Seventh FRAMEWORK PROGRAMME
FP7-ICT-2009-5 - ICT-2009-1.6
Future Internet experimentally-driven research

SPECIFIC TARGETED RESEARCH OR INNOVATION PROJECT

Deliverable D4.2

Experimental methodology, scenarios, and tools

Project description	
Project acronym:	**EULER**
Project full title:	**Experimental UpdateLess Evolutive Routing**
Grant Agreement no.:	**258307**

Document properties	
Number:	**FP7-ICT-2009-5-1.6-258307-D4.2**
Title:	**Experimental methodology, scenarios, and tools**
Responsible:	**Universite Pierre Marie Curie (UPMC)**
Contributor(s):	**Alcatel-Lucent Bell (ALB)**
	Institut National de Recherche en Informatique et en Automatique (INRIA)
	Interdisciplinary Institute for Broadband Technology (IBBT)
	Universitat Politècnica de Catalunya (UPC)
	University Pierre and Marie Curie (UPMC)
Dissemination level:	**Public (PU)**
Date of preparation:	**July 2012**
Version:	**1.1**

List of authors.

Affiliation	Author(s)
INRIA	David Coudert, Christian Glacet, Luc Hogie, Aurélien Lancin, Nicolas Nisse
ALB	Dimitri Papadimitriou
UPMC	Fabien Tarissan, Matthieu Latapy, Daniel Bernardes, Amélie Medem
IBBT	Wouter Tavernier
UPC	Jordi Perellò Muntan

Abstract.

This deliverable describes the experimental evaluation methodology that will be followed and the experimental scenarios that will be executed in tasks T3.3 and T4.3 to realize simulations and emulation-based experiments, respectively. The objective in specifying a systematic experimental methodology is to ensure repeatability, reproducibility, and verifiability of the simulation and emulation results. While defining the experimental methodology, ensuring that the settings and running conditions and accordingly the measurement procedures are reproducible or equivalent conditions can be recreated and/or translated between evaluation cases guarantees the validity of our experiments. The routing models and associated procedures proposed in WP2 task T2.2 will be experimented on representative topology settings as modeled following the results of WP3, tasks T3.1 and T3.2.

This deliverable also provides a detailed description of the experimentation an measurement tools that will be used to realize simulation experiments (as part of task T3.3) and emulation experiments (as part of task T4.3). For each tool, we further document the functionality, the properties but also the know and/or discovered limitations together with the assessment on how they can or cannot be utilized to realize the planned simulation/emulation experiments. According to this assessment any required modification or customization to be further realized is further documented.

Contents

Chapter 1

Introduction

In the context of the EULER project, several routing models and algorithms scoping applicability to the Internet are designed and developed. The range of routing paradigms the project investigates is relatively large: it covers a spectrum ranging from dynamic compact routing to geometric routing and its multiple variants, e.g., updateful and updateless. Further, by relying on the results of topology modeling investigations performed in WP3, these paradigms are expected to take benefit of the statistical and structural properties of the Internet topology and better characterization of its dynamics. Ultimately, the resulting routing paradigms would lead to distributed and dynamic routing schemes that are specialized for the Internet while taking into account its short-term dynamics and its long-term continuous evolution.

The aim of this document is to describe the common experimental methodology that will be followed by project partners for evaluating the behavior and the performance of the routing schemes through numerous experimentation. These experiments comprise both simulation and emulation, which play here a complementary role. Indeed, each of them has its own advantages and drawbacks. Simulations (in particular, simulation by discrete event) is well suited for experiments involving spatial metrics, structure, and dimensions as simulation enables handling of large-scale topology cases and produces results that are easier to tune, reproduce, and compare. On the other hand, emulation has the advantage of being more realistic thus providing valuable insight in temporal metrics and behaviour (although on smaller scale and at a higher cost). Moreover, emulation experiments enables to check the realism of simulation results and simulation experiments to extend the applicability of emulation results. Indeed, results obtained by means of simulation and emulation experiments are complementary when the experimental scenarios are commonly specified and their execution adapted (without introducing any bias) to the simulation or emulation environment.

The experimental methodology reported in this document will allow i) measuring and analyzing the functional and performance metrics described in deliverable D4.1, and ii) evaluating the results obtained against the criteria described in deliverable D4.1. The described methodology shall lead to verifiable, reliable, repeatable, and reproducible simulation and experimental results. For this purpose, a set of scenario for conducting simulation and experimentation is documented. Both static and dynamic scenarios are considered, where dynamics includes network topology growth, situations of single and multiple topology failures as well as routing policy changes. These scenarios rely on the models reflecting the Internet topology as produced within task T3.1, and on measurement-based topology modeling performed within task T3.2. Specific classes of network topologies will also be used for stressing the routing schemes in both static and dynamic contexts. Next, this deliverable presents the documentation of the tools that have been developed within task T4.2 in order to perform our investigations. These tools include graph libraries allowing to analyze efficiently network topologies, the implementation of several models of graphs, and a dedicated network simulator for investigating on the routing schemes.

This document is organized as follows. The experimental methodology is presented in Chapter 2. Then, in Chapter 3, we present the set of scenarios that will be used to conduct simulation and emulation experiments of routing schemes and their corresponding procedures. Next, in Chapter 4, we document in Section 4.1.1 the *DRMSim* network simulator. In the Section 4.1.2 of this chapter, we describe the *Grph* graph library tailored to network simulation and graph analysis. We also document several tools used for specific purpose by project partners and to the development of which project partners contribute. These tools includes the *Sage* open-source mathematics software system (Section 4.4.1 page 37).

Chapter 2

Experimental methodology

The EULER project strongly relies on the use of experiments to evaluate the functionality and the performance of the routing schemes designed as part of WP2. The specification of these experiments therefore is crucial, and it must follow a rigorous, systematic, and effective methodology. This chapter discusses these key methodological questions raised by such experiments, the details of which being defined in the next chapters.

2.1 General considerations

Our experiments rely on scenarios, which we first discuss. Given a scenario, one has to define appropriate parameters for running it, which we present after that. Size of instances used in experimentations being a key issue, we discuss that next. Finally, two key concerns are presented, namely robustness of observations and performance evaluation.

2.1.1 Scenarios

In the context of our project, an experiment consists in simulating or emulating a running network routing a given traffic demand. The key components of such experiments are the network topology (which may be modeled as a simple graph or a much more detailed topology with node types, link capacities, etc), a routing scheme (which defines how the traffic demand is handled by the network) and a traffic demand (which nodes sends data to with other(s)).

Each of these components may be modeled in many ways, at very different levels of abstraction, with key impact on the experiments we are able to run (for instance, if the topology is very details, then simulations cannot be very large) and on their outcome (different topology/routing schemes/traffic demands will lead to different results).

We will detail which models we consider for each of these components in the next sections. From a methodological point of view, the key point is that we have a set of models for each of the three components, and that in principle any combination (triple) of models may be considered.

2.1.2 Choice of model parameters

Once a choice of three models for topology, routing scheme and traffic demand is made, thus defining a scenario, there are still difficult choices to make before being able to run an experimentation: one has to decide each parameter of each chosen model.

Choosing appropriate parameters for a single model is already a difficult task. As a consequence, choosing appropriate parameters for all models in a scenario may be extremely challenging. Indeed, it is impossible to scan the wide variety of possible values for parameters of these models (their combination induces an exponential inflation of possible choices). Instead, some decisions must be made for reducing the number of possible choices.

As our aim is to run *realistic* experiments which reflect how routing would really perform in real settings, we take benefit of the measurement-based nature of EULER project to choose a reasonable number of appropriate parameters.

A typical example is topology modeling: while there exists a wide variety of topology models, from Erdös-Réníy random graphs to bottom-up models of network components and their interconnection, each with its own set of parameters, we will use information from actual measurements to choose these parameters. Current state-of-the-art is to rely on properties observed from maps, and we will follow this approach when appropriate (or when no other choice is possible). Going further, we conduct in EULER measurements targeting the accurate estimation of specific properties, like the physical degree of core routers (*i.e.* their number of interfaces). As many models use prescribed degree distribution as a key parameter, this knowledge allows us to choose the appropriate degree distribution (without having to try many different distributions).

This approach is very general and central in EULER experimentations: although it remains far from complete, we have an unprecedented knowledge of actual properties of the Internet topology, routing schemes and traffic demand. As a consequence, we may choose appropriate values for model parameters without having to try wide ranges of possible choices, which is essential for limiting the number of experiments to be conducted without reducing their outcome.

2.1.3 Size of simulations/emulations

A key parameter for experiments is the size of involved objects: number of nodes and links in considered topologies, amount of traffic, etc. Also, as experimental approaches result in outputs which depend on the particular instances considered, one must repeat experiments with different instances to study their robustness; then, choosing an appropriate number of instances may be difficult.

Obviously, as our experiments aim at studying the behaviour of routing protocols in the Internet environment, the best choice is to run them in settings reflecting this environment (and its evolution). As a consequence, topologies should have a size comparable to the one of the Internet, or at least comprise a sufficiently representative number of AS, traffic should reflect actual demand on the Internet, etc. This means that one may consider millions of nodes and users, or even more.

Unfortunately, this is not reachable for many experimental tools. Only the most simple ones (like graph generators) are able to handle such sizes, most being very limited (sometimes to a few hundred nodes, like in the case of detailed emulation tools). Even then, simulating traffic demand from many nodes to many others leads to untractable computations. It is therefore crucial to handle size issues carefully in our experimentations.

The first situation is when one has to choose a size of objects involved in experimentations (topologies and traffic demand, mainly). Current best practice is to use the largest possible objects, in the hope that the behavior observed using them will reflect their behavior on the Internet in a reasonable way. This is not sufficient however, and in EULER we will systematically study the impact of object sizes on the outcome of each experiment: we will run experiments with different object sizes and observe how results vary. This will provide accurate information on the impact of object size, and we may have to extrapolate observations to infer the results we would obtain with larger objects (and the Internet) if we were able to conduct experimentations at such large scales.

Another key issue is the study of robustness of observations with regard to the particular instances used in experimentations: observations may vary much depending on the specific topology or traffic demand considered, as well as arbitrary choices made during routing. It is crucial for EULER to both estimate the typical behaviors and how far from these typical situations one may be in practice. To experimentally explore this, we will run several instances of experiments with the same parameters and study the distribution of results. We may in particular distinguish situations where the results are homogeneous (all results are close to a typical situation) and the situations where they have huge variability. This in itself is a criterion for performance of the considered setting (its predictability).

2.1.4 Performance evaluation

Once the methodology for conducting experimentations is described, we still have to define a methodology for performance evaluation. Performance evaluation involves the specification of a performance model, performance measurements, and performance results analysis.

1. *Performance model*: the specification of a performance model defines the significant aspects of the way in which a proposed or actual system operates in terms of resources consumed, accessed, scheduled, etc. together with the various delays by processing and/or physical/hardware limitations (such as speed, bandwidth of communications, access latency, etc.). A performance model provides useful information on how the proposed vs BGP routing system/model will or does actually work. Based on the information contained in the performance model, the interpretation of the execution of the routing process model (by means of simulation or emulation) provide further insight into the routing system's behavior, and can be used to identify where the routing model design is inadequate.

