project, please contact the GTS service managers.

Edit User Details

Username

TestUserManual

Name

Test User

e-mail

Password *(necessary only for password change)*

Confirm Password

Role

user

Project

TestManual

Reset

Figure 6: Update or reset user profile

During a login process you may find a disclaimer that indicates that the testbed facility is currently under maintenance and its use may be limited (Figure 7).

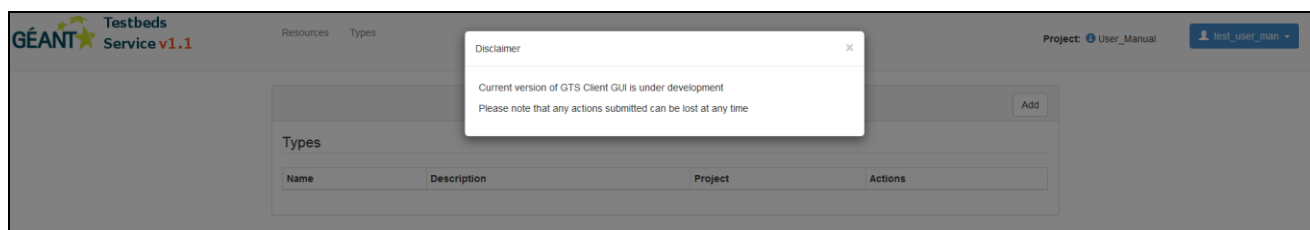


Figure 7: Disclaimer

4 Working in the testbed environment

The GTS GUI acts as an agent to the testbed resources and allows a user to build and monitor an isolated domain of interacting resources, i.e. a user's testbed is independent from other users' testbeds and can be operated without any impact on other testbed environments.

After logging in to GTS and naming and creating a new project, the user builds a testbed description for the Testbed Control Agent (TCA). In a next step, the testbed description document is fed to the Resource Manager (RM) which in turn parses the document, allocates resources, and sets up the testbed control plane. The testbed is then activated and the user controls it via the GUI.

For example, the user may decide to define router nodes at first and may then browse and select a set of PC resources to act as the end systems. Each of these resources has data plane ports defined as integral components of the resource. The user must then define the connectivity among these nodal resources by indicating how the various nodal resources are to be interconnected via these network links. At the end the user saves the testbed description to the project's repository and submits the testbed description to the SA2 Service Engine for processing. For some experiments it may be necessary to select virtual resources that are geographically apart or at certain locations. In such use cases users may select a resource's location from the list of available installations.

The SA2 service engine analyzes the testbed description for any obvious errors or omissions, and finding nothing critical, the Service proceeds to locate and reserve the indicated resources. Note: this is an iterative process – the researcher may have only defined a partial set of resources and wishes to reserve these first, where upon the user will then define additional resources to be incorporated into the testbed. Upon successfully reserving all resources, the researcher requests the SA2 Service Engine to instantiate the testbed. As resources are initialized and brought into service, control of those resources are passed to the researcher's own testbed control agent. Instantiating resources is complete, when the selected resources have been activated.

The user can program the desired testbed environments and reserve resources using a Domain Specific Language (DSL) description or select resources via the GUI (for more information on DSL please see Chapter 6). This reservation of resources also allows a user to specify start and end times during when these resources should be activated and made available for experimentation. Only when all requested resources are available the reservation process completes with success.

Reserved resources can then be **activated** by the user and at this time all resources will actually be allocated. At some time during an experiment a user may wish to **deactivate** the resources, but keep the reservation of resources valid for use at a later point in time. Once an experiment is finished and the project is no longer needed, the user should also **release** all resources so that resources can be made available to other projects. Resources can also be **queried** for status information.

4.1 Defining a testbed with DSL

After you have logged in as a registered user you are now able to upload Domain Specific Language (DSL) descriptor files that set up the type of experimental environment that you would like to have in your testbed (for more information on the DSL please see Chapter 6).

You can define your own resource types or classes with DSL code (for example: an OpenFlow switch connected to a controller). To add a DSL file, please click on “Types” (top navigation menu, Figure 8) and then select “Add” (Figure 9) which will open up a window (see Figure 10) where you can browse through your files to upload an already existing DSL file. Click on ‘Submit’ to finish your testbed description.

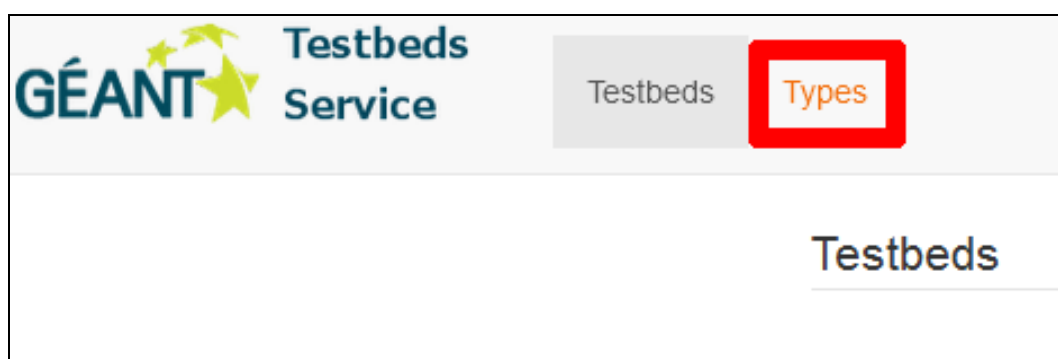


Figure 8: Select “Types”

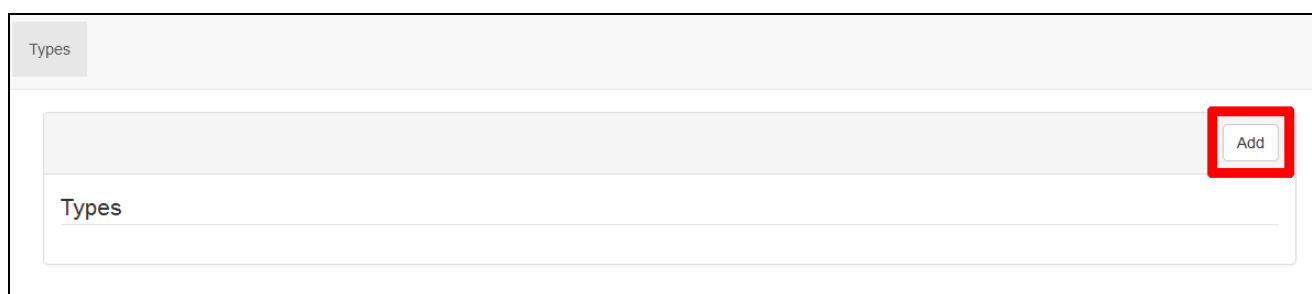


Figure 9: Add “Type”

Define New Type

Project
TestManual

DSL

DSL script file
Browse... No file selected.

Submit

Figure 10: Submitting DSL code via an already existing file

Another option is to configure your testbed by filling in your DSL description using the editor window of the GUI (Figure 11): Just type in your code and click on 'Submit'.

Define New Type

Project

TestManual

DSL

Triangle {
description = "Triangle testbed with three hosts."
id="TriangleTestbed"

host {
id="h1"
port { id="eth1" }
port { id="eth2" }
}

host {
id="h2"
port { id="eth1" }
port { id="eth2" }
}

host {
id="h3"
port { id="eth1" }
port { id="eth2" }
}
}

DSL script file

Browse... No file selected.

Submit

Figure 11: Submitting DSL code directly

4.2 Reserving testbed resources

After submitting your DSL code the system takes you back to the 'Types' screen and you will see the new testbed type you have added to your project (Figure 12)


| Types | | |
|----------|------------------------------------|---|
| Name | Description | Actions |
| Triangle | Triangle testbed with three hosts. |   |

Figure 12: Added testbed type

When you click on the type's name you have read-only access to your DSL code. For a created testbed type you can choose the following actions:

- You can RESERVE the testbed by clicking on the green button.
- The red button allows you to 'Undefine' or delete the type and the DSL file is dropped.

After you selected the green button to reserve your testbed you will see the following information on your screen that lists your new resource type (Figure 13):



| Testbeds | | | | | |
|-------------|-----------------|----------|----------------------|---|--|
| Provider ID | ID | Status | Type | Actions | |
| + 321 | TriangleTestbed | RESERVED | Testbed (Triangle) 6 |   | |

Figure 13: Reserved new resource type

You can expand the listed information to show you more details on the individual resource components that were reserved as part of your new type by clicking on the '+' sign next to the Provider ID on the left (Figure 14):



| Testbeds | | | | | |
|---------------------------|-----------------|----------|----------------------|---|--|
| Provider ID | ID | Status | Type | Actions | |
| 321 | TriangleTestbed | RESERVED | Testbed (Triangle) 6 |   | |
| Host-1464089637720 | h3 | RESERVED | Host | | |
| Host-1464089638483 | h2 | RESERVED | Host | | |
| Host-1464089639090 | h1 | RESERVED | Host | | |
| OpenNsaLink-GT-a18136acac | l3 | RESERVED | Link | | |
| OpenNsaLink-GT-162254d444 | l2 | RESERVED | Link | | |
| OpenNsaLink-GT-67c6322cd4 | l1 | RESERVED | Link | | |

Figure 14: Listing of reserved resources

The reservation process is complete when your resources show the status 'RESERVED'.

Note: In case you specify a certain location for a resource and your request cannot be granted because not enough VMs are available at that time at that location then your request may go through if you resubmit your DSL without the specification of a location (if that is not a requirement for your experiment in some way).

