A Toolchain for Simplifying Network Simulation Setup*

Lorenzo Saino, Cosmin Cocora and George Pavlou
Department of Electrical and Electronics Engineering
University College London
London, UK
{l.saino, c.cocora, g.pavlou}@ee.ucl.ac.uk

ABSTRACT

Arguably, one of the most cumbersome tasks required to run a network simulation is the setup of a complete simulation scenario and its implementation in the target simulator. This process includes selecting a topology, provision it with all required parameters and, finally, configure traffic sources or generate traffic matrices.

Many tools exist to address some of these tasks. However, most of them do not provide methods for configuring network and traffic parameters, while others only support a specific simulator. As a consequence, a user often needs to implement the desired features personally, which is both time-consuming and error-prone.

To address these issues, we present the *Fast Network Simulation Setup (FNSS)* toolchain. It provides capabilities for parsing topologies from datasets or generating them synthetically, assign desired configuration parameters and generate traffic matrices or event schedules. It also provides APIs for a number of programming languages and network simulators to easily deploy the simulation scenario in the target simulator.

Categories and Subject Descriptors

C.4 [**Performance of Systems**]: Modeling Techniques; I.6.4 [**Simulation and Modeling**]: Model Validation and Analysis; I.6.5 [**Simulation and Modeling**]: Model Development—*Modeling methodologies*; I.6.7 [**Simulation and Modeling**]: Simulation Support Systems—*Environments*

General Terms

Design, Performance, Experimentation

Keywords

network topology, network simulation, link capacity, modeling, traffic matrix

*Source code, binaries and documentation of the FNSS toolchain is available at `http://fnss.github.com`

Simutools 2013, March 05-07, Cannes, France

DOI 10.4108/icst.simutools.2013.251735

1. INTRODUCTION

The setup of a realistic network simulation scenario is usually a lengthy and delicate process comprising various tasks.

The first task consists in selecting a suitable network topology. Such a topology can either be parsed from datasets of inferred topologies, such as RocketFuel [34] or the CAIDA AS relationships dataset [4] or synthetically generated according to various models, such as [17], [8], [7], [37] or [10]. Alternatively, it is also possible to use canonical topologies such as stars, rings or dumbbells.

Second, after selecting the topology, it is necessary to configure it with all required parameters to be used in the simulation. These include link delays, capacities, weights, buffer sizes and configuration properties of all protocol stacks.

Third, it is necessary to assign a traffic matrix to the topology or decide how the traffic will be modeled, such as deciding on the number of concurrent flows, their origin and destination and their characteristics.

Finally, all this configuration has to be implemented on the target simulator before simulations can be run.

The execution of all these tasks is cumbersome and error-prone, since there are no publicly available tools automating the entire process. In fact, although there are tools taking care of some of the tasks, such as the parsing or the generation of topologies, they do not support the entire setup chain and are generally bound to a specific simulator. As a result, a user is required to integrate heterogeneous software components or develop his/her own code to set up a complete simulation scenario.

Apart from possibly requiring a considerable amount of time, this process can also lead to an increased amount of mistakes affecting the reliability of simulations. In fact, the lack of a framework for automating simulation setup may lead users to configure network and traffic characteristics using unrealistic models. In addition, even if appropriate models are selected, defects may be introduced in their actual implementation. For all these reasons, it would be highly desirable having a tool supporting the entire setup chain.

To address these issues, we present here the *Fast Network Simulation Setup (FNSS)* toolchain. FNSS is simply a software library providing tools for easily executing all the tasks listed above. It allows users to parse topologies from various datasets or from other generators, as well as generate them according to the most common models. These topologies can then be configured with all required parameters and matched with appropriate traffic matrices or traffic source configurations. A fully configured simulation scenario can be

exported to a set of XML files which can then be imported by the desired simulator. FNSS provides adapters for the `ns-2` [5] and the `ns-3` [23] simulators as well as generic Java, C++ and Python APIs to enable an easy integration of FNSS with other simulators. In particular, by providing generic APIs for the most common programming languages, we hope to contribute to increase the reliability and reduce the setup complexity of simulations run with custom-built simulators, which are very common.

The methods provided by FNSS for generating and configuring network topologies are commonly used in literature, with the exception of those used for link capacity assignment. In fact, for this task, apart from providing commonly adopted models, we devised and implemented novel algorithms which we argue they provide a more realistic link capacity assignment than state-of-the-art methods. In this paper we present these new models and demonstrate their effectiveness by evaluating their performance on a number of real network topologies.

This FNSS toolchain is publicly available as open-source software and it is released under the terms of the BSD licence, with the exception of the ns-2 and ns-3 adapters, which are released under the terms of the GNU GPLv2 license in order to meet the requirements set by ns-2 and ns-3 licenses on the licensing of derivative work. This tool has already been extensively adopted in the context of the EU FP7 COMET project [14], where it has been used to set up and run the simulations whose results were presented in [32], [30] and [13]. Source code and documentation are available here.[1]

The remainder of this paper is organized as follows. Section 2 provides an overview of the related work. Section 3 describes the FNSS toolchain by explaining its architecture and design and illustrating its features. Section 4 introduces our new link capacity assignment algorithms and evaluates their performance. Section 5 presents a complete example of how FNSS can be used. Finally, conclusions are drawn in section 6.

2. RELATED WORK

Many tools have been presented in literature to address some of the issues related to the setup of network simulation scenarios. However, most of them only focus on the generation or parsing of topologies. Very little work has been done to support the entire setup process, being not tied to a single simulator.

The most common network simulators, such as ns-2 and ns-3 already provide support for parsing topologies from various sources, including RocketFuel and CAIDA datasets. However, these datasets merely consist of unconfigured network topologies. In fact, with the exception of a small subset of RocketFuel topologies which include estimations of link weights [27], all remaining network and traffic parameters are not known and, therefore, they need to be manually provisioned in the simulator.

The situation is similar for synthetically generated topologies. There are many freely-available tools capable of generating large scale topologies in accordance to the most common models. Such tools include BRITE [28], Inet [38], GT-ITM [1] and aSHIIP [35]. However, also in this case, generated topologies cannot be used without further configuration. The only exception is BRITE which includes a method for assigning link capacities and delays. However, as we will explain in more detail in section 4, its capacity assignments are unrealistic and, anyway, the assignment of link weights, protocol stacks and traffic characteristics are not supported in any of these tools.