2. *Performance measurement*: many performance metrics may be used for this purpose, like routing path stretch, routing table size (per-node memory space required to store routing table entries), computational complexity of the routing algorithm, routing paths link/node centrality or convergence time, and comparing them is a challenge in itself. The performance metrics relevant for EULER are described in deliverable D4.1.

3. *Performance analysis*: our performance analysis includes i) performance comparison against performance criteria associated to each performance metric (the performance criteria relevant for EULER are described in deliverable D4.1), ii) comparison of performance measurement results obtained for the reference routing model (BGP) and those obtained for the routing models designed in the context of WP2 (the performance comparison criteria relevant for EULER are described in deliverable D4.1), and iii) performance assessment against design routing process model as determined by the performance model.

2.1.5 Complementarity between simulation and emulation experiments

In general, spatial measures are more easily achieved in simulation environments whereas temporal measures are more easily achieved by means of emulation. Simulations (in particular, simulation by discrete event) is well suited for experiments involving spatial metrics, structure, and dimensions as it enables handling of large-scale topology cases and produces results that are easier to tune, reproduce, and compare. On the other hand, emulation has the advantage of being more representative of the actual execution of procedures thus providing valuable insight in temporal metrics and behaviour (although on smaller scale and at a higher cost).

Moreover, emulation experiments enables to check the realism of simulation results and simulation experiments to extend the applicability of emulation results. Indeed, results obtained by means of simulation and emulation experiments are complementary when the experimental scenarios are commonly specified and their execution adapted (without introducing any bias) to the simulation or emulation environment.

2.2 Simulation methodology

Following Shannon [Shannon75], simulation is the process of designing a model of a real system and conducting experiments with this model for the purpose either of understanding the behavior of the system or of evaluating various strategies (within the limits imposed by a criterion or set of criteria) for the operation of a system. Ingalls [Ingalls02], further defines simulation as the process of designing a dynamic model of an actual dynamic system for the purpose either of understanding the behavior of the system or of evaluating various strategies (within the limits imposed by a criterion or set of criteria) for the operation of a system. Simulation can thus be seen as the process of exercising a model to characterize the behavior of the modeled entity process, or system over time.

Simulation is one of the most widely used techniques for i) understanding, characterizing and analyzing the behavior of complex systems, ii) construct theories or hypotheses that account for the observed behavior, iii) use the model to predict future behavior, that is, the effects that will be produced by changes in the system.

2.2.1 Simulation model

As depicted in Figure 2.1, simulations may be deterministic or stochastic, static or dynamic, continuous or discrete. In this figure, the term system refers to a group of objects that are joined together in some regular interaction or interdependence toward the accomplishment of some purpose.

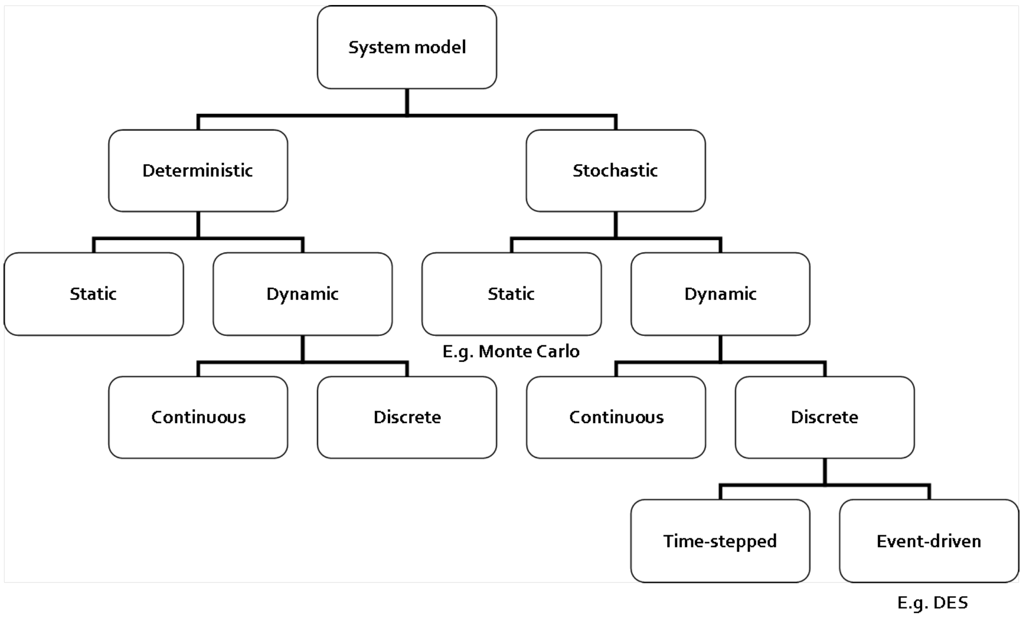

Figure 2.1: Simulation model.

The following sections describe in more details the simulation models that will be used in the context of the EULER T3.3 task.

2.2.1.1 Discrete Event Simulation (DES) Model

Discrete Event Simulation (DES) is commonly used in computer communication networks. DES simulation is characterized by being i) stochastic: some state variables are random variables, ii) dynamic: evolution over time, and iii) discrete: state variables change instantaneously at only a countable number of points in time upon occurrence of events (instantaneous occurrence that may change the system state).

2.2.1.2 Continuous Simulation Model

In a continuous model, the state variables change in a continuous way, and not abruptly from one state to another (infinite number of states). Continuous variables are typically described by differential equations. The simulation consisting in solving sets of differential equations numerically over time. Note this doesn't prevent that discrete events can occur over time that affect the continuously changing variables.

2.2.2 Methodology

As previously introduced, performance measurement and analysis by simulation requires to specify a theoretical model (of the system under study) from which a performance analysis can be performed. Using the feedback from this performance analysis a behavioural/conceptual model is then built enabling the development of a simulation specification model. The later can then be converted into a computational model. Measuring the metrics on the execution of model provides the information to compare the obtained results with those of the theoretical model. Figure 2.2 depicts the flow chart used for systematic performance evaluation and analysis process.

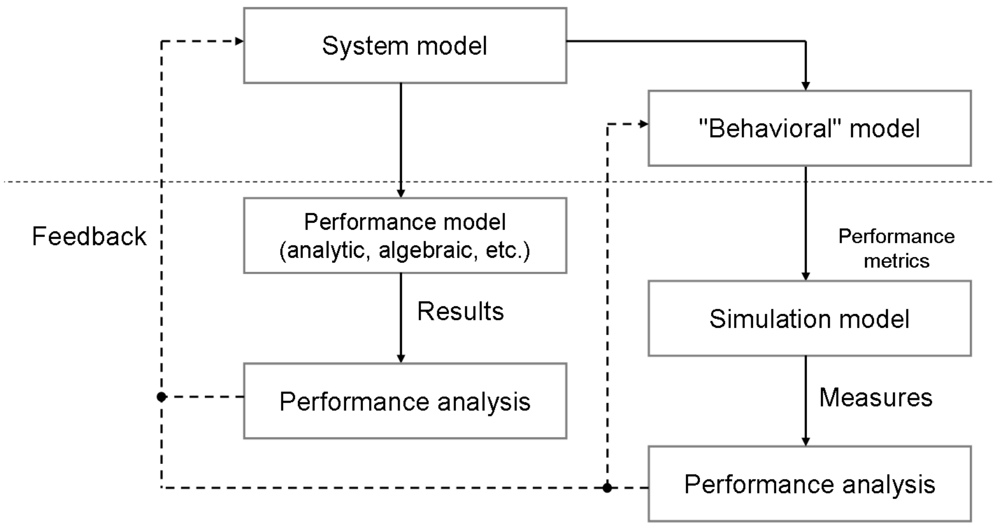

Figure 2.2: Evaluation by simulation methodology.

The (typically iterative) process to build and develop a simulation model are the following:

1. Determine the goals and objectives of the simulation
2. Build a conceptual model including state variables, which variables are static and dynamic, for the latter are variations continuous and discrete, etc.
3. Convert the conceptual model into a specification model of the simulation; the specification typically describes simulation procedures (pseudo-code) and data structures
4. Convert the specification model into a computational model, i.e., executable computer program (note: selection of the programming language is part of this step and consists in determining whether a general-purpose programming languages or a Special-purpose simulation language would be used to develop the program)
5. Verify (verification process): Ask the question: Did we build the model right?
   (a) Determine whether the computational model executes as intended by simulation specification model.
   (b) Determine whether the computational model implements assumptions made about the behavior of the real systems (as transposed in the behavioral model)
   (c) Techniques include: tracing/walk-through, continuity tests (sensitivity tests, i.e., slight change in input should yield slight change in output, otherwise error), degeneracy tests (perform execution with extremes values, e.g., lowest and highest), consistency tests (similar inputs produce similar outputs

6. Validate (validation process): Ask the question: Did we build the right model?
    (a) Determine whether the conceptual model is representative of the actual system being analyzed. Can the conceptual model be substituted, at least approximately for the real system?
    (b) The validation process also involves determining whether the computational model is consistent with the actual system being analyzed.

Note that the term simulation is frequently used to refer to the computational model itself (program).

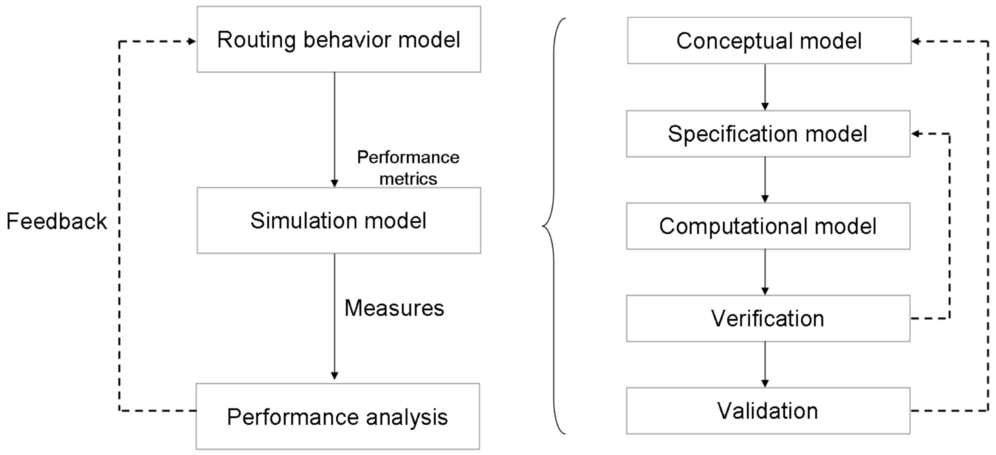

Figure 2.3: Simulation methodology.

Once the simulation program is verified and validated, simulation studies can be performed. Simulation studies include

1. Design simulation experiments:
    (a) Determine the input parameters that should be varied and their interval as well as the initial conditions for proper initialization (note that some characteristics of the environment may need to be included in the experiment if not accounted as part of the simulation model); this step is critical as it provides means for decreasing the run time of the simulation but still may not provide confidence for stable conditions.
    (b) Determine the variables to measure (at which frequency, upon which class of event (event-driven), etc. taking into account the tradeoff between too much data (that would in turn require the use of techniques for reducing the amount of collected to a usable form) and too little data (that would in turn introduce the need for representing data by statistical distributions);
    (c) Determine the execution time taking into account the tradeoff between the resource consumption (by very long runs) and amortization of effects of resulting from transient state will be amortized

2. Execution of the simulation program and record tuples of the form `{<initial_condition; input>; <running_condition; output>}` referred to as observations or data.

