Universitat de Girona
Escola Politècnica Superior

Treball final de màster

Estudi: Màster en Enginyeria Informàtica

Títol:
Simulador i visualitzador de robustesa de xarxes
(Network robustness simulator and visualizer)

Document:
Treball

Alumne:
Sergio Gómez Cosgaya

Tutor: Jose Luís Marzo Lázaro
Departament: Arquitectura i tecnologia de computadors
Àrea: Arquitectura i tecnologia de computadors

Convocatòria (mes/any)
Setembre 2017

Contents

List of Figures

List of Tables

Chapter 1

Introduction

1.1 Introduction, motivations and aims of the project

In a full communicated world where people around the world communicates each other through big networks it is essential to take care of these networks and make them available at much as possible. As a network must deal against attacks and failures, an analysis of them is mandatory, starting from checking its characteristics to executing failure simulations.

Related to network analysis, the research group **BCDS**[1] of University of Girona has been working for a while, so when I joined BCDS I started working on this area.

1.1.1 Motivations

In the research group, there was significant work done about network robustness analysis, that is analysing how a network is affected against several attacks. To analyse robustness, different experiments had to be executed with different kind of attacks and attack combinations which produced several experiments to execute.

This project is related to this previous work done and the main goal was to make these experiments easier. Previously, all executions had to be done manually and individually from a command line. Moreover, executions could not be executed in background, that means that who executed the experiments had to wait until executions finish and it was impossible to analyse results in a graphical way because they only had a static network visualizer which with a high number of nodes make the visual analysis impossible to achieve.

The main motivation was to start working in this topic and help the research group making an automatic tool or (from now referred as **simulator**) which could help them to execute their experiments easily. Also, another important motivation was to start working in a research group, learning how *research world* is.

1.1.2 Objectives

The main objective is to develop the automatic simulator which will be able to execute several experiments. The main functionalities it must have are:

- Simulator must be available worldwide through web access.
- Execute multiple experiments automatically from a single call.

[1]Broadband Network Control and Management. Official website: http://bcds.udg.edu

- Execute experiments in background, allowing the user to quit the simulator and executions do not stop.
- Full data results download availability.
- Nice results and networks visualizations.
- Simulator must have different workspaces to not merge different type executions and have users management.

Also, the simulator is going to have many more functionalities, but they are explained with detail in a further section.

Chapter 2

Viability study

In this chapter a brief viability study is detailed. First we need to explain some considerations to understand the taken decisions. This project is done in a university research group background, that means that the viability study probably should differ if being in a business background. This project has been developed practically entirely by myself (i.e. only one developer) but in this viability analysis it is supposed that different kind of roles can be hired (e.g. a developer, a designer, etc.).

2.1 Development cost

In order to calculate the development cost, this formula is used:

dev cost = initial cost + extra cost + developers cost

Necessary tools to develop the application are (at minimum):

- One computer for developer with medium specifications requirements.
- One computer for designer with high specifications requirements.
- Development IDE for developer (e.g. Atom).
- Development IDE for designer (e.g. Atom plus Adobe suite).
- Testing server with medium specifications requirements.

Initial cost is calculated by necessary devices cost:

DEVICES	PRICE (euro)
Computer with medium resources	750
Computer with high resources	1200
Testing server with medium resources	1500

TABLE 2.1: Initial costs

Assuming all tools have to be bought, calculated initial cost is:

initial cost = 750 + 1200 + 1500 = **3450 euro**

Extra cost are all costs related to development IDEs and their licenses:

TOOLS	PRICE (euro)
Development IDE - Atom	Free
Designing IDE - Adobe suite	60 / month

TABLE 2.2: Extra costs

As Adobe suite has a price of 60 euro each month we assume buy a 1 year license. With that in mind calculated extra cost is:

extra cost = 0 + 60x12 = **720 euro**

As said before, this study is done assuming the existence of one developer and one designer. The first one is assumed to have a full-time schedule and the second one a half-time schedule. The estimated developing time is around 4 months (approximately 17 weeks) so developers costs are based on:

EMPLOYEE	HOURS	PRICE (euro)
Web developer	40 / week	15 / hour
Designer	20 / week	110 / hour

TABLE 2.3: Developers costs

Calculated developer cost is:

developers cost = (17 x 40 x 15) + (17 x 20 x 20) = **17000 euro**

Finally, development cost is:

dev cost = 3450 + 720 + 17000 = **21170 euro**

2.2 Production cost

For production there are two options. First one is to buy a Virtual Private Server (VPS) in any cloud provider which will have a monthly cost and second one is to buy a physical server which only has a bigger unique cost. In our case we decided to buy a physical server in order to not have our data in the cloud. Physical server bought is:

SERVER SPECIFICATIONS	HOURS
Server name	HPE ProLiant ML350e
CPU	Intel® Xeon® E5-2400 v2
RAM	64GB DDR3 RDIMM
HDD	2TB SATA
Price	1750 euro

TABLE 2.4: Server specifications

Production cost is:

production cost = **1750 euro**

Chapter 3

Methodology

The methodology used during the project has been similar to the agile methodology. Agile methodologies are focused on improving quality and the efficiency of the corresponding project development. Therefore, the whole process is planned in short iterations in order to be able to deal with unpredictable changes and readjust the development orientation. All phases in process seen in figure 3.1 can be described as:

Previous study and decisions – A depth study of bibliography and existing studies done have to be done to see what is yet done and what can be set as starting point. A study and a well thought decision about how to achieve our objective is very important for not reinventing the wheel.

Specify application requirements – Before implementing nothing, it is important to define which requirements and functionalities our application is going to have. This can take some time, because is better to lose some time here than having to stop the developing process afterwards due a bad requirements specification.

Iteration planning – Here a iterative process is started. All requirements have to be planned in tasks or iterations which are going to be implemented following some order (e.g. first server-side tasks).

Tasks developing – For each task planned a iterative process must be followed. Before implementing a previous application status analysis should be done to understand how application is in that moment, then developing can be started finishing the process with an elaborated testing. If testing was successful you can continue with another task, otherwise process will come back to previous status analysis.

Update production version – The last step of the process when a task or functionality has been successfully tested is to send it to production servers. There is no need to update each task immediately when a task has been ended, some tasks can be updated at once.

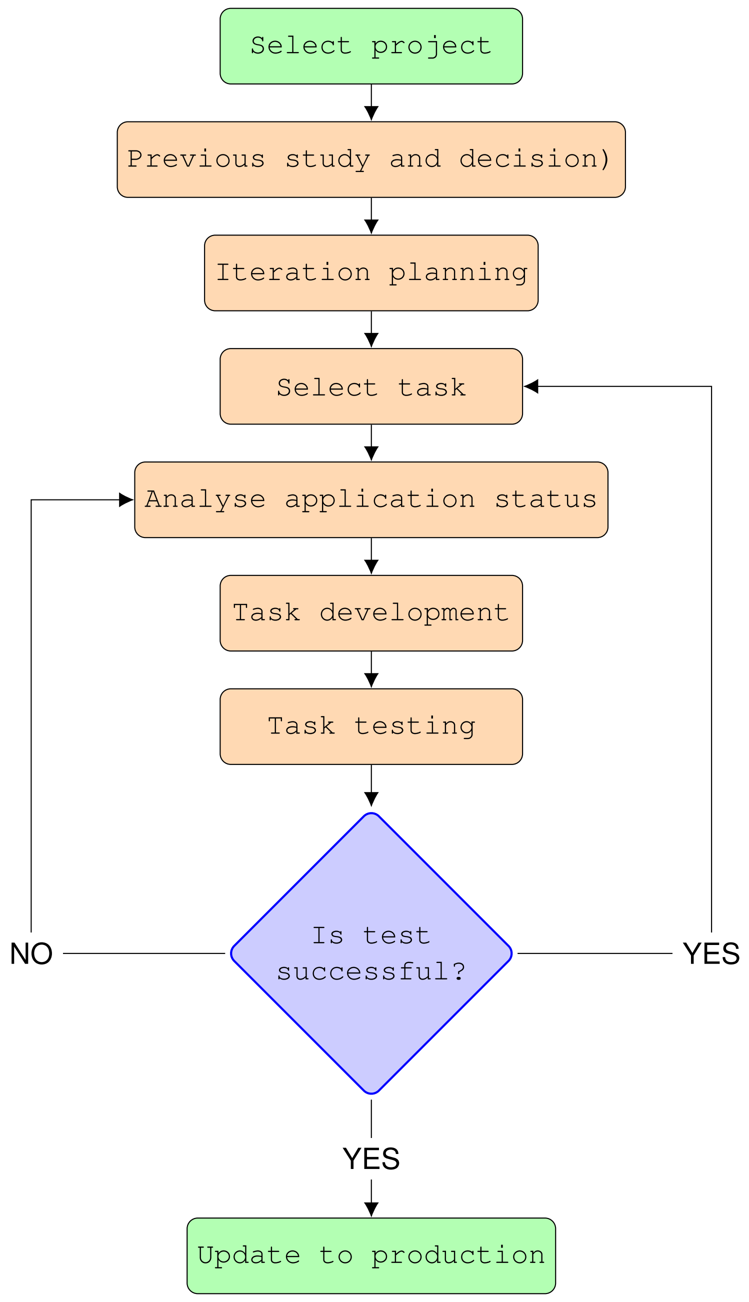


FIGURE 3.1: Methodology workflow diagram.

Obviously, sometimes you have to break this methodology when a new requirement needs to be added to application and as working in a research group, sometimes we need some functionality which we didn't realise about it.

To control all tasks we use a web-based project management application named as Trello[1]. It lets us organizing all tasks in different groups (i.e. notes, to do, doing and done) and assigning every task to a member.

[1]Trello offical website: https://trello.com/

Chapter 4

Planning

I started working in this project in September 2016, so all planing starts from this moment. All phases in planning can be seen in figure 4.1 and are detailed below. Is important to say that I have not been working on this project in full time because I had to combine it with working on other topics the research group is working on.

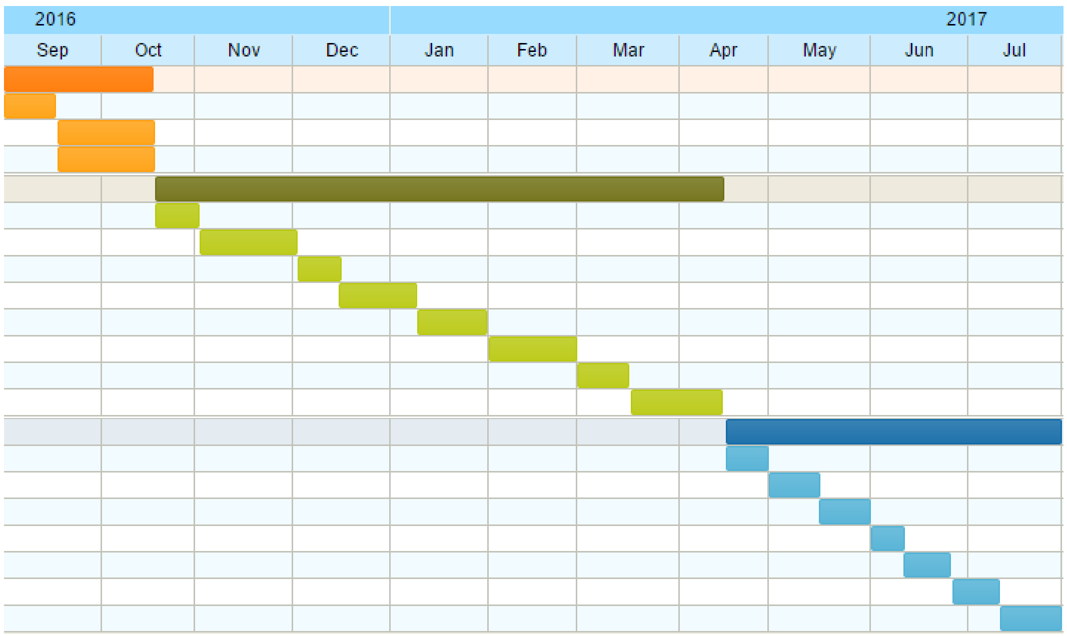


FIGURE 4.1: Project planned phases.