4.3 Activating testbed resources

Before *reserved* resources can be used, they must be *activated*. Click on the green button on the right to start the activation process (Figure 15).


| Testbeds | | | | | |
|-------------|-----------------|----------|----------------------|---|--|
| Provider ID | ID | Status | Type | Actions | |
| + 321 | TriangleTestbed | RESERVED | Testbed (Triangle) 6 |  | |

Figure 15: Activate a resource

The activation of resources prompts the user to indicate a start time and end time. The activation can happen in two ways:

1. If you specified your reservation time window (start time and end time) in your DSL, the resources will automatically go from reserved state to active when the time window starts (Figure 16). During that defined time window (i.e. when your reservation is valid) you can actually deactivate and activate your resources as you wish.
2. If you did not specify a reservation window in your DSL, your reservation is valid starting from the current time until practically forever. This will likely change in later versions of GTS! In this case you can activate and deactivate your resources whenever you want.


| Select Reservation to activate | | | | | |
|---|-----------------|----------|----------------------|---------|--|
|  Reservation Id: 321 Start time: 24-May-2016 11:33:00 GMT End time: 31-Jan-2526 22:59:59 GMT | | | | | |
| Provider ID | ID | Status | Type | Actions | |
| + 321 | TriangleTestbed | RESERVED | Testbed (Triangle) 6 | | |

Figure 16: Reservation start and end time

You can follow the status of your resources while it is changing from 'RESERVED' to 'ACTIVE' (Figure 17):


| Provider ID | ID | Status | Type | Actions |
|---------------------------|-----------------|------------|----------------------|---|
| 321 | TriangleTestbed | ACTIVATING | Testbed (Triangle) 6 |  |
| Host-1464089637720 | h3 | ACTIVE | Host | |
| Host-1464089638483 | h2 | ACTIVE | Host | |
| Host-1464089639090 | h1 | ACTIVATING | Host | |
| OpenNsaLink-GT-a18136acac | l3 | ACTIVE | Link | |
| OpenNsaLink-GT-162254d444 | l2 | ACTIVE | Link | |
| OpenNsaLink-GT-67c6322cd4 | l1 | ACTIVE | Link | |

Figure 17: Activating resources



| Provider ID | ID | Status | Type | Actions |
|---------------------------|-----------------|--------|----------------------|---|
| 321 | TriangleTestbed | ACTIVE | Testbed (Triangle) 6 |   |
| Host-1464089637720 | h3 | ACTIVE | Host | |
| Host-1464089638483 | h2 | ACTIVE | Host | |
| Host-1464089639090 | h1 | ACTIVE | Host | |
| OpenNsaLink-GT-a18136acac | l3 | ACTIVE | Link | |
| OpenNsaLink-GT-162254d444 | l2 | ACTIVE | Link | |
| OpenNsaLink-GT-67c6322cd4 | l1 | ACTIVE | Link | |

Figure 18: All resources are activated

For activated resources the status turns to 'ACTIVE' (Figure 18); please allow a few minutes for processing as all resources are created dynamically and not just picked from an already existing pool.

4.4 Setting reservation windows

This version does not allow the setting of specific start- and end times yet for activation and deactivation processes. Simply select the provided default settings by clicking on them (Figure 19) (**Please note:** Closing the window is not enough to choose the default values!):



Figure 19: Selecting default start- and end-times

4.5 Deactivating testbed resources

At some time during an experiment a user may wish to deactivate the resources, but keep the reservation of resources valid for use at a later point in time. In other words: Deactivated resources are not released and returned to other users until the experimenter actually *releases* the resources. Deactivated resources can be returned to an active state by simply activating them again at any time during the experiment.

To deactivate a resource, simply click on the yellow button on the right (Figure 20):



| Testbeds | | | | | |
|---------------------------|-----------------|--------|----------------------|---|--|
| Provider ID | ID | Status | Type | Actions | |
| 321 | TriangleTestbed | ACTIVE | Testbed (Triangle) 6 |   | |
| Host-1464089637720 | h3 | ACTIVE | Host | | |
| Host-1464089638483 | h2 | ACTIVE | Host | | |
| Host-1464089639090 | h1 | ACTIVE | Host | | |
| OpenNsaLink-GT-a18136acac | l3 | ACTIVE | Link | | |
| OpenNsaLink-GT-162254d444 | l2 | ACTIVE | Link | | |
| OpenNsaLink-GT-67c6322cd4 | l1 | ACTIVE | Link | | |

Figure 20: Deactivation of a resource

The start- and end-time reservation window will pop up again where you can specify the time when your resources are to be deactivated (in this version only available with default setting); select the (default) setting by clicking on the line with the start and end times (closing the window is not enough!) and wait for the status of the resources to change to 'RESERVED' again (Figure 21). But that does not mean that they are released (please see section 4.6 for more details); instead they remain reserved for the user to be available for the next activation process.

| Select Reservation to deactivate | | | | | |
|--|-----------------|--------|----------------------|---------|--|
| <div> Reservation Id: 321 Start time: 24-May-2016 11:33:00 GMT End time: 31-Jan-2526 22:59:59 GMT </div> | | | | | |
| Provider ID | ID | Status | Type | Actions | |
| 321 | TriangleTestbed | ACTIVE | Testbed (Triangle) 6 | | |

Figure 21: Selecting an end-time

4.6 Releasing testbed resources

Resources that are no longer of interest should be *released* so that these resources can be made available to other projects and users.



| Testbeds | | | | | |
|-------------|-----------------|----------|----------------------|---|--|
| Provider ID | ID | Status | Type | Actions | |
| + 342 | TriangleTestbed | RESERVED | Testbed (Triangle) 6 |   | |

Figure 22: Releasing resources

To release the testbed resource click on the red button on the right as marked in Figure 22.

4.7 Querying details

The individual resources that were reserved for your testbed become visible when you click on the '+' sign to the left of the Provider ID of your resource (Figure 23); each resource can then also be accessed by clicking on its link (Figure 24):




| Testbeds | | | | | |
|--|-----------------|----------|-----------------------------------|---|--|
| Provider ID | ID | Status | Type | Actions | |
|  42 | TriangleTestbed | RESERVED | Testbed (Triangle) 6 |   | |

Figure 23: Obtaining details via Provider ID




| Testbeds | | | | | |
|---|-----------------|----------|-----------------------------------|---|--|
| Provider ID | ID | Status | Type | Actions | |
|  342 | TriangleTestbed | RESERVED | Testbed (Triangle) 6 |   | |
| Host-1465200373058 | h3 | RESERVED | Host | | |
| Host-1465200374206 | h2 | RESERVED | Host | | |
| Host-1465200375170 | h1 | RESERVED | Host | | |
| OpenNsaLink-GT-abdd0d2a18 | l3 | RESERVED | Link | | |
| OpenNsaLink-GT-4eccec2ca07 | l2 | RESERVED | Link | | |
| OpenNsaLink-GT-276c394f16 | l1 | RESERVED | Link | | |

Figure 24: List of individual testbed resources

By clicking on a resource you can also obtain more information on ports, locations as well as reservation start and end times (Figure 25):

| Resource: Host | | | | | | |
|--------------------|-------------------------------|----------|-------------------------------|-----------|----------|---------|
| Provider id | Id | Status | Description | Ports | Location | Console |
| Host-1465200373058 | h3 | RESERVED | | eth1 eth2 | ams | |
| Reservations | | | | | | |
| Id | Start Time | | End Time | | | |
| 891 | Mon, 06 Jun 2016 08:06:00 GMT | | Thu, 31 Jan 2526 23:59:59 GMT | | | |

Figure 25: List of individual testbed resources

To obtain a tree structure of your testbed resource and to see port adjacencies or adjacencies for external ports, click on the Provider ID itself (Figure 26 and Figure 27):



| Testbeds | | | | | | |
|-------------|-----------------|----------|----------------------|---|--|--|
| Provider ID | ID | Status | Type | Actions | | |
| 342 | TriangleTestbed | RESERVED | Testbed (Triangle) 6 |   | | |

Figure 26: Access to tree structure


| Tree Structure | |
|--|--|
|  | |
| Port Adjacencies for Nested Resources | |
| Nested Resources | Adjacent Ports |
| Host-1465200373058 | Host-1465200373058.eth1 ⇌ OpenNsaLink-GT-abdd0d2a18.dst Host-1465200373058.eth2 ⇌ OpenNsaLink-GT-4ecec2ca07.dst |
| Host-1465200374206 | Host-1465200374206.eth1 ⇌ OpenNsaLink-GT-4ecec2ca07.src Host-1465200374206.eth2 ⇌ OpenNsaLink-GT-276c394f16.dst |
| Host-1465200375170 | Host-1465200375170.eth1 ⇌ OpenNsaLink-GT-276c394f16.src Host-1465200375170.eth2 ⇌ OpenNsaLink-GT-abdd0d2a18.src |
| OpenNsaLink-GT-abdd0d2a18 | OpenNsaLink-GT-abdd0d2a18.dst ⇌ Host-1465200373058.eth1 OpenNsaLink-GT-abdd0d2a18.src ⇌ Host-1465200375170.eth2 |
| OpenNsaLink-GT-4ecec2ca07 | OpenNsaLink-GT-4ecec2ca07.dst ⇌ Host-1465200373058.eth2 OpenNsaLink-GT-4ecec2ca07.src ⇌ Host-1465200374206.eth1 |
| OpenNsaLink-GT-276c394f16 | OpenNsaLink-GT-276c394f16.dst ⇌ Host-1465200374206.eth2 OpenNsaLink-GT-276c394f16.src ⇌ Host-1465200375170.eth1 |
| Adjacencies for External Ports | |
| No external ports for this testbed | |

Figure 27: Tree structure and port adjacencies

4.8 Internet Access Gateway (IAGW)

In order to be able to upload software to your testbed environment there is an Internet Access Gateway (IAGW) you can reach using the VPN information that you defined during your initial user registration process. The VPN is managed by a VM set up at the GTS Central Server Facility and remains in place as long as at least one of your hosts is active.

4.8.1 Using IAGW with Windows

For systems using versions of Windows set up a new VPN network connection via the Control Panel and its Network and Internet options.

To set up the VPN use the IP address (Figure 29) shown when you click on the blue 'i' icon next to your project name (and Figure 28: VPN informationFigure 28) and also the VPN user name and password you defined during the initial project registration process.