3. Analyze the output of the simulation program execution towards production of results. Analysis comprises:

(a) Results verification (correctness): test whether the results obtained are in accordance to the assumptions made about the behavior of the real systems (as transposed in the behavioral model)

(b) Input validation (representativity): validate assumptions about input parameter values and distributions. This step is often associated to the output validation phase.

(c) Results/output validation (representativity): test whether the results obtained are representative of those obtained either by real systems or theoretical model of the system

2.2.3 Simulation Platform

As deploying newly designed routing protocols on the Internet is not practicable at a large-scale, simulation is an unavoidable step to validate their properties. However, the increasing requirements in terms of routing information processing (CPU) and storage (memory) introduce several challenges for the simulation of state-full routing protocols on large-scale topologies (comprising tens of thousands of nodes).

For these reasons, the EULER project focused its development on the Dynamic Routing Model simulator *DRMSim*, a discrete event simulation platform which addresses the specific problem of large-scale simulations of routing models running on large networks. The motivation for developing this simulator lies in the limitation of existing simulation tools in terms of the number of nodes they can handle and in the models they propose. Before describing the *DRMSim* simulator, the following section motivates the introduction of such platform and compares to existing routing simulators.

2.2.3.1 Positioning

We have to distinguish three classes of simulators when it comes to distributed routing: (routing) protocol simulators, routing configuration simulators, and (routing) model simulators.

- Simulators dedicated to the performance measurement and analysis of the routing protocol (procedures and format) at the microscopic level. These can be further subdivided between simulators specialized for BGP protocol specifics, simulators dedicated to routing protocols and general protocol simulators. The ns [?] discrete-event simulator that relies on the BGP daemon from Zebra [?] belongs to the second sub-category. This daemon can be used to build realistic inter-domain routing scenarios but not on large-scale networks due to the low level execution of the protocol procedures. On the other hand, the SSFNet [?] discrete event simulator, relies on the implementation of the BGP protocol that was tailored and validated for the needs of a BGP-specific simulator. In SSFNet, a simulated router running BGP maintains its own forwarding table. It is thus possible to perform simulation with both TCP/IP traffic and routing protocols to evaluate the impact of a change in routing on the performance of TCP as seen by the end systems (hosts, terminals, etc.).

- Simulators dedicated to simulation of BGP protocol operations including the computation of the outcome of the BGP route selection process by taking into account the routers' configuration, the externally received BGP routing information and the network topology but without any time dynamics. These simulators can be used by researchers and ISP network operators to evaluate the impact of modified decision processes, additional BGP route attributes, as well as logical and topological changes on the routing tables computed on individual routers assuming that each event can be entirely characterized. Topological changes usually comprise pre-determined links and routers failures whereas logical changes include changes in the configuration of the routers such as input/output routing policies or IGP link weights. These simulators are thus specialized and optimized (in terms, e.g., of data structures and procedures) to execute BGP on large topologies with sizes of the same order of magnitude than the Internet since these simulators are not designed to support real-time execution. These simulators usually support complete BGP decision process, import and export filters, route-reflectors, processing of AS_path attributes and even custom route selection rules for traffic engineering purposes, and BGP policies. Simulators like SimBGP [?] or C-BGP [?] belong to this category. These simulators are gradually updated to incorporate new BGP features but are complex to extend out of the context of BGP.

- Simulators dedicated to the simulation of routing models, category to which *DRMSim* [?] belongs. Designed for the investigation of the performance of dynamic routing models on large-scale networks, these simulators allow execution of different routing models and enable comparison of their resulting performance. Simulators in this category consider models instead of protocols, meaning they do not execute the low level procedures of the protocol that process exact protocol formats but their abstraction. Thus these simulators require specification of an abstract procedural model, data model, and state model sufficiently simple to be effective on large-scale networks but still representative of the actual protocol execution. However, incorporating (and maintaining up to date) routing state information is also becoming technically challenging because of the amount of memory required to store such data. In practice, processing of individual routing states impedes the execution of large-scale simulations. *DRMSim* addresses this issue by means of efficient graph-based data structures. Moreover, by using advanced data structures to represent routing tables, *DRMSim* can still run simulation whose number of nodes exceeds ten thousands.

All simulators previously cited here above share many properties. Like *DRMSim*, they all rely on Discrete-Event Simulation (DES). However, on one hand, BGP simulators, in order to keep an acceptable level of performance, optimize their procedures and data structures for BGP protocol executions; thus, they can not be easily extended to accommodate other routing protocol models. On the other hand, general routing protocol simulators designed to investigate the effects of routing protocol dynamics are usually limited to networks of few hundred nodes; thus, preventing large-scale simulations of state-full routing protocols over networks comprising of the order of ten thousands nodes.

Because deploying newly designed routing protocols on the Internet is not practicable at a large-scale, simulation is an unavoidable step to validate their behavioral and performance properties. Unfortunately, the simulation of inter-domain routing protocols over large-scale networks (comprising tens of thousands of nodes) becomes a real issue [?] due to the increasing routing information processing (CPU) and storage (memory) they require. To the best of our knowledge, no simulator provides the capability to investigate in-depth the behavior and performance properties of routing schemes when applied to large-scale topologies (comprising tens of thousands of nodes).

2.2.3.2 Description

Section 4.1.1 details the Dynamic Routing Model simulator *DRMSim* which addresses the specific problem of large-scale simulations of (inter-domain) routing models on large networks.

2.3 Emulation methodology

The main purpose of WP4 is the experimentation of designed routing schemes in representative settings. Whereas simulation techniques abstract parts of the experimented components and its environments (potentially allowing very large scale experiments), emulation-based experimentation targets the evaluation of lower-level implementations (component prototypes) of the designed routing schemes. For example, an emulated BGP router, functionality behaves exactly in the same way as a production quality BGP router. In this case there is a one-to-one mapping between the defined functions and their implementation. However, emulation experiments require more implementation effort, as well as more hardware resources to evaluate a given network setup.

This section outlines the planned methodology for the emulation experiments planned in the context of WP4. This involves the definition of the used platform, the used procedures, the components to be experimented and the scenario's and their means for experimental verification.

2.3.1 Emulation platform

The iLAB.t virtual wall, experimental facility at IBBT, will be used in WP4 to execute the experiments associated to T4.3. This experimental emulation facility consists out of 100 servers connected through a non-blocking Ethernet switch. The 100 high-end servers composing the experimental facility cluster come each with dual CPU dual core 2.0GHz, 4GB RAM, 4 or 6 Ethernet network interface cards (Gb/s) - PCI express, and 4x80GB disks - software RAID 0 configuration. The servers are interconnected by a Force10 E1200 non-blocking Ethernet switch (1.68 Tb/s - 1 Billion pps) that scales up to 672 ports. Each server is connected with 4 or 6 Gigabit Ethernet links to the switch. Once a given number of nodes is reserved, arbitrary logical topologies can be created between the nodes (taking into account the available number of network interfaces). These topologies are set up automatically using the control software of the virtual wall (Emulab, see below) such that the Force10 Ethernet switch interconnects the reserved nodes in the desired logical topology relying on VLANs.

Remote access to the iLAB.t control system is possible without any restriction on functionality. As such, large dedicated experimental setup can be built very fast. The resulting setups are the repeatable and enable dedicated experiments within confined environment where experimental parameters can be controlled.

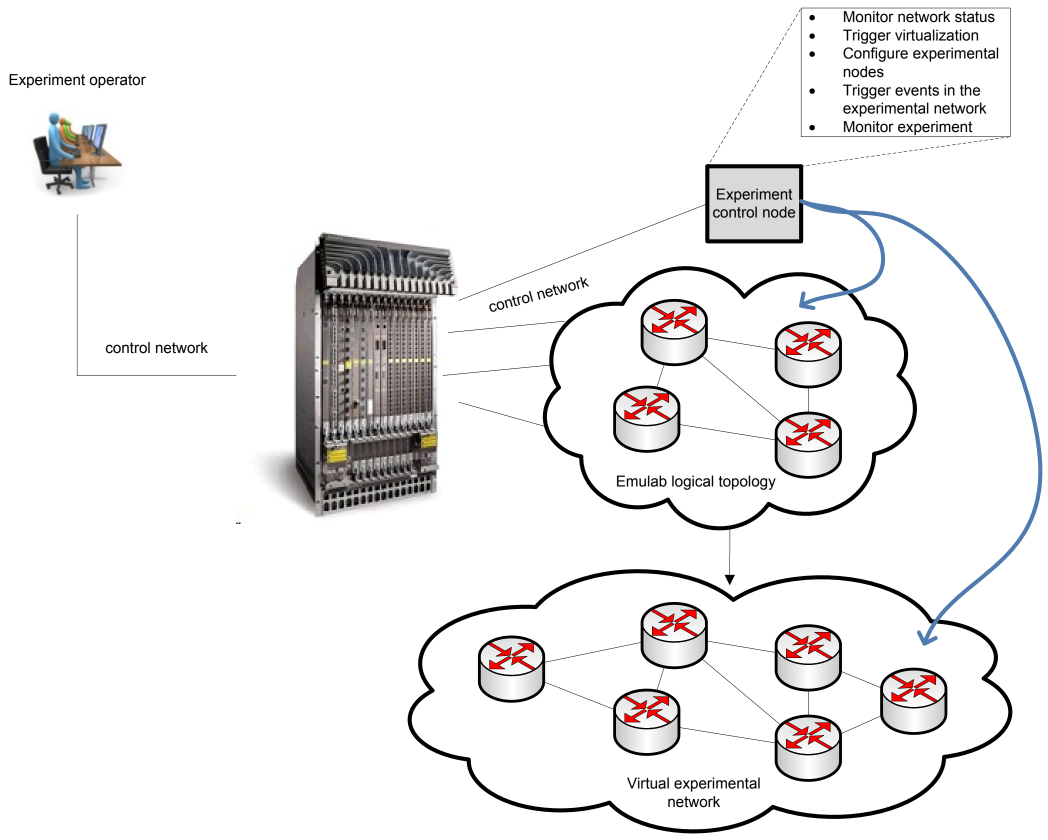

Figure 2.4: Emulation platform setup

Emulab The University of Utah has developed Emulab (www.emulab.net), which is control software for a very flexible emulation experimental facility based on servers. These servers can run a variety of operating system images and can be interconnected dynamically in topologies as needed for the experiments. In this setup, network links (bandwidth, delay) can be emulated through software. This software has been developed for controlling local PC clusters. The software enables performing automatic and remote configuration, by swapping in and out operating system images. As such, large dedicated experimental facility can be built very fast.

The Emulab software is used to control the ilab.t virtual wall experimental facility. It makes possible to fully control the servers and the interconnecting network (without executing experiments themselves). More precisely, it installs the desired operating system image on each server and configures the Ethernet switch with the right VLAN configuration. It also incorporates multiple users, a web interface and an archive for experiment setups. However, the user still needs to define and execute its experiments once the ilab.t virtual wall is configured.

The Emulab software provides the following features to the user:

- Web-based interface with dedicated user accounts;
- The user can graphically draw a testbed layout with nodes and interconnections (or provide this through an NS2 configuration file);
- The user can select an OS image (Linux, BSD, Microsoft Windows) for each node;
- After the user confirms: all nodes are installed and the necessary interconnections are made;
- After this, the user has full root access to the nodes for executing the experiments;
- Afterwards, the testbed is swapped out again, but available for future experiments with the same setup. As such, the nodes can be reused by another user;
- Nodes are installed in parallel in a very fast way (couple of minutes).

