Grant Agreement No.: 610764
Instrument: Collaborative Project
Call Identifier: FP7-ICT-2013-10

PANACEA

Proactive Autonomic Management

of Cloud Resources

D4.3: Preparation of the integrated environment, simulation data, scenarios and results

Version 1.0

Work package	WP4
Task	Task 4.3
Due date	31/10/2015
Submission date	31/10/2015
Deliverable lead	QoS Design
Version	1.0
Authors	Ahmad Al Sheikh, Anouar Rachdi
Reviewers	Olivier Brun, Eduardo Huedo Cuesta, Michel Diaz

Abstract	This document describes the strategy that will be used to validate the main outcomes of the self-managing routing overlay system and ensure that it meets its design specifications. We then describe the experimental platforms that will be used for the implementation and evaluation of the aforementioned system. For each of them, we describe in details the methodology that will be used, as well as the expected results.
Keywords	Overlays, test and validation plan, simulation, emulation, testbed validation

Document Revision History

Version	Date	Description of change	List of contributor(s)
0.9	15/10/2015	First Draft	Ahmad Al Sheikh, Anouar Rachdi
1.0	29/10/2015	Final release	Ahmad Al Sheikh, Anouar Rachdi

Disclaimer

Project co-funded by the European Commission in the 7th Framework Programme (2007-2013)		
Nature of the deliverable:	R	
Dissemination Level		
PU	Public	**X**
PP	Restricted to other programme participants (including the Commission Services)	
RE	Restricted to bodies determined by the PANACEA project	
CO	Confidential to PANACEA project and Commission Services	

EXECUTIVE SUMMARY

In this document, we describe the approach used for the validation of the autonomic routing overlay and to ensure that it meets its design specifications.

We first present the two software environments that were built to create the experimental platforms. The first one is a simulation environment built upon NEST IP/MLPS. The second one is an emulation platform driven by QoS Design's simulation kernel. The emulation platform takes benefit of the CORE emulator. In addition, NEST Entreprise was adapted to act as a hypervisor for the SMART module (the module that monitors the overlay network performances and highlights routing decisions). SMART is the name of the open source software that what previously referred to as PON (PANACEA Overlay Network).

For each environment, we describe in detail the methodology that will be used to experiment the outcomes of the SMART module.

We finally present a test plan with a clear scenario related to Use Case 1 (cloud web hosting). We also present preliminary results for the simulation environment. Results concerning Use Case 2 are not included in this document and are to be published in the final report.

Table of Contents

EXECUTIVE SUMMARY **3**
TABLE OF CONTENTS **4**
LIST OF FIGURES **5**
ABBREVIATIONS **7**
1 INTRODUCTION **8**
2 THE FIRST APPROACH: SIMULATION **9**
2.1 NEST IP/MPLS evolutions 10
2.1.1 Overlay editor 11
2.1.2 Simulation kernel 14
2.1.3 Simulation process 17
2.2 NEST ENTREPRISE evolutions 18
2.2.1 Adverse event editor 19
2.2.2 Overlay path widget 21
2.2.3 Topology widget 22
2.2.4 QoS performance widget 23
3 THE SECOND APPROACH: EMULATION **25**
3.1 Core platform configuration 26
3.2 The use of application VMs 27
3.3 Description of the emulation process 29
4 VALIDATION AND PRELIMINARY RESULTS **31**
4.1 Validation process 31
4.2 Simulation results 32
5 CONCLUSION AND FUTURE WORK **38**
6 REFERENCES **39**

LIST OF FIGURES

Figure 1: NEST IP/MPLS....9
Figure 2: NEST Entreprise....10
Figure 3: A network connecting five clouds....11
Figure 4: Network overlay editor....12
Figure 5: Overlay network topology configuration....12
Figure 6: The configured overlay network....13
Figure 7: Visual representation of the created overlay network....13
Figure 8: Route for the overlay path from "Cloud 1" to "Cloud 5"....14
Figure 9: Evolution of the number of connections between two communicating nodes....15
Figure 10: An example on the evolution of the maximum interface load....16
Figure 11: The simulation process....18
Figure 12: Adverse events editor....20
Figure 13: Adding an interface failure....20
Figure 14: Adverse events timeline....21
Figure 15: Overlay path widget....22
Figure 16: Visualise overlay path between proxies....23
Figure 17: Modification of the overlay path and forwarding via an intermediary proxy....23
Figure 18: Evolution of end-to-end delay metric for an overlay path....24
Figure 19: Evolution of available bandwidth for an overlay path....24
Figure 20: CORE emulator GUI....25
Figure 21: An emulated network created in CORE....26
Figure 22: Testing connectivity between emulated network routers....27
Figure 23: Connecting external VMs to CORE network....28
Figure 24: Sending traffic between VMs across the CORE network....29
Figure 25: Several application VMs connected to the CORE emulator....29
Figure 26: The emulation process....30
Figure 27: Checkout phase in e-commerce....32
Figure 28: Maximum interface load for the test network....33
Figure 29: Interface loads at peak hours....34
Figure 30: Default network path for the overlay link "Cloud 5 -> Cloud 3"....35
Figure 31: End-to-end delay evolution for the overlay link "Cloud 5 -> Cloud 3"....35
Figure 32: Overlay path proposed by SMART....36

Figure 33: Available bandwidth evolution for the overlay link "Cloud 5 -> Cloud 3" 37

ABBREVIATIONS

AS	Autonomous System
BGP	Border Gateway Protocol
CORE	Common Open Research Emulator
CPU	Central Processing Unit
FCFS	First Come First Served
GRE	Generic Routing Encapsulation
GUI	Graphical User Interface
HSRP	Hot Standby Router Protocol
IP	Internet Protocol
IS-IS	Intermediate System to Intermediate System
LSP	Label-Switched Path
MP-BGP	Multi-Protocol BGP
MPLS	Multi-Protocol Label Switching
NEST	Network Engineering and Simulation Tool
OS	Operating System
OSPF	Open Shortest Path First
PE	Provider Edge
PON	PANACEA Overlay Network
QoS	Quality of Service
RAM	Random Access Memory
SLA	Service Level Agreement
SLO	Service Level Objective
SMART	Self-MAnaging RouTing overlay
TPC-W	Transaction Processing Performance Council Web e-commerce Benchmark
VM	Virtual Machine
VPN	Virtual Private Network

1 INTRODUCTION

Even small degradations in the performance and availability of modern computer services can have a considerable business impact. These services usually require continuous operation over time, always maintaining the response time below an acceptable threshold, in order to fulfil the user requirements, avoid lost revenue and reduced productivity.

In spite of the increased adoption of clouds, many companies are still reluctant to deploy mission critical applications in the cloud. The main reason for this reluctance is that, currently, cloud-computing platforms do not provide the availability and performance levels [3, 4, 5, 6] required by these applications.

The primary goal of PANACEA is to tackle and ensure these requirements on the deployment of services in the cloud. This is achieved through the design and development of innovative solutions for a proactive and autonomic management scheme of cloud services. PANACEA addresses both the failures and performance degradations that can occur at the node level and at the network level.

In this document we focus on the validation of the SMART (previously called PON) module, which is developed by LAAS-CNRS and is part of the PANACEA solution. SMART's main role is to monitor the communication links of the overlay network and to discover and track the optimal paths according to application-specific metrics.

We are mainly interested in the ability of SMART to bypass and recover from many inevitable anomalies [2], and to autonomously optimize performances of the overlay network under changing conditions. In other words, we want to make sure that SMART meets its design specifications, that is, that it effectively provides a self-healing and self-optimizing communication system on top of the Internet network.

This validation process is carried out using both simulation and emulation environments. This approach is able to provide large-scale experimentation and more flexibility in controlling the system behaviour, which is not possible in a real-life environment (Internet).

Using simulation, for example, we use QoS Design's simulation tools to easily describe the desired topologies and services, introduce time specific events and anomalies, and monitor the behaviour of SMART and of its decisions to overcome performance degradations and increase performance in the overlay network. In this context, we are validating the self-optimizing and self-healing properties of the overlay network.

Similarly, the emulation provides a controlled environment in which we can run real applications. Here, we can again design the desired network platform using adequate emulation tools, and create our overlay network while having SMART running and monitored. As in the simulation environment, anomalies will be introduced to deteriorate performances. SMART should under these conditions propose solutions to remedy these network degradations. The proposed solutions could be analysed using QoS Design's environment.

The rest of this document is organized as follows. Section 2 first describes in detail the simulation environment and the modules used/developed to carry out the simulation process. Section 3 then illustrates the emulation process, the tools used to create this environment, and the associated configuration. In Section 4, we describe the validation process and the test plan in addition to some preliminary results on the simulation. Section 5 finally concludes this document and discusses future work.

2 THE FIRST APPROACH: SIMULATION

In this approach, we will focus on creating an environment able to run the simulation of the overlay network. For this purpose we will use NEST IP/MPLS support to create the underlying topology infrastructure (Router, Links, Servers, etc.) and the overlay network (overlay proxy nodes). A distributed application will also be mapped on the overlay network.

As a reminder, NEST IP/MPLS provides a unified framework for wired and wireless networks. It supports several protocols (OSPF, IS-IS, BGP, etc.), inter-AS routing policies, and many other features. It supplies a global simulation kernel able to handle large volumes of traffic flows, and various decision support tools such as resilience and bottleneck analyses, IP routing optimizations, MPLS traffic engineering and LSP optimal placement and protection.

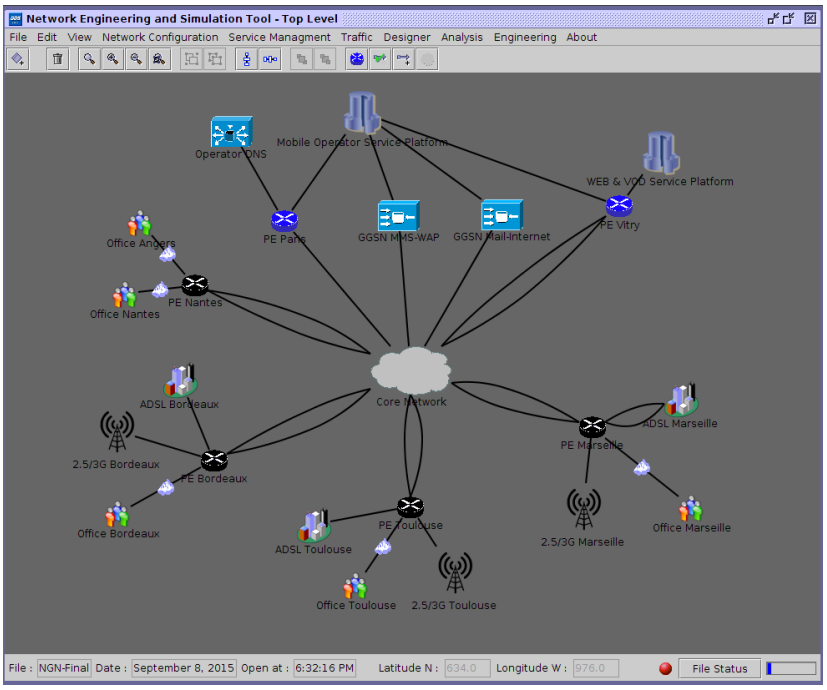


Figure 1: NEST IP/MPLS

In order to represent the evolution of the global network over a period of time, NEST Entreprise is utilised. NEST Entreprise is a workbench that allows automatic control of IP/MPLS networks. It provides the user with powerful tools for supervision, dynamic analysis (real-time) and intelligent control of networks. Coupled with NEST IP/MPLS, NEST Entreprise provides a powerful integrated workbench for supervision and intelligent control of IP/MPLS networks. It relies on simulations in the loop and autonomic control algorithms powered by real-time measurements.

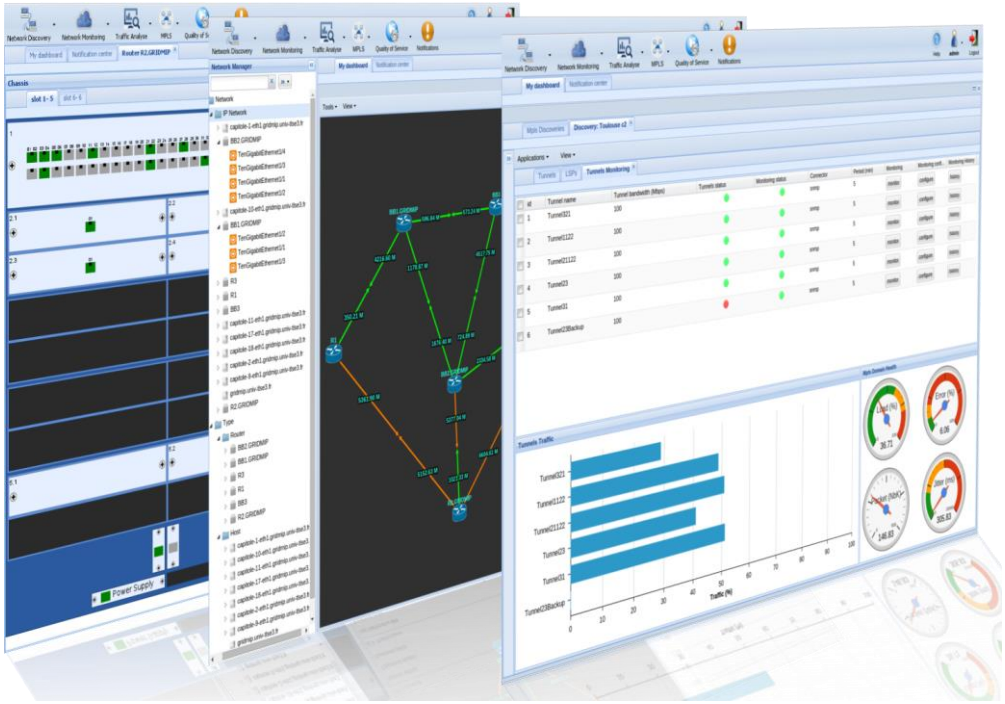


Figure 2: NEST Entreprise

NEST Enterprise will be used to focus on overlay network performances by highlighting the metrics related to the overlay links used by the distributed application. The evolution will be based on stochastic events generation within the simulation process. This framework will also be adapted to act as a hypervisor of the overlay network system. It will provide real-time supervision of the overlay performances and the decisions made.

The simulation approach has two benefits. It will allow performing large-scale experimentations that cannot be achieved in real life due to feasibility and economic cost issues. It will also allow stressing the overlay network under certain conditions to validate the self-optimizing and self-healing properties of the overlay network.