Different phases are:

Previous study – Before developing, it is necessary to study all theoretical concepts about networks and search any others studies done in bibliography. This phase was divided in three sub-phases, first I had to understand what previous work done in research group does and how, then I took some time studying all network theoretical concepts.

Simulator development – This phase represents all time spent developing the main tool which allows us executing all simulations and experiments easily from a web application. Sub-phases were divided in defining requirements, developing the basic user interface from where to start developing everything else, developing all users and workspaces control and adding the ability of executing single simulations or a set of them. Then some time is spent improving the execution performance where basically code was parallelized in threads and computer clusters reducing executing time. Finally a testing phase was done, executing some experiments we needed. Last sub-phase is related to a stay I did in **TUM**[1] during first two weeks of April, so I had to have developed all previous phases before April.

Visualizer development – This phase is similar than previous one but in this case the tool developed helps us to visualize all networks and all experiment results in a nice visualizer. Sub-phases are divided in different visualizer features.

[1]Technical University of Munich. Stay is detailed in following section.

Chapter 5

Previous concepts

In this section, the necessary previous concepts to understand what the tool developed in this project can do are detailed and explained. First some basic networks theoretical concepts are exposed, followed by all metrics used to analyse these networks and finished with an explanation of the whole process to obtain the robustness value of a network against attacks.

5.1 Basic network concepts

The aim of this section is to introduce basic network concepts. These concepts will be used along this document.

Formally, **a network is defined as a set of interconnected elements with a given purpose**. There are a lot of examples that satisfy this definition: water distribution networks, the power grids, social networks, subways networks, etc. Mathematically, a network is defined as a **graph** which can be defined as a structure used to represent relations between elements. A graph (G) can be represented as an union of a nodes set (V) and an edges set (E):

$$G = (V, E)$$

Graphs can be differentiated by the type of edges, unidirectional (**directed graphs**) or bidirectional (**undirected graphs**). Graphs elements[1] have labels to identify them and can have meta-data (or attributes). If edges meta-data represents the importance of each edge inside the network, then the graph can be described as a **weighted graph**.

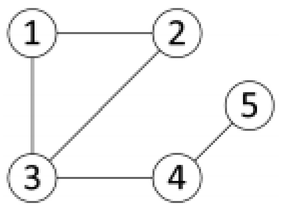
5.1.1 Graph theory

Graphs can represent either real networks or abstract networks, e.g. a graph can represent the relationship between football players and theirs clubs. Graphs can be created synthetically for research purposes. Graph theory includes wide and diverse topics, therefore in this section only basic graph theory concepts are explained, all of them related to robustness.

Adjacency matrix, $A(G)$ – It is a matrix (see figure 5.1) that represents which nodes are connected to each other. A 0 value means no adjacency between nodes

[1]Graph elements is equal to the union of graph nodes and edges, that is talking about elements is the same than talking about nodes and edges.

and a 1 value means both nodes are connected. For weighted graphs, the value represents edge weight.

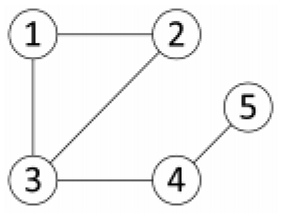


$$\begin{pmatrix} 0 & 1 & 1 & 0 & 0 \\ 1 & 0 & 1 & 0 & 0 \\ 1 & 1 & 0 & 1 & 0 \\ 0 & 0 & 1 & 0 & 1 \\ 0 & 0 & 0 & 1 & 0 \end{pmatrix}$$

FIGURE 5.1: Graph adjacency matrix example.

Laplacian matrix, $L(G)$ – The matrix representation of a graph is know as Laplacian matrix (see figure 5.2). It is denoted by $L(G) = D(G) - A(G)$, where $D(G)$ is the degree matrix of the graph which has node degrees on its diagonal and zeroes elsewhere.

$$L_{i,j} = \begin{cases} \delta_i, & \text{if } i = j \\ -1, & \text{if } i \neq j \text{ and } (i,j) \in E \\ 0, & \text{otherwise} \end{cases}$$



$$\begin{pmatrix} 2 & -1 & -1 & 0 & 0 \\ -1 & 2 & -1 & 0 & 0 \\ -1 & -1 & 3 & -1 & 0 \\ 0 & 0 & -1 & 2 & -1 \\ 0 & 0 & 0 & -1 & 1 \end{pmatrix}$$

FIGURE 5.2: Graph Laplacian matrix example.

5.2 Robustness and metrics

Studying the solidness of a network against attacks (nodes and/or edges failures) is the main purpose of the work done in the research group. For us, robustness is defined as **how a network behaves against attacks or failures** and our way to evaluate its value is to calculate different metrics on the network with which we are able to quantify a numeric robustness value. We work with robustness values between 0 and 1 where 0 (or less) means network is totally affected and inoperative and 1 means networks has not been affected for failures and it is working as normally.

As stated above, **metrics** are used to get a quantitative value for each network or element characteristic. Metrics give a way to compare graphs properties detailing different features of the network or graph. A metric can be from as simple a basic concept in graph theory (e.g. graph diameter or nodes average degree) to a complex process (e.g. calculate throughput in a network when removing elements).

5.2.1 General metrics classification

The metrics used in this project are organized in different groups, depending on what they can tell us regarding the robustness:

Structural – Structural metrics measure the classical parameters of a graph. Some of them measure the density (such as nodal degree) and others the size, such as the diameter. All structural metrics used are:

- Average nodal degree
- Heterogeneity
- Average shortest path length
- Diameter
- Efficiency
- Effective resistance
- Number of spanning trees
- Clustering coefficient
- Assortativity
- Symmetry radio
- Largest eigenvalue

Fragmentation – Fragmentation metrics address the number of components of a network. It should be taken into account that they are useless for connected networks (i.e. they do not measure anything until the network is unconnected). Available fragmentation metrics are:

- Largest connected component
- Fractional size largest component
- Average two terminal reliability
- Degree of fragmentation

Connectivity – Connectivity metrics measure how difficult is to break the network when removing elements. In contrast to fragmentation metrics, they can not be used when the network is fragmented. All available connectivity metrics are:

- Edge connectivity
- Vertex connectivity
- Algebraic connectivity
- Natural connectivity

Centrality – Centrality metrics can measure how important are the elements in the graph, i.e. if they are at the middle of most of the paths. All available centrality metrics are:

- Degree centrality
- Node betweenness centrality
- Edge betweenness centrality
- Closeness centrality
- Eigenvector centrality

Functional – Functional metrics represent those metrics that evaluate functional behaviours of network. The easiest example is to calculate throughput through the graph. Functional available metrics are:

- Throughput (TUM)[2]
- Link usage (TUM)

Due to the high number of available metrics, most of them have special relationships between them when expressing the same graph feature. When two metrics express the same feature, they can have different kind of relationship:

- One metric has a lower computation cost.
- One metric is harder to compute but it gives more information.
- One metric represents the inverse than the other.

This can become an issue when computing more than one metric than measure the same, so only one metric should be calculated for reducing execution cost.

5.3 Failures and attacks

As said before in this document, BCDS research group studies how networks are affected against attacks or failures over the network. A failure on a network is defined as the physical loss of an element (node or edge). In a graph, failures are modeled removing that failed elements from its structure. When talking about attacks in this document, we are referencing to several failures generated by one single attack. According to different characteristics, attacks can be classified in:

Node failure or edge failure – Both elements can fail but failing a node also affect to its connected edges as seen in figure 5.3.

[2]TUM metrics were developed using their routing algorithms.

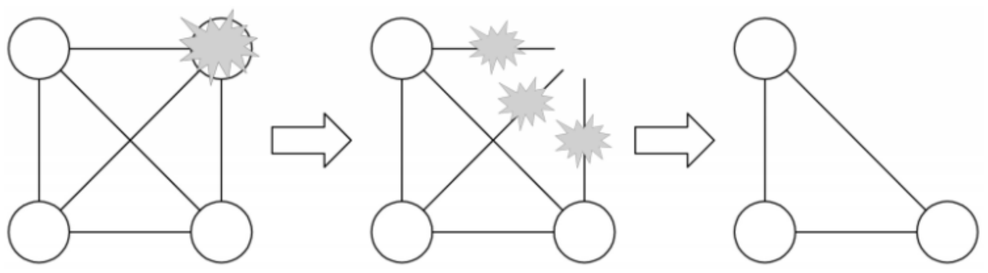


FIGURE 5.3: Node failure also affects its edges.

Static or dynamic attacks – A static attack is like a snapshot of the attack on the network and every attack is independent of each other. Dynamic attacks have continuity in time and grow over the network (see figure 5.4). Epidemic attacks (e.g. a virus spreading over a computers network) are an example of dynamic attacks.

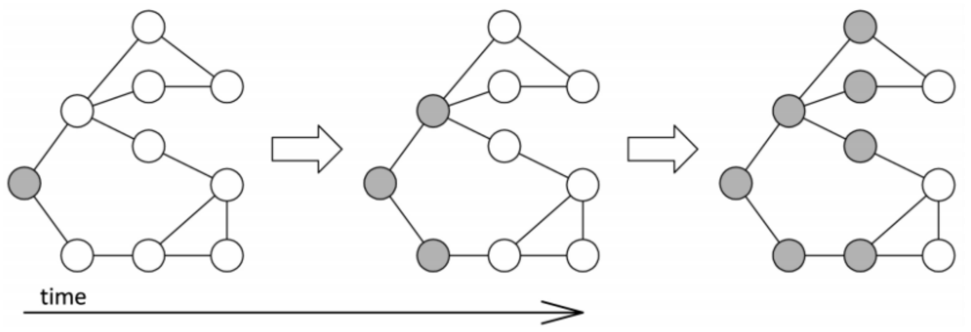


FIGURE 5.4: Attack spread process over a network.

Strategical behaviour – When stressing a network is important to study how to attack it, trying to damage it as much as possible. Different strategies can be used to attack networks in many ways. In this section we explain which strategies we use in our project:

- Random attacks:
  In random attacks, attacked elements are selected randomly, even if elements attacked are nodes or edges. Theoretically, this method is the less effective one, cause randomizing doesn't look for a way to make more damage to the network.

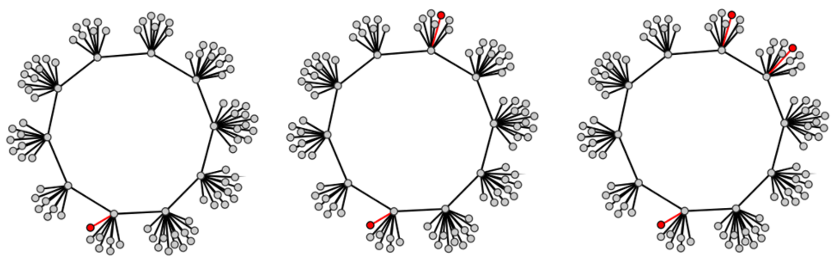


FIGURE 5.5: Random attack to nodes example. Each node is selected randomly

- Targeted attacks:
  Unlike randomizing, targeting attacks look for the most important elements in network and select them to be removed. For example, the node with a larger node's betweenness centrality is selected when attacking nodes and the edge with a larger edge's betweenness centrality should be selected in case of attacking edges. Obviously, the parameter used to select the desired elements can be changed.