To the best of our knowledge, the only work attempting to move in a similar direction of FNSS is represented by the *NSF Frameworks for ns-3* project [2], although its scope is widely different. This project aims at building a framework for automating the network simulation and data collection in ns-3. Its objectives are only marginally overlapping with FNSS. In fact, while the *NSF Frameworks for ns-3* project aims at building a complete framework for repeating experiments with different parameters and collecting results, which is outside FNSS scope, it only targets the ns-3 simulator while FNSS aims at being a cross-platform tool, targeting also custom-built simulators. In addition, scenario generation code from *Frameworks for ns-3* is not available yet to the community, except for a BRITE parser for ns-3. Finally, *Frameworks for ns-3* does not include traffic matrices generation, as it does not address flow-level simulations.

3. THE FNSS TOOLCHAIN

3.1 Architecture and design

The FNSS toolchain comprises a core library and several adapters. The core library implements all functionalities required to parse or generate topologies, configure them, generate traffic matrices and event schedules and save them in XML files. The adapters are separate modules capable of parsing the XML files generated by the core library and use them to deploy a simulation scenario in the target simulator.

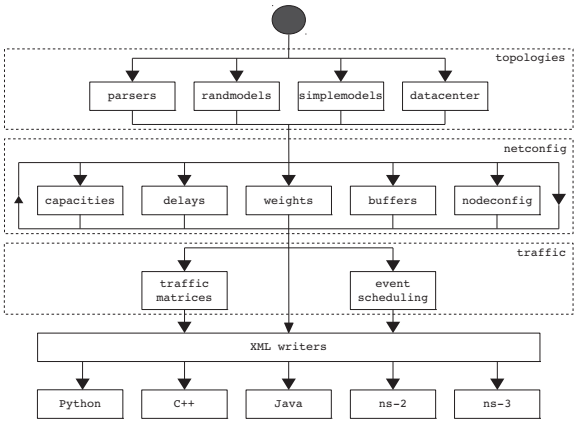

Figure 1: FNSS workflow

The core library is entirely written in Python, with some of its functionalities built on top of the NetworkX library [22]. We selected Python as the programming language for two main reasons. First, its high-level programming constructs would allow FNSS users to generate complex network simulation scenarios with a few lines of code. Second, the availability of NetworkX, a well-designed and actively-maintained library for graph manipulation and visualization, provides useful tools to manipulate topologies created with FNSS.

[1] `http://fnss.github.com`

As depicted in figure 1, which shows the workflow on which FNSS design is based, the core library has been implemented following a modular architecture. All core functionalities are implemented in 11 modules, contained in 3 packages, namely `topologies`, `netconfig` and `traffic`. The code functionalities are allocated to the various packages as follows:

- `topologies`: contains all functions allowing users to parse or synthetically generate network topologies and export them to an XML file.
- `netconfig`: contains all functions required to assign configuration parameters to network topologies, precisely: link capacities, link weights, link delays, buffer sizes, protocol stacks and applications.
- `traffic`: contains functions for synthetically generating traffic matrices and event schedules and to export them to XML files.

In the core library, the entities required to represent a simulation scenario are modeled with objects belonging to three categories:

- **Topologies:** they represent fully configured network topologies. There are three different topology classes: `Topology`, `DirectedTopology` and `DatacenterTopology`.
- **Traffic matrices:** they represent traffic matrices. A static traffic matrix is represented by a `TrafficMatrix` object, while a dynamic traffic matrix is represented by `TrafficMatrixSequence` object.
- **Event schedules:** they represent a schedule of events labeled with an execution time. In the core library, the class modeling schedules of events is called `EventSchedule`

All these objects can be serialized and saved in XML files, which can then be parsed by the adapter for the desired target simulator. Currently FNSS provides adapters for ns-2, ns-3 as well as for other generic simulators through the C++, Java and Python APIs. However, the strategies used to support the various simulators are different. Supported simulators are integrated as follows:

- **ns-2:** The ns-2 adapter is a Python script which takes as input a topology (in the form of either an XML file or an FNSS Python object) and converts it into a Tcl script containing all the code needed to set up the topology in ns-2.
- **ns-3:** The ns-3 adapter comprises a set of C++ classes which parse topology and event schedule XML files, deploy the configuration specified in the topology file and schedule the execution of the tasks specified in the event schedule in the ns-3 environment.
- **Generic Java, C++ or Python simulator:** The generic APIs provide capabilities to parse the XML files of topologies, traffic matrices or event schedules and convert them into objects for the target language.

In the remainder of this section, we will describe in more detail all functionalities provided by FNSS.

3.2 Network Topologies

FNSS allows users to create topologies in a variety of different ways:

- **Import a topology from a dataset:** topologies can be parsed from the RocketFuel ISP maps [34], the CAIDA AS relationships dataset [4], the Topology zoo dataset [26] and the Abilene network topology [3].
- **Import a topology from other topology generators:** FNSS supports the import of topologies generated using BRITE [28], Inet [38] and aSHIIP [35].
- **Generate a synthetic random topology:** the models supported are Barabási-Albert [8], extended Barabási-Albert [7], Erdős-Rényi [17], Waxman [37] and Generalized Linear Preference (GLP) [10].
- **Generate a datacenter topology:** the models supported are two- and three-tier [9], fat tree [6] and BCube [21].
- **Generate a simple topology:** FNSS supports the generation of the following canonical topologies: k-ary tree, dumbbell, line, star, ring and full mesh.

A parsed or generated topology is an instance of either the `Topology`, `DirectedTopology` or `DatacenterTopology` classes, all which extend NetworkX `Graph` class. This design decision makes it possible to directly use the graph algorithms provided by the NetworkX library on such topologies.

In the following section we explain in more detail the synthetic and datacenter topology models provided by FNSS and discuss how to select appropriate model parameters.