2.1 NEST IP/MPLS evolutions

In the aim of achieving simulation goals, the NEST IP/MPLS environment (UIs and modules) was first adapted to the PANACEA requirements. The undergone modifications that were designed and implemented allow the user to:

- Define the underlay network infrastructure, including routers, links, servers, and associated network parameters (protocols, metrics, etc.). The user is provided with a rich library of equipment models to create or replicate the physical network infrastructure. In addition, the user can define existing services, such as MPLS LSPs or VPNs, and provide their configurations. As an example, the LSP tunnels can be placed manually or through the help of the automatic LSP placement module.
- Define overlay network proxies that represent overlay access nodes onto which, distributed applications are mapped. These proxies are connected to the related equipment (such as routers) of the underlay network (the network infrastructure).
- Define the overlay network between the aforementioned proxies, built upon the

physical underlay real network. The underlay network acts as an "IP fabric" that provides connectivity between the proxies (e.g. by implementing MPLS tunnels).

- Generate backbone traffic to modify the network state and allow experimentation on shortest path computations of the overlay links. During a given interval of time (day, week, etc.), traffic matrices are generated based on user-defined parameters for each time instance (e.g. a traffic matrix every 5 minutes). These traffic matrices are used in conjunction with user-defined perturbations and failures to simulate the network evolution and its impact on the overlay metrics.
- Simulate the network behaviour (routing tables, traffic forwarding, etc.) and deduce network performance metrics which impact overlay path decisions. These metrics are communicated to SMART to optimise overlay paths when necessary.

In the following we will give the main details of the key modifications of the NEST IP/MPLS environment, mainly concerning overlay network creation, simulation and performance evaluation, and its interaction with SMART.

For illustration purposes, a simple example network will be considered hereafter (shown in Figure 3). It includes five backbone routers in a mesh configuration and five access routers to which clouds (and consequently their overlay proxies) are connected. The aforementioned clouds constitute the nodes of the overlay network.

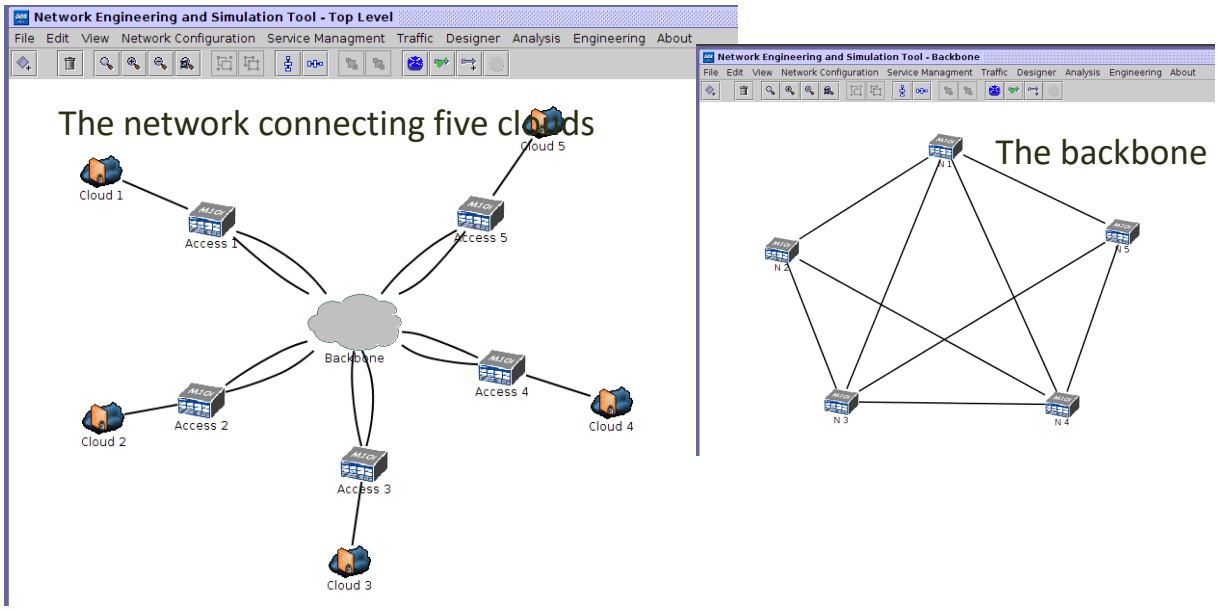


Figure 3: A network connecting five clouds

2.1.1 Overlay editor

After creating the desired topology, by using NEST's equipment library and as shown in Figure 3, the user can proceed to configure his overlay network. An overlay editor was conceived for this purpose (Figure 4).

This editor allows adding several network overlays with different configurations, including the supported applications and the type of the architecture, which is "Full Mesh" in our example. Each network overlay is configured simply by defining the overlay nodes (proxies) that

constitute it. We can do so by pressing the "Configure Topology" button.

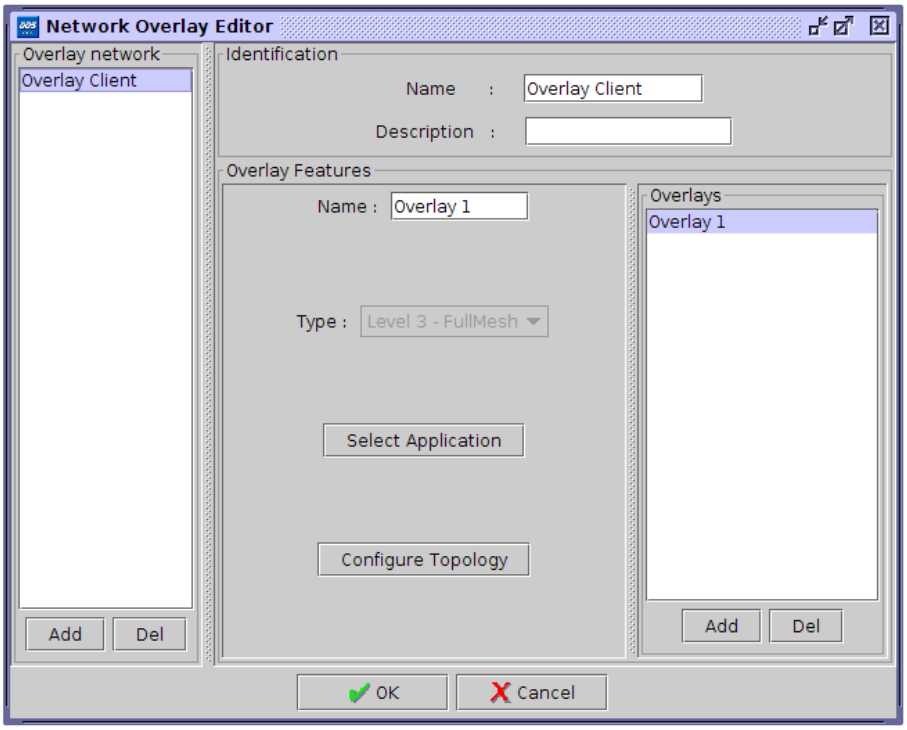


Figure 4: Network overlay editor

Figure 5 shows the overlay network topology editor. This editor allows the user to select the overlay nodes (proxies) that constitute the overlay network. We first start by selecting the access routers (PE Nodes) to which the overlay proxies are connected, and then the proxies themselves. After filling out the Service Level Objectives (SLO) for the overlay, we obtain a configuration similar to the one given in Figure 6.

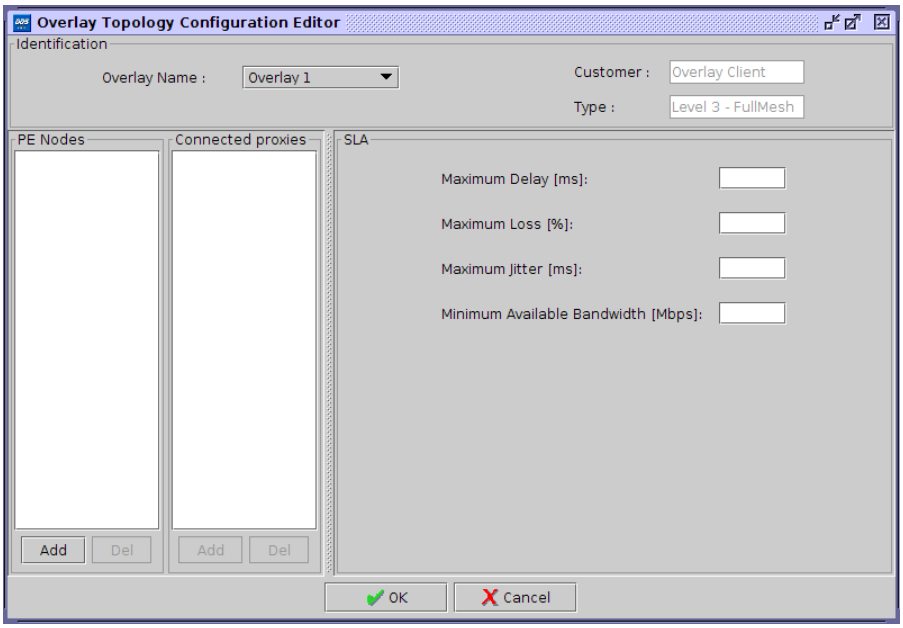


Figure 5: Overlay network topology configuration

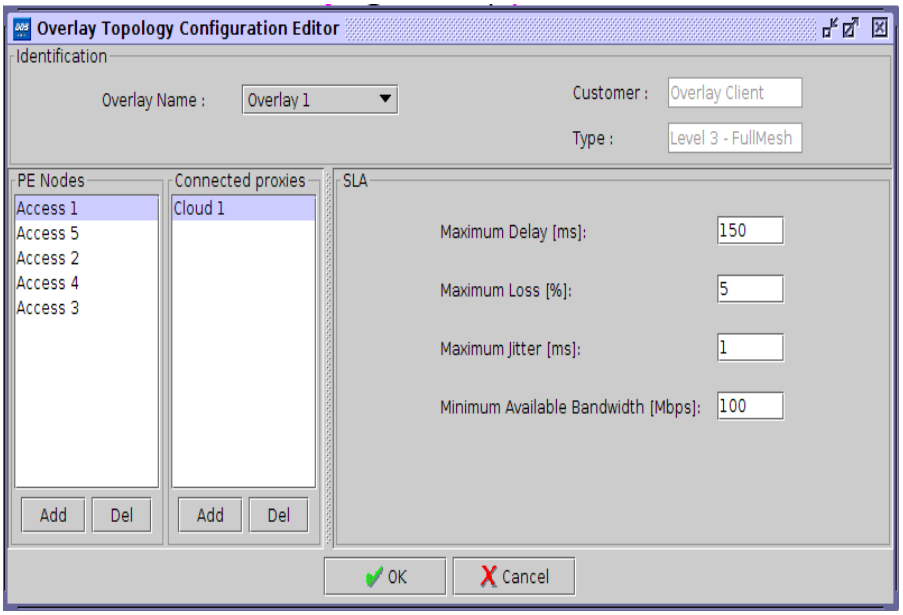


Figure 6: The configured overlay network

By closing this editor and selecting the created overlay in the "Overlays" list, we can visualise the overlay network graphically as shown in Figure 7 (based on the example of Figure 3).

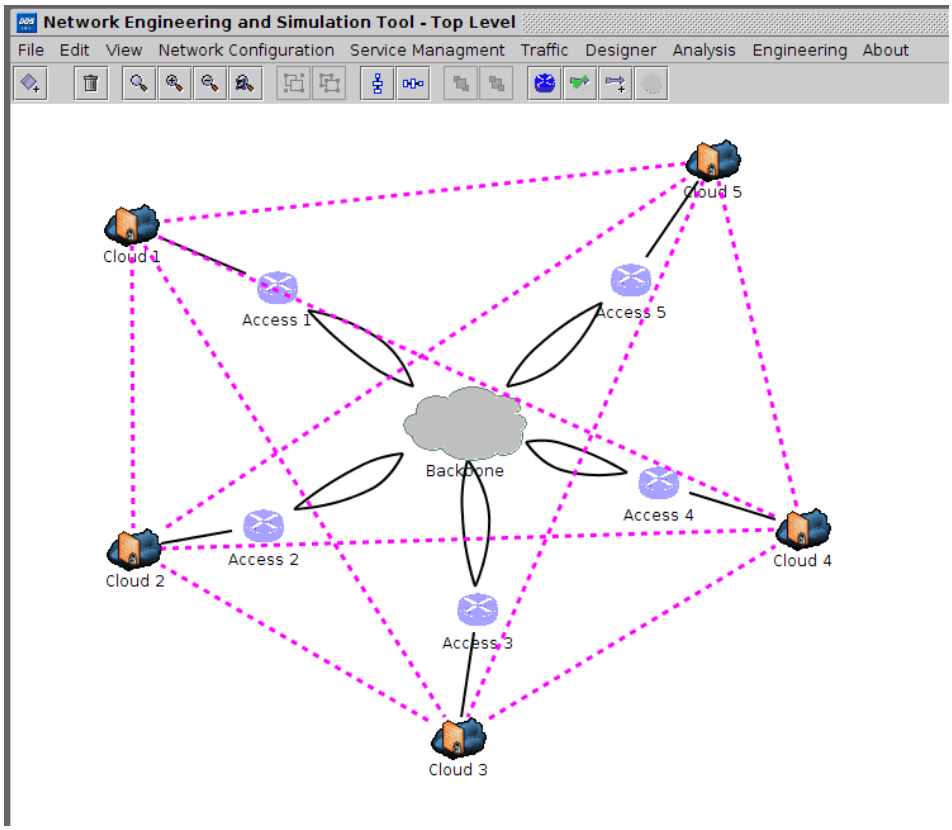


Figure 7: Visual representation of the created overlay network

NEST IP/MPLS will afterwards route the overlay links in the underlay infrastructure based on

the (network level) configured protocols and associated parameters. As an example, without optimisation, the overlay path from "Cloud 1" to "Cloud 5" uses the internet solution and is assumed to be routed as shown (in yellow) in Figure 8.