  In the targeted method there are two options. On one hand, there are **targeted simultaneous** attacks where all attacked elements are selected in one step (e.g. the 10 nodes with more node's betweenness centrality), on the other hand, **targeted sequential** attacks where attacked elements are selected step by step sequentially looking for the most important element of the network every time after attacking some other before (e.g. look for the node with more node's betweenness centrality after removing previously other elements).

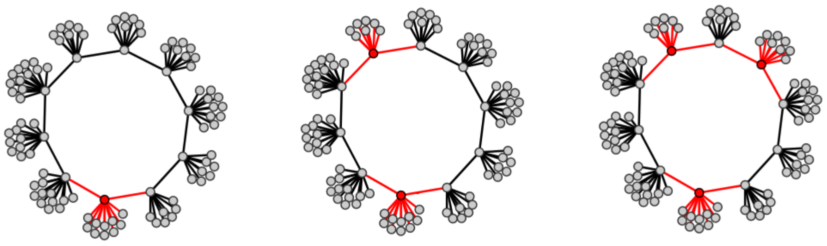


FIGURE 5.6: Targeted attack to nodes example. Each node is selected by its importance inside the network. Note that although it can looks like random it selects the central nodes.

- Epidemic attacks:
  When using the epidemic method, the attack simulates an epidemic spread over the network, that is selecting randomly the first infected node and then sequentially select nodes that are connected to an infected node following a probabilistic function. This method only works with node attacks because it has no sense to have infected edges.

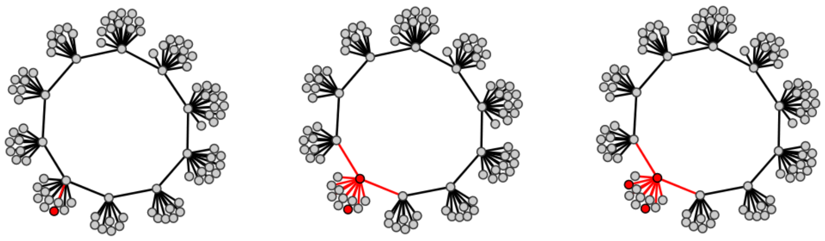


FIGURE 5.7: Epidemic attack to nodes example. Next selected node has to be connected to an infected node.

Chapter 6

Robustness process

The main goal of this project is to develop an application to make experiments execution easier to execute. To understand what application should achieve and how it should work it is mandatory to explain how whole experiment process works. The purpose of this chapter is to explain that process in a base background.

6.1 Input variables

First, ab experiment should be set up, that is defining all experiment input variables that define itself:

Network – Obviously a network has to be selected in order to generate attacks on it. We use networks coded in GraphML[1] format which is one of the most accepted format for representing graphs.

Attack strategy – We need an attack strategy to generate the list of elements failed. Valid strategies are random, targeted simultaneous, targeted sequential, epidemic and electrical cascade[2].

Element – Experiments can be over nodes or edges but not over both at same time, so it is needed to select whether to attack nodes or edges. Remember that removing nodes implies also removing its edges, so removing nodes is always more effective than removing edges.

Quantity of failed elements – One input variable tells how many elements are attacked. It can be defined as a discrete value (e.g. attack up to 20 nodes) or percentage (e.g. attack 10% of nodes, where in a 100 nodes network 10 nodes should fail).

Number of combinations – In a single experiment several different attack combinations are made on the network, so we need an input variable to tell how many combinations execute. In next section it is explained why making many combinations is needed.

Seed – When using randomizing options a seed value has to be passed as parameter. We use a seed to be capable of executing again exactly same experiments (using the same seed we get same results). If no seed is passed a default one is used.

[1]GraphML is a comprehensive and easy-to-use file format to represent graphs. It is based on XML. Website: http://graphml.graphdrawing.org/

[2]Electrical cascade attack strategy has been added recently and it has not been explained in previous concepts chapter.

Metrics – Finally, we need to select which metrics are going to be used in order to find the robustness of the experiment. They are represented as a list of metrics, with no maximum number of metrics. It is important to remind that many of the metrics available express the same feature as explained in chapter 5.

6.2 Process steps

When all input variables are set, then a process starts. All steps in this process are importing the network, calculating initial metrics, generating all combinations attacks, calculating all metrics for each combination, performing a PCA (Principal Component Analysis), calculating robustness value and generating results.

6.2.1 Importing the network

This is the first step but obviously it is mandatory. The network is read using the R package named **igraph**[3] from a GraphML format file.

6.2.2 Initial metrics computation

Once network is imported to a graph next step is to compute all desired metrics on the network by default, that is without any failure. This metrics are know as initial metrics ($t0$). As each metric robustness value range is from 0 to 1, $t0$ values represents the maximum robustness value of each metric (i.e. 1). For example, if a $t0$ value of a metric is 20, if later we calculate again this metric with failures in network getting a result of 15, then robustness value at that point will be 0.75 (15/20). From here, we have a $t0$ metrics list that is will be used later on this process.

6.2.3 Attacks generation

As said in input variables section, there is an input variable which tells how many attacks combinations are going to be generated, from now this number of combinations is defined as M. This is done because we need to execute enough different attacks to have sense results because, for example, if attack is random it can happen that it selects the most important nodes which will look like as random is the more damaging strategy, and it is not the case.

When using random attacks there is no problem generating several combinations because it should select different elements in each generation. In targeted attacks, but, if you always select the most important element all combinations will have same elements selected. To avoid this, a probabilistic function is used to select an element from a list of the most important elements (e.g. a probabilistic function than select a node from 20 nodes with higher degree). Finally, in epidemic attacks only first failed element is selected randomly in each combination, then epidemic algorithm follows selecting other elements using an epidemic model.

6.2.4 Metrics computation

After attacks generation has finished we have a M list of attacks combinations. As explained in input variables section, there is a parameter that tells the quantity of

[3]Website: http://igraph.org/

elements attacked in each attack. From now this quantity of elements attacked is defined as P. In this step all metrics are computed, but also they are computed step by step from first element failed to P elements failed.

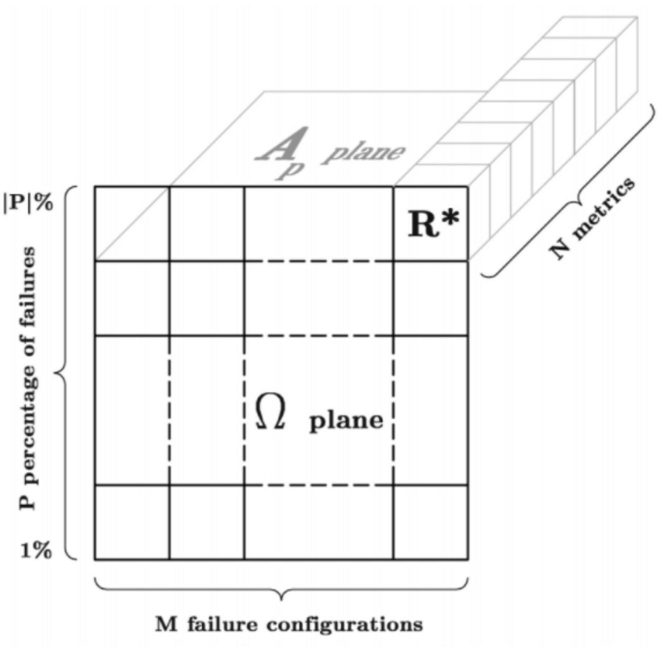


FIGURE 6.1: Robustness metrics 3D matrix representation.

For example, if P is set as 20 elements to fail, all metrics will be computed with 1 element removed, with 2 elements removes, with 3 elements removes and so on until with 20 elements removed. From here we should have a *3D matrix* representing all $PxMxN$ metrics results where N is the length of metrics to compute (see figure 6.1).

6.2.5 Principal component analysis

Once all metrics have been calculated, it is important to know which metrics give us more information. To achieve this, a PCA (Principal Component Analysis) is done to all metrics results. Doing this PCA will result to a list of N elements with the weight (or importance) of each metric and a PCA relevance value (from 0% to 100%) that tell us if selected metrics are good enough for the experiment. Also the PCA give us all correlations between each metric, which can tell us some metrics express the same. Further detailed information could be found in bibliography referenced.

6.2.6 Robustness value and surface

Robustness value can be expressed as the sum of all metrics information. This value gives us a robustness numeric value for each PxM combination, that is how robust is the network at that. As for each PxM combination we have N metrics results, we need to join them into a unique robustness value (from now defined as R^*). As R^* values range is from 0 to 1, the easiest way to join these metrics results should be a simple average but as we did the PCA before, we can use the metrics weight list to generate that R^* final value.

Then R^* is denoted by:

$$R^* = \sum_{i=1}^{N} t_i w_i$$

where t_i is the metric value and w_i is the weight obtained from PCA.

At this point, we have a matrix with PxM robustness values. Also we calculate a unique R^* value that represents all experiment robustness value, that is the robustness value of a network against a desired attack strategy. We use all matrix average to calculate this unique R^* value.

With this unique R^* value we can now compare different experiments, but it's only comparable by an unique number which many times it doesn't say anything interesting. In order to get a more interesting way to compare experiments and also to have a visual way of compare them we generate a **robustness surface**. The robustness surface is simply a coloured table representing all PxM robustness values, starting from red when $R^* = 0$ to green when $R^* = 1$. Obviously, when more green the robustness surface is, most robust the network is. We can see an example in figure 6.2.

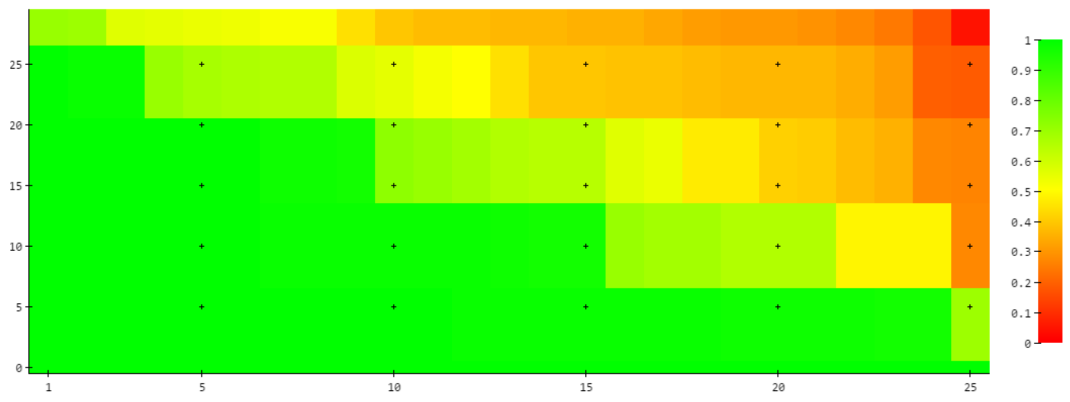


FIGURE 6.2: Omega robustness surface.

It has to be explained that robustness surface seen in figure 6.2 has been previously ordered by rows. That is done because ordering rows makes the robustness surface nicer and more understandable. This ordered robustness surface is named as **omega robustness surface**.