The WP4-related parts of EULER website and mailing lists contain further information on using the ilab.t platform (information on accounts, manuals, etc.).

The resulting experiment setup consists of the following elements, as illustrated in Figure 2.4:

1. **the experiment operator**: end-user which triggers/configures/controls the experiment from its laptop/desktop
2. **the experiment control node**: a node which is part of the defined Emulab experiment and whose purpose is exclusively to control the execution of the experiment (it is thus not connected via data interfaces towards other parts of the experimental setup).
3. **the Emulab logical topology**: the logical experiment network as swapped in by the Emulab software (this does not include any virtualization)
4. **the virtualized experimental network**: the emulated network to be used for experimentation, resulting from using virtualization technology on top of the swapped-in Emulab network topology

2.3.2 Experiment workflow

Emulation-based experimentation in the context of task T4.3 follows a structured process ensuring repeatability and automation. Figure 2.5 illustrates the different elements and phases in the envisioned experimentation workflow.

The experimentation process follows 7 phases:

1. **Swap-in preparation**: definition of the software images and Emulab-topology to be used.
2. **Swap-in execution**: during this process the virtual wall control software loads the configured software images on the reserved nodes and interconnects the nodes according to the pre-configured topology (as described in an ns2 format)

3. **Experiment preparation**:

   (a) Definition of the virtualization process enabling multiple emulation nodes to run on existing physical nodes of the Emulab experiment.

   (b) Configuration of the experiment controller node.

   (c) Experimentation environment monitoring to verify of the configured initial experiment conditions are valid (including for example the validation of link connectivity).

   (d) Definition of the *experimentation script.* This script steers the entire execution (time sequence) of the experimentation, including the setup of network traffic, the addition/removal of network elements, etc.

4. **Experiment execution**, resulting into experiment-related output (logfiles, tracefiles, etc.)

5. **Post-processing**: the process of deducing usable (and statistically valid) numeric and graphical conclusions as an outcome of the experiment.

2.3.3 Experimental components

Experimentation relying on the emulation (prototyping) of routing schemes consists of two major components:

- **routing functionality**: consisting of all control functionality of developed schemes in WP2 related to: discovery, pre-processing, routing table production, and resolution functionality (which includes locator/identifier functionality in relationship to existing IP addressing/subnetting). Experimentation in T4.3 focuses on these components.

- **forwarding functionality**: consisting of functionality responsible for receiving and transmitting data traffic. The forwarding process used by traditional routing schemes is based on lookups in forwarding tables (Forwarding Information Bases or FIBs). The forwarding decision in greedy routing is based on a set of distance comparisons between a node's neighbors and the destination coordinate of a received packet. Experimentation of forwarding functionality is optional in the context of task T4.3, and is mainly of value in the context of stress tests.

2.3.4 Enabling experiment events and measurements

Once the initial experiment setup has been completed (bootstrapping), the execution of the experiment can start. As illustrated in Figure 2.6, the resulting experimentation consists of monitoring the configuration and the state(s) of running elements as well as the activation of given network events such as network topology changes or the generation of network traffic (only for the evaluation of forwarding functionality). This process is steered by the experiment script. Experiment scripting functionality requires verifying evaluated experiment conditions. For example, if links are disabled in the script, logging functionality will maintain the state of the affected link after the planned disabling step. Post-processing functionality will then allow to verify if planned experiment conditions have taken place during the run. More information about using experiment scripts can be found in Section 4.4.2.3.

Experimental performance is characterized through the experimental metrics as defined in deliverable D4.1. Three classes of metrics can be distinguished. The metric definitions are taken from deliverable D4.1 and are extended with respect to their measurement in the experimental setup. An overview is given below.

1. **Routing performance metrics**

(a) *Routing path length*: the number of nodes along the routing path from source to destination as produced by the routing protocol. The routing path length can be exactly measured in emulation experiments. This measure allows computing the resulting stretch of a routing scheme. For routing schemes that maintain the actual routing path ((abstract) node sequence) in routing tables, the routing path length can be determined by counting the number of (abstract) nodes composing the routing path to each destination. For routing schemes that maintain or greedily compute a partial/local routing path information, in particular, the next-hop neighbor towards each destination, and in order to avoid implementing the forwarding functionality in the emulated prototype, this metric can be measured by implementing ICMP-alike packet forwarding in the control plane, allowing to record the number of hops of these packets sent between nodes. When divided by the corresponding topological path length, the resulting ratio, referred to as the multiplicative stretch, is expected to remain close to 1 (small and upper bounded deviation from the actual topological path length defines one of the main routing quality criteria).

(b) *Routing table size*: the number of routing table entries and their total size (expressed in terms of memory space) required to store them per node. Note this number/size should be sub-linear with respect to the number of nodes/reachable prefixes. This metric can be measured in emulation through the implementation of table counters per node.

(c) *Computational complexity* (of the routing algorithm): this metric can be measured by determining i) the number of CPU cycles and the memory space required to compute each routing path and ii) the time required to locally compute each routing path with respect to the input size. Computational complexity can be verified in emulation by recording a counter/timestamp for every atomic step performed locally during the routing path computation.

(d) *Communication cost*: the rate (x size) of routing protocol messages exchanged between nodes that are needed by the routing protocol to properly operate. Note that routing protocol messages may include topology information and/or routing information; both types of information are referred to as routing protocol information. The exchange rate can be measured in emulation by monitoring the number of routing protocol messages exchanged per time unit/per event between adjacent nodes. Recording the size of each message enables to further determine the instantaneous capacity and estimate the time required to exchange these messages.

(e) *Connectivity time*: the time needed for a newly added destination to become reachable through the existing network topology (connected component of the topology). In case of routing protocol information push: this time includes the propagation of the routing protocol messages and thus the corresponding information across the network topology. In case of routing protocol information pull: this time accounts for the query/response delay (and associated resolution if any) of this new routing protocol information by an existing node. In the emulation framework, this can be measured i) by adding additional information to the logging interface (which can be checked after the experimentation run), and ii) by sending ICMP-alike packets on the path between the affected nodes (see previous point on routing path length).

(f) *Convergence time*: network topology change(s) (e.g. link/node failure) or routing protocol change(s) (e.g. re-configuration) result in routing states change. Upon occurrence of such event, several destinations become unreachable (e.g., loss of connectivity) and remain potentially unreachable until routing states (re-)converge on the new topology. The time elapsing between the occurrence of such event and the time at which all destinations are again reachable is an indirect way to deduce the convergence time of routing protocol states (one assumes that achieving full reachability implies routing state convergence). This metric can be measured in emulation through the investigation of time stamps of logfiles of routers[1], notification to control node, or packet loss measurement between affected traffic streams[2]

   (g) *Configuration time*: number of actions to perform off-line configuration of newly added elements into the topology being network partition(s), node(s), link(s), or destination(s). The associated operations can be modeled in emulation and have a relationship to the number of steps/commands needed in the experimentation script.

2. **Forwarding performance metrics**

   The implementation of forwarding functionality in the context of emulation is not required to determine the performance of the routing scheme. Such implementation will thus only be considered in the second stage of experimentation. This section lists the possible forwarding performance metrics and indicates how they will be measured in the experimental setup.

   (a) *Forwarding table size*: the number of forwarding table entries and their total size per node (that should be sublinear with respect to the number of nodes/reachable prefixes). The emulation setup can measure this by implementing and maintaining node counters forwarding table.

   (b) *Forwarding delay*: the time needed to determine the outgoing interface/port of a packet from its incoming attributes (including destination coordinates, destination address, destination label, etc.) using the local forwarding table. This metric is of particular importance for routing protocols that assume more than one lookup/online computation operation to forward individual packets. Measurement of forwarding delay can be implemented by timestamping individual packets at the monitored node(s) before the forwarding process starts and after the forwarding process completes. Another mean to measure the forwarding delay is to measure the total delay experienced when packets traverse a node and substract the propagation, transmission and queuing delay.

   (c) *(Aggregated) Throughput*: average rate of packets generated by a given (set of) source that is successfully delivered to a given (set of) destination along a (common) forwarding path. This metric can be approximated in emulation through the use of configurable traffic generation/reception tools such as iperf (possibly adapted to make them compatible with the routing scheme to be evaluated).

   (d) *Delay*: the time interval elapsing between sending a packet from a source node, and the reception of that packet at the destination node. The delay can be approximated in emulation (half round-trip time) through the use of Ping-like functionality (implementation of ICMP-type packets).

   (e) *Jitter*: the difference in end-to-end delay between selected packets in a sequence of packets flowing from a given source to a given destination. This metric can be measured in emulation by extending end-to-end traffic measurement tools such as iperf or D-ITG.

   (f) *Packet loss*: this occurs when one or more packets of data traveling across a network fail to reach their destination. This metric can be measured in emulation by extending end-to-end traffic measurement tools such as iperf or D-ITG.