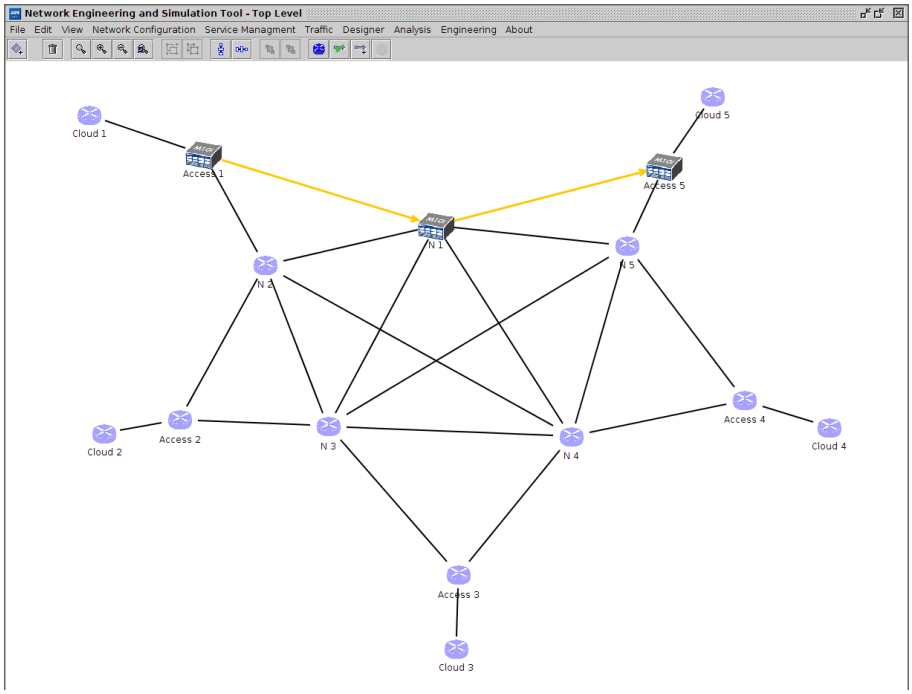


Figure 8: Route for the overlay path from "Cloud 1" to "Cloud 5"

2.1.2 Simulation kernel

Following the underlay and overlay network creation and configuration, NEST IP/MPLS will allow:

1. Network traffic generation
2. Network simulation and performance evaluation

The traffic generator allows the user to create, for a given topology, the expected traffic between the source/destination pairs. The user has the ability to restrict ingress and egress nodes in the network so as to exclude backbone and intermediary nodes from the traffic matrices. Remember that a traffic matrix describes the traffic intensity between source/destination pairs. Services are also implemented to characterize the traffic. Services define the upward and downward rates of communication, packet sizes and several other attributes for a traffic flow.

The traffic matrix generation is based on several parameters, including:

- The time frame: start and end dates
- Instance interval: the time step between two traffic matrix generations
- The services to be used: P2P (FTP, VoIP, etc.) or P2S (Video Streaming, Email, etc.)
- The distribution of the number of connections between source/destination pairs (global, per service or per source/destination pair)

For example, the user is able to generate, for a network he created, traffic matrices for a period of 24 hours with 5 minutes intervals. He could define, as an example, Poisson distributions for the variation of the number of connections for the defined services. Figure 9 shows the evolution of the number of connections, for a given service, between two communicating nodes. The eventual bandwidth is deduced from the upward/downward rate of a single connection of the service.

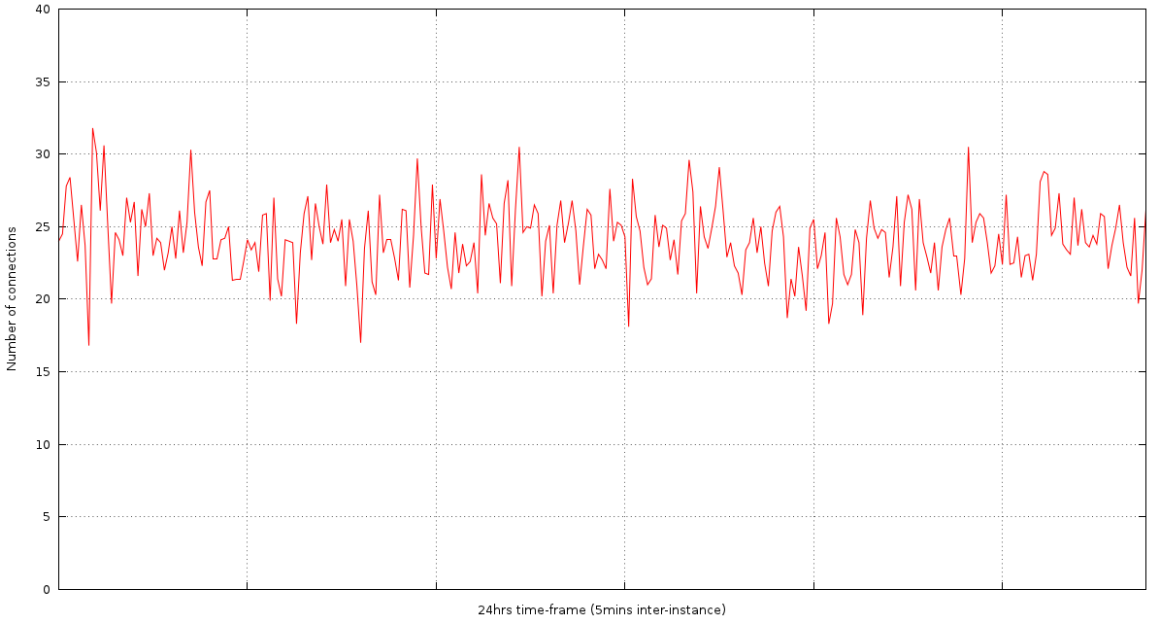


Figure 9: Evolution of the number of connections between two communicating nodes

The evolution of the set of connections between source/destination couples will induce the evolution of the traffic in the network and the loads on the link interfaces. As many services might also be enabled or disabled by the user during the day hours, congestion can occur on certain interfaces. As a result, we could obtain a maximum interface load evolution similar to that of Figure 10.

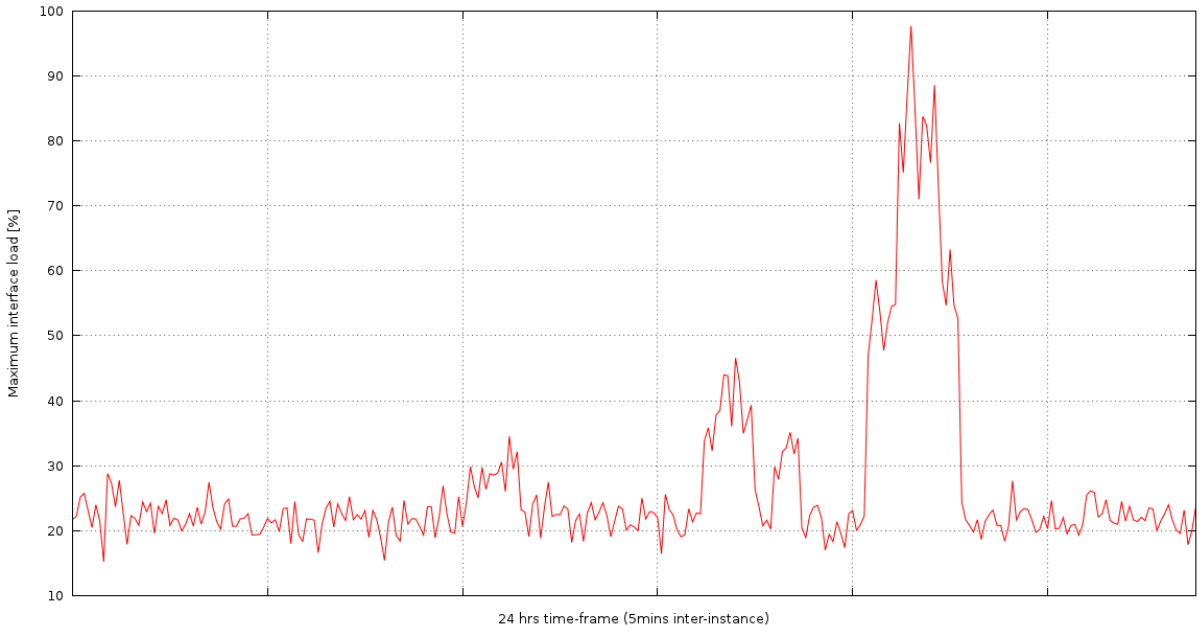


Figure 10: An example on the evolution of the maximum interface load

Once the network and the traffic matrices are created, the simulation kernel proceeds with the simulation process. This simulation sequentially iterates through the instants of the defined time frame to compute the routing tables and to forward the associated traffic in the network. Note that the considered traffic matrices are either generated as described earlier, or imported from traffic matrix logs or archives.

In computing the routing tables, the simulation kernel implements the configured protocols. It supports multiple network technologies such as IP, MPLS, or Metro-Ethernet, and well-known protocols such as OSPF, IS-IS, BGP, MP-BGP, HSRP, MPLS, and others.

Forwarding traffic in the network will abide by the implemented protocols. The simulation kernel will compute traffic routes, between source/destination pairs, according to the configured metrics and protocol parameters.

For example, in the case of OSPF, load balancing is applied when needed, and the traffic flow is forwarded along multiple routes. MPLS FECs (Forwarding Equivalence Class) are also taken into consideration. Whenever necessary, traffic is injected in the adequate LSP tunnels and routed along its paths.

The simulation kernel further considers traffic perturbations and equipment failures created by the user, as will be discussed in Section 2.2.1. For each time instant, the simulation kernel also computes key performance metrics for the overlay links, such as end-to-end delay, residual bandwidth, and even packet loss. These metrics will be used by SMART to evaluate the adequacy of the routes used for the overlay paths.

Concerning the first performance metric for an overlay link, i.e. the residual bandwidth, it will be simply the available bandwidth on its route. The kernel will evaluate the links/interfaces traversed by the overlay link and deduce the minimum available bandwidth.

As for the end-to-end delay and loss metrics, and for sake of simplicity, an analytical solution of finite M/D/1/N queues [7] was proposed and implemented for PANACEA. This model includes closed-form formulae for the steady-state queue-length distribution, for the average queue length and the average waiting time. The latter measures allow interpreting the end-to-end delay and loss through a given overlay path (the route that is constituted of a series of successive links/interfaces).

In this approach, each network interface is considered as an $M/D/1/N$ queue. The $M/D/1/N$ model is a finite capacity queuing system, with $N-1$ places in the waiting room (buffer in our case). Packets arrive according to a Poisson process at rate λ. The packets are served according to the FCFS (First Come First Served) discipline and the service time of each packet is T (a function of the interface speed and the packet size). All packets, which upon arrival, enter a full buffer system are rejected (dropped) and do not further influence the simulation. Since the finite buffer acts as a regulator of the queue size, no a priori assumption about the utilisation factor $\rho = \lambda T$ is needed. The waiting time W_N in the $M/D/1/N$ queuing system is [7]:

$$W_N = \left(N - 1 - \frac{\sum_{k=0}^{N-1} b_k - N}{\rho b_{N-1}}\right) T$$

Where $b_0 = 1$ and, for $n \geq 1$,

$$b_n = \sum_{k=0}^{n} \frac{(-1)^k}{k!} (n-k)^k e^{(n-k)\rho} \rho^k$$

Therefore, the end-to-end delay for an overlay path can be evaluated as the sum of the waiting times in the traversed interface queues in addition to the propagation delay (the time needed for the packet to propagate on the physical link between two routers, which is proportional to the distance).

As for the loss, in an $M/D/1/N$ queue it is equivalent to the probability of having a full buffer

$$P_N = 1 - \frac{b_{N-1}}{1 + \rho b_{N-1}}$$

Then, along a path, or a set of traversed interfaces Π, the total loss is as follows,

$$L = 1 - \sum_{i \in \Pi} (1 - P_N^i)$$

The simulation kernel implements the aforementioned expressions to analytically compute delay and loss metrics for the overlay paths that are used at that moment. These metrics are sensitive to the state of the network and the loads on the interfaces. High traffic loads traversing a given link/interface may greatly impact the end-to-end delay (and eventually loss) of overlay paths routed through this interface.

2.1.3 Simulation process

The simulation environment is based on the interaction between NEST's simulation kernel and SMART. After simulating the network behaviour and evaluating its key performances, the simulation kernel exchanges with SMART the needed information, i.e. the overlay link metrics, such as end-to-end delay or available bandwidths. SMART analyzes the metrics of monitored

overlay links, based on its learning algorithm, and detects whether some paths can be optimized.

As an example, after being informed of the evolution of end-to-end delay of the monitored overlay links, let us assume that SMART detects degradation in performance for a certain path between two proxies.

Then, by analyzing the overlay network, it is able to find an alternative solution to overpass this degradation. This means that it has to modify the definition of the overlay path, for instance by routing it via a third proxy. Of course, this leads to a new route definition, finally leading to a modification and the improvement of the real physical path in the network.

This modification would eventually ameliorate the end-to-end delay, as this new delay was coming from a certain network event, such as a path congestion or a router failure. In another example, SMART may detect that an alternate route of an overlay path, forwarded via a third proxy, is always better whatever the state of the physical network.

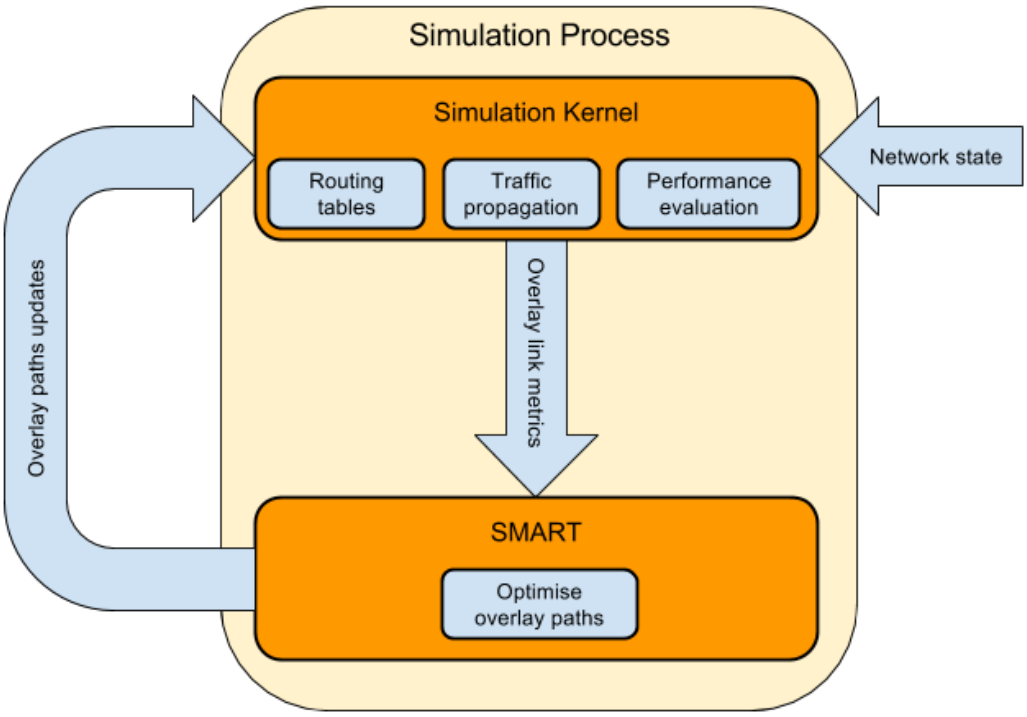


Figure 11: The simulation process

2.2 NEST ENTREPRISE evolutions

The work of Section 2.1 allowed the creation of the overlay network along with its underlay infrastructure, generating network traffic that evolve in a specified time frame, and simulating the network behaviour to deliver to SMART key performance metrics that will allow optimising the overlay paths.