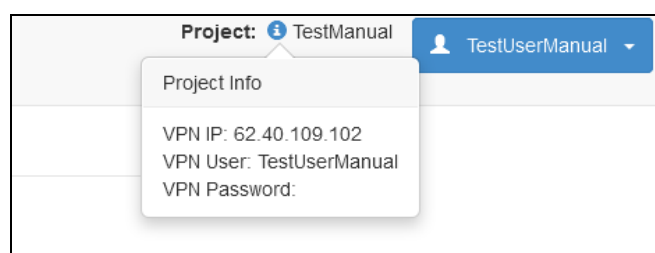


Figure 28: VPN information

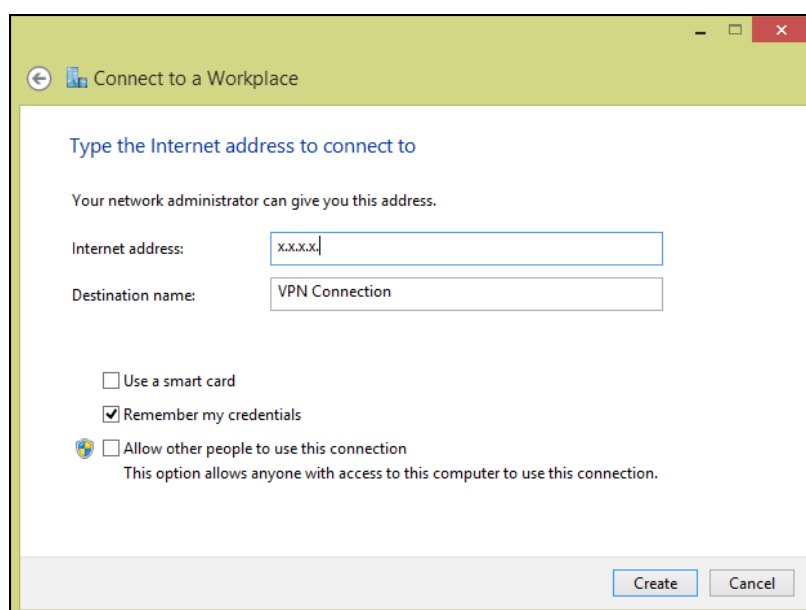


Figure 29: VPN IP address

Once you are connected via VPN, use a file transfer program like WinSCP or PuTTY to transfer software to a testbed host. Before you use WinSCP or PuTTY access the console of the host that you would like to transfer to by clicking on the appropriate host (Figure 30) which will then provide a console link (Figure 31).

| Resources | | | | | | |
|------------------------------|---------|--------|-----------|---------|--|--|
| Provider ID | ID | Status | Type | Actions | | |
| 2491 | 8963620 | ACTIVE | Testbed 6 | | | |
| Host-1425297436046 | h3 | ACTIVE | Host | | | |
| Host-1425297436919 | h2 | ACTIVE | Host | | | |
| Host-1425297437784 | h1 | ACTIVE | Host | | | |
| OpenNsaLink-TA-T863cc624f840 | l3 | ACTIVE | Link | | | |
| OpenNsaLink-TA-T6ee5ff17d8b1 | l2 | ACTIVE | Link | | | |
| OpenNsaLink-TA-T115cf06baffe | l1 | ACTIVE | Link | | | |

Figure 30: Accessing the console of a host

| Resource: Host | | | | | | |
|--------------------|----|--------|-------------|-----------|----------|---|
| Provider id | Id | Status | Description | Ports | Location | Console |
| Host-1465200373058 | h3 | ACTIVE | | eth1 eth2 | ams | http://62.40.126.12:6080/vnc_auto.html?token=b71365e5-3c17-45a8-b148-a3076bd19c62 |

| Reservations | | | |
|--------------|-------------------------------|-------------------------------|--|
| Id | Start Time | End Time | |
| 891 | Mon, 06 Jun 2016 08:06:00 GMT | Thu, 31 Jan 2526 23:59:59 GMT | |

Figure 31: Console link

Click on the console link; you will then be prompted for user name and password (gts, gts). Look for the IP address provided on the screen for `eth0` (Figure 32); this is the IP address you should use for your file transfer program along with user name and password (gts, gts). If you don't see an IP address here, use the `ifconfig` command to find out the IP address for interface `eth0`. You are then ready to transfer your software to your host.


```

Ubuntu 14.04.1 LTS sa2host tty1
sa2host login: gts
Password:
Last login: Fri Mar  6 10:11:08 CET 2015 on tty1
Welcome to Ubuntu 14.04.1 LTS (GNU/Linux 3.13.0-37-generic x86_64)

* Documentation:  https://help.ubuntu.com/

System information as of Fri Mar  6 10:11:08 CET 2015

System load:  0.0               Processes:            68
Usage of /:   16.3% of 6.76GB    Users logged in:     0
Memory usage: 1%               IP address for eth0: 172.16.0.11
Swap usage:   0%

Graph this data and manage this system at:
https://landscape.canonical.com/

gts@sa2host:~$ _

```

Figure 32: IP address for eth0

4.8.2 Using IAGW with Debian / Ubuntu

For a VPN connection to GTS using Debian / Ubuntu proceed with the following steps:

- Get into root environment
\$ sudo su
- Switch into directory 'peers'
cd /etc/ppp/peers
- Setup up a VPN configuration file 'projectname' and adjust the values marked in red (Figure 33):

```
### Build VPN connection to GTS VPN server
pty "pptp <VPN IP> --nolaunchpppd --nobuffer --timeout 10"

### GTS VPN - user name
name <VPN user name>

### This command reconnects to GTS VPN server after a disconnect
persist

### MTU value should be standard value.
mtu 1400

### used for scripts in ip-up / ip-down
ipparam <YOURPROJECTNAME>

### Terminate after n consecutive failed connection attempts.
### A value of 0 means no limit. The default value is 10.
maxfail 0

### IMPORTANT: Do not use this option for the GTS VPN connection.
### Otherwise all packets are routed over this connection.
### Please comment it out
#defaultroute

### Some useful settings.
remotename <YOURPROJECTNAME>
lock
noauth
nobsdcomp
nodeflate

#Require encrypt
require-mppe-128
```

Figure 33: Adjust VPN configuration file

To find out the correct IP and user name that you should set in this VPN configuration file click on the blue 'i' icon next to your project name (Figure 34) and use the values you set during your initial project registration and then save the file.

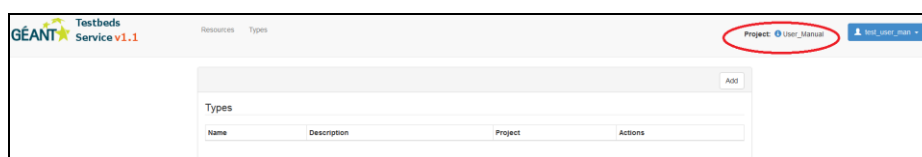


Figure 34: VPN information

- In the next step adjust the file 'chap-secrets' indicating your VPN user name, your project name and VPN password that you set during your initial project registration (Figure 35):

```
# Secrets for authentication using CHAP
# client      server      secret      IP addresses
# IMPORTANT: <VPN PASSWORD> MUST be set in double quotes
<VPN USER> <YOURPROJECTNAME> "<VPN PASSWORD>" *
#end
```

Figure 35: Adjust file 'chap-secrets'

- Then switch to the directory `ip-up.d`
`cd ip-up.d/`
- and set the project route with an executable file '`yourprojectname.route`' (Figure 36)

```
#!/bin/sh
# IMPORTANT: Use here the network IP of interface eth0 of your
# GTS VM (NOT the VPN IP!)
if [ "${PPP_IPPARAM}" = "<YOURPROJECTNAME>" ]; then
    /sbin/route add -net <YOUR NETWORK IP>/24 dev <PPP INTERFACE NAME>
fi
```

Figure 36: Set project route with executable file

and make sure you use the network IP address of `eth0` of your host. To find this IP address click on the appropriate host (Figure 37) which will then provide a console link (Figure 38).

| Resources | | | | | |
|------------------------------|---------|--------|---------|---------|--|
| Provider ID | ID | Status | Type | Actions | |
| 2491 | 8963620 | ACTIVE | Testbed | | |
| Host-1425297436046 | h3 | ACTIVE | Host | | |
| Host-1425297436919 | h2 | ACTIVE | Host | | |
| Host-1425297437784 | h1 | ACTIVE | Host | | |
| OpenNsaLink-TA-T863cc624f840 | l3 | ACTIVE | Link | | |
| OpenNsaLink-TA-T6ee5ff17d8b1 | l2 | ACTIVE | Link | | |
| OpenNsaLink-TA-T115cf06baffe | l1 | ACTIVE | Link | | |

Figure 37: Accessing the console of a host

| Resource: Host | | | | | | |
|--------------------|----|--------|-------------|---------------|----------|---|
| Provider id | Id | Status | Description | Ports | Location | Console |
| Host-1425297436919 | h2 | ACTIVE | | port22 port21 | AMS | http://62.40.126.10:6080/vnc_auto.html?token=85b2f30a-3612-429a-8688-5bb5be0d83fb |

| Reservations | | |
|--------------|-------------------------------|-------------------------------|
| Id | Start Time | End Time |
| 2554 | Mon, 02 Mar 2015 11:57:00 GMT | Thu, 31 Jan 2526 22:59:59 GMT |

Figure 38: Console link

Click on the console link; you will then be prompted for user name and password (gts, gts). Look for the IP address provided on the screen for `eth0` (Figure 32); this is the IP address you should set as **<YOUR NETWORK IP>** in the file `'yourprojectname.route'` as shown in Figure 36.

In the file `'yourprojectname.route'` you also define the **<PPP INTERFACE NAME>** (see Figure 36) as `pppx` where 'x' would be '0' if you have no other VPNs defined (or else would be incremented to the appropriate number, but please **note**: only count VPNs set up with `ppp`, not VPNs set up with OpenVPN).