3.2.1 Synthetic models

FNSS can create synthetic topologies according to the most common models. These are:

- **Erdős-Rényi [17]:** it generates simple random topologies $G(n, p)$, where n is the number of nodes and p is the probability that a pair of nodes is connected by an edge.
- **Waxman [37]:** it generates topologies in which the probability that two nodes are connected by an edge depends on their distance. Such probability is equal to:

$$p(u, v) = \alpha e^{-d(u,v)/\beta L} \quad (1)$$

  where $\alpha, \beta \in (0, 1]$ are required model parameters and $L > 0$ is the maximum distance between two nodes. By increasing the value of α raises the edge density, while decreasing the value of β makes the probability function decay faster, resulting in a greater density of short edges in comparison to long ones.

  In FNSS, topologies generated with this model have edges annotated with their Euclidean distance, which makes it possible to later assign link weights and delays accordingly.
- **Barabási-Albert [8]:** it generates scale-free networks whose node degrees are power-law distributed. Such topologies are created by adding nodes according to a *preferential attachment* model, whereby new nodes

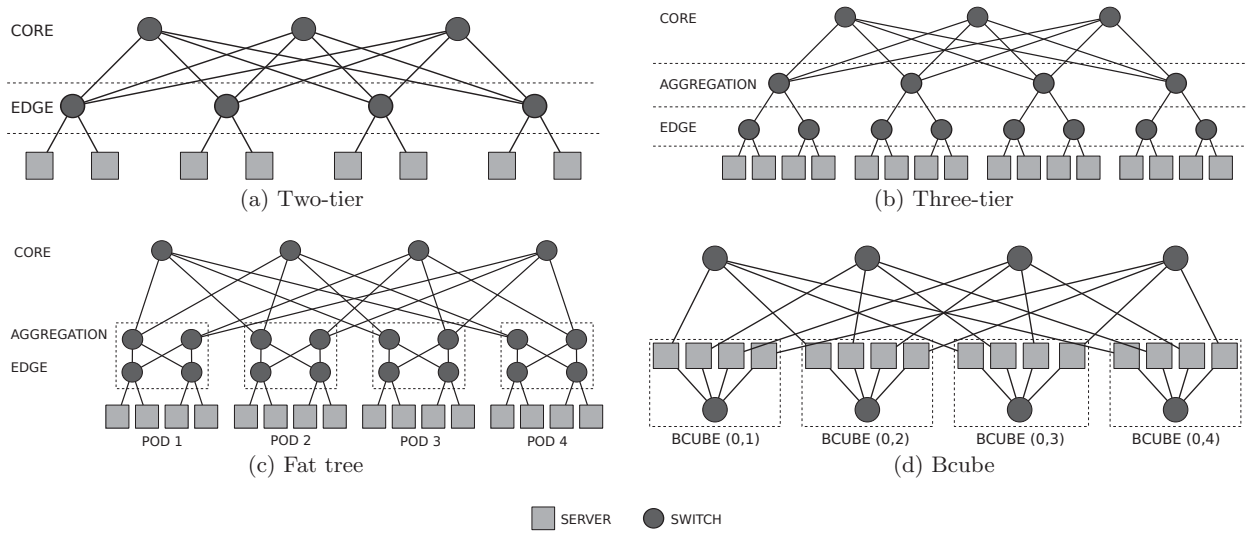

Figure 2: Models of datacenter topologies provided by FNSS

joining the network are more likely to attach to nodes with a greater degree. In such a topology, the probability that a node has degree k is k^{-3}, irrespective of its size and of the value of model parameters. Topologies generated with this model, as well as with other models that provide scale-free graphs, are generally well-suited to model AS-level Internet topology, since, as demonstrated by [19], Internet topology exhibits power-law characteristics.

- **Extended Barabási-Albert [7]:** it is an extension of the Barabási-Albert model that takes into account the presence of local random events such as the addition of nodes or links or the rewiring of existing links in the construction of the topology. The result is still a scale-free topology, however the exponent of the node degree distribution varies.
- **Generalized Linear Preference (GLP) [10]:** it is another model capable of generating scale-free networks in a similar manner to the two models previously presented. It is a small extension of the extended Barabási-Albert model that comprises an additional parameter $\beta \in (-\infty, 1)$ which allows users to fine-tune the intensity of the preferential attachment. Decreasing the value of β reduces the preference given to high degree nodes for attachment.

3.2.2 Datacenter models

The dramatic increase in the popularity of cloud computing services has triggered a considerable amount of research focusing on datacenter issues. As a result, novel topologies have been proposed in order to achieve specific objectives, such as performance improvement, cost reduction or simplification of wiring. In order to address the increasing need for tools to evaluate datacenters, FNSS has been equipped with functionalities for generating the most common datacenter topologies. The topologies, depicted in figure 2, are:

- **Two- and three-tier [9]**: these are arguably the most common datacenter topologies. They are characterized by the fact that switches are organized in two tiers (core and edge) or three tiers (core, aggregation and edge). Three-tier topologies are generally capable of supporting hundreds of thousands of servers, while two-tier topologies can only support up to five to eight thousand servers.
- **Fat tree [6]:** it is a recently proposed topology, designed to build large-scale datacenter using only commodity k-port switches appropriately interconnected in a way similar to a Clos network [15].
- **BCube [21]:** it is a topology specifically designed for shipping-container based, modular datacenters. In this model, switches are never directly connected to each others and servers also accomplish packet forwarding functions.

3.3 Topology Configuration

3.3.1 Link Capacities

As briefly mentioned in section 1, with respect to link capacity assignment, FNSS implements both commonly used and newly proposed methods.

The common methods implemented by FNSS are:

- **Constant capacity**: all links are assigned a fixed user-defined capacity.
- **Manual assignment**: the user can assign specific capacities to selected links.
- **Random capacity assignment**: the user can assign capacities randomly. To use this model, the user needs specify a set of allowed capacities and the probability of each capacity being assigned to a link. Apart from allowing users to manually specify the Probability Density Function (PDF), FNSS provides utility methods for assigning capacities according to uniform, power-law and Zipf-Mandelbrot distributions.

The newly proposed models allow users to assign link capacities proportionally to a number of topological centrality metrics. These models are formally introduced and evaluated in section 4.

3.3.2 Link Delays

Link delays can be assigned in three different ways:

- **Geo-distance**: if the topology has been parsed from a dataset providing the geographical coordinates of the nodes, it is possible to assign link delays proportionally to the estimated length of the link, calculated as the Euclidean distance between the two endpoints. The user can specify the desired value of specific propagation delay to model different types of transmission media.
- **Constant delay to all links**: all links are assigned the same delay.
- **Manual delay assignment**: the user can manually select a link or a set of links and assign them a specific delay.