On another level, NEST Entreprise was adapted to give the possibility to schedule adverse events that influence the simulation process, such as injecting equipment failures or traffic

perturbations. Features were also introduced to allow tracking the evolution of the network state and the influence of SMART decisions.

We would like to emphasize that NEST Entreprise also functions in real-time in a hypervisor context. At each instant (e.g. every 5 minutes), real data, such as interface loads, is captured from the live system in addition to SMART decisions on overlay link paths. This will allow the real-time tracking of performance in the system.

This environment allows:

- Visualising graphically the network topology, identifying key network elements such as routers and overlay nodes.
- Tracking the evolution of the network state. This includes displaying interface load evolution using adequate colour codes (based on the level of congestion). These loads are updated based on simulation results (or could be coming from real-time data using probes and network agents).
- Displaying information on the overlay links, including their sources, destinations and latest performance metrics.
- Displaying the overlay paths used for the overlay links in the network (influenced by SMART decisions). Different colours are used to identify intermediary proxies used for routing.
- Plotting the evolution of overlay path metrics while highlighting the gain in performance from implementing SMART decisions.

In the following we will describe the features integrated into NEST Entreprise for PANACEA needs. We start off with the adverse event editor, to then present several widgets that allow monitoring overlay network paths and tracking the evolution of the associated performance metrics.

2.2.1 Adverse event editor

As was mentioned in earlier sections, the simulation can be influenced by traffic perturbations or failures injected by the user. In order to achieve this, several features were added to NEST Entreprise. The Adverse event editor, shown in Figure 12, allows adding within two time intervals:

- Node failures such as the failure of routers.
- Link failures between two nodes.
- Interface failures from one node to another.
- Traffic perturbation, or in other words increase or decrease the traffic intensity between nodes for specific services.

As an example, adding an interface failure (Figure 13) is done by selecting the interface to fail and the time interval within which the failure occurs.

Figure 12: Adverse events editor

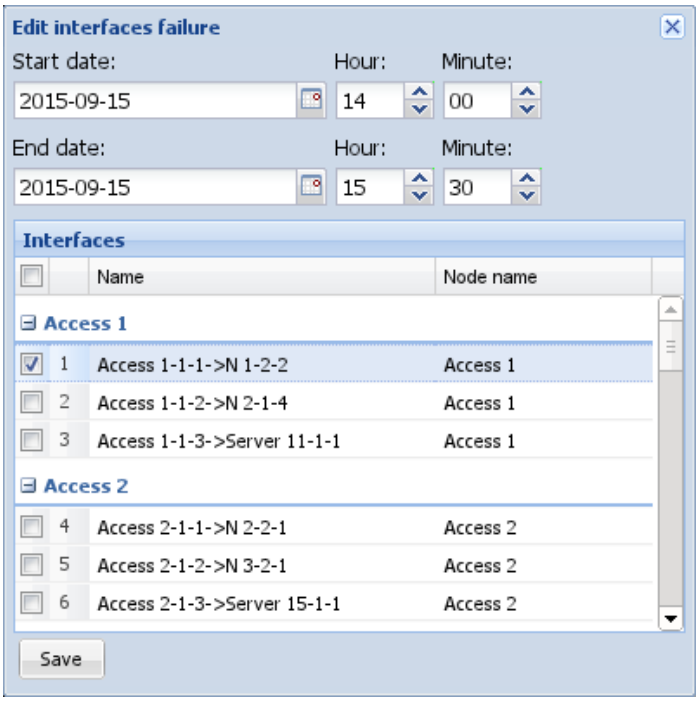


Figure 13: Adding an interface failure

The user can at any moment visualise a timeline of the configured failures and perturbations as shown in Figure 14. The user can also modify or delete any of the events.

These events are then taken into account in the simulation kernel as described in Section 2.1.2. At each time instant within the simulation, the kernel will shutdown failed equipment at that moment to exclude them from the simulation. It will also detect traffic perturbations to increase or decrease traffic volumes as specified.

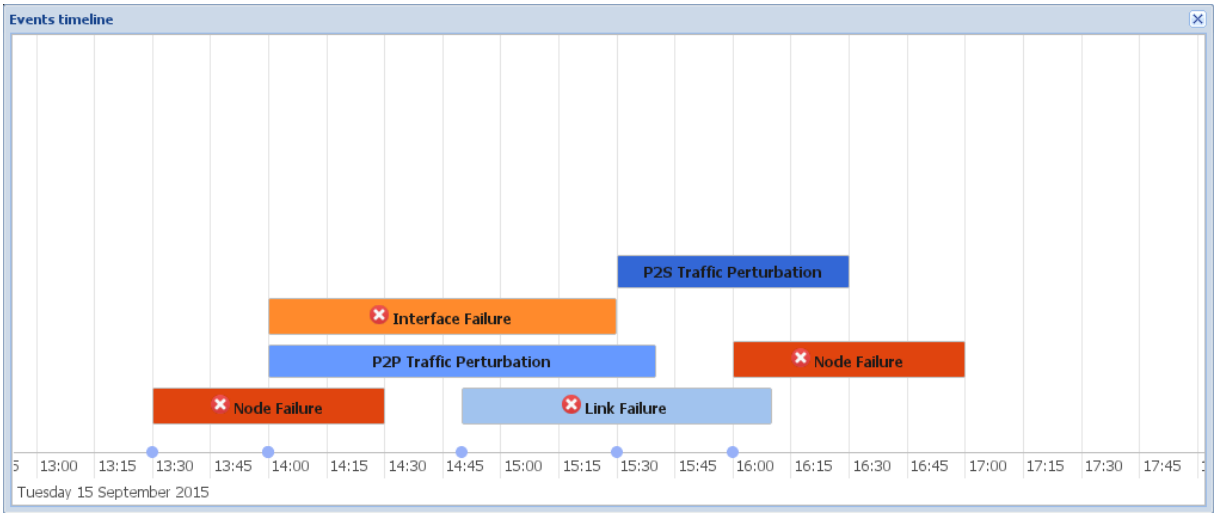
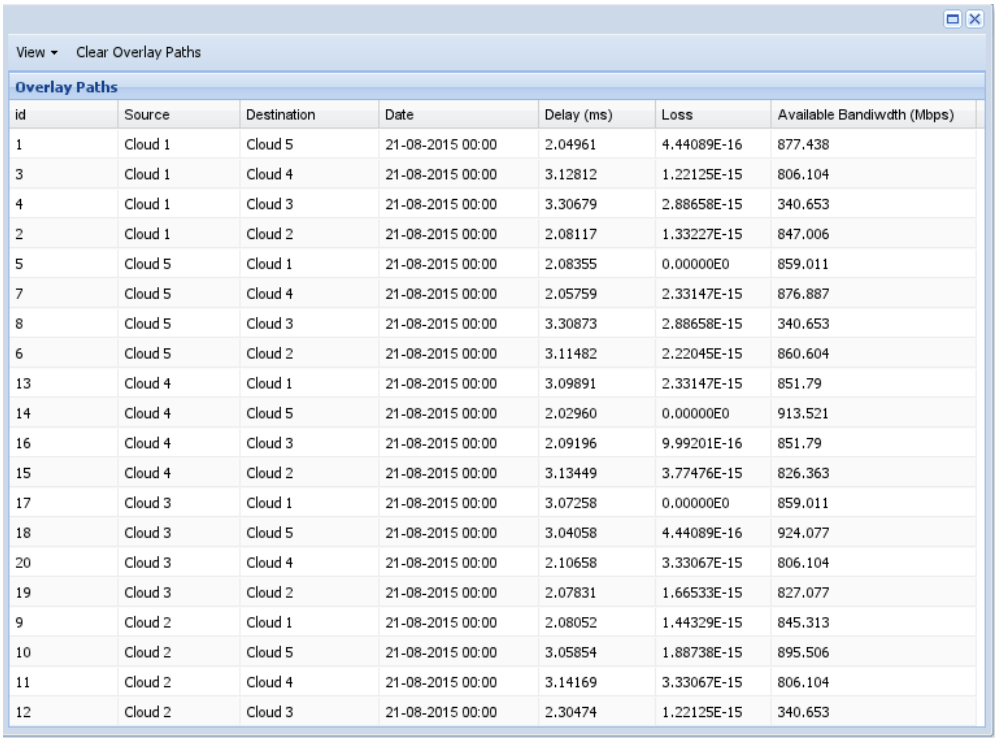


Figure 14: Adverse events timeline

2.2.2 Overlay path widget

In order to follow the evolution of the overlay paths in time, the overlay path widget allows displaying pertinent information on overlay links between proxies. The overlay paths are listed as shown in Figure 15. In addition to identifying the source and the destination, this widget allows verifying the latest performance metrics (delay, loss, and bandwidth). The date at which the metrics are updated is also indicated. The information is dynamic and is refreshed at each step.



View ▾ Clear Overlay Paths

Overlay Paths

id	Source	Destination	Date	Delay (ms)	Loss	Available Bandiwdth (Mbps)
1	Cloud 1	Cloud 5	21-08-2015 00:00	2.04961	4.44089E-16	877.438
3	Cloud 1	Cloud 4	21-08-2015 00:00	3.12812	1.22125E-15	806.104
4	Cloud 1	Cloud 3	21-08-2015 00:00	3.30679	2.88658E-15	340.653
2	Cloud 1	Cloud 2	21-08-2015 00:00	2.08117	1.33227E-15	847.006
5	Cloud 5	Cloud 1	21-08-2015 00:00	2.08355	0.00000E0	859.011
7	Cloud 5	Cloud 4	21-08-2015 00:00	2.05759	2.33147E-15	876.887
8	Cloud 5	Cloud 3	21-08-2015 00:00	3.30873	2.88658E-15	340.653
6	Cloud 5	Cloud 2	21-08-2015 00:00	3.11482	2.22045E-15	860.604
13	Cloud 4	Cloud 1	21-08-2015 00:00	3.09891	2.33147E-15	851.79
14	Cloud 4	Cloud 5	21-08-2015 00:00	2.02960	0.00000E0	913.521
16	Cloud 4	Cloud 3	21-08-2015 00:00	2.09196	9.99201E-16	851.79
15	Cloud 4	Cloud 2	21-08-2015 00:00	3.13449	3.77476E-15	826.363
17	Cloud 3	Cloud 1	21-08-2015 00:00	3.07258	0.00000E0	859.011
18	Cloud 3	Cloud 5	21-08-2015 00:00	3.04058	4.44089E-16	924.077
20	Cloud 3	Cloud 4	21-08-2015 00:00	2.10658	3.33067E-15	806.104
19	Cloud 3	Cloud 2	21-08-2015 00:00	2.07831	1.66533E-15	827.077
9	Cloud 2	Cloud 1	21-08-2015 00:00	2.08052	1.44329E-15	845.313
10	Cloud 2	Cloud 5	21-08-2015 00:00	3.05854	1.88738E-15	895.506
11	Cloud 2	Cloud 4	21-08-2015 00:00	3.14169	3.33067E-15	806.104
12	Cloud 2	Cloud 3	21-08-2015 00:00	2.30474	1.22125E-15	340.653

Figure 15: Overlay path widget

2.2.3 Topology widget

NEST Entreprise features a topology widget that allows, as in NEST IP/MPLS, to graphically display the network infrastructure along with the overlay proxies. This topology represents a network created in NEST IP/MPLS (or a real discovered one using network discovery agents). The user is allowed to zoom in or out of certain areas, modify node placement and visualise interface loads as time goes by.

In order to visualise the evolution of overlay paths in time, and in conjunction with the overlay path widget, the topology widget allows focusing graphically on a given communication link between two proxies, as that shown in Figure 16.

As the performance metrics evolve within the simulation time frame, SMART might issue a modification on the communication between the two proxies and ask a third proxy to be used to forward traffic, so as to enhance the specified performance criterion.

For example, as the path in Figure 16 might change in time, it is automatically updated and the topology is refreshed with the new route as represented in Figure 17. Here, two colours are used to indicate the segmentation of the path and the forwarding via an intermediary proxy. This modification allows the traffic to avoid certain links in the network, the ones that actually cause the degradation in performance.