[1]time offset estimation will be required between different experimental nodes

[2]some form forwarding functionality will be needed

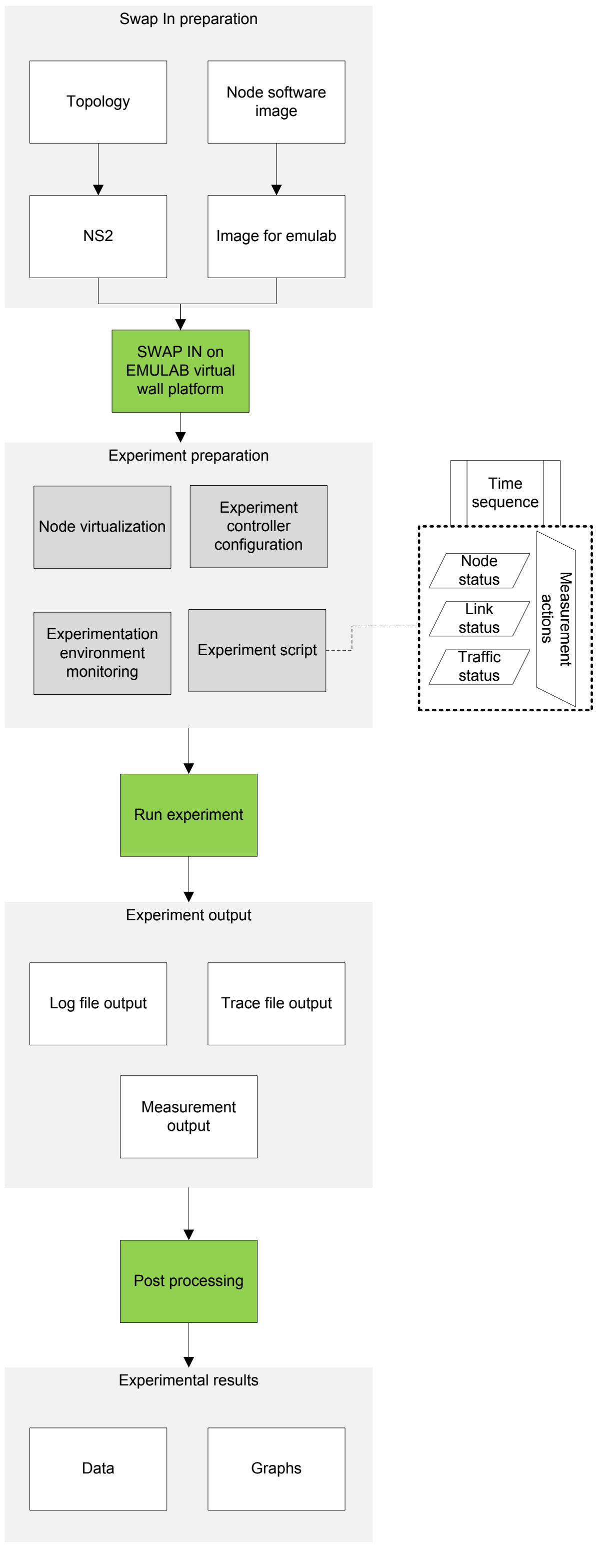

Figure 2.5: Experimentation flow in emulation

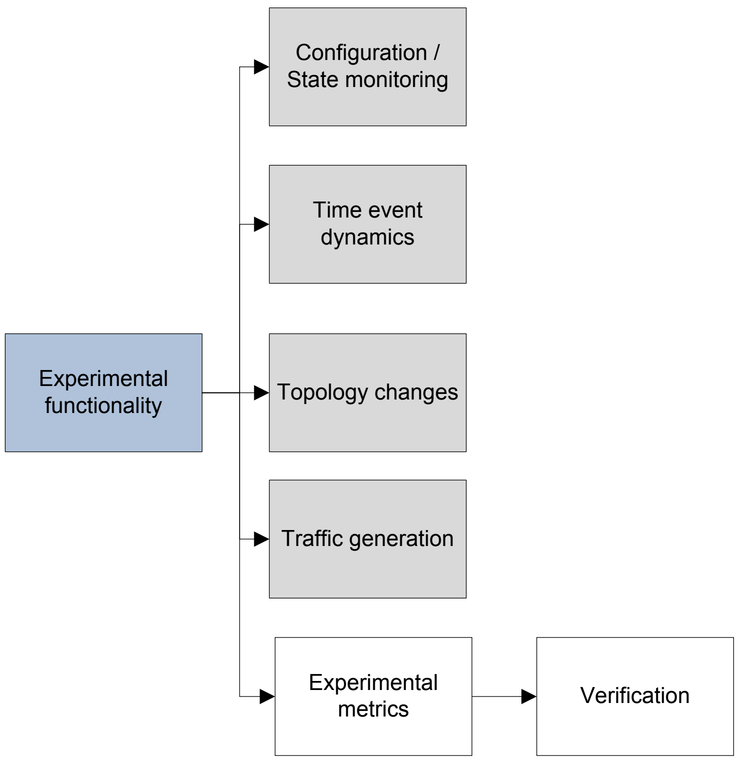

Figure 2.6: Experimental functionality

Chapter 3

Scenarios

EULER experimentations strongly rely on the notion of scenario, which define the topology, the addressing, and the traffic demand we consider for each routing scheme experiment. Note that several routing experiments can be performed without involving any actual traffic/forwarding. These scenarios are instances obtained either from measurements or from models. Therefore, we need to describe which measurements and models we consider.

3.1 Topologies

There are many ways to choose a topology representing the underlying network on which the experiment takes place. Such topologies may be directly obtained through measurements which provide *maps* of the Internet. In EULER experiments, we use state-of-the-art maps like the ones provided by CAIDA and DIMES. However, such maps are known to be partial and biased, which may have an influence on experiment outcomes. EULER solves this issue by conducting reliable measurements of Internet properties (like its degree distribution) and then use graph models to generate graphs which fit these properties. In a way, these artificial graphs are closer to the real Internet than any map. In addition, we use models to generate graph families with different properties in order to study their impact on routing performances. There, the key concern is not that the model topology should be similar to the actual Internet topology, but that we are able to tune topology properties in order to identify desirable vs undesirable properties for the (future) Internet topology.

We therefore use three kinds of topologies:

- Internet maps
  - CAIDA [1] provides Internet maps at IP and AS levels collected on a regular basis since 2004.
  - DIMES [2] provides AS-level maps of the Internet collected from a large number of distributed monitors (approximately one thousand).
  - RouteViews [3] collects BGP tables since 1997 on a set of BGP routers, making it possible to build dynamic AS maps.
  - Radar [4] collected for several months in continuous IP-level maps from a hundred monitors every few minutes, making it possible to study the dynamics of ego-centered views.
- Measurement-based artificial graphs

[1]http://www.caida.org/home/
[2]http://www.netdimes.org/
[3]http://www.routeviews.org/
[4]http://www-rp.lip6.fr/~latapy/Radar/

  - One key objective of EULER is to obtain the first accurate and reliable estimate of the degree distribution of the Internet topology. This distribution is a key parameter to many graph models, including the Configuration Model. We will therefore use artificial graphs with the appropriate degree distribution as topology models.
  - Another approach consists in taking ebefit from new measurement approaches based on `mtrace` which give a bipartite view of the Internet, more accurate than classical view; we will combine this with random bipartite graph models in EULER to obtain artificial graphs reflecting this property.

- Model graphs [5]
  - ER random graphs are used as a baseline for topology modeling, and give insight on the impact of density on experimentations.
  - Random graphs with prescribed degree distribution make it possible to investigate the impact of degree distribution.
  - Bipartite random graphs adds to this the possibility to tune clustering properties using the bipartite structure and study their impact.
  - Albert-Barabasi graphs rely on the notion of preferential attachment and make it possible to explore the impact of this feature on our results.
  - BRITE, a parameterized topology generator which aims at modeling different aspects that lead to the power law behavior observed in Internet-like topologies such as preferential connectivity, incremental growth, node placement strategies, and connection locality.

Note that, as explained in deliverable D3.1, each of these models has its own interest in the context of EULER. For instance, although ER graphs are very unrealistic models for the Internet, they provide a baseline for evaluation of results with other models. Similarly, random graphs with prescribed degree distribution make it possible to focus on this specific and important property. Other listed models are more complex but also more representative, enabling in turn to capture more subtle Internet topology-related features.

In addition to these topologies, EULER has a special interest in *dynamic* topologies. Several of the Internet maps described above have been collected for years, and so capture some kind of dynamics. In particular, RouteViews and Radar data capture this dynamics at a rather high frequency, below one day or even one hour. In addition to this real data, in a way similar to what we do with static topologies, we model the dynamics of topology in a way which maps the observations from measurements and in ways which allow us to study the impact of dynamic features on our experimentations.

We consider in particular models capturing the following dynamics:

- load balancing, *i.e.* routers which forward traffic towards a given destination over several paths in order to distribute their incoming load,
- congestion-induced routing changes, such capability requires to enhance the routing path computation with traffic-driven processes (e.g. monitor the traffic rate per unit of time per routing path)
- link and node failures modeled by their (temporary) removal from the network,
- link switch, consisting in replacing two links $a--b$ and $c--d$ by links $a--c$ and $b--d$.

In a measurement-based approach, such dynamics are tuned in a way which fits real-world observations; in a more exploratory way, we make experiments with wide variety of parameter values in order to gain insight on their impact on observations.

[5] See deliverable D3.2 for more details on each model and deliverable D3.1 for a comparative study of the weaknesses and strengths of each model.

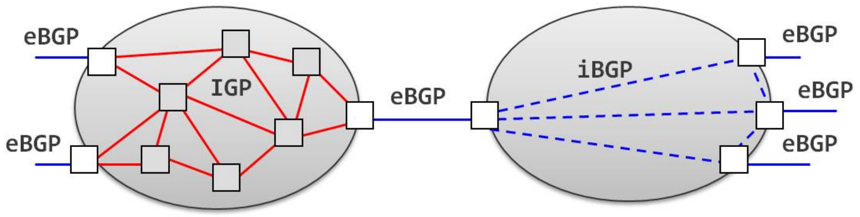

Figure 3.1: Inter-AS vs intra-AS router control configuration in BGP

3.2 Network partitioning and addressing

The formal topology description in terms graphs comprising edges and vertices is not sufficient as input for the experimentation in the context of Task T4.3. The EULER-designed routing schemes intend to provide an alternative to the BGP-based inter-AS routing which distributes and maintains the required information to ensure the reachability of/connectivity to IP (sub)networks from remote end-hosts/terminals.

The nodes of the graph on which BGP routing operates are ASes. At the time of writing this document, the size of this topology is of the order of 40K ASes. An AS can be seen as an abstract node consisting itself of a router topology modeled as a graph and that may comprise up to thousands of nodes. Only a limited number edge routers participate in the inter-AS routing of BGP. Within a given AS, edge routers operate such that other ASes are not informed about the internal structure of that AS (note that from some traffic engineering attributes of BGP such as the Multi-Exit Discriminator (MED) one may derive the behaviour of routing paths crossing neighboring ASes). In the context of BGP, one distinguishes between iBGP for intra-AS routing exchanges between BGP routers located at the edge of a given AS, and eBGP for routing exchanges between BGP routers belonging to different ASes (see Figure 3.1).

In the context of EULER experimentation (Task T4.3), the designed routing schemes are evaluated against BGP on the following aspects for what concerns network partitioning in AS and IP addressing space segmentation in (sub)networks:

1. The evaluation of mechanisms which are responsible for the distribution of host/IP (sub)network connectivity (i.e., reachability)

2. The evaluation of mechanisms responsible for intra-AS control activity

3. The evaluation of AS partitioning mechanisms (in order to determine how many logical routing levels an alternative routing scheme would have to support).

Besides a suitable mapping of graph nodes to ASes, adequate IP addresses need to be assigned to graph nodes, and subnets need to be associated to graph nodes (and subgraphs). Whereas for some existing topologies (such as the CAIDA network topology), some of these details might be known, for synthetically generated topologies, this information needs to be generated. For this reason, scripts can be developed, as mentioned in Section 4.4.2.1.

3.3 Traffic

Once a topology has been determined, and addresses (i.e., locators) have been assigned, what we have basically is a addressable network topology: a network infrastructure without traffic between hosts/terminals attached to the network. The last step towards running experiments therefore is to model traffic demand to be routed in the network.

Modeling traffic demand is a challenging task, for which current state-of-the-art is relatively poor (as compared to modeling of topology or routing schemes, for instance). In most cases, experiments only consider the very simple scenarios where one node sends traffic to all other nodes, and where all nodes send traffic to all others. We will consider these scenarios as baselines for our traffic demand modeling, and we develop in EULER a measurement-based approach aimed at better capturing the features of real traffic.

First notice that file exchanges between Internet users nowadays contribute for a great fraction of all Internet traffic, typically more than half of it. Thanks to our measurements of exchanges in a large-scale P2P system, we have deep insight on such exchanges among millions of users, involving millions of files: we know which peers seek which files and when, and we also know which other peers provided them to the seeking peers. Based on this data, we design models of file spreadings among users of a P2P system in order to capture the main features of the induced traffic demand. Thanks to these models, we are able to generate traffic much more realistic than previously, and use it in simulations in our global framework (topology and routing).

Our measurement-based traffic models therefore typically consists in file exchanges between Internet users which mimic offer and demand encountered in a large P2P system. They consist in file spreadings among users (nodes in the network) based on simple spreading mechanisms tuned to fit real-world observations. Such models, detailed in the next chapter, generate lists of exchanges between users that the network has to route. Although this is not fully realistic, it is by far more realistic than current approaches and it is a good complement to one-to-all and all-to-all experiments.