3.3.3 Link Weights

Link weights can be assigned in four different ways:

- **Proportionally to the inverse of link capacity**: the weight assigned to each link is proportional to the inverse of its capacity.
- **Proportionally to link delay**: the weight assigned to each link is proportional to its delay.
- **Constant weight to all links**: all links are assigned the same weight (default is 1).
- **Manual weight assignment**: the user can manually select a link or a set of links and assign them a specific weight.

In addition, if the topology has been parsed from a dataset already providing link weight information, such as the Abilene topology [3], weights are automatically assigned.

3.3.4 Buffer Sizes

Buffer sizes can be assigned in four different ways:

- **Equal to network bandwidth-delay product:** This method follows the rule of thumb, originally proposed by [36], that the buffer sizes of Internet routers should be set to the average RTT experienced by TCP flows traversing them multiplied by the link bandwidth. In FNSS, this is calculated as follows:
  1. Identify all shortest paths traversing the specific buffer.
  2. Calculate the RTT for each of these paths by summing the propagation delay of each link in the path.
  3. Average the RTT and multiply it by the capacity of the link associated to the buffer.
- **Proportionally to link capacity**: In case link propagation delays are not known or are set to 0, it is possible to assign a buffer size equal to $k \times C$ where C is the capacity of the link and k is a constant.
- **Constant buffer size**: All buffers in the topology are assigned a fixed size.
- **Manual assignment**: A user can manually assign specific sizes to a single buffer or a set of buffers.

3.3.5 Protocol Stacks and Applications

As already mentioned above, FNSS provides the capability to preconfigure protocol stacks and applications and deploy them on selected nodes of a topology. Each types of stacks and applications are identified by a specific name (e.g. `http_server`) and a set of key-value attributes stored in a dictionary object. For example, an HTTP server application may have the following properties:

```
server_props = {
    'port': 80,
    'max_threads': 200,
    'avg_obj_size': '150KB'
    'conn_keep_alive': True
}
```

The application thus defined is appended to the selected node(s) of the topology object. When the topology is saved to an XML file, all stack and application information is saved as well so that it can be easily imported by the target simulator.

3.4 Event Scheduling

Another feature of FNSS is the ability to produce event schedules and export them to an XML file. An event schedule is represented in the core library by an object of the `EventSchedule` class. Such a schedule is a sorted list of events labeled with an execution time. Each event, similarly to applications and protocol stacks, is modeled as a dictionary of key-value attributes.

For example, an HTTP request event could be represented by the following set of attributes:

```
event = {
    'client_ip':   '192.168.1.24'
    'proxy':       '192.168.1.100:8080'
    'method':      'GET'
    'url':         'http://www.ucl.ac.uk/'
    'User-Agent':  'fnss-client'
    'Connection':  'keep-alive'
}
```

3.5 Traffic Matrices

FNSS is capable of synthetically generating traffic matrices with given statistical characteristic which can be used for network simulations.

A Traffic Matrix (TM) is a representation of aggregate traffic volumes being carried by a network at a specific time interval, whose elements T_{ij} represent the average traffic volume from ingress node i to egress node j.

There exist two types of traffic matrices: *static* or *dynamic*. A static traffic matrix reports the traffic volumes collected at a single point in time. Differently, a dynamic matrix contains a sequence of traffic volumes collected at different times.

In a communication network, traffic volumes follow diurnal patterns [33]. Therefore, to appropriately model network traffic over long time spans, it is recommendable to adopt dynamic traffic matrices which are cyclostationary over a

period of 24 hours. However, it has been observed [29] that over shorter timescales, generally under one hour, traffic variations can be accurately modeled using just stationary dynamic traffic matrices. Static traffic matrices, instead, can only be reliably used to model network traffic at a single point in time.

From a mathematical perspective, *cyclostationary*, *stationary* and *static* traffic matrices can be represented as follows. A cyclostationary traffic matrix can be represented as:

$$T_{ij}(t) = X_{ij}(t) + W_{ij}(t) \quad (2)$$

Where $X_{ij}(t)$ is the mean traffic volume from i to j, periodic of period $T = 24h$ and $W_{ij}(t)$ is a zero-mean random variable modeling random fluctuations of traffic volumes.

A stationary traffic matrix can be represented as:

$$T_{ij}(t) = X_{ij} + W_{ij}(t) \quad (3)$$

Where X_{ij}, unlike the cyclostationary model, is time-invariant.

Finally, a static traffic matrix can be represented as:

$$T_{ij}(t = t_0) = X_{ij} \quad (4)$$

where, since the matrix comprises a single set of traffic volumes, there are no volume fluctuations over time.

FNSS is capable of synthetically generating both static, stationary and cyclostationary traffic matrices. This is realized following the process proposed by Nucci *et al.* in [29], which comprises the four following steps.

First, we generate the mean rates for all flows between each ingress and egress node X_{ij}. Mean volumes are realizations of a lognormal random variable $ln\mathcal{N}(\mu, \sigma^2)$. It should be noticed that at this step we simply generate as many mean values of traffic volume as the number of Origin-Destination (OD) pairs in the network but we do not map them to specific OD pairs yet.

Second, in case a dynamic (either stationary or cyclostationary) traffic matrix is desired, we generate random fluctuations for each OD pair. These fluctuations are realizations of a zero-mean normal random variable. As shown in [29] and [11], the standard deviation of such fluctuations σ and the mean traffic volumes are related by the following power law:

$$\overline{x}_{i,j}(t) = \psi \sigma_{ij}^{\gamma} \quad (5)$$

Nucci *et al.* [29] reported that in Sprint Europe and Abilene [3], this assumption about power-law relationship holds. In particular, the values that best fit the distribution are ($\gamma = 0.8$, $log\psi = -0.33$) for Sprint and ($\gamma = 0.93$, $log\psi = -0.31$) for Abilene.

Third, in case a cyclostationary traffic matrix is required, traffic volumes are multiplied by a one-mean sin function oscillating between $1-\delta$ and $1+\delta$ in order to simulate diurnal traffic oscillations.