- This file `'yourprojectname.route'` should then be made into an executable file
`chmod 700 yourprojectname.route`
- To connect with the VPN use
`pon projectname`
- To disconnect the VPN use
`poff projectname`
- **To test:** Type
`ifconfig`

and look for your `pppx` with address 172.16.0.1 (Figure 39):

```
ppp0      Link encap:Point-to-Point Protocol
          inet addr:172.16.0.1  P-t-P:172.16.0.253  Mask:255.255.255.255
          UP POINTOPOINT RUNNING NOARP MULTICAST  MTU:1396  Metric:1
          RX packets:14584 errors:0 dropped:0 overruns:0 frame:0
          TX packets:13996 errors:0 dropped:0 overruns:0 carrier:0
          collisions:0 txqueuelen:3
          RX bytes:2568663 (2.4 MiB)  TX bytes:966234 (943.5 KiB)
```

Figure 39: check for `pppx`

- and then check if you can ping your host, for example
`ping -c 5 172.16.0.10`
- if your ping does not work, check with
`ip route show`
if you have the following entry:
`172.16.0.0/24 dev ppp0 scope link`
- **To upload software:** ssh into your host with user name and password (gts,gts)
`ssh gts@172.16.0.10` and upload files with
`scp filename gts@172.16.0.10:`
- **Please note:** Currently GTS does not support automatic security updates on VMs. It is therefore recommended to switch into root environment on the host with password (gts)

```
$sudo su  
and update / upgrade the list of available packages with  
# aptitude update  
# aptitude upgrade
```

4.9 Persistent Shared Project Folder

GTS provides a shared folder inside a project, which can be accessed from all hosts that are part of a testbed. The shared folder `project-share` can be found in the home directory of the `gts` user, so usually the path to it is `/home/gts/project-share`. The file contents in that folder are persistent while the project exists, so releasing a testbed will not have any effect on the contents of that folder. A user can store up to 3GB of data in this `project-share`.

4.10 Example: Working with GTS OpenFlow switches (OFXs)

The following steps guide you through a DSL example¹ that sets up a simple OpenFlow environment with two OpenFlow switches (OFXs)², a controller and three Virtual Machines (VMs). Each of the two OpenFlow switches is at a different location (in Ljubljana (LJU) and in Bratislava (BRA)) and they are both connected:

Step 1: Define the VM that will be the controller

```
TwoOFXwithController {
  description = "Two OFX with one controller VM. The OFXs are at
different locations and connected."
  id = "TwoOFXwithController"

  host {
    id="controller"
    port { id="eth1" }
    port { id="eth2" }
  }
}
```

A host called “controller” is defined as controller; the controller has two ports, because it will later have to connect to each of the two OpenFlow switches in LJU and BRA.

Step 2: Define a host (VM) in Bratislava

```
host {
  id="dataBRAVM1"
  location="BRA"
  port { id="eth1" }
}
```

The host has one port for data transfer.

¹ The complete DSL code is available in the Appendix, Section VI

² In GTS OpenFlow hardware switches are separated into instances referred to as OFX switch fabrics.

Step 3: Define OpenFlow switch in BRA

The following DSL code adds an OpenFlow switch in BRA with IP address 10.10.10.1. The controller port for the switch is on port "pControl" (mode="CONTROL"). RYU [RYU-2014] by default launches the controller instance in port 6633, but in this example the port '9966' was specified:

```

ofx {
  id="ofxBRA"
  location="BRA"
  fabricIPv4="10.10.10.1"
  fabricSubnetMaskv4="255.255.255.0"
  controllerPort="9966"
  controllerIPv4="10.10.10.10"
  port { id="p1" }
  port { id="p2" }
  port { id="pLJU" }
  port {
    id="pControl"
    mode="CONTROL"
  }
}

```

Step 4: Connect components in BRA

As a next step, the host "dataBRAVM1" in Bratislava must be connected to the OpenFlow switch and the Switch must be connected to the controller. This is achieved by

- defining the required links ("l1BRA" for the data plane between host and OpenFlow switch and "l1CBRA" for the control plane between the controller and the OpenFlow Switch)
- **and** by defining the adjacencies, which actually link the objects:

```

link {
  id="l1BRA"
  port { id="src" }
  port { id="dst" }
}

link {
  id="l1CBRA"
  port { id="src" }
  port { id="dst" }
}

adjacency dataBRAVM1.eth1, l1BRA.src
adjacency ofxBRA.p1, l1BRA.dst

adjacency controller.eth1, l1CBRA.src
adjacency ofxBRA.pControl, l1CBRA.dst

```

Step 5: Add a second host in Bratislava

Now add a second host in BRA by adding the following DSL code:

```
host {
    id="dataBRAVM2"
    location="BRA"
    port { id="eth1" }
}

link {
    id="l2BRA"
    port { id="src" }
    port { id="dst" }
}

adjacency dataBRAVM2.eth1, l2BRA.src
adjacency ofxBRA.p2, l2BRA.dst
```

Step 6: Connect another OpenFlow switch in Ljubljana

In this example the second OpenFlow switch is located in Ljubljana (LJU) with IP address 10.10.11.1. The controller port for the switch is on port "pControl" (mode="CONTROL").

```
ofx {
    id="ofxLJU"
    location="LJU"
    fabricIPv4="10.10.11.1"
    fabricSubnetMaskv4="255.255.255.0"
    controllerPort="9966"
    controllerIPv4="10.10.11.10"
    port { id="p1" }
    port { id="pBRA" }
    port {
        id="pControl"
        mode="CONTROL"
    }
}
```

Step 7: Add a host to the OpenFlow switch in LJU, connect host to OpenFlow switch, connect OpenFlow switch to controller, and connect the two OpenFlow switches

```
host {
    id="dataLJUVM1"
    location="LJU"
    port { id="eth1" }
}

link {
    id="l1LJU"
    port { id="src" }
    port { id="dst" }
}

link {
    id="loFXBRAOFXLJU"
    port { id="src" }
    port { id="dst" }
}

link {
    id="l1CLJU"
    port { id="src" }
    port { id="dst" }
}

adjacency controller.eth2, l1CLJU.src
adjacency ofxLJU.pControl, l1CLJU.dst

adjacency dataLJUVM1.eth1, l1LJU.src
adjacency ofxLJU.p1, l1LJU.dst

adjacency ofxBRA.pLJU, loFXBRAOFXLJU.src
adjacency ofxLJU.pBRA, loFXBRAOFXLJU.dst
```

Don't forget to close the surrounding "TwoOFXwithController {" block with a "}".

Step 8: Configuring the controller interfaces

Once you have reserved your testbed, activate the resources and make sure that all resources are marked as 'ACTIVE'.

- find your controller in the list of resources
- click on the controller in order to get a link to its console (as you would do to get access to the console of any VM; see Figure 40, Figure 41, compare also with Figure 38)

| Testbeds | | | | | |
|--|----------------------|--------|-----------------------------------|---------|--|
| Provider ID | ID | Status | Type | Actions | |
| 344 | TwoOFXwithController | ACTIVE | Testbed (TwoOFXwithController) 12 | | |
| Host-1465203959772 | dataLJUVM1 | ACTIVE | Host | | |
| Host-1465203959994 | dataBRAVM2 | ACTIVE | Host | | |
| Host-1465203960200 | dataBRAVM1 | ACTIVE | Host | | |
| Host-1465203960368 | controller | ACTIVE | Host | | |
| OFX-1b602f0a-0239-46cc-9cf8-fa7c422e837f | ofxBRA | ACTIVE | OFX | | |
| OFX-da881a1c-fecd-48d1-9bdc-6f929c39b016 | ofxLJU | ACTIVE | OFX | | |
| OpenNsaLink-GT-ebda351c43 | l1CLJU | ACTIVE | Link | | |
| OpenNsaLink-GT-5d643ec80b | lOFXBRAOFXLJU | ACTIVE | Link | | |
| OpenNsaLink-GT-419fc7f1e9 | l1LJU | ACTIVE | Link | | |
| OpenNsaLink-GT-0074b43d85 | l2BRA | ACTIVE | Link | | |
| OpenNsaLink-GT-637ed2c659 | l1CBRA | ACTIVE | Link | | |
| OpenNsaLink-GT-ed6a596c4d | l1BRA | ACTIVE | Link | | |

Figure 40: Access to controller VM

| Resource: Host | | | | | | |
|--------------------|------------|--------|-------------|-----------|----------|---|
| Provider id | Id | Status | Description | Ports | Location | Console |
| Host-1465203960368 | controller | ACTIVE | | eth1 eth2 | prg | http://62.40.126.12:6080/vnc_auto.html?token=4b7506fa-920c-423e-a10c-be4d2d1dc04d |

| Reservations | | |
|--------------|-------------------------------|-------------------------------|
| Id | Start Time | End Time |
| 950 | Mon, 06 Jun 2016 09:05:00 GMT | Thu, 31 Jan 2526 23:59:59 GMT |

Figure 41: Link to controller console

- at the console you will be prompted for a user name and password (gts, gts)
- at the moment it is still necessary to manually configure the interfaces of the controller (in later versions of GTS it will be possible to simply set this via the GUI); this is because GTS currently does not support port mapping.
- **please note:** 192.168.x.x and eth0 should NOT be used, as they are currently reserved for internal processes!
- configure eth1 interface of controller for BRA (as already defined in the DSL example):

```
$ sudo su
# ifconfig eth1 up 10.10.10.10 netmask 255.255.255.0
and check for incoming ARP packets with
# tcpdump -i eth1
```

You will see the incoming echo request from the OFX like this:

```
#16:49:58.702223 IP 10.10.10.1.59938 > 10.10.10.10.6633: Flags [S], seq 688712374,
win 65535, options [mss 1460,nop,wscale 3,sackOK,TS val 4260253826 ecr 0], length 0
i.e. an OFX with IP addresss 10.10.10.1 requests IP 10.10.10.10.
```

- open the interfaces configuration file with your editor
/etc/network/interfaces
- add the following entries to the file:
auto eth1
iface eth1 inet static
address 10.10.10.10
netmask 255.255.255.0
network 10.10.10.0
- stop interface with
ifdown eth1
- start interface with
ifup eth1
- check if you can ping the OFX in BRA with IP address 10.10.10.1
ping -c 10.10.10.1
- repeat the same process for eth2 interface of controller for LJU (as already defined in the DSL example):
\$ sudo su
ifconfig eth2 up 10.10.11.1 netmask 255.255.255.0

Step 9: Example: Installing RYU as OpenFlow Controller

GTS OpenFlow switches support OpenFlow specification 1.3. Therefore we suggest to use RYU as OpenFlow controller in GTS, because it fully supports OpenFlow specification 1.3. To install the RYU OpenFlow controller the following steps are required:

- get root access with
\$ sudo su
- install dependencies
aptitude install python-setuptools python-pip python-dev libxslt-dev
libxml2-dev zlib1g-dev wget
- get the installation script from RYU
wget https://raw.githubusercontent.com/sdn2s-tw/ryuInstallHelper/master/ryuInstallHelper.sh
- make the script executable
chmod +x ryuInstallHelper.sh

- execute the installer
`# ./ryuInstallHelper.sh`
- this should install RYU applications and libraries. Follow the output messages during the installation process as in some cases errors may occur where missing packages have to be installed and the installer has to be run again
- you can find all the examples and test scripts under
`# /usr/local/lib/python2.7/dist-packages/ryu`
- to see if the controller works, you can change to the directory with applications
`# cd /usr/local/lib/python2.7/dist-packages/ryu/app/`
and run a simple application, e.g. `simple_switch_13.py`
- RYU uses port "6633" as default ; if you have specified another port in your DSL (such as `controllerPort="9966"` in the example above) you must also adjust the port number with
`# ryu-manager simple_switch_13.py --ofp-tcp-listen-port 9966`
- **please note:** GTS uses OpenFlow version 1.3 switches, therefore only apps for OpenFlow version 1.3 will work
- **please note:** In this version of GTS it is strongly discouraged to change the `datapathID` (Figure 42) that identifies your OpenFlow switch fabric as this identifier has to be absolutely unique!!