6.2.7 Results generation

Finally, the last step is to generate a final results file. This file will be a Excel format file which will include all generated attacks combinations, all $PxMxN$ computed metrics and all R^* values. Also it includes both unordered robustness surface and omega robustness surface as a table.

Chapter 7

System requirements

In this section all system requirements that application must accomplish are detailed. Both functional and non-functional requirements have to be defined before any implementation. We differ about simulator and visualizer requirements.

7.1 Functional requirements

Functional requirements are those that represents every functionality the system has to be able to perform. They are based on an user input and expect an output or a result.

Simulator functional requirements are:

Register and login – Be able to register and log in using different social networks (using a no-password user system).

Administrator managing – An administrator must be able to manage all users, simulations and networks in the system.

Different workspaces – Have your own private workspace to work in his your experiments.

Shared workspaces – Be able to have shared workspaces with other users.

Compute simulations – Compute simulations over a network using some parameters and metrics.

Compute set of simulations – Compute a set of simulations at once without executing each of them individually.

Computing again – Compute a simulation or a set of them instantly from another simulation or set previously computed.

See computed simulations – Be able to see all simulations and sets previously executed. Simulations from a workspace can not be seen from another workspace or user.

Download results – Download simulations results in a file (off-line availability).

Preview results – Preview results dynamically on application (on-line availability).

Remove simulations – Remove a simulation or a set of them from the system.

Import networks from public repository – Be able to use public networks by importing them from a public repository available in the system. Public networks are available for any user.

Upload new networks – Be able to upload custom or personal networks to the system. These networks have to be only available for user who uploaded it.

Generate networks – Be able to generate several kind of networks and to import them to system. These generated networks have to be only available for user who generated it.

Visualizer functional requirements are:

View networks synthetically – Be able to preview any network formatted in GraphML file in a synthetic way, i.e. the network shape is defined dynamically using forces between nodes and edges.

View networks on a map – Be able to preview any network formatted in GraphML file on a map. Network nodes must have geographical attributes (e.g. coordinates).

Focus on mouse-over – Focus elements when pointing them with cursror, i.e. reducing other elements visibility or increasing pointed elements visibility.

Analyse attributes – Be able to see network, nodes and edges attributes. Also be able to colourise elements depending on their attributes values.

Filter elements – Show only desired elements hiding elements that not satisfy our needs (e.g. show only edges with bandwidth higher than 10Gbps).

Search elements – Find any elements matching with a keyword inside whole network.

Visual settings – Be able to modify all visual settings (e.g. change nodes color and size, nodes positioning, etc.).

Dynamic computing – Be able to execute dynamically different analysis over the network visualized at the moment (e.g. calculating the diameter in whole network).

7.2 Non-functional requirements

Non-functional requirements are those that system are supposed to do but are not directly a functionality of the system.

Simulator non-functional requirements are:

Stability – Application have to be stable, even when conditions are not optimal. System should not turn off for no reason.

Worldwide availability – System have to be accessible anywhere.

Execution performance – Execution performance has to be better as possible. That is adding availability to execute more than one simulation at the same time and to efficiently use all hardware resources (e.g. panellizing executions over all available threads).

User permissions – User can only see his experiments and networks or those ones shared with other users.

User enabling – User can not use his account just after registering because administrator has to enable the user before. That helps no one without permissions can have access to application.

Execution time-out – System can detect when an execution is no longer working (e.g. it has entered into a loop) and stop it from executing.

Unmerged code – Simulator code have not to be merged with execution code. When changing execution code should not have to affect to simulator.

Friendly interface – Simulator must have an easy and understandable interface that make user interaction and learning faster.

Chapter 8

Studies and decisions

All decisions about how to implement the simulator and the visualizer are based on how the group had to execute experiments and what they need. In this section it is analysed how the execution had to be done, how it should has to be and how we decided to solve all issues and disadvantages in each step of the experiment process.

8.1 Launching an experiment

In fact, first step is to think about what experiment to execute but this step is purely a think step and has no relationship with the system itself. Then, since experiment is defined theoretically, next step is to launch or execute the experiment:

How it worked? – As said before in this document, execution code is made in R, so it could be executed locally in a personal computer or in a server. If executing in a server, the first step is to connect to that server using SSH what is not familiar for someone with no high computer knowledge. Then, there were two option to execute an unique experiment or simulation. On one hand it could be executed an *command prompt application* that ask the user all experiment parameters (i.e. executing manually step by step the simulation) and on the other hand there were the possibility of executing the experiment automatically passing all parameters through a unique *string text* with each parameter separated by a comma. The definition of this *string text* is:

id, element, attack_type, attack_parameter, "metric1, metric2,...", P, M, seed

where **id** is the experiment identifier (it can be whatever), **element** is which element is attacked (nodes or links), **attack_type** tells what attack type is used (random, sequential, etc...), **attack_parameter** gives extra parameters to desired attack, **"metric1, metric2, ..."** define all used metrics, **P** is the maximum number of attacked elements, **M** is the number of different attacks combinations and **seed** is the seed for random generations. All these parameters are better explained above in this document. An example could be for example:

exp1, node, targeted, sequential, "01-AND,02-HET,03-ASPL", 30, 20, 123456

Obviously, this methodology had the big disadvantage that it was very slow to execute a bunch of experiments and also you have to be very careful while editing the *string text* because a simple mistake could make execution fail. Also the user had to keep the computer turned on while waiting execution to finish what it added another disadvantage.

How should it work? – There is not a unique way of do it, but the basic requirements in this step is to have a tool able to automatically execute a set of experiments from a bunch of parameters performing all possible combinations (e.g. selecting many attack types as random and epidemic which it will execute a simulation with random attack and another simulation using epidemic attacks). The tool must have the ability of select all parameters and options in a easy way and without typing anywhere to avoid typing errors.

What was decided? – To solve it we decided to develope an easy web application with a computing page that can help all research group members executing many experiments at once. We decided that page to have several steps to set all experiment parameters:

1. Select workspace to use.
2. Select network to execute experiment on. Only 1 network can be selected.
3. Select experiment parameters as attack type, elements to attack, number of elements attacked, etc. In this step many options can be selected (e.g. user can select random and epidemic attacks plus attacking both nodes and edges). Then each combination will be executed.
4. Select metrics to use. From 1 to N metrics can be selected.
5. Start computation. System will start computing each combination as a unique simulation. User will be able to see the process output of each simulation.

Using this methodology a bunch of experiments is easily executable without having typing errors and as executions are done in server background user can turn off computer while all executions will not stop.

8.2 Viewing experiments logs

After computing experiments it has to be a way to check which previous experiments have been done and all their parameters.

How it worked? – That really didn't work. There were not a way to check previously launched experiments. The only previous experiments data saved is the results XML file inside a folder with the simulation ID. Inside that XML file all parameters were saved but it was a mess to opening every XML file to check what each simulation was about.

How should it work? – New tool should have a record or log page where all previous launched experiments should appear listed with all parameters used to define them.

What was decided? – As we did a computing page to compute new simulations, we decided to develope a log page where all simulations sets appear listed with the option of seeing its parameters clicking on each set. Then, a button to see listed all simulations inside a set was added too in the same way as sets are shown.

8.3 Recomputing experiments

Another important functionality is to have the ability of recompute a set of experiments easily. It is useful when changing the computation code (i.e. changing the robustness analysis code) to get new results with updated code.

How it worked? – Using the *string texts* mentioned above could be "*simplier*" than executing one by one manually entering all parameters, but you had to save all *string texts* somewhere and also there were the typing error issue. Another problem was that you had to execute one by one each combination and that you had to wait until execution finishes.

How should it work? – New tool should have the ability to read previous set of simulations parameters which what the simulations set was defined and use them to execute automatically another set of simulations identical to previous set. It should be fine to have the possibility of modifying the parameters before the new execution (e.g. removing epidemic attacks if you are no longer interested in them).

What was decided? – As we had a whole log of simulations sets launched previously with their parameters in logs page, we added an option to *clone* a single simulation or a set of simulations to the computation page. Then all parameters than user could select manually step by step in computation page are automatically filled and the application offers to user the options of execute again all simulations or to change any parameters before executing.

8.4 Getting results

Finally, the last step of the process is to get the simulations results. Not only numeric results are important but also the robustness surface.

How it worked? – There was only one output, the generated XML file with all results data. It includes all parameters, metrics calculation results, PCA relevance values, robustness values and generated attacks. You can't see the robustness surface if not having Microsoft Excel installed. Also there was the problem that to get the results file you should access the server through FTP (or go to directory if executed locally). Finally, another issue was not having the possibility of saving the execution output after the execution finished.

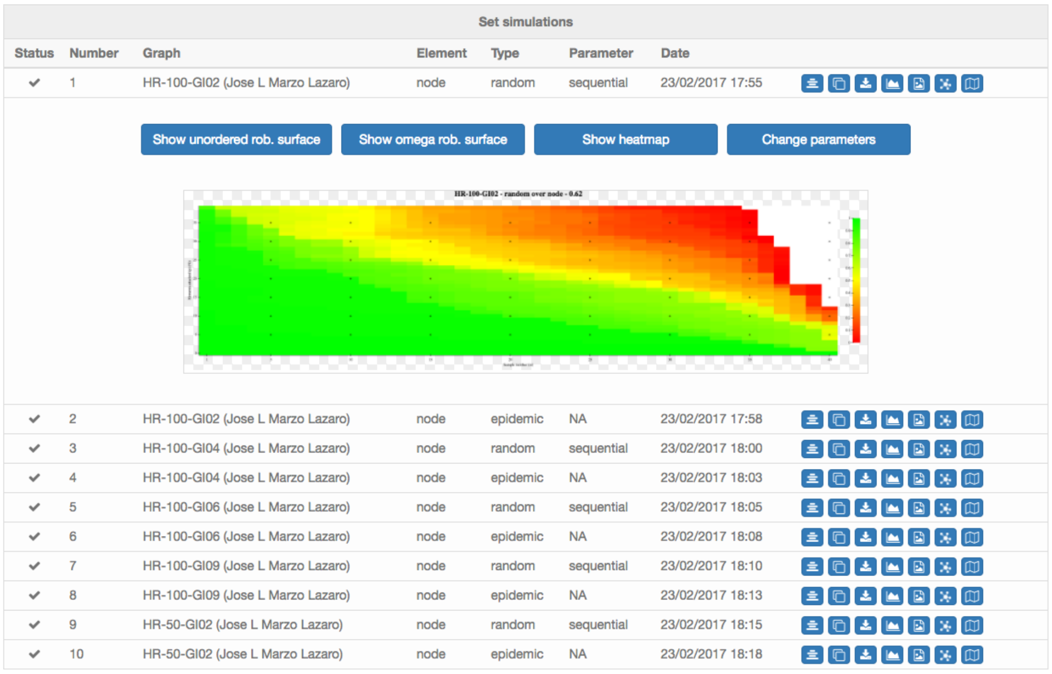
How should it work? – Desired tool should have the ability of downloading that XML file from each simulation directly from application. The tool should be able to show the robustness surface as an image and each simulation output directly in application.

What was decided? – We decided to add desired functionalities in logs page. When selecting a unique simulation inside a simulation set a bunch of options appear. Each button performs one of desired functionally, i.e. one button shows simulation output, another one downloads the XML file, another shows the robustness surface, etc... The process it is not easier than:

1. Select workspace to use.

2. Go to logs page. All sets should be listed.
3. Click on desired simulations set. All simulations from set should be listed.
4. Clink on a button to perform desired functionality. Another sub-view should appear when showing output or robustness surface of selected simulation.