Fourth, we assign traffic volumes to the target topology. This assignment is executed using the *Ranking Metrics Heuristic* proposed by [29]. This method comprises the following steps:

1. Sort all volumes X_{ij} in decreasing order
2. Sort all OD pairs in decreasing order of metric m_1:
$$m_1(n_1, n_2) = min(F_{out}(n_1), F_{in}(n_2)) \quad (6)$$
where F_{out} and F_{in} respectively the fan-out and fan-in capacity of a node.
3. If a tie occurs, subsort OD pairs in decreasing order of metric m_2:
$$m_2(n_1, n_2) = min(outdeg(n_1), indeg(n_2)) \quad (7)$$
4. If another tie occurs, subsort OD pairs in decreasing order of metric m_3:
$$m_3(n_1, n_2) = \frac{1}{max(NFUR(n_1), NFUR(n_2))} \quad (8)$$
where NFUR stands for *Number of Flow Under Failure* and corresponds to the maximum number of shortest paths traversing a node if one random link of the network is down.
5. Map sorted traffic volumes to sorted OD pairs

Finally, we calculate the utilization of each link assuming that shortest path routing is used and linearly scale traffic volumes to match the expected maximum link utilization.

In FNSS implementation, a static traffic matrix can be generated by invoking the function `static_traffic_matrix (topology, mean, stddev, max_u)` where `topology` is the topology for which the matrix is generated, `mean` and `stddev` are respectively the values of μ and σ used to generate the mean flows and `max_u` is the target maximum link utilization used for scaling traffic volumes. The function to be invoked for generating dynamic (stationary) matrices is `stationary_traffic_matrix(topology, mean, stddev, gamma, log_psi, n, max_u)` where `n` is the number of time intervals and `gamma` and `log_psi` are respectively the values of γ and $log\psi$ used to derive the standard deviation of random fluctuations. Finally, a cyclostationary traffic matrix can be generated by invoking the function `sin_cyclostationary _traffic_matrix(topology, mean, stddev, gamma, log_ psi, delta, n, periods, max_u)` where `delta` is the value δ of the sin function, `n` is the number of traffic samples per period and `periods` in the number of periods spanned by the matrix.

4. LINK CAPACITY ESTIMATION

While a large amount of work is available in literature regarding the modeling ([8], [7], [37], [10]) or the inference ([34], [16], [20]) of Internet topologies, almost no work exists on inference and modeling of link capacity distribution.

With respect to link capacity inference, although a number of methods for inferring link capacities have been proposed ([12], [25]), their practical applicability is limited because it is hard to accurately measure packet timing or congest network links to the point required for measuring link capacity before the ISP reacts [39].

Differently, with respect to link capacity modeling, to the best of our knowledge, the only work investigating properties of link capacity distribution in an ISP network is [24]. In this paper, the authors show that, in the backbone network of the Internet Initiative Japan (IIJ) ISP, the number of links with the same capacity follows a power law. The authors speculate that this relation could be due to the

fact that by allocating link capacities according to a power law would yield better network throughput performance in comparison to other distributions. These results, however, cannot be considered conclusive, first because they have been only validated on a single topology and second because the justifications proposed are not convincing.

The lack of commonly accepted models for link capacity distribution has resulted in various heterogeneous methodologies being adopted in practice.

The BRITE topology generator, for example, can assign either a constant user-defined capacity to all links or assign capacities randomly. In the latter case, the user can specify a minimum and a maximum capacity BW_{min} and BW_{max} and a distribution (uniform, exponential or Pareto) and the BRITE tool assigns to each link a random capacity $C = [BW_{min}, BW_{max}]$ according to the selected distribution. This method, however, cannot generate realistic capacity assignments. In fact, in this model link capacities can take any real value between minimum and maximum capacities while links in real networks can only have a discrete number of different capacities.

Another common approach ([39], [31]) is to select a set of discrete capacities and repeat simulations using different constant capacities or assign heterogeneous capacities according to various distributions and evaluate the sensitivity of results.

The use of this large variety of not validated models can have a negative impact on the reliability of simulations whose results are sensitive to capacity assignment. To address this problem, in the following sections, we analyse the problem of modeling link capacity distributions and propose novel methods for modeling the link capacity assignments of ISP backbone networks. Our evaluation, presented in section 4.2, shows that our methods provide more realistic assignments than commonly used random assignments.

4.1 Proposed solution

The problem we are addressing in this section is the following. Let $G(E, V)$ be a graph with vertices $V = \{v_1, v_2, ...v_n\}$ and edges $E = \{e_1, e_2, ..., e_m\}$ representing the topology of an ISP backbone network whose edges have capacities belonging to the set $C = \{c_1, c_2, ..., c_p\}$, with $|C| \ll |E|$, assigned according to $f_c : E \to C$. The objective is to find the most realistic estimate $\hat{f}_c$ of f_c, assuming that no additional information about the network apart from G and C is available.

Our intuition is that, in ISP backbone networks, capacities are not allocated to links randomly but according to various economic, technological and traffic constraints, including the *importance* of the links, which are also related to the *importance* of the nodes they connect. For this reason, we speculate that there should be some correlation between the amount of capacity assigned to a specific link and various metrics capturing the importance of the link itself or the link's endpoints. Our argument is therefore that important links are more likely to have greater capacities.

Based on this assumption, the solution proposed here is to identify a set of metrics m capable of capturing links importance and to assign link capacities c which are linearly proportional to the value of such metric. As a result, the capacity assigned to each link (u, v) is:

$$c(u, v) = c_i \in C : b_{i-1} \leq \tilde{c}(u, v) < b_i \tag{9}$$

where:

$$\tilde{c}(u, v) = b_{|0|} + (b_{|C|} - b_0)\frac{m(u, v) - m_{MIN}}{m_{MAX} - m_{MIN}} \tag{10}$$

and:

$$b_x = \begin{cases} c_1 - \frac{c_2 - c_1}{2}, & \text{if } x = 0 \\ \frac{c_i + c_{i+1}}{2}, & \text{if } 1 \leq x \leq |C| - 1 \\ c_{|C|} + \frac{c_{|C|} - c_{|C|-1}}{2}, & \text{if } x = |C| \end{cases} \tag{11}$$

In our analysis, we focus on three specific metrics m to capture the importance of edges in a graph. Such metrics are the *edge betweenness centrality*, the *degree centrality gravity* and the *communicability centrality gravity*.