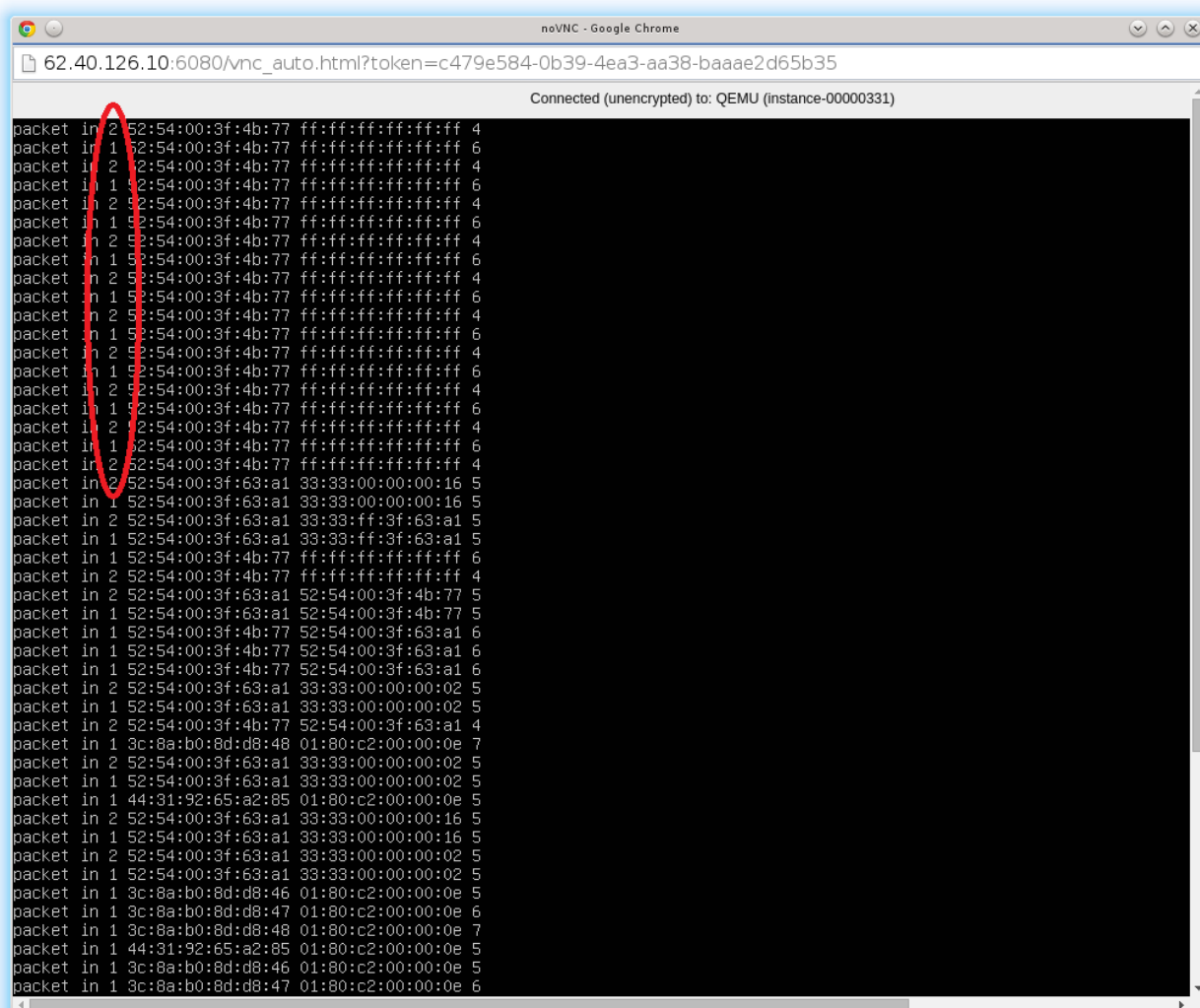


Figure 42: Example of a changed datapathID

5 Help and Support

For help and support please contact

- GÉANT Operations Centre
mailto: support@gts.geant.net; phone: +44 1223 866140

You also find this support information by clicking on 'About' at the bottom of the GTS home page (Figure 43).

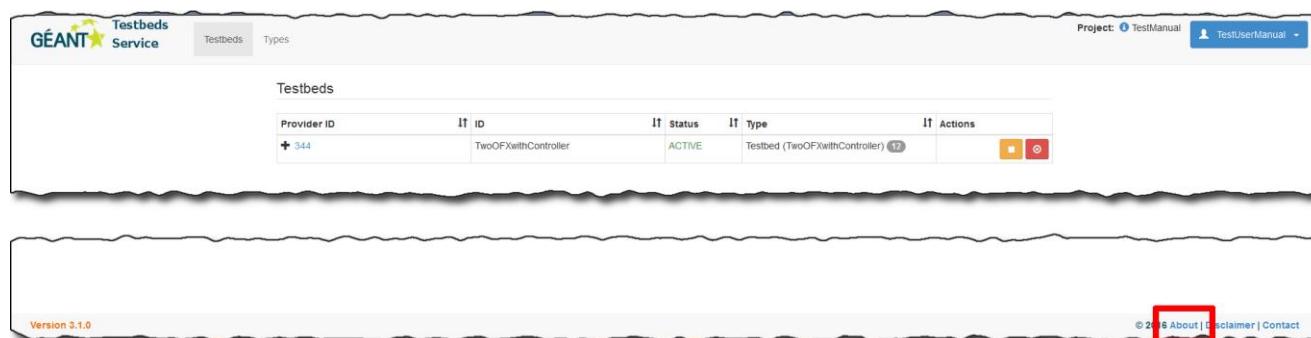


Figure 43: Access to support

In case you forgot your password, you can use the password hash calculator at <https://www.dailycred.com/article/bcrypt-calculator> to generate a hash for your new password (use 10 rounds for the hash generation). Please then send us your generated hash and we will update your account.

Additional information and training videos are available at <http://services.geant.net/gts/>.

6 Introduction to Domain Specific Language

6.1 DSL in GTS

GTS uses its own Domain Specific Language (DSL) to describe topologies. It is used by users to specify their requests, but also by GTS service to represent the allocated topology that conforms to users' requirements. DSL not only allows defining requests, but also offers Groovy [GRO-2014] programming language syntax that eases up the process of specifying scalable, complex topologies. In this section only DSL syntax is described. Readers should get familiar with Groovy programming language grammar if they want to use programming language features. For basic request definition knowing only DSL syntax is enough. The section is divided into two parts: Quick start and DSL grammar definition. The first offers introduction with examples where after five minutes users will know all concepts and should be able to define basic requests. The latter provides information about all resource types, their parameters and BNF (Backus–Naur Form) grammar of the DSL.

6.2 Quick Start

For every virtual resource there is a resource class or type, called a 'template' that defines all possible attributes of that resource. The process of creating a resource instance from this class template is referred to as 'instantiation'. In a testbed, a researcher can instantiate several instances of a resource and distinguish them by assigning each resource instance a unique name [DEL-D61].

Each resource class has

- a class name / an identifier of that class (not instance!) (name cannot start with a digit)
- external ports (where each port has a port name and a set of attributes for those ports)
- class attributes
- internal resources
- adjacencies among internal resources

- and a set of resource control primitives defined for that resource class.

Resources can be either atomic or composite. Atomic resources denote virtual resources that will be allocated for the user. Those that are composite are logical containers of other resources. This feature allows to define scalable, complex topologies, because composite types form reusable building blocks that can be combined together.

Before going into formal BNF grammar definition let's start with a simple example to get familiar with the DSL syntax. The code snippet below is a definition of a type containing one host. The type called *HostWithTwoPorts* is a composite that contains one host. The host has two interfaces called ports *p1* and *p2*. Each resource instance definition consists of a resource type name and a set of parameters and ports specified in curly braces. The parent-child relation is specified by defining the resource instance within the definition of the composite resource.

In this example the host has *HostWithTwoPorts* composite type as a parent.

```
type HostWithTwoPorts {  
    description = "Host that has two ports."  
  
    host {  
        id = "host"  
        port { id = "p1"}  
        port { id = "p2"}  
    }  
}
```

Let's imagine a more complex topology where we have three hosts connected to each other forming a triangle. We will reuse *HostWithTwoPorts* type to specify three hosts. Each resource can be connected with another resource via a transport resource called *link*. Links have exactly two ports called *src* and *dst* denoting source and sink. We need three hosts and three links.

```
type Triangle {
  description = "Triangle using PRG, LJU and BRA hosts."

  HostWithTwoPorts {
    id = "h1"
    host.location = "prg"
  }

  HostWithTwoPorts {
    id = "h2"
    host.location = "lju"
  }

  HostWithTwoPorts {
    id = "h3"
    host.location = "bra"
  }

  link { id = "l1" }
  link { id = "l2" }
  link { id = "l3" }
  ...
}
```

The *id* parameter in each resource is a user defined identifier that is unique only within the given DSL. It does not have to be a globally unique identifier. It helps users specifying properties and adjacencies. We used *ids* to define locations of internal nodes. The other benefit is when debugging what went wrong during instantiation of the request. Having hundreds of resources with meaningful identifiers greatly helps to find the cause of the problem in the request. The last step that we need in order to define the triangle are adjacencies between resources. We are using composites to connect to link resources.

```

type HostWithTwoPorts {
    description = "Host that has two ports."

    host {
        id = "host"
        port { id = "p1" }
        post { id = "p2" }
    }

    port { id = "p1" }
    port { id = "p2" }

    adjacency p1, host.p1
    adjacency p2, host.p2
}

```

In order to expose internal details, ie. ports of the underlying host, logical ports must be defined in the composite and connected to the host's ports. Then these logical ports will be used to connect to link ports. Having *HostWithTwoPorts* revisited we can define triangle adjacencies.

```

type Triangle {
    ...
    adjacency h1.p1, l1.src
    adjacency h2.p1, l1.dst
    adjacency h1.p2, l2.src
    adjacency h3.p2, l2.dst
    adjacency h2.p2, l3.src
    adjacency h3.p1, l3.dst
}

```

We specify that host *h1* port *p1* is connected to source of link *l1*. Link *l1* sink is connected to host *h2* port *p1* and so on. As you see we use user specified identifiers to help us denote specific resource instances.