An example of resulting robustness surface of an experiment viewed on the application tool can be seen in figure 8.1



Set simulations

Status	Number	Graph	Element	Type	Parameter	Date
✓	1	HR-100-GI02 (Jose L Marzo Lazaro)	node	random	sequential	23/02/2017 17:55
✓	2	HR-100-GI02 (Jose L Marzo Lazaro)	node	epidemic	NA	23/02/2017 17:58
✓	3	HR-100-GI04 (Jose L Marzo Lazaro)	node	random	sequential	23/02/2017 18:00
✓	4	HR-100-GI04 (Jose L Marzo Lazaro)	node	epidemic	NA	23/02/2017 18:03
✓	5	HR-100-GI06 (Jose L Marzo Lazaro)	node	random	sequential	23/02/2017 18:05
✓	6	HR-100-GI06 (Jose L Marzo Lazaro)	node	epidemic	NA	23/02/2017 18:08
✓	7	HR-100-GI09 (Jose L Marzo Lazaro)	node	random	sequential	23/02/2017 18:10
✓	8	HR-100-GI09 (Jose L Marzo Lazaro)	node	epidemic	NA	23/02/2017 18:13
✓	9	HR-50-GI02 (Jose L Marzo Lazaro)	node	random	sequential	23/02/2017 18:15
✓	10	HR-50-GI02 (Jose L Marzo Lazaro)	node	epidemic	NA	23/02/2017 18:18

FIGURE 8.1: Resulting simulation robustness surface viewing in the application.

8.5 Viewing networks and animated results

Another but not less important requirement application must offer is to visualize networks in a visualization tool where the user can look around all network aspects even though the network has a lot of elements on it. It should offer the option to zoom into the network, see relations between nodes, etc. Also, for understanding the results obtained with simulations execution it is important to see how attacks spread over the network, with the ability of seeing each attacks animated through the network.

How it worked? – R igraph package has the option to plot a network in a static image as seen in figure 8.2.

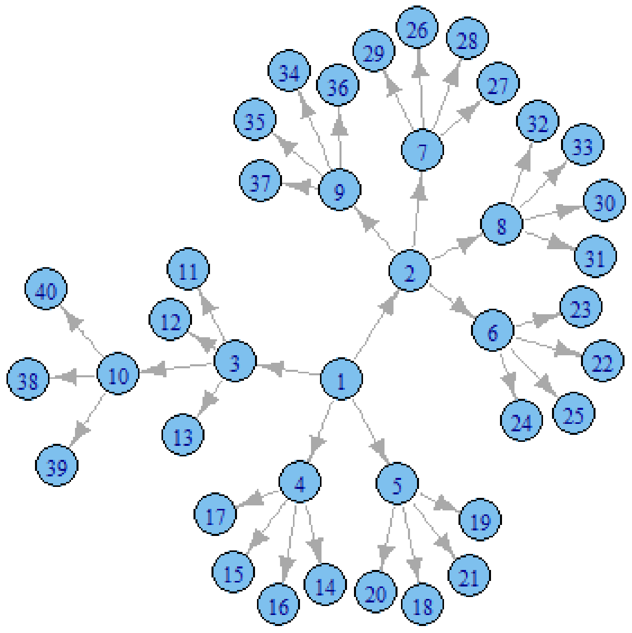
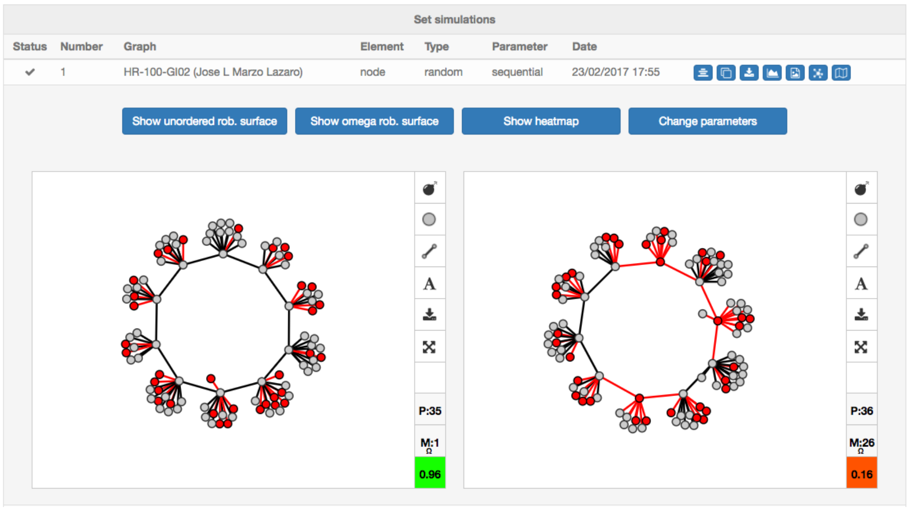


FIGURE 8.2: Sample tree network R plotting example.

The biggest issue of this representation is the amount of elements in the network because with a high number of nodes or a very dense network nodes can not be differentiated when they become overlapped. That plot option does not allow zooming or editing network shape and does not have any option to see attacks spreading over networks. Other issue is related to the need of executing the R igraph package to get any visualization, that means log into the server, and execute some code manually, what is not easy for none experimented users.

How should it work? – The tool should offer an easy accessible way to preview networks in their default state, that is without any modification on it (or without attacking it). Also when placing mouse cursor over a node or an edge, the tool should show all data about that element and focus related elements, that is showing all connected elements and hiding others). It should have the capability of zooming into the network which would help the user see better dense network zones. As said before, an important requirement is to see attacked networks and dynamically go through attack steps with an animated visualization and the tool should have a way to compare two attack instances over the same network.

What was decided? – We decided to add to our application a page to visualize networks with the ability to print generated attacks over it. It has been done as a separated application (not in the same page as simulator) because user can want to visualize networks without executing a simulation. Also the visualization tool have a *mini* version that can be used to compare two attacks in a simulation from the logs page as seen in figure 8.3



Set simulations

Status	Number	Graph	Element	Type	Parameter	Date
✓	1	HR-100-GI02 (Jose L Marzo Lazaro)	node	random	sequential	23/02/2017 17:55

FIGURE 8.3: Different attacks compared using the visualization tool. Attacked elements are marked as red and alive ones are painted with black.

Our visualization tool must be accessible from any computer connected to internet and user can upload their own networks to visualize them using the tool. User is able to zoom and move around network with simple mouse gestures and can see elements attributes when situating cursor over them.

8.6 Implementation decisions

We basically have two important requirements that our developed application must satisfy. First one is to be available from a computer worldwide without installing anything on it and second one is to not merge simulations execution code with application code. First requirement is solved by developing a web application which is available with any web browser and second requirement is achieved by adapting application to be compatible with simulation R code and not backside, that means application must simulate what research group members had to do manually.

Technical implementation details are explained in further chapters but here a basic explanation is following. In client side (what user computer executes) we have decided to use **HTML5** with **Javascript** and **JQuery** as every web application merged with new frequently used frameworks as **Bootstrap** and **AngularJS**. In visualization tool we decided to use the Javascript library **D3.js** that helps drawing amazing visualizations over a canvas inside a web page.

In server side, all code is based on **NodeJS** with a bunch of frameworks available for it. As simulations are supposed to be changing so frequently, we decided not to use a SQL database and use a **noSQL database**, in our case **MongoDB**, which allows changing database structure easily than a SQL one. The NodeJS server is going to act as a previous research group member, that is generating the *string text* of each simulation and execution the R code to launch simulations.

Chapter 9

Analysis and design

In this chapter, the analysis of the implemented project is presented, explaining in detail the design of its content. First, all simulation R process design is explained which is important to understand how then the developed web application is designed to work with it. Then both client-side and server-side design is detailed in order to see how they are structured.

9.1 Previous simulation R process analysis

This section explains only the design of the simulation process previous to web application development, that is how it worked before this project. As said in studies and decisions chapter, one requirement was to not change anything in this side and make new web application be adapted to existing simulation R process design.

Functionalities are grouped into different modules and **all the user interaction is done through a general command line interface module**, which has the proper options to access to all the other modules functionalities.

Basically, the other modules are classified in:

Auxiliary modules – They basically manage the input and output data, which is specified below.

Functional modules – They implement the main processes and calculations to satisfy the requirements.

The involved data is organized within the project folder structure, which is basically composed by:

"resources" folder – It includes all necessary data to execute simulations that don't change dynamically by simulation parameters as for example graphs files in GraphML format. For the correct functioning of the whole system, it is only required that the user provides the topology source files and declares the desired experiments setting parameters correctly.

"out" folder – It contains a subfolder for every experiment. Each subfolder contains the saved data related to the results of the main calculation processes (PCA process, robustness surface process, generated attacks and metrics...) in CSV format. It also contains a Microsoft Excel spreadsheet with all these results presented in a more human friendly way, that is why the interface module also provides the necessary communication between the user, the data and Microsoft Excel itself.

"conf" folder – It contains a configuration file with the basic environmental and path variables. In the beginning, it is necessary to set the value of some variables of the configuration file.

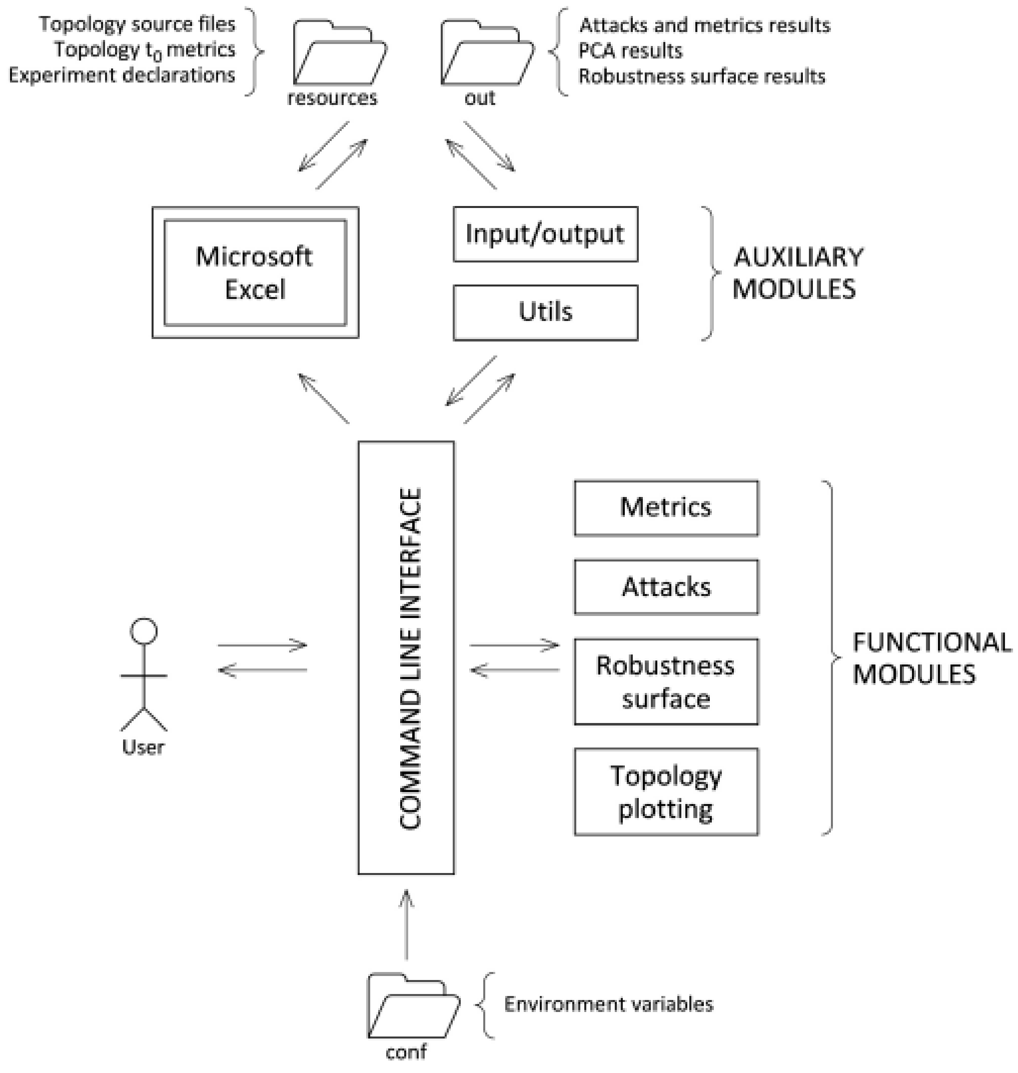


FIGURE 9.1: R process design scheme.