The edge betweenness centrality $c_B(e)$ corresponds to the number of shortest paths passing through a specific edge, formally defined as:

$$c_B(e) = \sum_{s,t \in V} \frac{\sigma(s, t|e)}{\sigma(s, t)} \tag{12}$$

where V is the set of nodes, $\sigma(s, t)$ is the number of shortest (s, t) paths, and $\sigma(s, t|e)$ is the number of those paths passing through edge e. In our problem, we assume that no information about link weights or routing tables is available. Therefore shortest paths are calculated using unitary link weight.

The degree centrality gravity of a link G_{C_D} corresponds to the product of the degree centralities (i.e. the number of neighbors) of the link's endpoints u and v.

If there are unidirectional links in the network, this is calculated as:

$$G_{C_D}(u, v) = \text{outdeg}(u) \times \text{indeg}(v) \tag{13}$$

Where u and v are, respectively, the egress and ingress nodes of the link. Otherwise, it is calculated as:

$$G_{C_D}(u, v) = \deg(u) \times \deg(v) \tag{14}$$

The communicability centrality gravity of a link corresponds to the product of the communicability centralities of the link's endpoint. The communicability centrality of a node [18], sometimes referred to as *subgraph centrality* corresponds to the number of distinct closed walks passing through that node. This metric captures how well connected is a node within a subgraph and is formally defined as:

$$G_{SC}(u, v) = \sum_{j=1}^{N} (v_j^u)^2 e^{\lambda_j} \times \sum_{j=1}^{N} (v_j^v)^2 e^{\lambda_j} \tag{15}$$

where v_j is an eigenvector of the adjacency matrix A of G corresponding to the eigenvalue λ_j.

We implemented functions in FNSS to calculate these three metrics (edge betweeness centrality, degree centrality gravity and communicability centrality gravity) and assign link capacities so that the capacity of each link is proportional to the value of the selected metric associated to the link, as expressed in equation 9.

4.2 Performance evaluation

We measure the performance of the proposed algorithms on five different backbone networks whose topology and link

Table 1: Network topologies

Topology	$\|V\|$	$\|E\|$	$\|C\|$	$\|E\|/\|V\|$	$\|E\|/\|C\|$
GEANT	23	38	4	1.65	9.5
GARR	42	51	7	1.21	7.29
Uninett	66	91	3	1.38	30.33
WIDE	29	30	2	1.03	15
RedIris	19	31	5	1.63	6.2

capacity assignments are known. These networks, also listed in table 1, are:

- **GEANT:** European academic network
- **WIDE:** Japanese academic network
- **GARR:** Italian academic network
- **RedIris:** Spanish academic network
- **Uninett:** Norwegian academic network

A first evidence of the existence of a correlation between capacities and centrality metrics is provided by the analysis of the Pearson's r between these two values. This evaluation, whose results are reported in figure 3, shows that r is always positive and in most cases its value is greater than 0.3, which indicates a medium to strong correlation between the two variables.

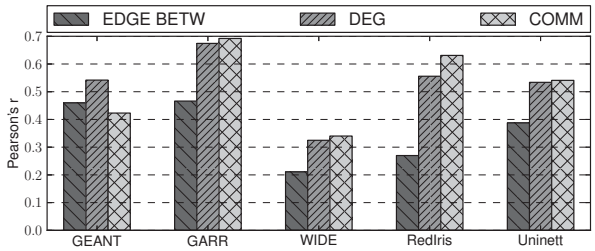

Figure 3: Pearson's r between link capacity and centrality metrics

To further validate the performance of our methods, we assign link capacities according to the algorithms presented above and measure how close our estimation is from real capacity assignments. Such closeness is measured with two metrics.

The first metric used, which we called *Matching Capacity Ratio* (MCR), measures the number of links whose estimated capacity corresponds exactly to the real capacity, normalized by the total number of links. This metric is formally defined as:

$$MCR = \frac{|\{e \in E | \hat{f}(e) = f(e)\}|}{|E|} \tag{16}$$

where E is the set of links, $f(e)$ and $\hat{f}(e)$ are, respectively, the real and assigned link capacity.

The second metric, which we called *Capacity Rank Error* (CRE), captures the Root-Mean-Square Error (RMSE) between real and assigned capacities, and is defined as:

$$CRE = \frac{1}{|C|} \times \sqrt{\frac{1}{|E|} \sum_{e \in E} \left[R(\hat{f}_e) - R(f_e) \right]^2} \tag{17}$$

where C is the set of capacities and $R(x)$ is a function for values $x \in C$ such that $R(x) = i \Leftrightarrow x = c_i$, that returns the rank of a capacity value in the set of capacities C. For example, if the set of capacities C comprises 10, 40 and 100 Mbps and a link has capacity $c = 10Mbps$, then $R(c) = 1$, if $c = 40Mbps$, $R(c) = 2$ and if $c = 100Mbps$, $R(c) = 3$. So, if on a link whose real capacity is $100Mbps$ the algorithm assigns a capacity $\hat{c} = 40Mbps$, then $R(\hat{c}) - R(c) = 1$, while if it assigns assigns a capacity $\hat{c} = 10Mbps$, then $R(\hat{c}) - R(c) = 2$

The performance of our algorithms, with respect to the two metrics defined above, is shown in figures 4 and 5, where they are compared with random assignments carried out using a uniform distribution. The RAND value represented in both figures refers to the median value of the metrics over 10^4 random assignments.

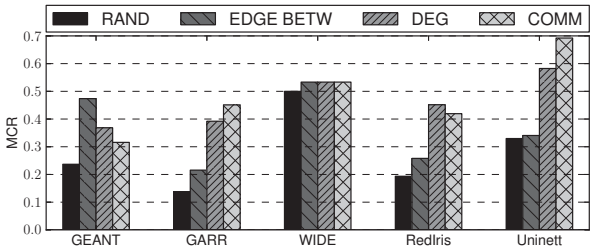

Figure 4: Matching Capacity Ratio (MCR)

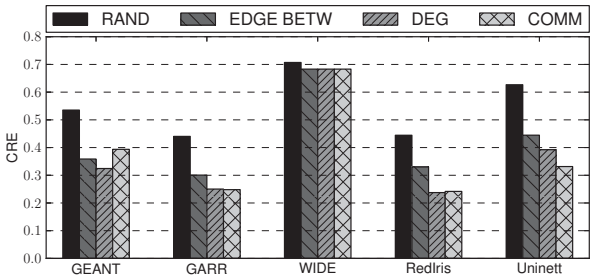

Figure 5: Capacity Rank Error (CRE)

These results show that in all cases considered, our algorithms yield better performance than random assignment, in terms of both MCR and CRE.