Groovy programming language features come in handy when there is a need to specify complex topologies with possibly hundreds of connections. It can be done by defining procedures for doing so. Even the Triangle example can benefit from using programming. This way we do not have to define an additional type called *HostWithTwoPorts*. Take a look at the example below. We use a loop three times to define three hosts and three links. To ease up iterating over resources we defined two lists *hosts* and *links*. The second loop is used to connect all resources together.

```

type Triangle {
    description = "Triangle using Groovy language iterators to define
adjacencies."
    id = "t1"

    def hosts = []
    def links = []

    3.times { idx ->
        def h1 = host {
            id = "host$idx"
            port { id = "p1" }
            port { id = "p2" }
        }
        hosts << h1

        def l1 = link {
            id = "link$idx"
            port { id = "src" }
            port { id = "dst" }
        }
        links << l1

        adjacency h1.p1, l1.src
    }

    3.times { idx -> adjacency hosts[(idx + 1) % 3].p2, links[idx].dst }
}

```

Appendix I: Resource Guide

Each resource type has a different set of properties that can be specified or read by users. This section provides a list of resource types and possible properties with description of their field types and possible ranges.

I. Composite types

Aggregate logical resources are specified by this type.

List of parameters:

| Parameter | Type | Description | Example |
|-------------|---|---|--|
| description | String <read, write, optional> | Characterizes the type for the user. | Host that has two ports. |
| id | String <read, write, optional> | User defined identifier. If not specified it is autogenerated by the service. | h1 |
| providerId | String <read-only> | Globally unique identifier of the resource instance. Resource instances have this value set by the service. | nsi20010495 |
| status | RESERVED, ACTIVATING, ACTIVE, DEACTIVATING, RELEASING <read-only> | Current state of the resource. | ACTIVE |
| ports | Map<id, port> <read only> | Map of ports defined by the user. | port { id = "p1" } port { id = "p2" } |

Example:

```
type HostWithTwoPorts {
  description = "Host that has two ports."

  host {
    id = "h1"
    port { id = "p1" }
    port { id = "p2" }
  }
}
```

II. Host

A virtual machine is depicted by this type. Each VM has the following components preinstalled:

- Ubuntu 14.04.1 LTS (GNU/Linux 3.13.0-37-generic x86_64)

List of parameters:

| Parameter | Type | Description | Example |
|------------|---|---|--|
| location | PRG, BRA, LJU, AMS, MIL, HAM, PAR, MAD, LON <read, write, optional> | location of the resource | PRG |
| id | String <read, write, optional> | User defined identifier. If not specified it is autogenerated by the service. | h1 |
| providerId | String <read-only> | Globally unique identifier of the resource instance. Resource instances have this value set by the service. | Host-1464179133289 |
| status | RESERVED, ACTIVATING, ACTIVE, DEACTIVATING, RELEASING <read-only> | Current state of the resource. | ACTIVE |
| ports | Map<id, port> <read only> | Map of ports defined by the user. | port { id = "p1" } port { id = "p2" } |

| Default configuration ³ | | Comments |
|------------------------------------|---------------------------------|---|
| Host { } | cpuSpeed == 2.3 | The minimum clock speed of the CPU in GHz. |
| | image = "<string>" | The ISO image of the operating system installed on the host by default. The current operation system is "Ubuntu 14.04" |
| | disk == 20 | Disk space coming with the host* defined in GB. <i>* persistent storage (NFS) of 3GB is also available in folder "project-share" of testbed.</i> |
| | consoleAccess == "<public URL>" | Public URL of the console access to the host. |
| port { } | Framing == "<Eth>" | Only Ethernet (802.1Q) framing is available. |

Example:

```

host {
    id = "h1"
    location = "prg"
    port { id = "p1" }
}

```

³ Currently only default configuration is possible.

III. Link

Represents a virtual circuit between resources. Always has exactly two ports for source and sink.

List of parameters:

| Parameter | Type | Description | Example |
|------------|---|---|--|
| id | String <read, write, optional> | User defined identifier. If not specified it is autogenerated by the service. | l1 |
| providerId | String <read-only> | Globally unique identifier of the resource instance. Resource instances have this value set by the service. | OpenNsaLink-GT-228a44578b |
| status | RESERVED, ACTIVATING, ACTIVE, DEACTIVATING, RELEASING <read-only> | Current state of the resource. | ACTIVE |
| ports | Map<id, port> <read only> | Map of exactly two ports source and sink | port { id = "src" } port { id = "dst" } |

| | Default configuration ⁴ | Comments |
|-----------------|------------------------------------|---|
| Link { } | capacity = {1...10000} | Capacity of the link defined in Mbps. Default value is 10 Mbps |
| port { } | Framing = "<Eth>" | Only Ethernet (802.1Q) framing is available |
| | lineRate = {1, 10} | Nominal line rate of the port defined in Gbps. Default value is 1 Gbps. |
| | directionality = {uniDir, biDir} | Directionality of the port. Default value is bidirectional: "biDir" |

Example:

```
link {
  id = "l1"
  port { id="src" }
  port { id="dst" }
}
```

⁴ Currently only default configuration is possible.

IV. OFX

OpenFlow switches are represented by this type. Please note that one of ofx ports must be denoted with mode = "CONTROL" to inform the service that this port is going to be a control port.

GTS currently uses Hewlett Packard switches of the HP 5900 Series which support OpenFlow specification 1.3:

HP Comware Software, Version 7.1.035, Release 2208P01
 Copyright (c) 2010-2013 Hewlett-Packard Development Company, L.P.
 HP 5900AF-48G-4XG-2QSFP+

Boot image: flash:/5900_5920-cmw710-boot-r2208p01.bin
 Boot image version: 7.1.035P05, Release 2208P01
 System image: flash:/5900_5920-cmw710-system-r2208p01.bin
 System image version: 7.1.035, Release 2208P01

List of parameters:

| Parameter | Type | Description | Example |
|----------------|--|---|--|
| location | PRG, BRA, LJU, AMS, MIL, HAM, PAR, MAD, LON <read, write, optional> | location of the resource | PRG |
| id | String <read, write, optional> | User defined identifier. If not specified it is autogenerated by the service. | ofx1 |
| providerId | String <read-only> | Globally unique identifier of the resource instance. Resource instances have this value set by the service. | OFX-5af7dc69-228c-4a93-8ed1-e1f66aa1245 |
| status | RESERVED, ACTIVATING, ACTIVE, DEACTIVATING, RELEASING <read-only> | Current state of the resource. | ACTIVE |
| ports | Map<id, port> <read only> | Map of ports defined by the user. | port { id = "p1" } port { id = "p2" } |
| controllerIPv4 | IPv4 address | IPv4 Address of the controller node | 10.10.100.100 |

| | | | |
|--------------------|--------------|-----------------------------------|---------------|
| controllerPort | Number | TCP port of the controller node | 6633 |
| fabricIPv4 | IPv4 address | IPv4 address of the fabric switch | 10.10.100.1 |
| fabricSubnetMaskv4 | IP mask | IP mask of fabric subnet | 255.255.255.0 |

| Default configuration ⁵ | | Comments |
|------------------------------------|------------------------------------|--|
| OpenFlowFabric { } | ctrlProtocol = { "tcp", "ssl" } | Protocol used by the controller to communicate with the fabric. Default protocol is "tcp". |
| port { } | Framing = "Eth" | Only Ethernet (802.1Q) framing is available. |
| | lineRate = { 1, 10 } | Nominal line rate of the port defined in Gbps. Default value is 1 Gbps. |
| | directionality = { uniDir, biDir } | Directionality of the port. |

⁵ Currently only default configuration is possible.

Example:

```

type OneOFX {
    description = "Two hosts connected to OpenFlow switch."

    host {
        id = "h1"
        location = "PRG"
        port { id = "port1" }
    }

    host {
        id = "h2"
        location = "PRG"
        port { id = "port1" }
    }

    link {
        id = "l1"
        port { id="src" }
        port { id="dst" }
    }

    link {
        id = "l2"
        port { id="src" }
        port { id="dst" }
    }

    ofx {
        id = "ofx1"
        location = "PRG"
        fabricIPv4="10.10.100.1"
        fabricSubnetMaskv4="255.255.255.0"
        controllerIPv4="10.10.100.100"
        controllerPort="6633"
        port { id = "port1" }
        port { id = "port2" }
        port {
            id="port3"
            mode="CONTROL"
        }
    }

    adjacency h1.port1, l1.src
    adjacency ofx1.port3, l1.dst
    adjacency h2.port1, l2.src
    adjacency ofx1.port2, l2.dst
}

```

External Domain

The External Domain resource [SOB-2015] represents an endpoint in some facility that is outside the GTS service area. This facility is reachable by a preprovisioned virtual circuit that has been established outside of the scope of a normal GTS testbed life cycle by the GTS service management team.