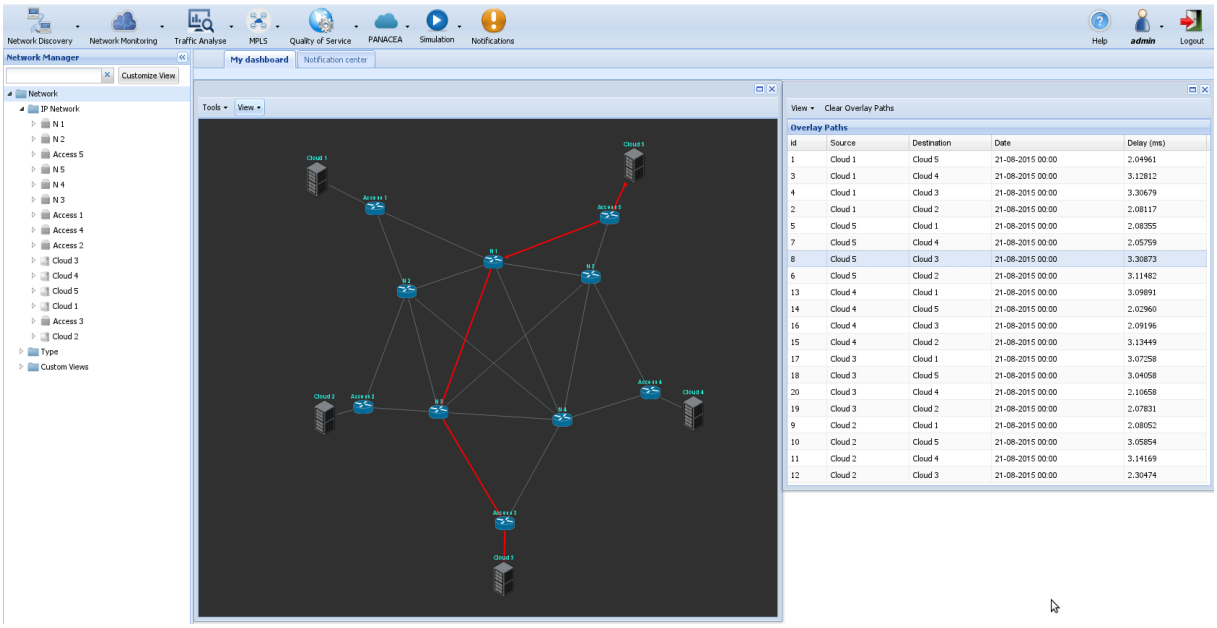


Figure 16: Visualise overlay path between proxies

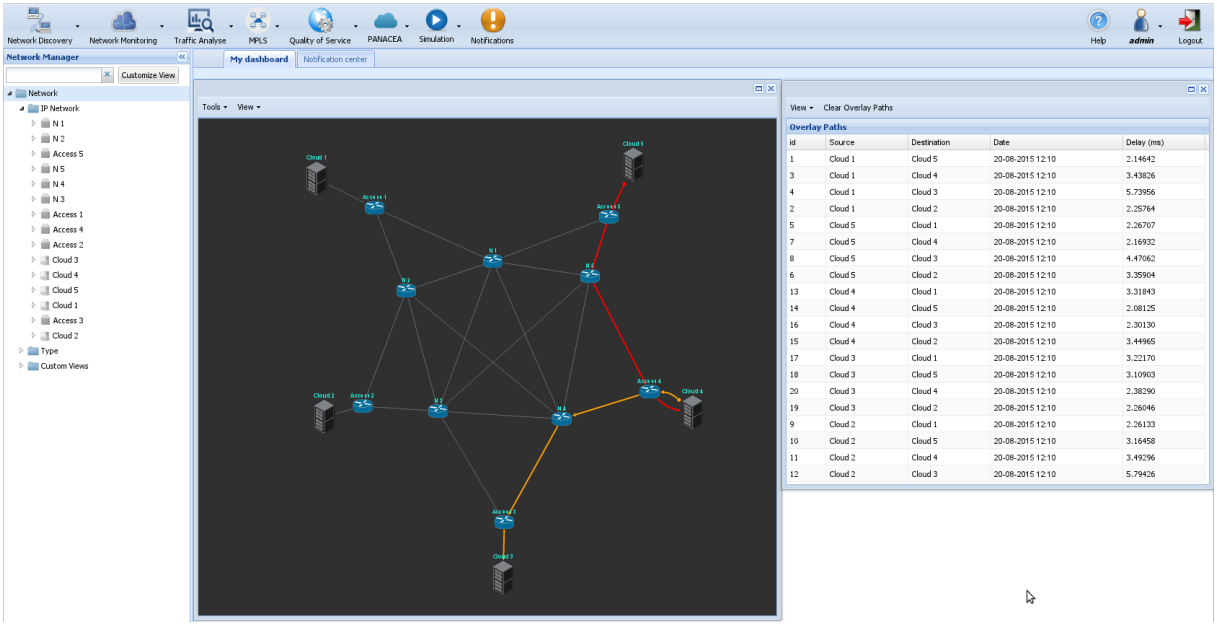


Figure 17: Modification of the overlay path and forwarding via an intermediary proxy

2.2.4 QoS performance widget

In addition to the visual representation of the overlay paths, NEST Entreprise allows plotting the evolution of overlay path metrics, such as the end-to-delay, loss, or available bandwidth.

Figure 18 represents the evolution of the end-to-end delay for a given overlay link throughout the simulation. The blue curve shows the delay that would be perceived if SMART decisions were ignored, i.e. the delay of the IP path. The green one shows the delay evolution when SMART decisions are implemented.

It is clear that at some points, the overlay path is modified so as to avoid network anomalies and to reduce the degradation in performance. In this example, the delay is maintained at 16ms instead of peeking at 28ms.

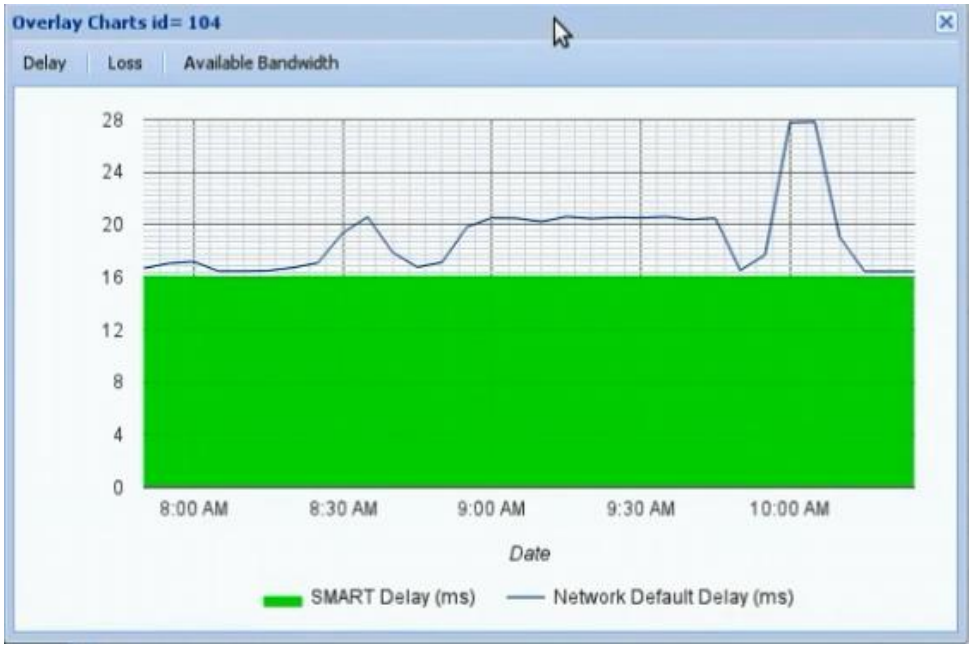


Figure 18: Evolution of end-to-end delay metric for an overlay path

Similarly, Figure 19 displays the evolution of the available bandwidth for a given overlay link. Here again, the green and blue curves correspond to the available bandwidth if SMART decisions were considered or ignored respectively. We can notice that SMART decisions lead to a great increase of the available bandwidth along the overlay path.

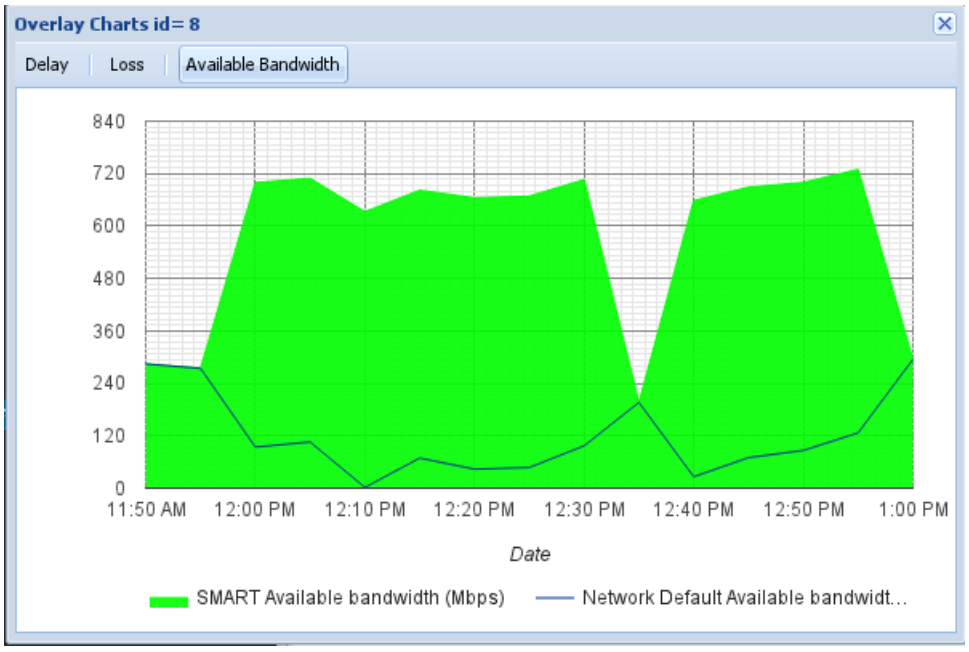


Figure 19: Evolution of available bandwidth for an overlay path

3 THE SECOND APPROACH: EMULATION

The emulation environment differs from the simulation one in that the network infrastructure is emulated and real VMs hosting real applications are used. This environment will have SMART configured and operational, meaning that we will deploy and activate on each VM a SMART proxy. The VMs hosting the real applications will be linked to the emulated environment as if they were connected to routers of a real network.

The emulation of the network is carried out using CORE emulator [1] (Common Open Research Emulator), an open-source network emulator developed at Boeing Research and Technology division.

CORE is a tool for emulating networks on one or more machines, interconnecting them, and even connecting them to live networks. CORE is typically used for network and protocol research, demonstrations, application and platform testing, evaluating networking scenarios, security studies, and increasing the size of physical test networks.

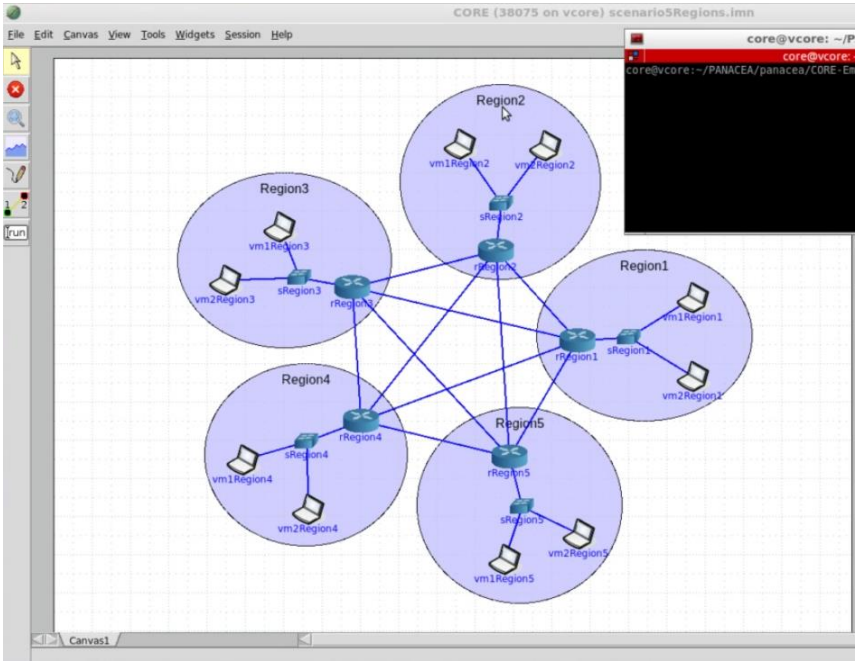


Figure 20: CORE emulator GUI

As it is not practical to inject large volumes of traffic in the emulated network (for scalability issues of doing so inside a virtual machine), the simulation process/kernel is used, in this context, to:

- Generate network traffic and inject them in a modelled version of the network
- Compute routing tables and forward traffic correspondingly
- Evaluate performance metrics
- Set link/interface parameters (such as delay) in the emulated environment to the values that correspond to the simulation (i.e. to obtain similar performances for the overlay links)
- Inject equipment failures (routers, interfaces, etc.) in the emulated network
- Exchange updates on overlay paths with SMART

This approach will allow the user to conduct stress testing and to assess the impact of the network, under chosen conditions, on the performance of real applications.

3.1 Core platform configuration

CORE is available for download in an already configured VM image. The latest VMware image incorporates version 4.7 of CORE. After some testing, it was concluded that the supplied VM is buggy as anomalies and unusual behaviour in main CORE functions were experienced.

To resolve this problem, the CORE environment had to be recreated from scratch. A lightweight VM, based on lubuntu 15.04 OS, was considered. This VM was configured with 1GB of RAM and given access to 2 CPU cores.

CORE (version 4.7) was first compiled from source. The kernel was patched and the supports for Apache, MySQL and SMART were added in the VM. CORE was further updated to version 4.8 which is more stable and includes several bug fixes.

Once a functional environment was obtained, a test network was created as shown in Figure 21. Several routers were used to constitute a network backbone. The link/interface parameters and capacities were also configured.

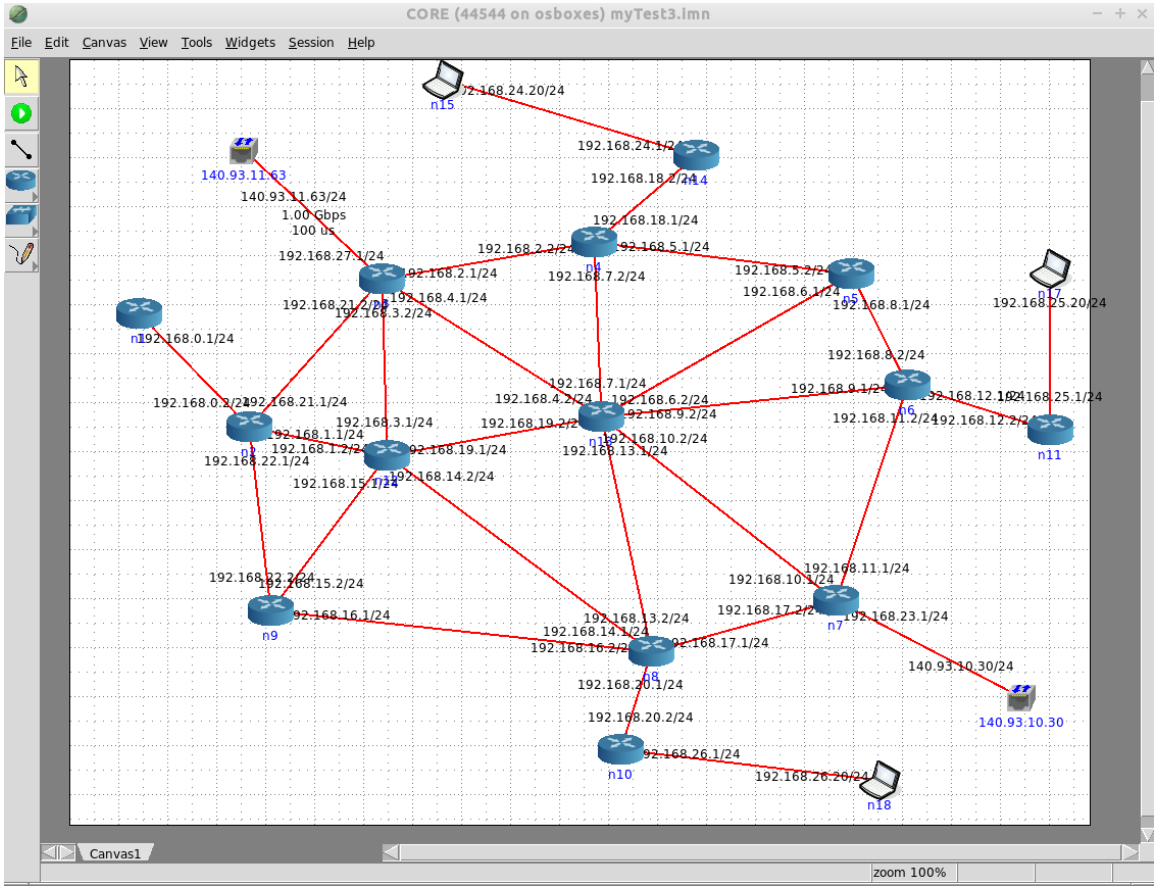


Figure 21: An emulated network created in CORE

In this example, the OSPF protocol was configured for the entire network. After running the emulation environment, the routers start by filling out and communicating routing tables. Once this initial phase is terminated, shell terminals could be launched on any of the routers. This serves to test connectivity between the network nodes (using ping or traceroute). In Figure 22,

a shell is launched on router "n9"to ping router "n11" on its interface of IP address "192.168.12.2". The intermediary hops on the route are also displayed using traceroute. This route depends on the OSPF metrics in the network.