The smallest improvements are noticed in the WIDE topology. The reason justifying this result is that this topology comprises 30 edges, 16 of which having one capacity and 14 another one. Therefore, in this particular case, uniform random assignment, which assigns on average each capacity to 15 edges, can achieve reasonably good performance. In all other networks, link capacities are not distributed as uniformly as in WIDE, therefore the performance upper bound of uniform random assignments is lower. Anyway, it should be noticed that even in this case, although the number of link with a given capacity fits very well a uniform distribution, our algorithms still perform better under both metrics considered.

Although further work is required to validate the effectiveness of these models, we think that these early results are encouraging and that these models could be successfully adopted to improve the reliability of network simulations. In any case, further studies of these models will be part of our future work.

In conclusion, we are confident that our algorithms could be used by network researcher to assign more realistic link capacities to networks with known topologies but unknown capacities, like inferred topologies such as those of the RocketFuel dataset.

5. EXAMPLE OF UTILIZATION

We report in this section a simple example that demonstrates how different features of the FNSS toolchain can be used to create a configured network topology and generate a traffic matrix. This specific example shows how to use the core Python library to parse a topology from the RocketFuel dataset, configure capacities, delays, weights and buffer sizes, generate a cyclostationary traffic matrix and save both topology and traffic matrix on XML files.

First, we import all functions of the FNSS core library:

```
from fnss import *
```

Then we parse a topology from the RocketFuel dataset:

```
topo = parse_rocketfuel_isp_map("file.cch")
```

At this point, we configure the parsed topology. We assign capacities of 1, 10 and 40 Gbps proportionally to edge betweenness centrality, weights proportionally to the inverse of the capacities, constant delays of 2ms and, finally, buffer sizes equal to the bandwidth-product delay.

```
C = [1, 10, 40]
set_capacities_edge_betweenness(topo, C, 'Gbps')
set_weights_inverse_capacity(topo)
set_delays_constant(topo, 2, 'ms')
set_buffer_sizes_bw_delay_prod(topo)
```

After fully configuring the network topology, we generate a cyclostationary traffic matrix with 7 periods of 24 samples.

```
tm = sin_cyclostationary_traffic_matrix(
       topo, mean=0.5, stddev=0.05,
       gamma=0.8, log_psi=-0.33, delta=0.2,
       n=24, periods=7, max_u=0.9
       )
```

Finally, we export topology and traffic matrix objects to XML files in order to be imported in the preferred target simulator.

```
write_topology(topo, 'topology.xml')
write_traffic_matrix(tm, 'traffic-matrix.xml')
```

6. CONCLUSIONS

This paper provided two main contribution.

First, it presented the the *Fast Network Simulation Setup (FNSS)* toolchain, a comprehensive library allowing network researchers and engineers to simplify the execution of the tasks required to generate a scenario for a network simulation. This library allows to generate topologies or import them from datasets, configure them with all required parameters and generate event schedules and traffic matrices. We plan to further enhance this toolchain as part of future work, by implementing more core feature and supporting more simulators, such as Omnet++.

Second, this paper proposed novel methods for modeling the distribution of link capacities in a backbone ISP network. These methods assign link capacities proportionally the values of a number of link centrality metrics. Although the design of these methods is simple, their accuracy, evaluated on a set of real network topologies, significantly outperformed the most commonly used models. Further refinements of these methods and a more comprehensive evaluation will be carried out as part of future work.

7. ACKNOWLEDGMENTS

This work was funded by the COMET project (ICT-248784) and by the Flamingo project (ICT-318488), both supported by the European Commission under its Seventh Framework Program.

8. REFERENCES

[1] GT-ITM: Georgia Tech Internetwork Topology Models. `http://www.cc.gatech.edu/projects/gtitm/`.

[2] Project Summary: CI-ADDO-EN: Frameworks for ns-3. `http://www.eg.bucknell.edu/~perrone/research-docs/NSFProjectSummary.pdf`.

[3] The Abilene topology and traffic matrices dataset. `http://www.cs.utexas.edu/~yzhang/research/AbileneTM/`.

[4] The CAIDA AS Relationships Dataset. `http://www.caida.org/data/active/as-relationships/`.

[5] The Network Simulator NS-2. `http://www.isi.edu/nsnam/ns/`.

[6] M. Al-Fares, A. Loukissas, and A. Vahdat. A scalable, commodity data center network architecture. In *Proceedings of the ACM SIGCOMM 2008 conference on Data communication*, SIGCOMM '08, pages 63–74, New York, NY, USA, 2008. ACM.

[7] R. Albert and A. Barabási. Topology of evolving networks: local events and universality. *Physical review letters*, 85(24):5234–5237, 2000.

[8] A. Barabási and R. Albert. Emergence of scaling in random networks. *science*, 286(5439):509–512, 1999.

[9] T. Benson, A. Anand, A. Akella, and M. Zhang. Understanding data center traffic characteristics. *SIGCOMM Comput. Commun. Rev.*, 40(1):92–99, Jan. 2010.

[10] T. Bu and D. Towsley. On distinguishing between internet power law topology generators. In *INFOCOM 2002. Twenty-First Annual Joint Conference of the IEEE Computer and Communications Societies. Proceedings. IEEE*, volume 2, pages 638 – 647 vol.2, 2002.

[11] J. Cao, D. Davis, S. V. Wiel, and B. Yu. Time-varying network tomography: Router link data. *Journal of the American Statistical Association*, 95(452):pp. 1063–1075, 2000.

[12] R. L. Carter and M. E. Crovella. Measuring bottleneck link speed in packet-switched networks. *Perform. Eval.*, 27-28:297–318, Oct. 1996.

[13] W. K. Chai, D. He, I. Psaras, and G. Pavlou. Cache "less for more" in information-centric networks. In

Proceedings of the 11th international IFIP TC 6 conference on Networking - Volume Part I, IFIP'12, pages 27–40, Berlin, Heidelberg, 2012. Springer-Verlag.