External domains are represented with the following syntax:

```
ExternalDomain {  
    id = "ex1"  
    location = "lab1"  
    port { id = "ep1" }  
}
```

Users may specify the following properties of the external domain:

- *id* – alphanumeric string, case sensitive; The ID is a user chosen name of the External Domain resource (unique within the scope of the Testbed DSL) that can be used within Adjacency specifications to reference a particular ED object/port pair, or for other debugging purposes during the life cycle of the testbed;
- *location* – choice of “AMS”, “BRA”, “LJU”, “PRG”, “LON”, “HAM”, “MIL”; case insensitive. The GTS pop location defines where the External Domain virtual circuit appears on the edge of the GTS domain.
- *Port id* - choice of “ep1”, or “ep2”. The port construct specifies a port ID that is preconfigured within GTS (by the service management team) to be the GTS endpoint for an External Domain virtual circuit. The port id must be either “ep1” or “ep2”. These port ids, at the associated location, are configured to represent different External Domains at different times. The user and the GTS service management team must coordinate to arrange for an external Domain facility to be attached to the GTS domain, and to identify the specific External Domain <location>/<portid> pair which will terminate the virtual circuit going to the particular External Domain. For convenience, the user is able to use the External Domain identifier and the port ID to reference the External domain in adjacency specifications.

To reiterate: It is the “location”/“port id” pair that maps to a particular External Domain at any particular time. And within the testbed DSL description, an External Domain resource instance can be specified and referenced much like a Host resource. Below you can find an example with a host “h1” which is defined to be in Prague that is connected to an external domain called “ProtoGENI-SL” which meets GTS in London and presents a port id “ep1” that other testbed resources can connect to. The host *h1* port *eth1* is connected via link *l1* to external domain *ProtoGENI-SL* port *ep1*.

Example:

```
ExternalExample {
    host {
        id = "h1"
        port { id = "eth1" }
        port { id = "eth2" }
    }

    link {
        id = "l1"
        port {id = "src" }
        port {id = "dst" }
    }

    ExternalDomain {
        id = "ProtoGENI-SL"
        location = "LON"
        port { id = "ep1" }
    }

    adjacency h1.eth1, l1.src
    adjacency ProtoGENI-SL.ep1, l1.dst
}
```

For more information please contact support@gts.geant.net.

V. BNF Grammar

`<property> ::= <key>=<value>`

`<properties> ::= <property>(<|\n>)<properties>`

`<port> ::= port { <properties> }`

`<atomic> ::= <atomic-type> { <properties> <ports> }`

`<atomic-type> ::= host | link | ofx`

`<composite> ::= type <name> { <properties> <resources> <ports> <adjacencies> }`

`<resources> ::= <resource> <resources>`

`<resource> ::= <composite> | <atomic>`

`<adjacencies> ::= <adjacency><adjacencies>`

`<adjacency> ::= adjacency <port-id>, <port-id>`

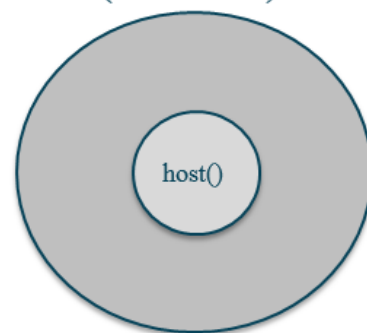
VI. Examples: One host

The following example shows how one host can be described with DSL:

```
testbed {
  id = "OneHost"

  host {
  }
}
```

testbed ("OneHost")

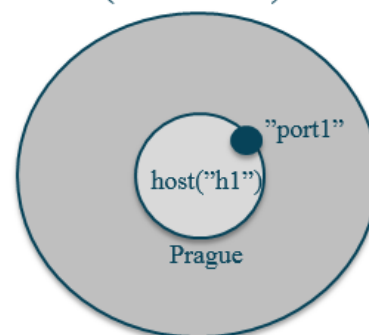


If you need to have one host at a specific location for your experiment then you must specify this in your DSL code:

```
testbed {
  id = "OneHost"
  description = "One host in Prague"

  host {
    id = "h1"
    location = "PRG"
    port { id = "port1" }
  }
}
```

testbed ("OneHost")



VII. Example: Two hosts linked together

This example shows in two steps how you can define two hosts and a link and connect them to each other:

Step 1: Describe both hosts and the link:

```
HostsLine {
    description = "PRG host linked
with BRA host"

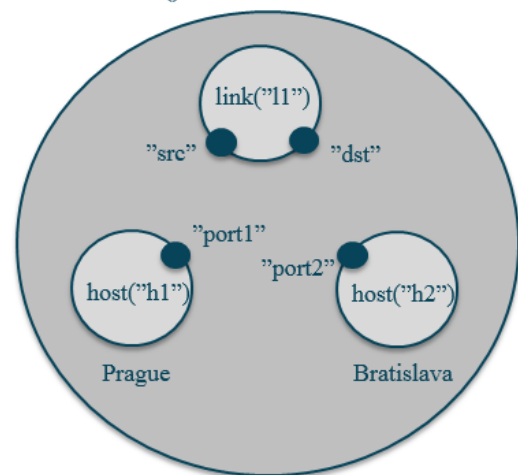
    host {
        id = "h1"
        location = "PRG"
        port { id = "port1" }
    }

    host {
        id = "h2"
        location = "BRA"
        port = { id = "port2" }
    }

    link {
        id = "l1"
        port { id = "src" }
        port { id = "dst" }
    }

    ...
}
```

HostsLine ()



Step 2: Describe the adjacencies to actually connect the entities:

```

HostsLine {
    description = "PRG host linked with
    BRA host"

    host {
        id = "h1"
        location = "PRG"
        port { id = "port1" }
    }

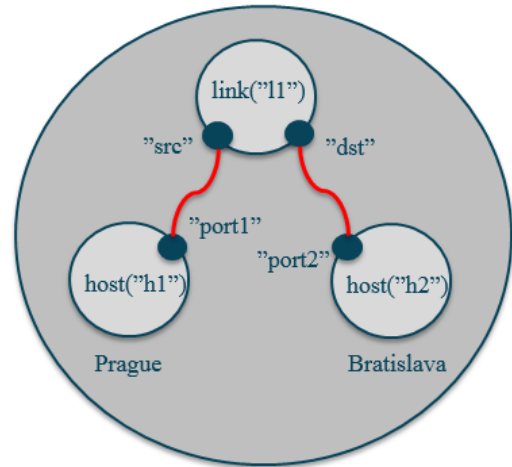
    host {
        id = "h2"
        location = "BRA"
        port { id = "port2" }
    }

    link {
        id = "l1"
        port { id = "src" }
        port { id = "dst" }
    }

    adjacency h1.port1, l1.src
    adjacency h2.port2, l1.dst
}

```

HostsLine ()



VIII. Example: Triangle between three locations

For a triangle to be set up between three different locations you need to first describe the three hosts and specify their locations in your DSL code:

Step 1: Describe hosts and locations:

```
type triangle {
  description = "Triangle between PRG,
  BRA, and LJU"

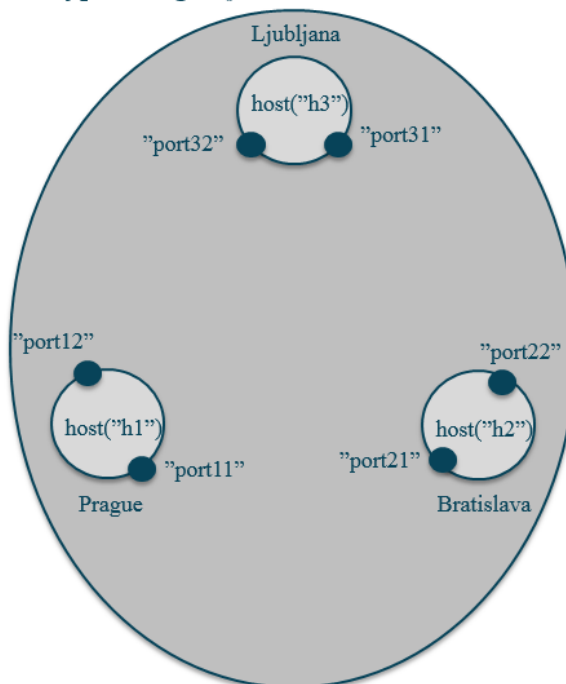
  host {
    id = "h1"
    location = "PRG"
    port { id = "port11" }
    port { id = "port12" }
  }

  host {
    id = "h2"
    location = "BRA"
    port { id = "port21" }
    port { id = "port22" }
  }

  host {
    id = "h3"
    location = "LJU"
    port { id = "port31" }
    port { id = "port32" }
  }

  ...
}
```

type triangle ()



Step 2: Describe the three links between them:

```

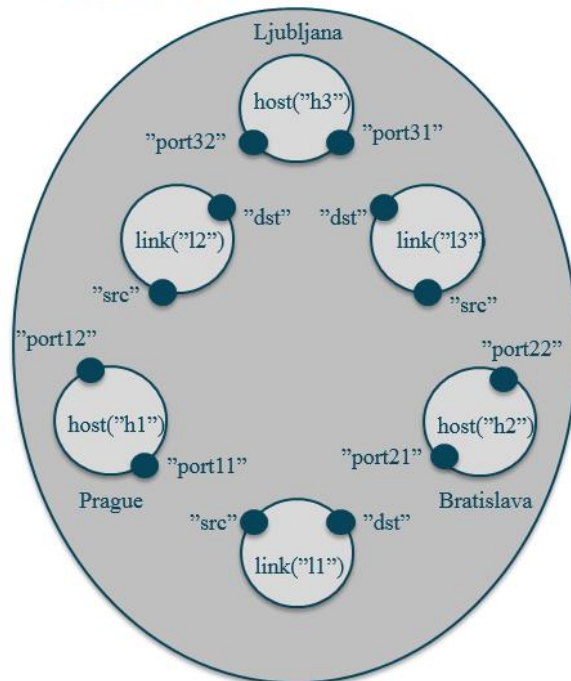
...
link {
    id = "l1"
    port { id = "src" }
    port { id = "dst" }
}

link {
    id = "l2"
    port { id = "src" }
    port { id = "dst" }
}

link {
    id = "l3"
    port { id = "src" }
    port { id = "dst" }
}
...