3.4 Specific scenarios

3.4.1 Stress test of a router

Several network conditions can introduce stress to the network and individual network nodes. A network node comprises functionality to receive packets, process them, and to transmit them on an outgoing interface. Buffering is used at the incoming and outgoing interfaces to avoid dropping packets while they wait to be processed by an entity having limited capacity.

The forwarding process introduces delay, measure by the difference between the packet arrival time in a router and the packet departure time. This delay can change under changing network conditions. When a router needs to process heavy bursts of network traffic (high number of packets within short time frame), buffers can filled up, resulting in dropped packets (packet loss) and increased delay for the packets that aren't dropped.

Therefore, as part of the experimentation of the forwarding plane, an experiment will be performed to characterize the correlation between the load a router is experiencing for given configuration of buffer sizes, and the introduced delay and packet loss. Increasing the traffic load (in Mbps) to levels which are close to, or even above the routers capacity is a meaningful stress test for characterizing the traffic sensitivity of a router.

The stress test of a router can be accomplished by wiring all interfaces of the tested router to a device which is able to produce and receive network traffic and measure the resulting delay, packet loss and other related metrics. The latter device can either be professional testing equipment, or a modified software router with sufficient capacity and measurement capabilities installed.

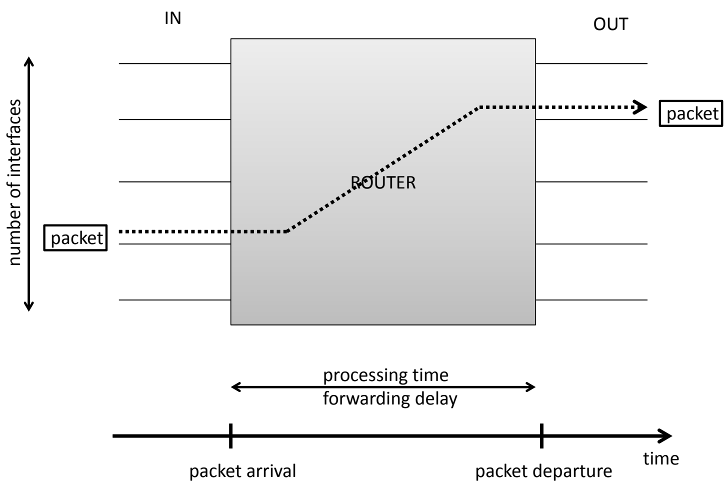

Figure 3.2: Forwarding delay of a router

Chapter 4

Tools

EULER experiments use a large variety of simulation and emulation tools, which we describe in this section. We grouped them into three categories: simulation tools/platforms, emulation tools/platforms, and other experimentation support tools.

In each of these sections, we listed tools/platforms by decreasing order of importance or generality. All tools/platforms are fully documented and provided in deliverable D4.3.

4.1 Simulation tools/platforms

4.1.1 DRMSim, a network simulator for the investigation of routing schemes

The objective of *DRMSim* is to provide a routing model simulator to the scientific networking community. *DRMSim* does not aim at performing accurate simulations at the forwarding level. Instead it focuses on the underlying routing layer itself, by exposing a clear and dedicated Application Programming Interface (API) to its users. In other words, it is devoted to the construction of routing tables and so to the evaluation of the behavior and performances of various distributed routing schemes (performance metrics are: stretch, size of routing tables, number of messages, adaptivity to topological modifications, etc.). Simulations are modeled according to the discrete-event paradigm.

Many researchers willing to conduct experimentations on the routing layer opt for widespread solutions like *ns-2*, OMNet++, etc. However, these protocol simulators do not permit easy access to their routing models since they are designed to provide reasonable accuracy of the physical layer, and maximum accuracy of the MAC protocols. The induced cost for achieving accuracy has a negative impact on the overall performance of their simulation engines. The *DRMSim* routing model for physical and MAC layer is designed as a compromise between performance and detail. In particular, performance is always prioritized as long as the routing models are respected. Therefore, *DRMSim* enables its users to conduct simulations on large network instances that cannot be handled by other simulators. In particular, when coupled with the *Grph* library (see Section 4.1.2, page 30), it enables the simulation of networks composed of more than 10 000 nodes. Also, its API makes it easier the experimentation of novel routing schemes in the team under various scenario including dynamic evolution of the topology and of the routing policies.

Up to now, *DRMSim* is primarily targeted to the FP7 STREP EULER project partners. However, as soon as it will have reached a higher level of maturity, it will be made available to a larger scientific community. To this end, *DRMSim* is already distributed under the terms defined by the General Public License GPLv3.

DRMSim is entirely written in Java, but it relies on the *Grph* library which runs pieces of native code (often C code compiled on-the-fly). The core of *DRMSim* consists of about 60,000 lines of code. The development of *DRMSim* has been done in collaboration between Alcatel-Lucent

Bell (ALB) and INRIA. Preliminary experiments showing the possibilities of DRMSim have been presented in [HPTM10].

References.

HPTM10 L. Hogie, D. Papadimitriou, I. Tahiri, and F. Majorczyk, Simulating routing schemes on large-scale topologies, Proceedings of 24th ACM/IEEE/SCS Workshop on Principles of Advanced and Distributed Simulation (PADS), Atlanta (GA), United States, pages 8p, May 2010.

***DRMSim* web site:** `http://www-sop.inria.fr/mascotte/projets/DCR/`

***DRMSim* source code:** available at `https:gforge.inria.fr/projects/drmsim/`

4.1.2 *Grph*, an efficient portable graph library tailored to network simulation and graph analysis

In the context of the EULER project, we have been developing a suitable graph library for conducting numerical and statistical analysis of graph topological properties, in collaboration with task T3.1. This graph library, called *Grph*, is also used by *DRMSim* as a support for routing scheme to evaluate topological properties that could facilitate routing. Also, this library will continuously been extended according the requirements of tasks T2.2, T3.1, and T3.3.

The main objective of *Grph* is to provide researchers and engineers a suitable graph library for graph algorithms experimentation and network simulation. *Grph* is mainly a software library, but it also comes with a set of executable files for user interaction and graph format conversion; as such, it can be used autonomously. Performance and accessibility are the primary targets of the *Grph* library. At every stage, it is designed to be efficient, both in terms of computation time and in terms of memory requirements. Its model considers mixed graphs composed of (un)directed simple- and hyper-edges. It allows to handle large dynamic graphs in the order of millions of nodes. *Grph* comes with a collection of base graph algorithms which is regularly augmented.

Grph uses the inherent parallelism of multi-core processors and multi-processor computers whenever possible, and performs caching of the results of graph algorithms in order to avoid recomputation (this is particularly useful in the context of network simulation). *Grph* integrates a bridge to native code (libraries or external applications) which allows the implementation of critical algorithms in C/C++. In order to provide this bridge, *Grph* resorts to on-the-fly compilation. Additionally, it provides a console-based interactive interface (shell), making experimentation more accessible.

So far, most known users of the *Grph* library are part of INRIA and part of the EULER project. *Grph* is distributed under the terms of a license defined by its contributors and is available for download. This license allows free usage and access to the source code. See `http://www-sop.inria.fr/mascotte/software/grph`.

Grph enables its users to conduct experiments on large graph instances that cannot be directly processed in Java. In particular, it enables the creation and manipulation of graph instances composed of several millions of nodes. Most researchers and engineers willing to deal with graphs from a programming point of view resort to JUNG or JGraphT (Java), to Boost (C++) or Sage (Python). Because of its implementation language, *Grph* is in direct competition with JUNG and JGraphT; these projects were very active in the past but exhibit very low activity during the last two years (i.e., no new release and almost no traffic on their mailing-list). In addition, a number of tools for statistics as well as for the computation of distributions, linear regressions, power-law exponents, etc. are also provided. A detailed list of algorithms is available at `http://www-sop.inria.fr/members/Luc.Hogie/grph/grph-algos.pdf`.

Our current and future plans include the design and development of a framework for the distribution of computational-intensive graph algorithms across a farm of servers as well as the addition of new property tests. These include bridging-centrality, localized bridging-centrality, group betweenness, vertex cover, growth, and hyperbolicity (exact, approximate, and heuristic algorithms). A preliminary implementation is already available but further algorithmic improvements are sought.

4.1.3 Generation of dynamic routing trees / Internet dynamics on artificial graphs

The Internet is a living system that evolves in time. Everyday, many nodes and links are added or removed, during planned maintenance or because of unexpected network failures. It is important to map the Internet topology, it is equally or even more important to understand its dynamics. The goal of our research is to study the dynamics of the Internet. We focus on the dynamics of the IP-level routing topology around a single node. We first measure this topology using our traceroute-like measurement tool: tracetree. We periodically probed the route to several destinations from a single monitor in the Internet. This results in a series of routing trees which represent different ego-centered views of the routing topology around the monitor. Then, we analyze the resulting data to identify relevant behaviors that characterize the dynamics. Finally, we propose a model whose initial goal is explanatory.

Our model reproduces on artificial graphs the measurements performed on the Internet. We represents the Internet IP-level topology as an artificial graph $G = (V, E)$, where vertices correspond to IP addresses and edges correspond to the IP-level links between two IP addresses. It incorporates four ingredients: the routing topology, the routes from the monitor to the destinations in this topology, load balancing, and routing modifications. For modeling each ingredient, we try to make the simplest choice possible, our goal being to obtain a baseline model which makes it possible to investigate the role of each component, and to which future and more realistic models should be compared. It follows the steps below :

1. First, generate an artificial graph G_0 and set i to 0.
2. From G_i, randomly select one node as the monitor and d nodes as the destinations (d is given).
3. From G_i, extract a routing tree T_i from the monitor towards the d destinations.
4. Add s changes to the topology of the graph G_i, which produces the graph G_{i+1}.
5. Repeat steps 2, 3 and 4, thus obtaining a series of trees T_0, T_1, T_2, ... that simulates periodic tracetree measurements.

This process generates a series of trees T_0, T_1, T_2, ... that simulates periodic tracetree measurements, on which we can conduct similar analysis as those we performed on real data.

4.1.3.1 Graph generation

The first step of the simulation is to generate the network graph. Let $G = (V, E)$ be an undirected and unweighted graph with V as the set of vertices ($|V| = n$) and E, the set of edges ($|E| = m$). Different proposals for the topology of G can be used, such as regular topologies, realistic network topologies or randomly generated topologies. In our work, we test simple random graphs, like Erdös-Rényi and power-law graphs. It is well known that these graph models may not reflect the reality of a network. However, its a good starting point for our work because it is simple to understand and to analyze.

4.1.3.2 Routing strategy

The result of a tracetree measurement from one monitor to many destinations is a routing tree. In our simulation, we assume that it is a routing tree of shortest paths between the monitor and the destinations. Several strategies may exist to compute shortest paths in a graph, we implement and test two of them, which we call (a) the *source-routing* (b) and the *hop-by-hop* model.

The *source-routing*: this model assumes that the monitor knows all the shortest paths through a given destination. Therefore, the monitor makes the choice of one shortest path among the other. As a result, for each destination, every shortest path has the same probability to be picked.