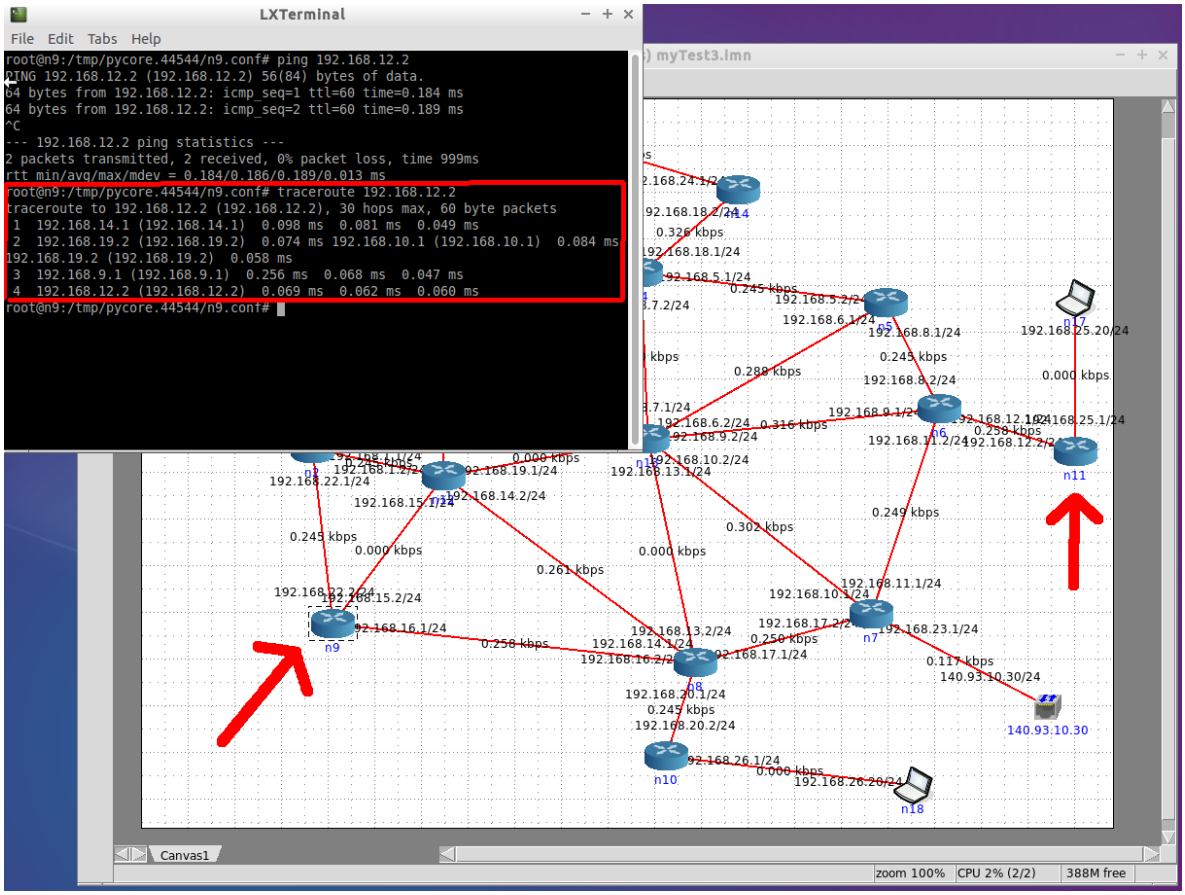


Figure 22: Testing connectivity between emulated network routers

3.2 The use of application VMs

Concerning the overlay network, CORE allows adding servers on which applications could be run. These servers will represent the overlay nodes on which specific applications will be installed. Unfortunately, the implementation of these servers was buggy and it was impossible to start Apache servers on them.

For this reason we decided to externalise the application VMs on which specified applications will be hosted (web application, TPC-W, etc.). Consequently, the CORE emulator will be used to emulate the underlay network only. The application VMs, which will have SMART proxies integrated, will be then connected to the CORE emulated network via adequate configuration.

In order to connect the application VMs to the emulated network; the tunnel tool in CORE had to be used. The tunnel tool builds GRE tunnels between CORE emulations or other hosts. The peer GRE tunnel endpoint may be another CORE machine or a host (such as Linux) that supports GRE tunnels.

To illustrate this, by referring to Figure 23, consider a VM having CORE configured and running, and another application VM acting as an overlay node, designated VM1. These VMs can be hosted on different machines. The only constraint is that they are able to communicate with

each other (depending on the network adapter configurations of the VMs).

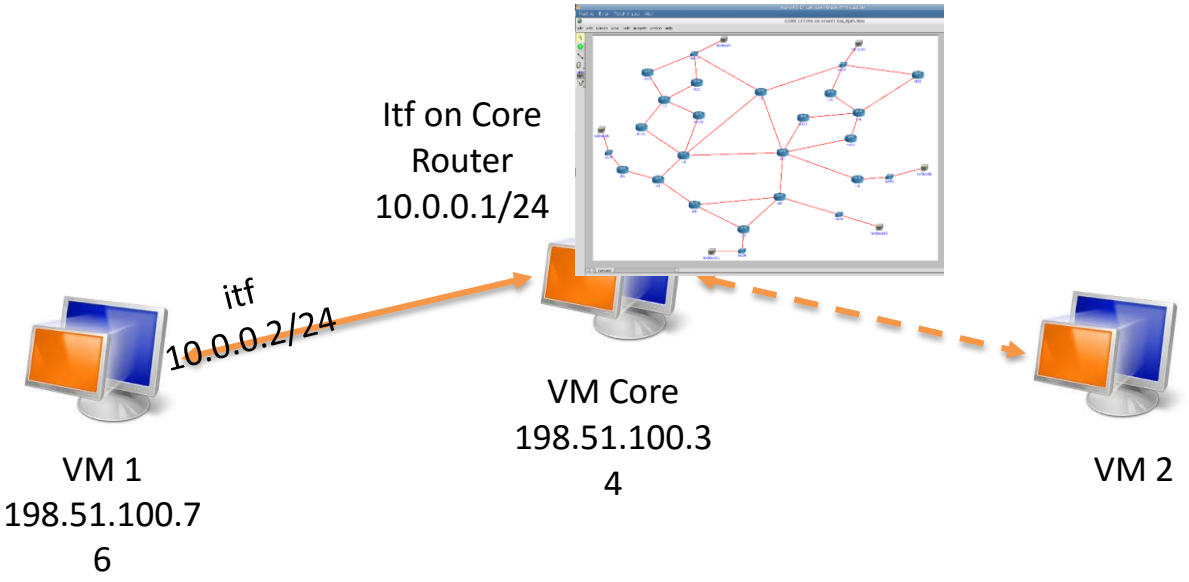


Figure 23: Connecting external VMs to CORE network

In order to connect the VM1 to a virtual router in CORE on an interface of IP=10.0.0.1/24, we should:

- Ensure that the VMs can reach each other (e.g. via ping)
- Add a tunnel in CORE and configure it with the IP address of VM1: 198.51.100.76
- Create a GRE tunnel on VM1 indicating the other endpoint, which is CORE with IP=198.51.100.34
- Create a virtual interface on the VM1 with an IP address from the subnet of the virtual router in CORE (10.0.0.x)
- Update the routing table of VM1to refer to the virtual interface that was created. This will permit reaching nodes in the emulated network

The command lines for the aforementioned configurations on VM1 are the following:

```
sudo ip link add gt0 type gretap remote 198.51.100.34 local 198.51.100.76 key 1
sudo ipaddr add 10.0.0.2/24 dev gt0
sudo ip link set dev gt0 up
sudo ip route add default via 10.0.0.1
```

To test this configuration, two application VMs were connected to a CORE emulated network as illustrated in Figure 24. Traffic generated by an http request from VM1 to VM2 was forwarded in the network. This traffic influenced the interface loads along the route between the two VMs. A real-time tracking of the interface load evolution can be visualized in CORE using a corresponding widget.

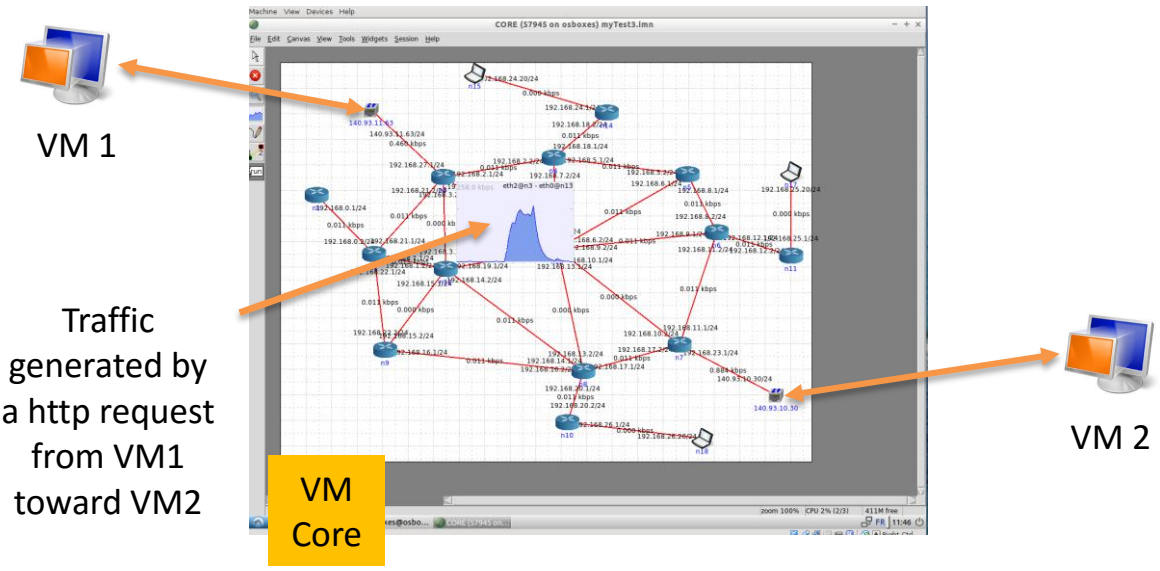


Figure 24: Sending traffic between VMs across the CORE network

Other application VMs could be connected to CORE in a manner similar to that indicated above. The aim is to configure several application VMs and connect them to an emulated network in CORE, as illustrated in Figure 25. These VMs will play the role of overlay nodes as indicated earlier.

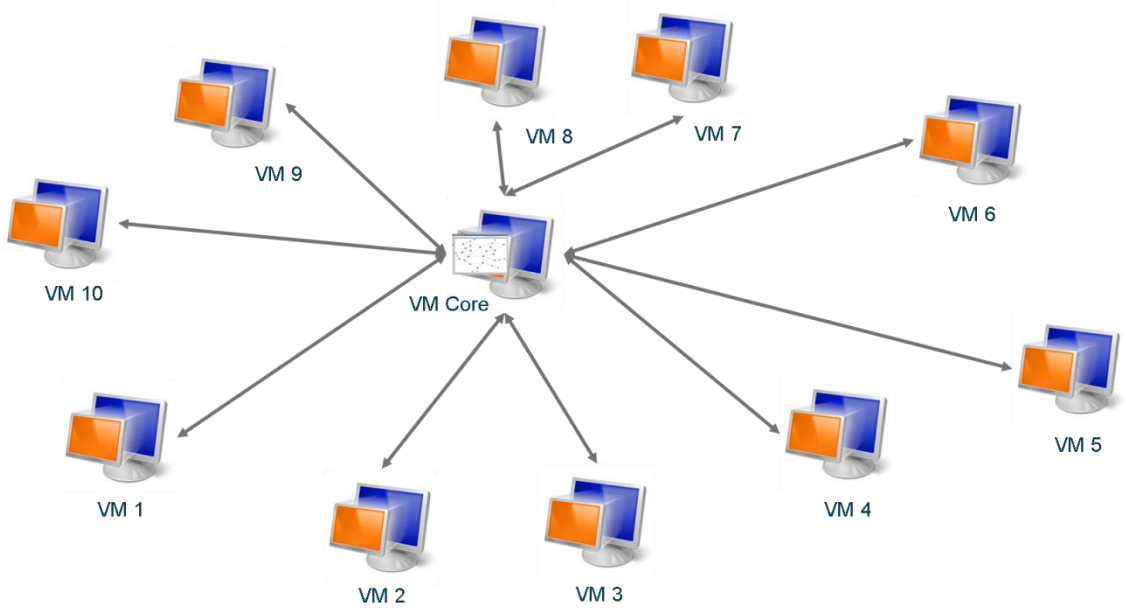


Figure 25: Several application VMs connected to the CORE emulator

3.3 Description of the emulation process

The emulation process will be based on the interaction between the emulated network, the application VMs and the simulation kernel as illustrated in Figure 26. The NEST simulation kernel will model the emulated network, generate background traffic based on user-specified

parameters, and evaluate the performance metrics. This simulation kernel will then set the interface metrics in CORE so as to correspond to the simulation results. The overlay paths between client VMs (VMC_i) and application VMs (VM_i) will be consequently influenced by these link/interface metric changes.

SMART will, whenever necessary, modify the overlay paths in a manner to enhance the overlay links performances. SMART informs the simulation kernel of such modifications.