```

type triangle ()



Step 3: Describe the adjacencies to connect them:

```

...
link {
    id = "l1"
    port { id = "src" }
    port { id = "dst" }
}

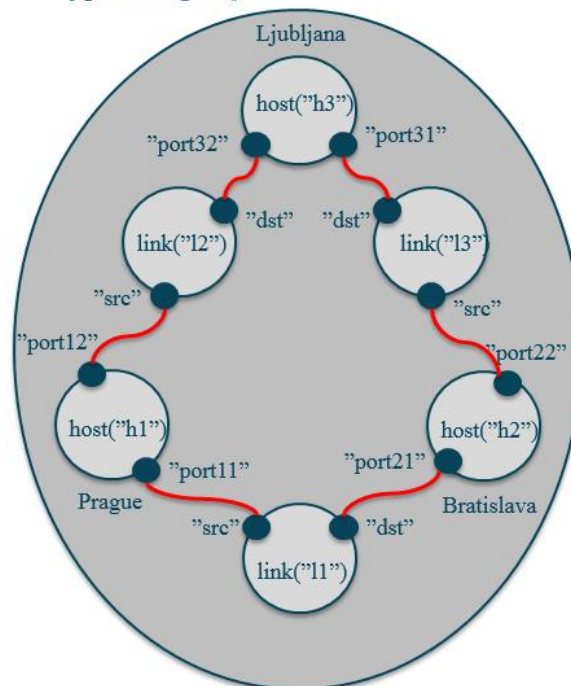
link {
    id = "l2"
    port { id = "src" }
    port { id = "dst" }
}

link {
    id = "l3"
    port { id = "src" }
    port { id = "dst" }
}

adjacency h1.port11, l1.src
adjacency h2.port21, l1.dst
adjacency h1.port12, l2.src
adjacency h3.port32, l2.dst
adjacency h2.port22, l3.src
adjacency h3.port31, l3.dst
}

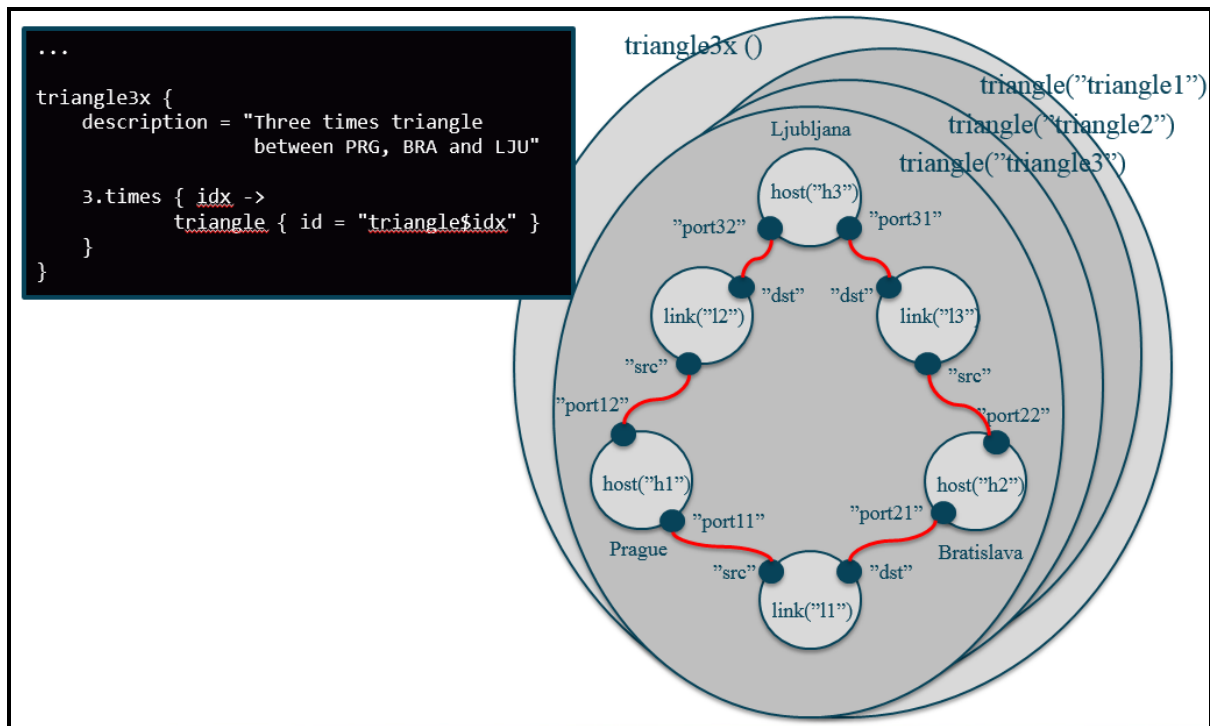
```

type triangle ()



IX. Example: Three triangles built using DSL code iterations

The following describes three triangles being setup between Prague, Bratislava and Ljubljana:



```

type Triangle {
    description = "Triangle using Groovy
language iterators to define adjacencies."
    id = "t1"
    def hosts = []
    def links = []

    3.times { idx ->
        def h1 = host {
            id = "host$idx"
            port { id = "p1" }
            port { id = "p2" }
        }
        hosts << h1

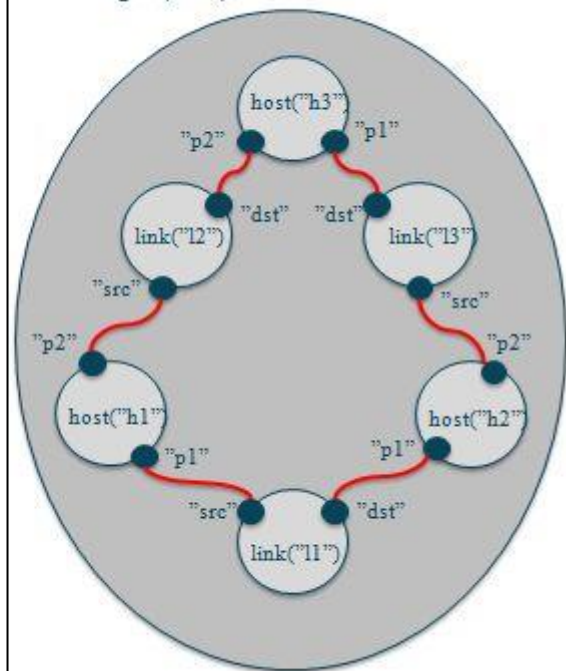
        def l1 = link {
            id = "link$idx"
            port { id = "src" }
            port { id = "dst" }
        }
        links << l1

        adjacency h1.p1, l1.src
    }

    3.times { idx -> adjacency hosts[(idx +
1) % 3].p2, links[idx].dst }
}

```

triangle("t1")



X. Example: Two OpenFlow switches, one controller and three hosts

```
TwoOFXwithController {
  description = "Two OFX with one controller VM. The OFXs are at different locations and
connected."
  id = "TwoOFXwithController"

  host {
    id="controller"
    port { id="eth1" }
    port { id="eth2" }
  }

  host {
    id="dataBRAVM1"
    location="BRA"
    port { id="eth1" }
  }

  ofx {
    id="ofxBRA"
    location="BRA"
    fabricIPv4="10.10.10.1"
    fabricSubnetMaskv4="255.255.255.0"
    controllerPort="9966"
    controllerIPv4="10.10.10.10"
    port { id="p1"}
    port { id="p2"}
    port { id="pLJU"}
    port {
      id="pControl"
      mode="CONTROL"
    }
  }

  link {
    id="l1BRA"
    port { id="src" }
    port { id="dst" }
  }

  link {
    id="l1CBRA"
    port { id="src" }
    port { id="dst" }
  }

  adjacency dataBRAVM1.eth1, l1BRA.src
  adjacency ofxBRA.p1, l1BRA.dst

  adjacency controller.eth1, l1CBRA.src
  adjacency ofxBRA.pControl, l1CBRA.dst

  host {
    id="dataBRAVM2"
    location="BRA"
    port { id="eth1" }
```

```

}

link {
    id="l2BRA"
    port { id="src" }
    port { id="dst" }
}

adjacency dataBRAVM2.eth1, l2BRA.src
adjacency ofxBRA.p2, l2BRA.dst

ofx {
    id="ofxLJU"
    location="LJU"
    fabricIPv4="10.10.11.1"
    fabricSubnetMaskv4="255.255.255.0"
    controllerPort="9966"
    controllerIPv4="10.10.11.10"
    port { id="p1"}
    port { id="pBRA"}
    port {
        id="pControl"
        mode="CONTROL"
    }
}

host {
    id="dataLJUVM1"
    location="LJU"
    port { id="eth1" }
}

link {
    id="l1LJU"
    port { id="src" }
    port { id="dst" }
}

link {
    id="l0FXBRAOFXLJU"
    port { id="src" }
    port { id="dst" }
}

link {
    id="l1CLJU"
    port { id="src" }
    port { id="dst" }
}

adjacency controller.eth2, l1CLJU.src
adjacency ofxLJU.pControl, l1CLJU.dst

adjacency dataLJUVM1.eth1, l1LJU.src
adjacency ofxLJU.p1, l1LJU.dst

adjacency ofxBRA.pLJU, l0FXBRAOFXLJU.src
adjacency ofxLJU.pBRA, l0FXBRAOFXLJU.dst
}

```


References

| | |
|----------|--|
| DEL-D61 | Deliverable D6.1 (DS2.3.1): Architecture Description: Dynamic Virtualised Packet Testbeds Service |
| FAR-2014 | F. Farina, P. Szegedi, J. Sobieski, GÉANT World Testbed Facility - Federated and distributed Testbeds as a Service facility of GÉANT, published at FIDC 2014 in Karlskrona, Sweden, 2014 |
| GRO-2014 | GROOVY, http://groovy.codehaus.org/ |
| RYU-2014 | RYU Project Team, RYU SDN Framework, http://osrg.github.io/ryu-book/en/Ryubook.pdf , available on September 29, 2014 |
| SOB-2015 | J. Sobieski, S. Naegele-Jackson, B. Pietrzak, M. Hazlinsky, F. Farina and K. Kramaric, GÉANT Testbeds Service (GTS) - GÉANT Testbed Service - External Domain Ports: A demo on multiple domain connectivity, European Workshop on Software Defined Networks (EWSDN), Bilbao, Sept. 30 - Oct. 2, 2015 |
| SZE-2014 | P. Szegedi, User's Training 1 – The Basics |