The *hop-by-hop*: This model tries to mimic the behavior of real routers on the Internet. Usually to route data, Internet routers search in their routing tables the closest neighbors to destinations and send them the data. Therefore, our *hop-by-hop* model does not choose a shortest path, instead it chooses a node. Each node on the shortest path has to choose one of its neighbor. A straightforward process to implement this model between a single monitor and a given set of destinations is the following (1) Extract all shortest paths between the monitor and each destination; (2) For each destination, traverse all existing shortest paths to fill the routing table of each node with the appropriate neighbor; (3) Build a routing tree on top of the resulting subgraph of shortest paths. We build routing tree in the same principle as tracetree. Each routing path is traversed from the destination to the monitor. The traversal is interrupted when we reach a node that is already in the tree. During our experiment, we find this process highly resource consuming because we have to do many traversals of the tree; specially for large graphs (more than $500,000$ and $1,000,000$ nodes). We improve our implementation by performing a simple *breadth-first search (BFS)* from the monitor towards the destinations and then, by removing all branches that do not lead to the destinations

4.1.3.3 Network changes

In the final step, we model how the Internet evolves during time. While probing the Internet with active measurements, it appears that the path from the monitor towards a given destination may not be the same at different rounds. Previous studies already highlights many factors behind the dynamics of Internet paths, some of them being *load balancing* and *route evolution*. The dynamics in Internet may be observable on its topology or on the routing. For instance, load balancing along different paths materialize routing dynamics.

At first, in order to simulate load balancing, each node chooses at random the next node on a shortest path to the destination, and we therefore implement a *random BFS*. It generates a shortest-path tree from the monitor to the destinations by considering the neighbors of explored nodes in a random order. These routing trees will therefore be different from one random BFS to the next, even if the underlying graph does not change.

Second, in order to account for route evolution, we propose to rely on simple graph modifications that can affect the shortest paths. We use a simple approach based on link rewiring. Given our graph $G = (V, E)$, we test the two following approaches for link rewiring.

Link swap. It consists in choosing uniformly at random two links (u, v) and (x, y) [1] and swap their extremities, *i.e.* replace them by (u, y) and (x, v). This approach has a the main interest that it conserves the degree distribution of all nodes in G. This features is particularly important when analyzing the dynamics of graphs with specific degree distribution. For instance, ensures that, when running the simulation on a power-law graph for example, we still end up with a power law graph at each round(improve).

link removal. Another approach for graph rewiring is to simply remove a link (x, y) at random, and then create a new link (u, v) between two nodes u and v chosen at random. This procedure may not conserve the degree distribution of the initial graph. To do so, we can choose nodes x and y such their degree are the same as the nodes u and v, respectively. We implement both variants of this model in our code.

[1] We choose them such that the four nodes are distinct.

4.2 Generation of P2P traffic demand / Epidemic Spreading Simulation

Diffusion phenomena are ubiquitous in complex networks, for example: the spread of virus on contact networks, gossip on social networks and files in peer-to-peer (P2P) networks. In these contexts, epidemiological models have established themselves as reference in the study of information spreading. In particular the SIR model is a standard choice since it is a model based upon few assumptions and can be characterized with one parameter, namely the *spreading probability* p. In our study of real file spreading in P2P networks, we have decided to simulate file spreadings and compare them with real data to assess the pertinence of this model. More precisely, we have estimated the spreading probability assuming the real data was generated by this model – to calibrate the model – and than we have performed the simulations. To do so, we have implemented a simple, discrete-time version of the standard SIR model as follows.

Given a network, represented by a graph, each file spreading corresponds to an independent epidemic in the graph, in which each node is in one of the following states: *susceptible*, *infected* or *non-interacting* (sometimes denoted *removed*, hence the acronym SIR). Susceptible nodes do not possess the file and may receive it from an infected node, thus becoming infected. Infected nodes, in turn, spread the file to each of its neighbors, independently, with probability p and become promptly non-interacting thereafter. Although non-interacting nodes remain in this state, infected nodes may unsuccessfully try to infect them sending the file. In this setting, the simulation inputs for each file spreading are:

- A spreading parameter $p \in [0, 1]$
- An underlying network represented by a graph $G = (V, E)$
- A list of initial providers (nodes which possessed the file at $t = 0$)
- A simulation bound (on time or on the number of infected nodes)

The simulation proceeds as follows: for each file F, we begin with the initial providers in an infected state and the other nodes in a susceptible state. At each step, infected nodes will infect each of its neighbors with probability p, becoming non-interacting afterwards. The epidemic continues as long as there are active infected nodes and as long as the simulation bound is not attained. The simulation output will be a list of spreading events, each represented by the following 4-tuplet: $(t\ P\ C\ F)$, where $t > 0$ is a timestamp, and the other three integers are unique ids for provider $P \in V$, client $C \in V$, and transmitted file, F. In other words, each entry in the output trace corresponds to a transmission of F by P to C at time t.

The artificial spreading trace generated by the simulation can be compared to the real trace, typically in terms of the number of infected nodes, and the maximum number of hops a file has done from its original source to the final destination. In fact from these traces it is possible to build a *spreading cascade* which is a directed graph capturing several quantitative and qualitative features of the diffusion. These cascades are the current focus of our study on real information diffusion.

4.2.1 Python-based simulator for greedy routing in the hyperbolic plane

IBBT developed a custom simulation environment in Python/C++ in support for greedy routing, having the following functionality:

- generation, loading and storing of network graphs
- implementation of several spanning tree algorithms for large graphs
- functionality to determine coordinates in the Hyperbolic plane (Poincare disk) for nodes in a graph, based on a spanning tree (greedy embedding in the hyperbolic plane)

- functionality to determine coordinates using arbitrary precision
- functionality to determine tree coordinates for nodes in a graph, based on a spanning tree (greedy embedding in a regular tree)
- failure evaluation and recovery mechanisms to evalute the robustness of embeddings
- functionality to evaluate the performance of several embeddings with respect the path length (stretch)
- functionality to simulate the distributed behavior of spanning tree algorithms

The graph-based algorithms were implemented on top of the existing Networkx-library, and numerical algorithms were built using the scientific packages Numpy and Scipy. Some parts of the code were optimized in C++ (either Cython or Weave) to allow for large scale network simulations. The evaluation of multiple floating point precision values was performed using the GNU MPFR C library developed by INRIA. The distributed simulation is based on the simulation functionality of the SimPy.

4.3 Emulation tools/platforms

4.3.1 Click Modular Router

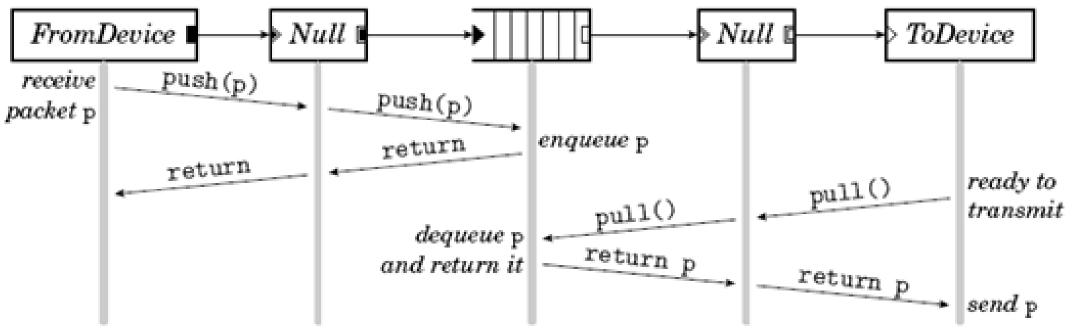

Figure 4.1: Click Element chain

The CLICK Modular Router is an open source platform developed in C++ that allows to let a Linux PC to act as a configurable router or switch. Therefore it is often referred as an emulation platform. The Click architecture is built in a modular way, such that the configuration of a router or switch boils down to the interconnection of a chain of connected modules, or elements which process packets or frames. The interconnection of the elements, which enables packet hand off between the different elements, is defined in what is called a click script. Elements can be of two types: push and pull. In push elements, frames are initiated by the source and pushed downstream, whereas in pull elements the trigger comes from the sink which pulls packets into its direction. Another aspect of elements is that they can have several incoming and outgoing such as to receive and send frames from and to different other elements. Finally elements can be configured through configuration strings at the moment their interconnection is defined in a click script. Once the click-script has been defined and started, elements can still be accessed or reconfigured through the use of handlers, which are a type of Unix sockets through which certain attributes at C++ class level can be tweaked.

The framework has a lot of pre-built elements, for examples for classification, queuing or even elements which perform Ethernet specific functions such as Spanning Tree control or a act as Ethernet learning switch. Besides pre-built elements, Click can be extended with custom made elements in C++. As the example below will illustrate, special elements called FromDevice and ToDevice are elements which enable to capture and send packets or frames to the standard Ethernet interfaces in Linux. The configuration of a click router or switch can be represented as a graph. In the next example the basic configuration of a simplified IEEE 802.1d bridge is shown. The typical packet flow, starting at the point of arrival from one of the Ethernet interfaces, is that it first undergoes a classification such as to distinguish between control and data plane frames, the first class is processed by the spanning tree element, the second class goes through a learning switch (Etherswitch). Before and after the learning elements called suppressors are used, such as to force that certain ports are blocked as a result of the spanning tree (this is also the reason why these elements are referenced in the configuration of the STP element).

Click has also proven to reach acceptable performance numbers, for example the paper ¡93¿Measuring Click¡92¿s Forwarding Performance¡94¿ by Felipe Huici has shown that a router running Click on commodity hardware can forward at rates of hundreds of Mbps even for minimum-sized packets. More specifically, an Opteron 2.0GHz computer with 2GB of memory and Intel Pro 1000MT Server adapters with polling drivers can forward at rates of as much as 361 Mbps, largely outperforming Linux native forwarding rates of approximately 265 Mbps under the same setup.

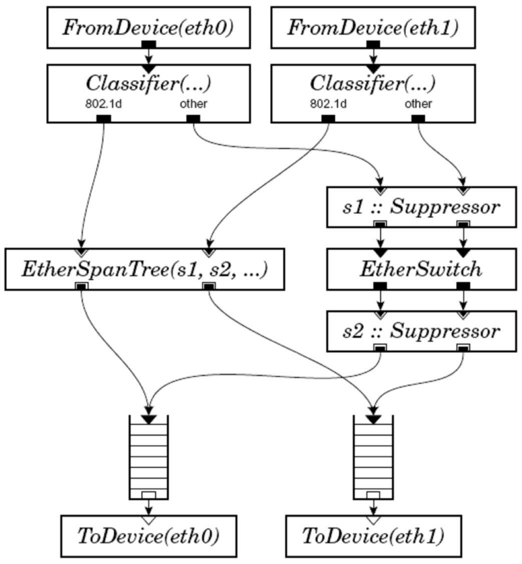

Figure 4.2: An Ethernet switch implemented in Click

There are currently two drivers that can run Click router configurations, a Linux in-kernel driver and a user-level driver that communicates with the network using Berkeley packet filters [McCanne and Jacobson 1993] or a similar packet socket mechanism. The user-level driver is most useful for profiling and debugging, while the in-kernel driver is good for production work.

One of the advantages of the Click platform is that click configurations can not only be used for emulation purposes, but also for simulation goals in combination with ns2. This setup, typically referred to as nsclick, can make use of several click nodes in a ns2 simulation test.