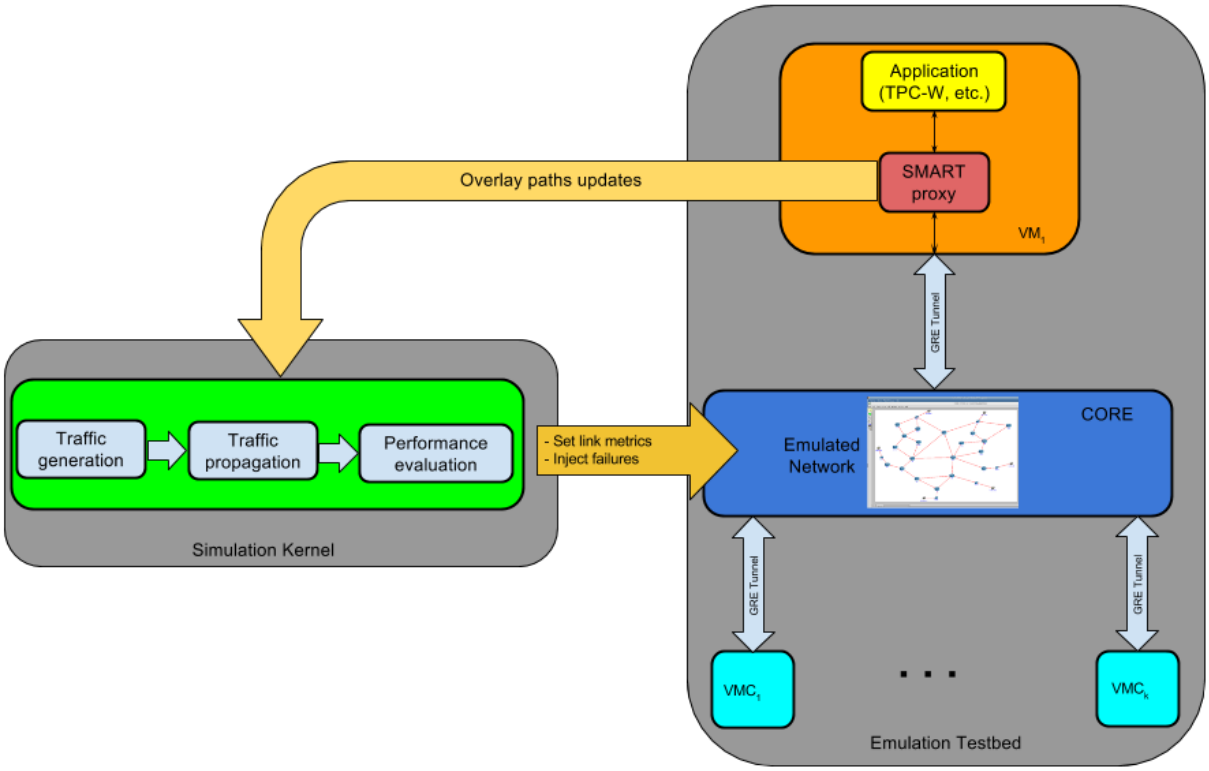


Figure 26: The emulation process

4 VALIDATION AND PRELIMINARY RESULTS

4.1 Validation process

Validation for both the simulation and emulation processes will be carried out as follows. Considering first the simulation environment,

- Traffic is generated and routed in the network.
- Adverse events are introduced (failures, traffic perturbations) during certain time-frames to degrade performances.
- The performance metrics are logged for the overlay links throughout the simulation. This is done while disregarding SMART decisions on one hand, and implementing them on the other.
- Gain in performance is then compared between the two situations and the significance of using SMART is highlighted. We will consider
    - the end-to-end delay metric which will influence the response time perceived by the users, and
    - the available bandwidth on the overlay link which will influence the volume of requests that could be served.

As for the emulation environment,

- The application VMs are configured with clients that will inject http requests towards a web service.
- The number of requests is set so as to obtain a low network utilisation.
- During a given time-frame, and while deactivating SMART, we increase the number of requests to generate congestion and degrade performances. Significant increase in the end-to-end delay (response time) is, for example, detected.
- Now activating SMART under the same conditions, we should demonstrate the improvement of response times.

In particular, in the context of Use Case 1, we will consider the checkout phase of an e-commerce site. After adding products to his cart, the customer will proceed to checkout to finalize his order. It was shown that 57% of shoppers abandon the site and cancel their orders after waiting longer than 3 seconds [8, 9]. Among them, 80% will never return to the aforementioned site.

We can notice here that the response time is an important factor to be considered by e-commerce sites. The faster the response, the more the customer is satisfied.

This delay is usually affected by two factors:

1. The delay of request treatment by the server(s)
2. The end-to-end network delays

In our use case, we will assume that the delay introduced by the server(s) is 2.7 seconds. This will mean that the network delay has to at most 300 milliseconds. In the experiments, the simulation kernel will be configured to inject traffic that lead to an average of 250 milliseconds end-to-end delay on some overlay links (between the application VMs).

At some point, a certain perturbation will be introduced to cause an increase in this delay, say 80 milliseconds, so that the aforementioned 3 seconds are exceeded. SMART should propose an alternate overlay path that reduces this delay and keep the overall response time (including server delays) under 3 seconds.

Figure 27: Checkout phase in e-commerce

4.2 Simulation results

Within the context of the aforementioned use case, some experiments were conducted on the network example of Figure 3. Traffic matrices were generated over a 24 hour period, along with some traffic perturbations, to obtain the maximum traffic load shown in Figure 28. We can clearly notice peek traffic midday.

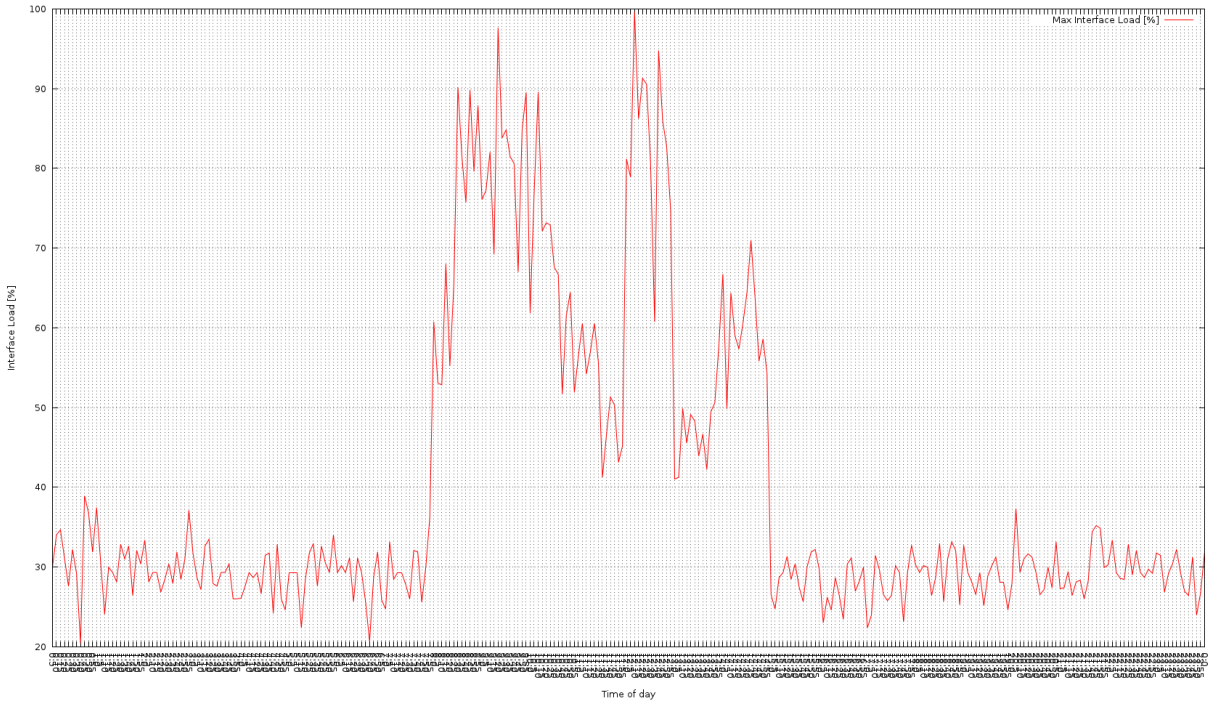


Figure 28: Maximum interface load for the test network

In idle conditions, the average load in the network is around 30%, whereas it goes up to around 80% or 90% in peak hours. Figure 29 demonstrates the network interface load state at peak time and gives the three mostly loaded interfaces (purple indicates the heaviest load).

These interfaces will certainly impact the performance metrics of the overlay links that traverse them. Consequently, SMART should detect this deterioration in performance and investigate alternate overlay paths.

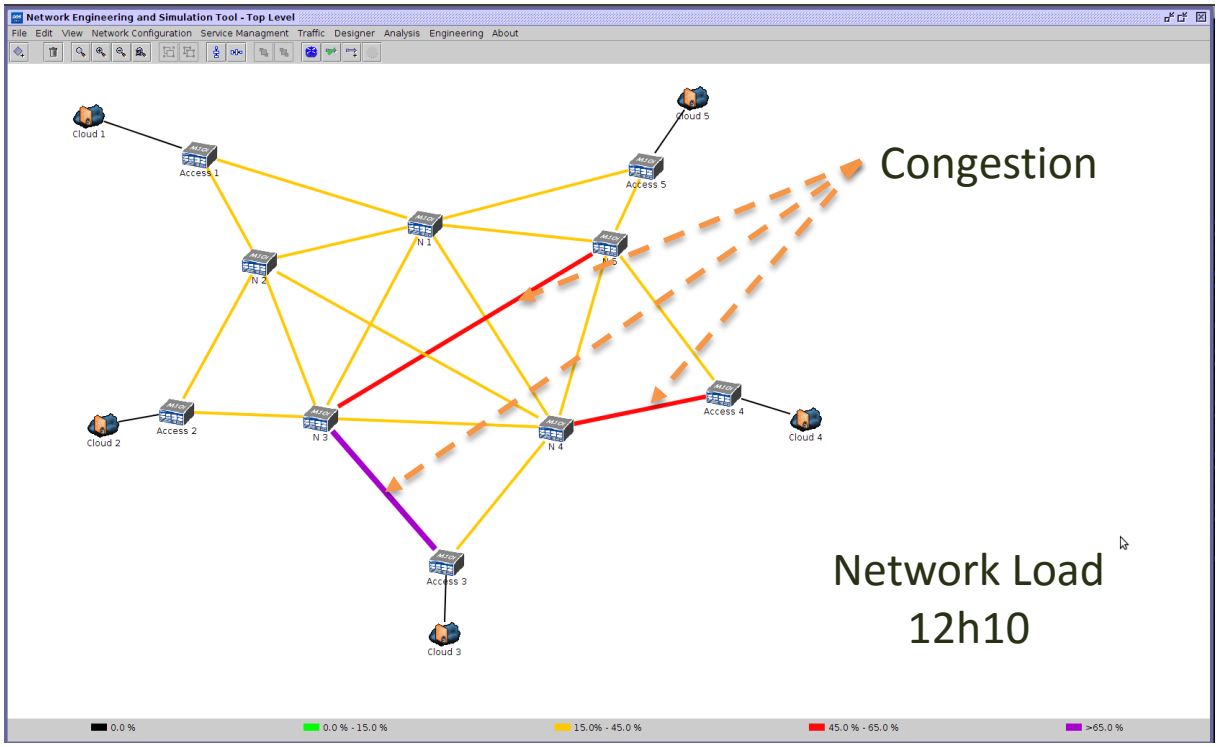


Figure 29: Interface loads at peak hours

In a first setting, SMART is configured to optimise the end-to-end delay metric. The objective here is to keep the end-to-end network delay below 300 milliseconds for the overlay links, as was discussed in the Use Case 1 example earlier. To illustrate SMART interventions, we will focus on the evolution of the end-to-end delay on the overlay link "Cloud 5 -> Cloud 3".

In idle conditions, this overlay link is routed through the default (usual) network path illustrated in Figure 30. It is clear here that this overlay link passes by default on an interface/link that is heavily loaded during peak hours.

Figure 31 demonstrates the end-to-end delay evolution for the overlay link throughout the simulation. The red curve represents the delay on the default network path between "Cloud 5" and "Cloud 3", whereas the blue one represents the delay on the path as proposed by SMART.

It is clear that SMART intervenes when necessary and proposes an alternate path to minimize the delay. At noon, as the delay rises to around 314 milliseconds, SMART reduces this delay to around 284 milliseconds by using an intermediary proxy (cloud) for the path routing.

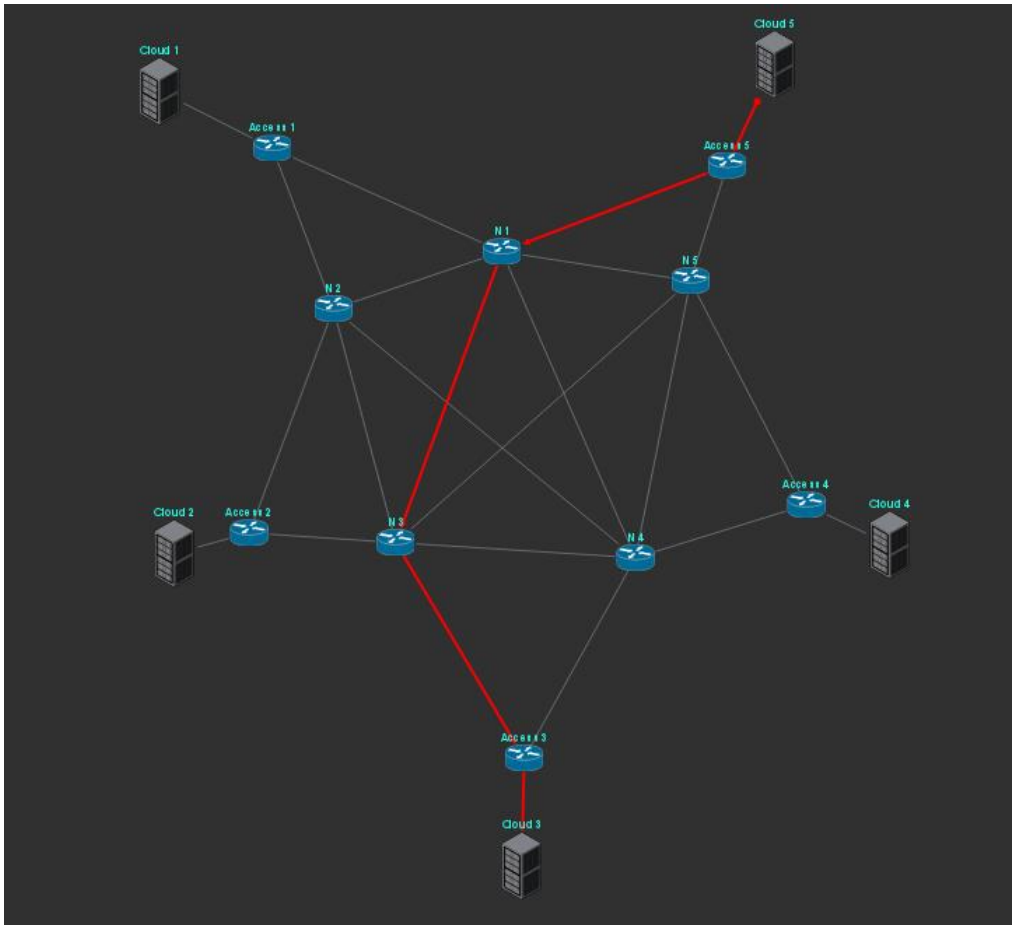


Figure 30: Default network path for the overlay link "Cloud 5 -> Cloud 3"