In figure 9.1, can be seen all modules and folders structure explained above. We can also see a user actor which developed web application will act as.

9.2 Web application analysis

In this section an analysis of whole new developed system including the web application and the R simulation process. All logical modules can be seen in figure 9.2 and are explained below:

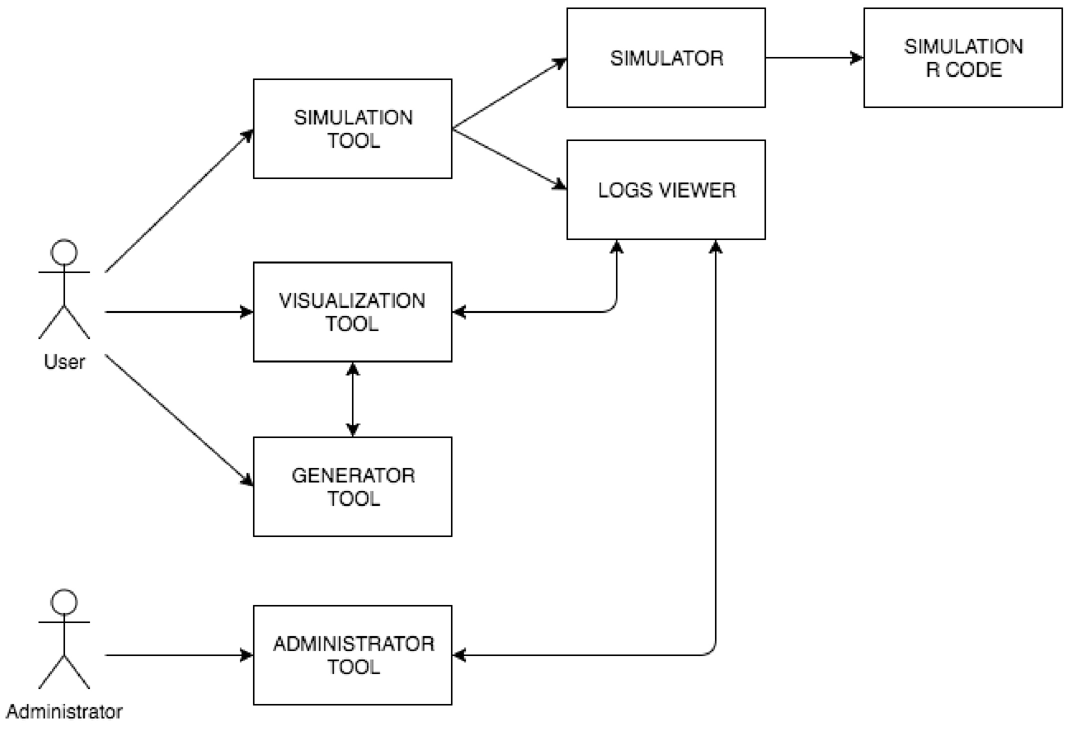


FIGURE 9.2: Web application modules.

Simulation tool – This module implements the functionality of executing all set of simulations. It gives user all options to define each experiment and executes it. It is divided in two sub-modules, on one hand there is the simulator itself which performs the computation acting as the user who executes the existing previous simulation R code. On the other hand, there is the logs viewer module which maintain records about all simulations launched and it is able to get their results. Logs viewer module also is able to get those simulation results and show to user through the visualization tool module.

Visualization tool – This module is able to make nice visualizations of networks and let users to manage and interact with them. For representing simulations results as resulting attacks over a network it needs to ask for simulations results to logs viewer module.

Generator tool – This module helps users to generate desired networks topologies and import them to whole system. User is able to select from a large list of available topologies and then, before importing the generated network, is able to visualize it using the visualization tool module.

Administrator tool – This module let the administrator manage all simulations done by all users. Administrator is able to see all simulations executions, download their results, remove desired simulations, stop simulations executions, etc. but it is not able to execute new simulations and, for this reason, the administrator module is only connected to logs viewer module.

In next chapter is explained how these logical modules are implemented because in implementation some of these modules are implemented together.

9.2.1 Use cases

All use cases related to application functionalities can be seen in figure 9.3.

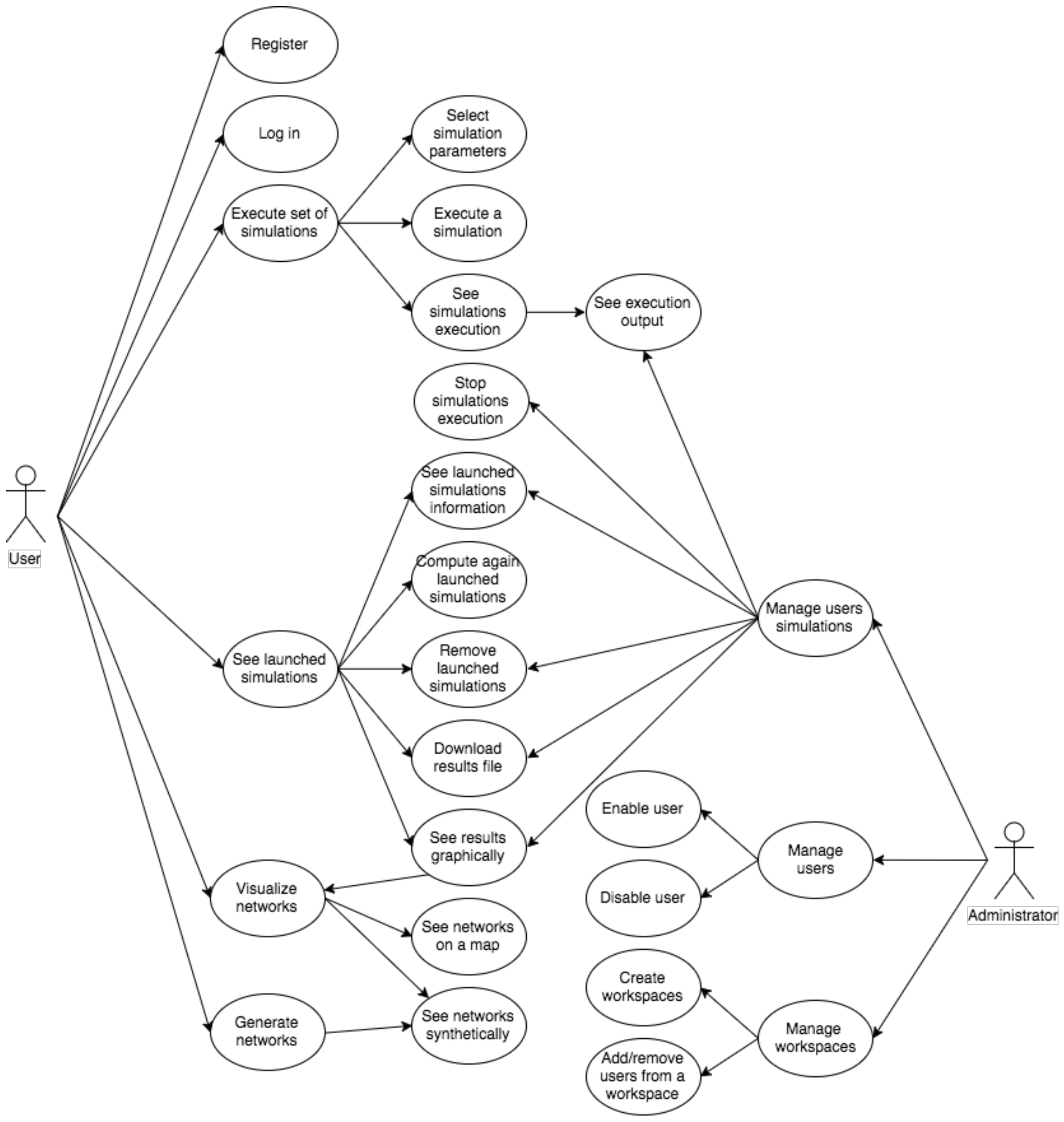


FIGURE 9.3: Application use cases diagram.

Basically it can be seen all functionalities an user is able to do and which of these the administrator is able to do too. Administrator can also do some other functionalities related to manage simulations, users and workspaces.

Below some of these use cases are detailed step by step using use case tables. Only the most important use cases are detailed.

Compute new set of simulations

Computing a new set of simulations is the main functionality of this project, so here is defined how this use case works:

COMPUTE NEW SET OF SIMULATIONS	
Actor	User.
Description	Prepare and execute a new set of simualtions.
Precondition	User is logged in and is enabled by administrator.
Principal flow	1. User selects workspace 2. User selects network 3. User selects simulation parameters as attack type, elements, number of elements attacked, seed, etc. 4. User selects desired metrics to use. 5. User selects start computation button. 6. For every combination (from all parameters selected): 6.1 Simulation combination is executed. 6.2 While executing, user can see output. 6.3 Simulation ends. 6.4 If simulation finished successfully 6.4.1 User can download results. 6.5 End if. 7. End for.
Postcondition	New simulations set is being executed and available in logs page.

TABLE 9.1: Compute a new set of simulations: use case table

Compute a simulation again

Many times it is necessary to compute again a previously launched simulation, for example when changing something in the code. Process steps are defined in use case table above:

COMPUTE A SIMULATION AGAIN	
Actor	User.
Description	Select a simulation and compute it again.
Precondition	User is logged in and is enabled by administrator. Simulation has been launched before.
Principal flow	1. User selects workspace. 2. User selects simulation set. 3. User selects compute again buttom from desired simulation. 4. Application shows the ready to compute page. 5. If simulation parameters are not as user wants: 5.1 Change attack parameters. 6. End if 7. User selects start computation button. 8. For every combination (from all parameters selected): 8.1 Simulation combination is executed. 8.2 While executing, user can see output. 8.3 Simulation ends. 8.4 If simulation finished successfully 8.4.1 User can download results. 8.5 End if. 9. End for.
Postcondition	Simulation has been launched again and is available in logs page.

TABLE 9.2: Compute a simulation again: use case table

Visualize attack over a network

Once a simulation is done, a dynamic visualization of the attacks generated over the network is very helpful to understand what happened. This use case is detailed above:

VISUALIZE ATTACK OVER A NETWORK	
Actor	User.
Description	Visualzie a generated attack over a network using the visualizer.
Precondition	User is logged in and is enabled by administrator. Simulation has been launched before and has ended successfully.
Principal flow	1. User selects workspace. 2. User selects simulation set. 3. User selects desires simulation from set simulations list. 4. User selects the open visualizer button. 5. Application opens visualizer. 6. Visualizer ask for network data in JSON format to API Rest. 7. If network has not yet cached to JSON format. 7.1 API Rest convert network structure to JSON and cache it. 8. End if. 9. API Rest responses with network cached in JSON. 10. Visualizer displays network. 11. User selects attack settings option. 12. User selects attack combination to visualize. 13. Visualizer ask for attack data to API Rest. 14. API Rest responses with attack data. 15. Visualizer displays attack over the network. 16. While user wants to continue visualizing attack 16.1 User move the slider to move over the attack propagation. 16.2 Visualizer displays the desired attack moment over the network. 17. End while.
Postcondition	User has seen attack propagation dynamically over the network.