4.3.2 Quagga

Quagga is an open-source routing protocol suite providing implementations of different IP protocols such as Open Shortest Path First (OSPF), Routing Information Protocol (RIP) and Border Gateway Protocol (BGP). The software is developed in C and it is available for UNIX platforms, in particular Linux, Solaris and BSD.

The Quagga architecture consists of a core module (i.e., zebra), which acts as an abstraction layer to the UNIX kernel, and a set of client modules each one corresponding to the implementation of a specific routing protocol. Hence, the zebra module provides an API through which overlying routing protocol modules can access to the kernel routing table and network interfaces (Figure 4.3).

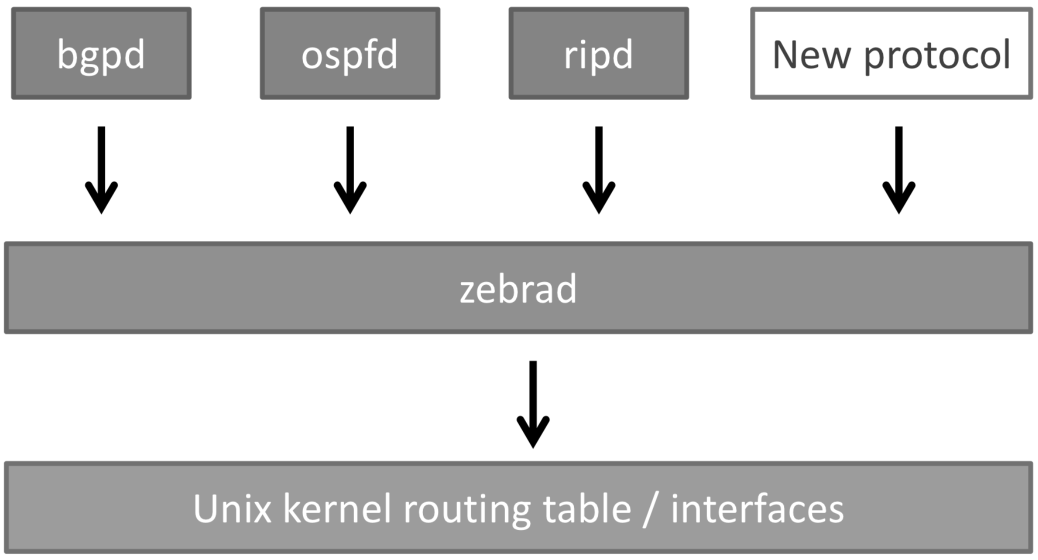

Figure 4.3: Quagga architecture: the zebra daemon (zebrad) interfaces the client routing protocol daemons (bgpd, ospfd and ripd) with the Unix kernel. New client protocol daemons can easily be incorporated in the architecture.

During bootstrap, the zebra daemon is started first and the desired routing protocol/s daemon/s is/are started afterwards. In this way, zebra is able to interface the communication between the routing daemons and the kernel, thus providing access to the routing table and network interfaces. For configuration, all daemons in Quagga are equipped with a command-line interface, which follows a CISCO IOS-like syntax. Alternatively, a pre-defined configuration file can also be used.

Quagga provides an implementation of OSPF in its versions 2 and 3, which fulfill the standard specifications defined in [RFC2328] and [RFC2740], respectively. While OSPFv2 is the standard version for IPv4, version 3 is the OSPF implementation for IPv6. Standard compliant implementations of RIPv1 [RFC1058], RIPv2 [RFC2453] and the RIP version for IPv6 (i.e., RIPng [RFC2080]) are also provided in the platform. The BGP-4 implementation of Quagga follows the specifications in [RFC4271]. Furthermore, this implementation incorporates the multi-protocol extensions defined in [RFC4760] in support of multicast and IPv6. Specifically, the Subsequent Address Family Identifier field defined in [RFC4760] contains the Network Layer Reachability Information (NRLI) used for multicast forwarding.

The Quagga software contains a rich development library to facilitate the implementation of protocol/client modules, coherent in configuration and administrative behavior. This library includes a set of implemented structures for easy data management (such as lists, hash tables, etc.), different structures to develop socket-based communications rapidly (IP, UDP or TCP) and a module to facilitate the implementation of finite state machines, which are the basic elements necessary to develop any routing protocol.

4.4 Other Experimentation support tools

4.4.1 *Sage*

Sage (see http://www.sagemath.org) aims to provide the arsenal mathematicians, researchers, and students need in order to perform calculations. The basic concept is to combine the power of many established software packages under a common Python-based interface. Even more than that, it provides powerful and unique algorithms in its own library. The mission of *Sage* is to *create a viable free open source alternative to Magma, Maple, Mathematica, and Matlab.*

EULER's interest in *Sage* comes from the combination of a large collection of state-of-the-art graph algorithms, including bridges with `NetworkX` (see http://networkx.lanl.gov/), a collection of algebra computation algorithms, and a simple interface with main linear programming solvers (GLPK, CBC, CPLEX, Gorubi). We use it both for research and educational purposes.

Sage was initially targeted to mathematicians and has now extended to a large community. It is used in computer science (e.g., algorithmics, combinatorics, graph theory, geometry, etc.) and in education. It is freely available and distributed under the General Public License. The community of users is large and distributions are downloaded approximately 6 500 times per month. The communities of both users and developers are animated through mailing-lists (about 50 messages per day for the developers community) and periodic workshops gathering developers and main users.

Through its large library of algorithms and user-interface, *Sage* constitutes an advance in the field of graph algorithms experimentation. By providing high-level building blocks, it facilitates the implementation of complex graph algorithms, thus allowing for rapidly testing a hypothesis. Furthermore, *Sage* allows for sharing the effort of optimizing the implementation among a community (e.g., one may propose a faster implementation of an existing algorithm).

Sage is mainly written in Python, Cython, and C by a worldwide community of developers (180 contributors for version 4.7.2 released on 2011-10-29, including 3 members of INRIA). It has around 300MB of source code. The user interface is in Python which makes it intuitive for beginners, while advanced users can also use Cython and C for implementing or optimizing demanding algorithms.

INRIA members have already contributed to more than 100 graph algorithms, added interfaces to specific linear programming solvers, and actively contributed to the improvement of the documentation. In particular, Nathann Cohen (former Ph.D. student of INRIA) wrote tutorials on graph theory and on linear programming in *Sage*, and he contributed to the French translation of the book entitled "A tour of *Sage*".

Sage provides a set of algorithms for generating graphs and testing properties. An updated list of available topology generators and property tests is documented in the online reference manual of *Sage* (http://www.sagemath.org/doc/reference/sage/graphs/graph.html), and a list of graph theory software included in or interfaced with *Sage* is documented at http://wiki.sagemath.org/graph_survey.

However, one should be aware of *Sage*'s limitations. In particular, the data structures chosen for simplifying both the implementation in Python and the ease of use (especially for beginners) are not fully optimized. Consequently, the memory consumption could be quite large, and some basic operations are slow compared to other software (e.g., the *Grph* software presented in Chapter 4.1.2 page 30). Furthermore, the size of graphs that can be manipulated with *Sage* is currently limited to 65535 ($2^{16} - 1$) nodes. The main argument for this limitation is that the computation time of quadratic (or more) algorithms on larger graphs is enormous and requires a careful control of the memory usage. Also, one should prefer a dedicated implementation in C.

Within our current and future plans, in collaboration with task T3.1, we aim at contributing to *Sage* by proposing our implementation of some missing topology property tests and new random graph generators for inclusion into future releases of *Sage*. This effort is also relevant in the context of the dissemination of the EULER's algorithmic results to a broader audience.

4.4.2 Experimentation support tools

4.4.2.1 AS-numbering and IP-addressing information generation

In order to perform emulation-based routing experimentation on a graph which only consists of node- and link descriptions, a tool needs to be developed to generate AS names, IP addresses and IP address ranges connectivity in relation to the graph's nodes. A Python-based tool will be implemented to generate this data based on a text file containing the graph topology in terms of links and nodes.

4.4.2.2 Configuration of virtual experiment topologies

Given the limited number of physical nodes (about 100) on the testbed infrastructure of EULER (ilab.t), virtualization techniques will be needed to emulate topologies having thousands of nodes, in order to experiment realistic network scenario's in the context of inter-AS routing.

In order to maximize the number of emulated routers on a given physical testbed node, network namespaces and Linux containers (LXC[2]) will be used as virtualization technology. The *network namespace*[3] is a private set of network resources assigned to one or several processes. These have their own set of network devices, IP addresses, routes, sockets and so on. Other processes outside of the namespace cannot access these network resources, neither know they exist.

That allows:

- virtualization : the processes inside the network namespaces do not know anything about the network resources outside the namespace and use the resources without conflicting with other network namespaces. For example:
  - several network namespaces can have eth0 and lo network devices.
  - several apache servers listening on *:80 can be launched into different network namespaces.
- isolation : the processes cannot access to the network resources which are outside the namespace. For example:
  - a process cannot sniff traffic related to another network namespace.
  - a process cannot shutdown an interface belonging to another network namespace.

The above virtualization technology allows to emulate tens to hundred[4] routers on a single physical testbed node.

A configuration tool is under development to deploy larger scale experiments on the virtual wall relying on LXC technology. Given a topological description, its related addressing, software and its configuration, the tool targets to automatically generate and execute virtualized network environments mapping to the topology given as input.

4.4.2.3 Experiment execution script support

Once the virtualized experimental testbed has be deployed, the real experiment can be started. The latter involves the evaluation of the implemented routing schemes with respect to their performance metrics as described in D4.1 under dynamically changing network conditions. For the purpose of reproduceability, experiment scripts will be used, as started from the control node (see Section 2.3.1).

The experiment script hardcodes the roadmap of the experiment in terms of: software to start and stop, topology dynamics, network traffic generation, wait instructions, logging actions

[2]http://lxc.sourceforge.net/

[3]description taken from the referred website

[4]using OLSR daemons, in the context of EULER, detailed scalability of this technology still needs to be evaluated with respect to Quagga router software

to be made, etc. The experiment script can rely on UNIX batch scripting, Python scripts or OMF[5]-based control.

[5]http://mytestbed.net/

Chapter 5

Conclusion

This deliverable documents the experimental systematic methodology that will be applied, the representative scenario that will be used, and the tools that will be executed to i) experiment the routing schemes developed in WP2, ii) evaluate their functionality and performance, and iii) compare their functionality and performance, in particular against BGP routing. These experiments include both simulation and emulation on scenarios. These scenarios to become handful for routing experiments must at least comprise the specification of a topology and an addressing model. Experiments involving traffic and thus the specification of traffic models, can be realized once the routing scheme itself is experimentally validated but involve the development of a forwarding model (in simulation experiments) and forwarding component (in emulation experiments).

As simulation and emulation experiments involve different levels of abstraction of the routing process, the specification of the routing experiments themselves will be specifically documented in Deliverable D3.5 and Deliverable D4.5. By keeping these scenarios independent of the type of experiment (simulation vs emulation), the EULER project aims at exploiting their complementarity.

EULER simulation and emulation experiments will also use a large variety of simulation and emulation tools. For this purpose we have grouped them into three categories (simulation tools/platforms, emulation tools/platforms, and other experimentation support tools) and document them by decreasing order of importance or generality. The specification/implementation of the tools are further reported in deliverable D4.3.