[14] W. K. Chai, N. Wang, I. Psaras, G. Pavlou, C. Wang, G. de Blas, F. Ramon-Salguero, L. Liang, S. Spirou, A. Beben, and E. Hadjioannou. Curling: Content-ubiquitous resolution and delivery infrastructure for next-generation services. *Communications Magazine, IEEE*, 49(3):112 –120, march 2011.

[15] C. Close. A study of non-blocking switching networks. *The Bell System Technical Journal*, 32(2):406–424, 1953.

[16] X. Dimitropoulos, D. Krioukov, M. Fomenkov, B. Huffaker, Y. Hyun, k. claffy, and G. Riley. As relationships: inference and validation. *SIGCOMM Comput. Commun. Rev.*, 37(1):29–40, Jan. 2007.

[17] P. Erdős and A. Rényi. On the evolution of random graphs. *Magyar Tud. Akad. Mat. Kutató Int. Közl*, 5:17–61, 1960.

[18] E. Estrada and J. Rodriguez-Velazquez. Subgraph centrality in complex networks. *Physical Review E*, 71(5):056103, 2005.

[19] M. Faloutsos, P. Faloutsos, and C. Faloutsos. On power-law relationships of the internet topology. In *Proceedings of the conference on Applications, technologies, architectures, and protocols for computer communication*, SIGCOMM '99, pages 251–262, New York, NY, USA, 1999. ACM.

[20] R. Govindan and H. Tangmunarunkit. Heuristics for internet map discovery. In *INFOCOM 2000. Nineteenth Annual Joint Conference of the IEEE Computer and Communications Societies. Proceedings. IEEE*, volume 3, pages 1371–1380, 2000.

[21] C. Guo, G. Lu, D. Li, H. Wu, X. Zhang, Y. Shi, C. Tian, Y. Zhang, and S. Lu. Bcube: a high performance, server-centric network architecture for modular data centers. In *Proceedings of the ACM SIGCOMM 2009 conference on Data communication*, SIGCOMM '09, pages 63–74, New York, NY, USA, 2009. ACM.

[22] A. A. Hagberg, D. A. Schult, and P. J. Swart. Exploring network structure, dynamics, and function using NetworkX. In *Proceedings of the 7th Python in Science Conference (SciPy2008)*, Pasadena, CA USA, Aug. 2008.

[23] T. Henderson, S. Roy, S. Floyd, and G. Riley. ns-3 project goals. In *Proceeding from the 2006 workshop on ns-2: the IP network simulator*, page 13. ACM, 2006.

[24] S. Hosoki, S. Arakawa, and M. Murata. A model of link capacities in isp's router-level topology. *Autonomic and Autonomous Systems, International Conference on*, 0:162–167, 2010.

[25] V. Jacobson. Pathchar: A tool to infer characteristics of internet paths, 1997.

[26] S. Knight, H. Nguyen, N. Falkner, R. Bowden, and M. Roughan. The internet topology zoo. *Selected Areas in Communications, IEEE Journal on*, 29(9):1765 –1775, october 2011.

[27] R. Mahajan, N. Spring, D. Wetherall, and T. Anderson. Inferring link weights using end-to-end measurements. In *Proceedings of the 2nd ACM SIGCOMM Workshop on Internet measurment*, IMW '02, pages 231–236, New York, NY, USA, 2002. ACM.

[28] A. Medina, A. Lakhina, I. Matta, and J. Byers. Brite: An approach to universal topology generation. In *Modeling, Analysis and Simulation of Computer and Telecommunication Systems, 2001. Proceedings. Ninth International Symposium on*, pages 346–353. IEEE, 2001.

[29] A. Nucci, A. Sridharan, and N. Taft. The problem of synthetically generating ip traffic matrices: initial recommendations. *ACM SIGCOMM Computer Communication Review*, 35(3):19–32, 2005.

[30] I. Psaras, W. K. Chai, and G. Pavlou. Probabilistic in-network caching for information-centric networks. In *Proceedings of the second edition of the ICN workshop on Information-centric networking*, ICN '12, pages 55–60, New York, NY, USA, 2012. ACM.

[31] L. Qiu, Y. R. Yang, Y. Zhang, and S. Shenker. On selfish routing in internet-like environments. In *Proceedings of the 2003 conference on Applications, technologies, architectures, and protocols for computer communications*, SIGCOMM '03, pages 151–162, New York, NY, USA, 2003. ACM.

[32] L. Saino, C. Cocora, and G. Pavlou. Cctcp: A scalable receiver-driven congestion control protocol for content centric networking. In *IEEE ICC 2013 - Next-Generation Networking Symposium (ICC'13 NGN)*, Budapest, Hungary, June 2013.

[33] A. Soule, A. Nucci, R. Cruz, E. Leonardi, and N. Taft. How to identify and estimate the largest traffic matrix elements in a dynamic environment. In *Proceedings of the joint international conference on Measurement and modeling of computer systems*, SIGMETRICS '04/Performance '04, pages 73–84, New York, NY, USA, 2004. ACM.

[34] N. Spring, R. Mahajan, and D. Wetherall. Measuring isp topologies with rocketfuel. *ACM SIGCOMM Computer Communication Review*, 32(4):133–145, 2002.

[35] J. Tomasik and M.-A. Weisser. ashiip: Autonomous generator of random internet-like topologies with inter-domain hierarchy. *2012 IEEE 20th International Symposium on Modeling, Analysis and Simulation of Computer and Telecommunication Systems*, 0:388–390, 2010.

[36] C. Villamizar and C. Song. High performance tcp in ansnet. *SIGCOMM Comput. Commun. Rev.*, 24(5):45–60, Oct. 1994.

[37] B. Waxman. Routing of multipoint connections. *Selected Areas in Communications, IEEE Journal on*, 6(9):1617–1622, 1988.

[38] J. Winick and S. Jamin. Inet-3.0: Internet topology generator. Technical report, University of Michigan, 2002.

[39] M. Yu, W. Jiang, H. Li, and I. Stoica. Tradeoffs in cdn designs for throughput oriented traffic. In *Proceedings of the Eigth COnference on emerging Networking EXperiments and Technologies*, CoNEXT '12, New York, NY, USA, 2012. ACM.