TABLE 9.3: Visualize a generated attack over a network: use case table

All functionalities available in the visualization tool have the same sequence. Once the network is loaded, all options done over that need extra data make a request to API Rest service to get these data.

Enable user

Unlike previous explained use cases above, this one is done by the system administrator. By default users can be registered but they have no access to application until administrator enables it, avoiding non desired people to use the application without permission. The use case process is:

ENABLE USER	
Actor	Administrator.
Description	Administrator enables an user.
Precondition	User has been registered using a social network.
Principal flow	1. Administrator enters admin panel with admin password. 2. Admin page shows all registered users. 3. Administrator selects desired user. 4. Administrator selects the enable user. 5. Admin page ask for a user name. 6. Admin page make a request to API Rest to check user name. 7. API Rest responses the request. 8. If user name yet exists. 8.1 Return to step 5. 9. End if. 10. Admin page make a request to API Rest to enable user. 11. API Rest creates a new privat workspace to user. 12. API Rest enables user with entered user name. 13. API Rest respones the request. 14. Admin page reloads page with updated users data.
Postcondition	User has been enabled, a user name has been set to him and a private workspace has been created.

TABLE 9.4: Visualize a generated attack over a network: use case table

All administrator options follow the same request sequence to API Rest to manage all users, simulations and workspaces. API Rest has to be sure that keep all database and file system meaningful.

9.2.2 Data model

After analysing the basic functionalities, we have to explain how the data model is. The data model (or database) is not really big due there are not many entities to define. Basic entities are Simulation, User and Graph but also we need other entities as Folder, Set and Visualization. Relationships between them is shown in figure 9.4 and explanation of them is available below:

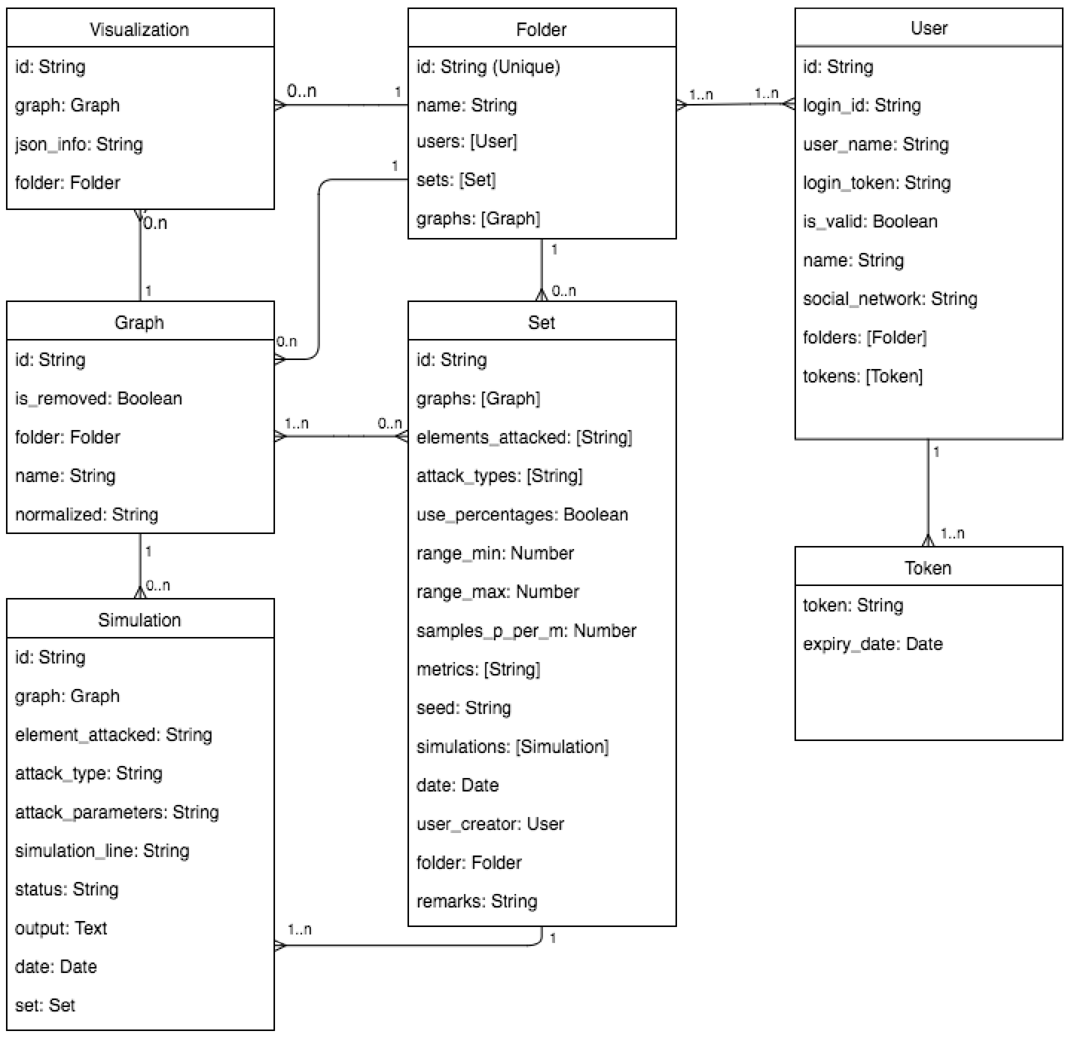


FIGURE 9.4: Database model.

User – Represents the final user that uses the web application. It has to register using a social network and it has references to his folders (or workspaces) and tokens. Attributes[1] info are:

- *login_id*: Social network identifier (e.g. email used to log in to Facebook). Example: **email@gmail.com**.
- *user_name*: User name in the system. Example: **sergiogc9**.
- *login_token*: Unique token string to identify user in selected social network (necessary for using social network API).
- *is_valid*: Boolean that identifies user as enabled or not enabled. Remember that users are not enabled until administrator enables them.
- *name*: Name to display in application. It is automatically filled when registering from a social network. Example: **Sergio Gómez**.
- *social_network*: Identifies the social network used. Example: **Facebook**.

[1]Only non relation attributes are detailed. Also entities have always an unique identifier that is also not detailed.

Token – It is needed to let sessions opened until a certain time. A token is valid until expiry date and every time an user log it has to use a valid token. Attributes are:

- *token*: The token itself in string format.
- *expiry_date*: The date since the token is not valid.

Folder – Represents a workspace[2]. It has references to users that have access to it, to all sets launched in the workspace background and to all graphs available in it. Remember that a graph is only related to a workspace. Entity attributes are:

- *name*: The name of the workspace that will be displayed in application. Example: **Personal Worskpace**.

Set – Contains a set of simulations. It has references to all simulations inside the set and to the folder it has been launched in. Remember that when executing a set, the system automatically creates all simulations combinations from all selected parameters and hence an array of related graphs are referenced too. A reference to the user that launched the execution is saved also. Set's attributes are:

- *elements_attacked*: Array with all elements to attack. Example: **["node","edge"]**.
- *attack_types*: Array with all attack types to generate. Example: **["random","targeted_sequental","epidemic"]**.
- *use_percentages*: Boolean that defines if range_min and range_max values are expressed in discrete values or in percentages.
- *range_min*: Integer that defines the minimum number of elements attacked. It can be expressed as discrete numbers or as percentages. Example: **1**.
- *range_max*: IInteger that defines the maximum number of elements attacked. It can be expressed as discrete numbers or as percentages. Example: **20**.
- *samples_p_per_m*: Define the number of attacks combinations to generated in each simulation. Example: **25**.
- *metrics*: Array with all metrics identifiers that will be computed in each simulation. Example: **["01-AND","02-HET","05-EFF","28-TRH"]**.
- *seed*: Seed to use when randomizing. If no value is entered a default one is set. Example: **123456789**.
- *date*: Date when set has been created.
- *remarks*: A sentence to describe the set. It is shown when listing all sets in application. Example: **"Epidemic attacks over real networks"**.

Simulation – Represents an unique simulation launched and executed. It is one of the combination generated by system when launching an experiment. It has references to used graph and to its set. It has several attributes explained below but other that are common to all simulations in the same set are available through the set and are not saved in the simulation entity:

[2]In server side code the concept of folders is used, but then in application we changed it to workspaces.

- *element_attacked*: The element attacked. It can be nodes or edges. Example: **"nodes"**.
- *attack_type*: Attack type to generate attacks. Example: **"epidemic"**.
- *attack_parameter*: Attack parameter when needed. When not needed it has a value of **"NA"**. Example: **"sequential"**.
- *simulation_line*: The string like command line used to execute simulations in R code previously to this project. Example:
  exp1, node, targeted, sequential, "01-AND,02-HET,03-ASPL", 30, 20, 123456.
- *status*: The status of the simulation execution. It can be **not_started**, **started** or **finished**.
- *output*: R execution output. It can be a large text.
- *date*: Date of execution start. It should be different than simulation's set date, because simulation execute can start after set is created.

Graph – Represents the graph or network available in application. Graphs are related to one folder or workspace so a reference to it is needed. Entity attributes are:

- *is_removed*: Boolean that tells if graph is removed. Graph must be marked as removed but not removed to let users visualize simulations launched before its removal, that is removing a graph only implies disabling it from selecting for new simulations.
- *name*: The name of the graph or network. The name is set from GraphML file name. Example: **GpENI_L2.graphml**.
- *normalized*: The normalized name. It is needed for ordering purposes and for saving GraphML file in server. Example: **gpeni_l2.graphml**.

Visualization – Handles a fixed position of a network. A graph has a unique representation in each folder or workspace, so a reference to a graph and a folder is needed. The unique visualization attribute is:

- *json_info*: A JSON format string with all graph nodes positioning using width and height percentages. Example: **"{"node1":{x:0.4,y:0.3}, "node2":{x:0.9,y:0.7}}"**.

9.3 Application functionalities

This section lists all web application functionalities. First, application users functionalities are listed followed by administrator ones. Then a most important functionalities behaviour depth explanation is detailed to give some background about how whole system works and see interaction between client-side and server-side.

9.3.1 User functionalities

Users have access to simulator, visualizer and generator. Each tool features are listed separately but some of these features are related to two tools.

Simulator user functionalities

- Register and log in using different available social networks with no need to use a password.
- Select a workspace to work with. An user can only use workspaces administrator has given him permission to use.
- Compute a new experiment, that means launching several simulations executions grouped in a set. Includes selecting a network, experiment parameters and metrics.
- See live execution logs when launching simulations.
- See previously launched simulations. Users first see launched sets and then can see a unique simulation selecting it. Includes seeing both set and simulations used data.
- Edit and compute again launched sets which includes automatically fill again all set parameters in computing page. Same functionality is available with a single simulation.
- Delete a set including deleting all simulations in it. All results files are also deleted too.
- See launched simulations output.
- Download launched simulations XML result file.
- See robustness surface as a bitmap (as an image).
- Compare two attack combinations using visualizer selecting both in a dynamic robustness surface. Dynamic robustness surface can be shown unordered or ordered (omega robustness surface).
- Preview simulation results in visualizer, including viewing network in synthetic mode or placing network on a map.