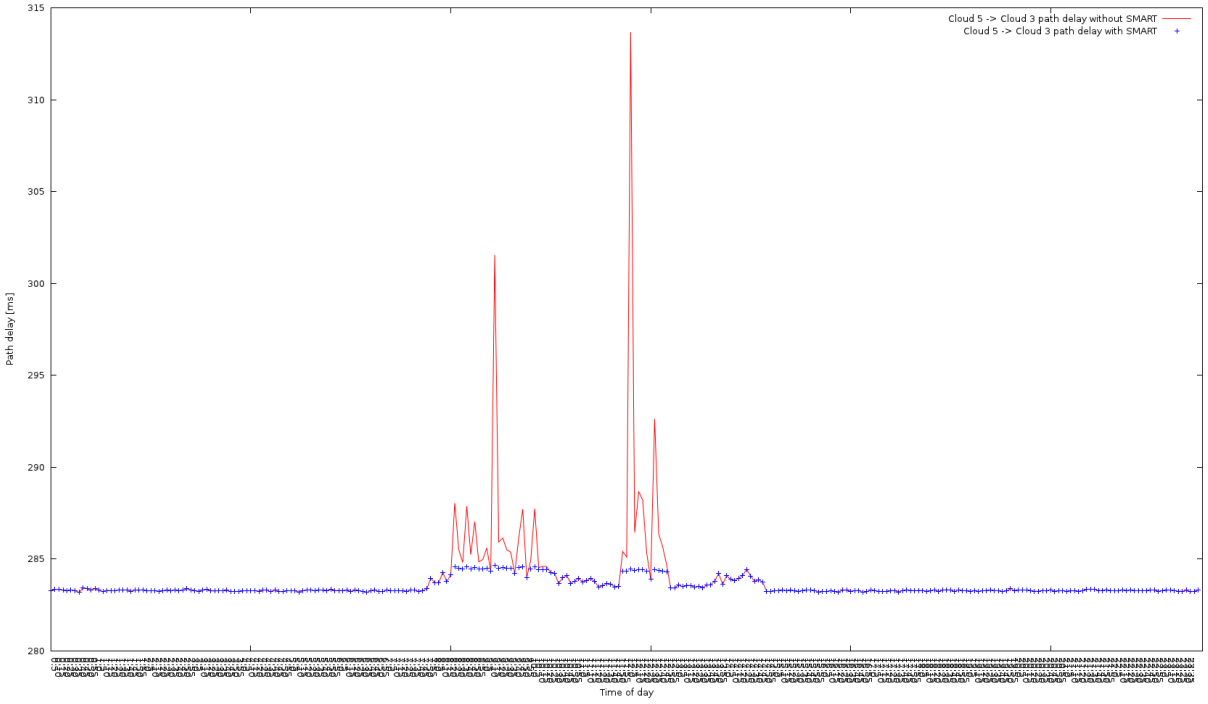


Figure 31: End-to-end delay evolution for the overlay link "Cloud 5 -> Cloud 3"

In order to reduce the delay below the critical threshold of 300 milliseconds, SMART proposed an alternate overlay path that avoids the heavily loaded link. This path is forwarded via "Cloud 4" as shown in Figure 32.

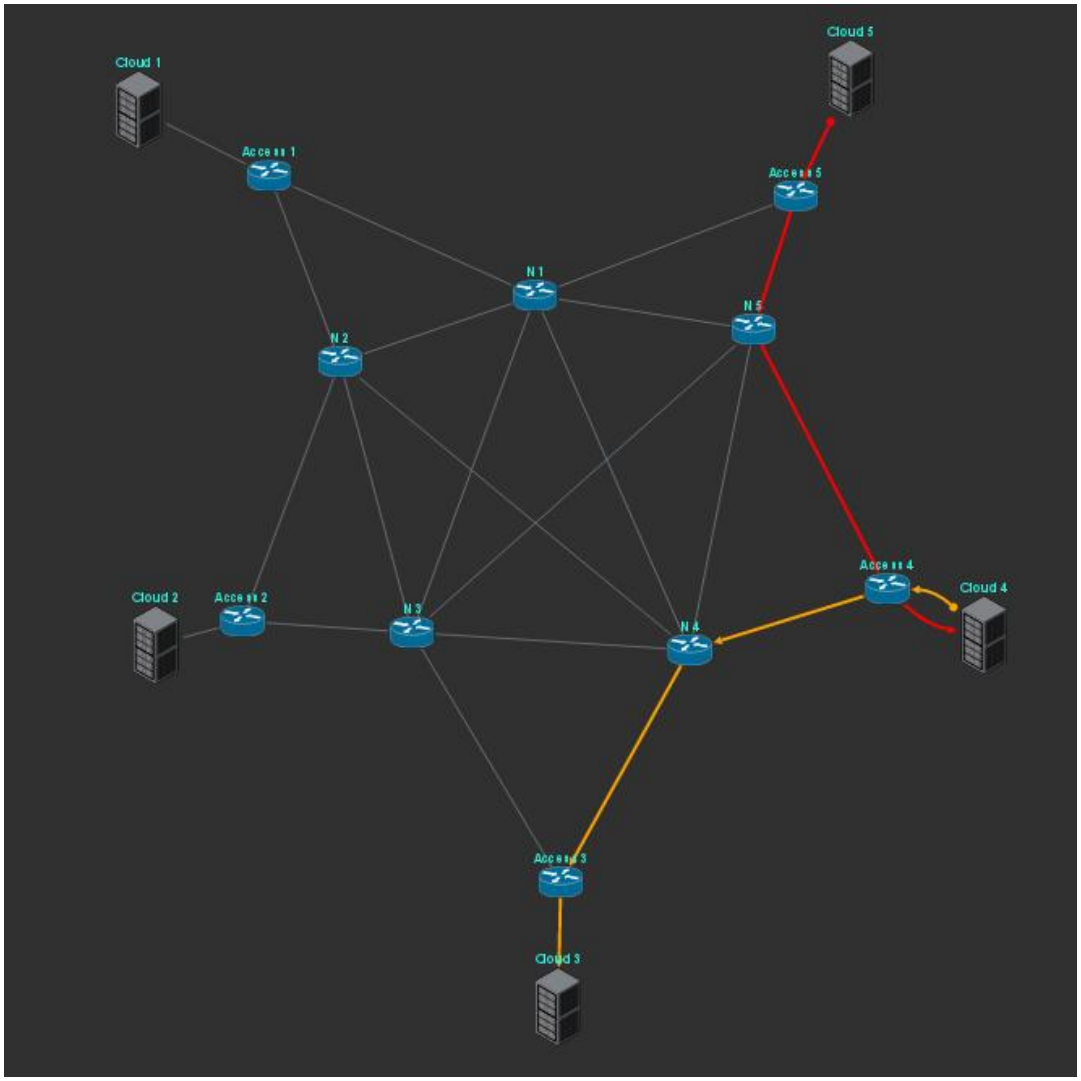


Figure 32: Overlay path proposed by SMART

In a second setting, SMART is configured to optimize the available bandwidth along the overlay path. Considering the same overlay link shown earlier, Figure 33 illustrates the evolution of the available bandwidth on the overlay link throughout the simulation.

Similarly here, the blue and red curves represent the evolution with and without SMART decisions respectively. The overlay link is never optimal along the default network path. SMART will propose an alternate path (passing through "Cloud 4") that provides a higher available bandwidth (an increase between 500 and 600 Mbps).

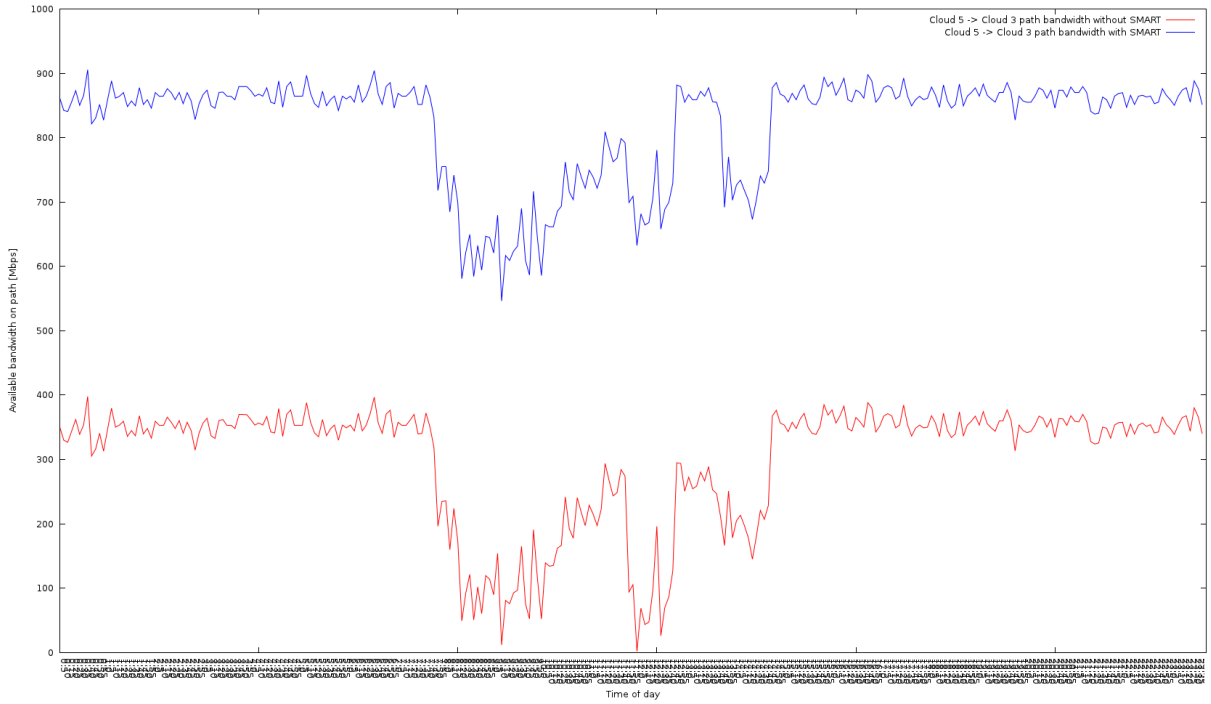


Figure 33: Available bandwidth evolution for the overlay link "Cloud 5 -> Cloud 3"

5 CONCLUSION AND FUTURE WORK

In this deliverable we have demonstrated the work done for the simulation and emulation environments. QoS Design's NEST IP/MPLS and Entreprise environments were enriched with features to adapt to PANACEA objectives. A simulation kernel was also developed for this matter. This kernel is responsible for network routing simulation, traffic propagation, and performance evaluation.

Implementing complex and real-world protocols (OSPF, BGP, MPLS, etc.), it carefully evaluates the network behaviour while considering user-defined events (failures, perturbations) and computes key metrics which influence SMART decisions on the overlay paths selection. NEST Entreprise will allow the monitoring of the network state and the evolution of overlay paths in time, along with their performance metrics.

The simulation environment is operational and ready to carry out desired experimentations. Preliminary results were obtained on a test network. We started by creating the desired underlay infrastructure and the overlay network. Traffic was then generated over a specified time-frame based on configured parameters (sources, destinations, services, distributions, etc.).

The routing and traffic forwarding were simulated to calculate performance metrics for the overlay links. These metrics were monitored by SMART to propose alternate overlay paths whenever necessary, to overcome performance degradations and provide gain in performances (reduce the response time for example).

Further work on the simulation environment includes selecting a more elaborate scenario and performing further experimentation to validate SMART decisions. Pre-defined network anomalies such as failures and traffic perturbations will be introduced to modify the network state. SMART decisions on overlay paths will then be tracked to highlight their gain on overlay links performances.

As for the emulation environment, it was prepared and various configurations were achieved. We have considered emulating the underlay network using the CORE emulator. We have also prepared application VMs which are able to host real distributed applications (such as TPC-W) and have SMART configured.

These application VMs will be connected to the emulated network using GRE tunnels. The aforementioned configuration was tested successfully. Multiple VMs where connected to a test CORE network and http requests were injected toward a web service. The traffic flow was then verified on the emulated network interfaces.

The emulation environment is now ready for further experimentation. We will first focus on running tests in the context of Use Case 1 of the project. In this use case, we will consider an e-commerce checkout example where customers tend to abandon their operation after a certain waiting time, as discussed in Section 4.1.

The simulation will, in this context, generate a nominal traffic matrix for which, related response times are observed below the given threshold (3 seconds). At some points, perturbations will be introduced to degrade the network delay, leading by such to a QoS violation. SMART decisions will be monitored to verify if they succeed in providing solutions to remedy this degradation.

Following the experimentations on Use Case 1, we will focus on Use Case 2. This work will be carried out by a collaboration involving QoS Design, LAAS-CNRS and Atos.

6 REFERENCES

[1] J. Ahrenholz, C. Danilov, T.R. Henderson, and J.H. Kim. Core: A real-time network emulator. In Military Communications Conference, 2008. MILCOM 2008. IEEE, pages 1-7, Nov 2008.

[2] Olivier Brun, Josu Doncel, and Christopher Thraves Caro. Shortest path discovery problem, revisited (query ratio, upper and lower bounds). Research report, LAAS-CNRS, October 2014.

[3] C. Labovitz, R. Malan, and F. Jahanian. Internet routing instability. IEEE/ACM Transactions on Networking, 6(5):515-526, 1998.

[4] Craig Labovitz, Abha Ahuja, Abhijit Bose, and Farnam Jahanian. Delayed internet routing convergence. SIGCOMM Comput. Commun. Rev., 30(4):175-187, August 2000.

[5] T. Leighton. Improving performance on the internet. Communications of the ACM, 52(2), February 2009.

[6] V. Paxson. End-to-end routing behavior in the internet. In in Proc. ACM SIGCOMM'96, pages 25-38, Stanford, CA, USA, August 1996.

[7] O. Brun and J.M. Garcia, Analytical solution of finite capacity M/D/1 queues, Journal of Applied Probability, pp. 1092-1098, 2000.

[8] What Leads to eCommerce Checkout Abandonment? http://savvypanda.com/blog/intermediate-level/what-leads-to-ecommerce-checkout-abandonment-infographic.html.

[9] State of the Union for Ecommerce Page Speed and Web Performance http://www.webperformancetoday.com/2015/04/15/new-findings-state-union-ecommerce-page-speed-web-performance-spring-2015/.