Visualizer user functionalities

- Upload new networks. Networks must be in GraphML format.
- Import a network from public repository to current workspace.
- Remove desired networks from workspace.
- Choose viewing networks synthetically or on a map.
- See nodes and edges data placing mouse cursor on them.
- Change nodes display positioning changing manually a node position.
- Save to server current nodes positioning.

- Load from server saved nodes positioning.
- Filter nodes and edges. Includes showing and/or hiding nodes and/or edges and filtering depending on attributes values.
- Paint nodes and/or edges depending on attributes values.
- Search for desired elements using a text value.
- Personalize networks aspect, changing nodes and/or edges size and colours.
- Download filtered network[3] in GraphML format.
- Download filtered network as displayed in SVG format.
- See nodes and/or edges attributes statistics.
- Compute network analysis (e.g. calculate degree, edge betweenness, etc.) over filtered network and see results in visualizer dynamically.

Generator user functionalities

- Select different kind of network topologies (e.g. tree, exponential, hypercube, dcell, etc.).
- Select parameters for each kind of topology.
- Preview generated network using visualizer.
- Add generated network to available networks list in workspace.

9.3.2 Administrator functionalities

Administrator is able to manage users, simulations and workspaces but it has some limitations as, for example, he can not launch simulations. All administrator available features are:

- Enable and disable users.
- Delete users.
- Add and remove workspaces.
- Add and remove users in a workspace.
- See all set and simulations launched.
- Same functionalities as users when seeing launched simulations, including downloading and previewing results.
- Remove launched sets.
- Stop current simulation execution.

[3]Filtered network is current displayed network in visualizer, that means whole network without hidden elements.

9.3.3 Functionalities workflow

Before analysing functionalities workflow is important to define both **client-side** and **server-side** concepts:

Client-side – It is all code executed in the user computer, i.e. in web navigator. All simulator, visualizer and generator tools are web-based applications, that means a web navigator is needed.

Server-side – Server-side code is these code executed by server. Includes both R simulation execution code, the API Rest needed for giving the data to client-side web application and database management.

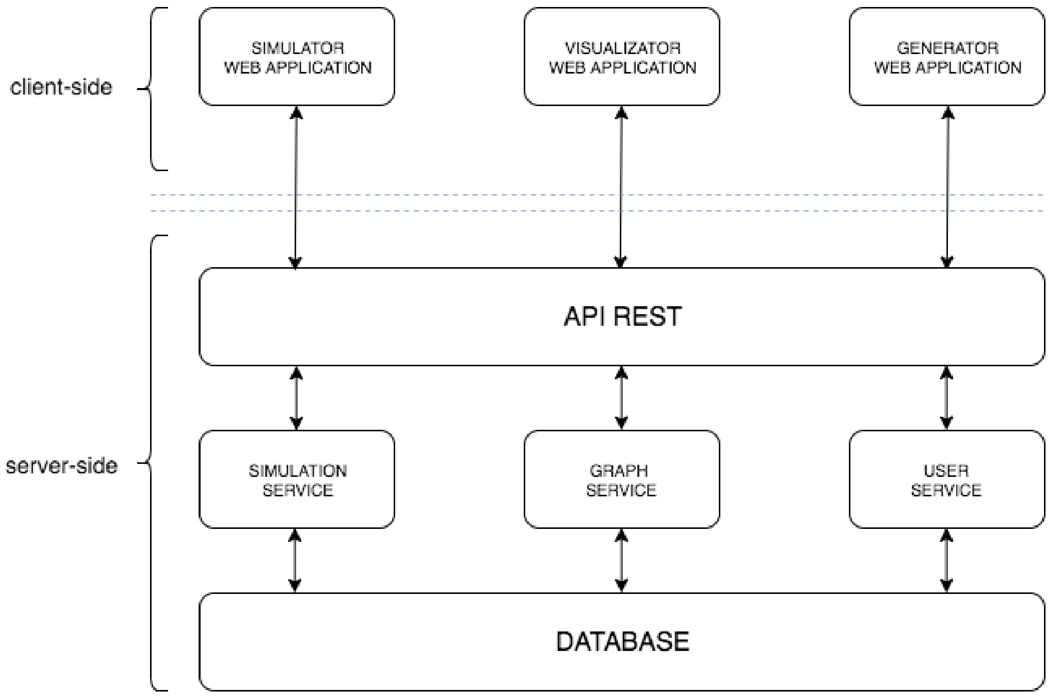


FIGURE 9.5: Logical server-side and client-side structure.

In figure 9.5 a **logical structure** of server-side and client-side interconnection can be seen. In next chapter a depth technical explanation about how it is really implemented is detailed. Client-side applications make requests to a unique API Rest that handle all these requests using different services depending on the request topic. These services act as intermediate between API Rest and database and simulation service is also the intermediate between whole web application system and R simulation execution code.

Compute new set of simulations workflow

Compute a new set needs user entering all experiment parameters and then API Rest has to generate all simulation combinations followed by queuing them and start execution first of them if possible. Workflow steps are (figure 9.6):

1. User selects workspace, network, attack parameters and metrics.
2. User starts set computation.

3. Web application makes a request to API Rest.
4. API Rest check parameters and if they are correct, continues with process passing them to simulation service.
5. Simulation service generates all simulation combinations with received data.
6. Simulation service responses request with the number of simulations generated.
7. Simulation service starts the computing process and web application starts simulations status check process, both detailed next.

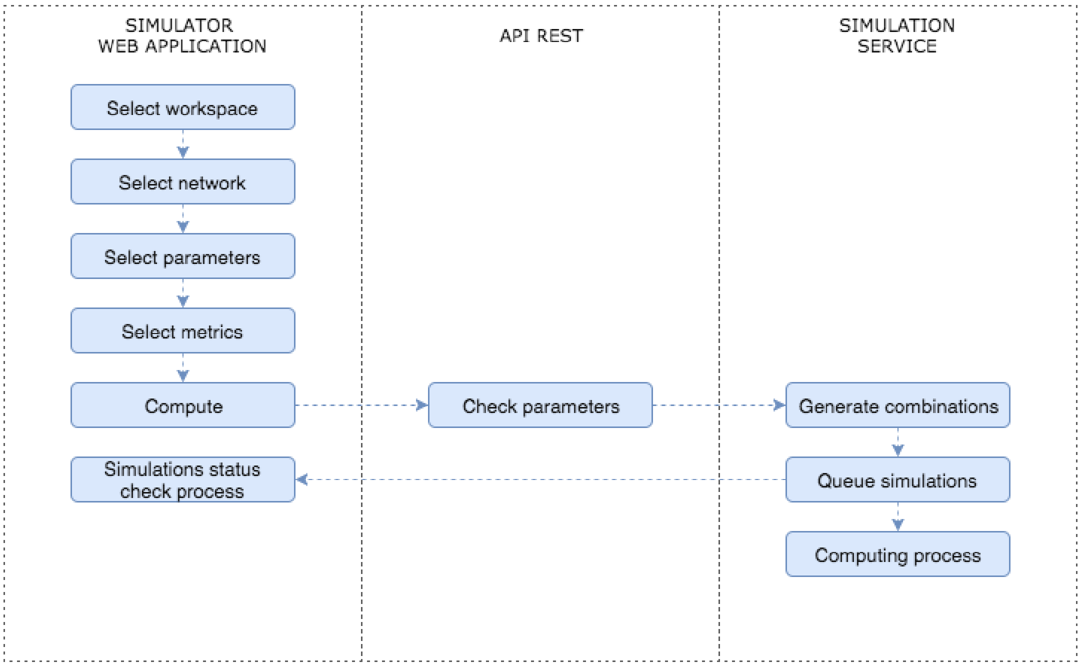


FIGURE 9.6: Compute new set of simulations workflow diagram.

Simulations status check process done by simulator web application gets the number of simulation combination generated. Its goal is to check simulation computation status starting from first one until all simulations have been computed. Workflow steps are (figure 9.7):

1. Simulator web application receives the number of generated simulations and their ids in request response.
2. Simulator web application queues all simulations to check.
3. While there are simulations queued:
   3.1. Get first queued simulation status.
   3.2. If not started display as not started, if running show output and if ended display as finished and remove simulation from queue.
4. Display all simulations have been computed message.

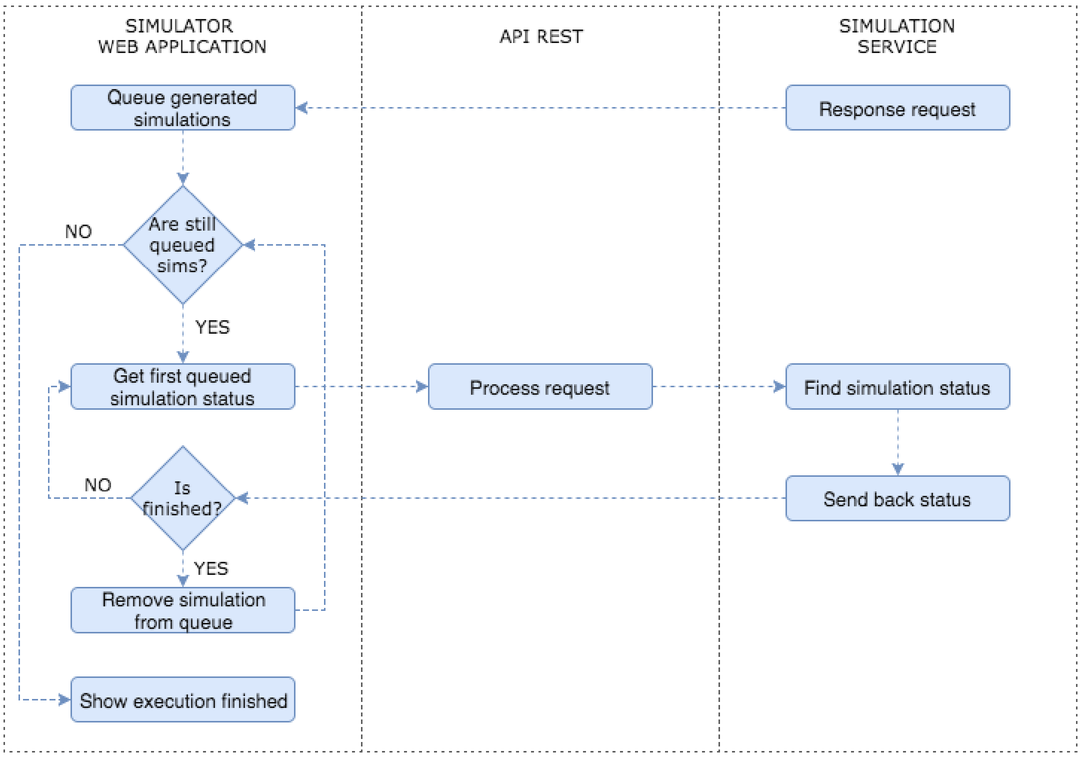


FIGURE 9.7: Check launched simulations computing status workflow diagram.

Computing process done by simulation service in server-side receives all generated simulations queued and its main goal is to compute all of them one behind other. Workflow steps are (figure 9.8):

1. Simulation service receives all generated simulations queued.
2. While queue still has a simulation to compute:
    2.1. Start simulation computation setting simulation status as *started*.
    2.2. While computing, save output into database.
    2.3. When computation finishes, if result file exists set simulation status as *finished*, otherwise set simulation status as *failed*.
    2.4. Remove simulation from queue.

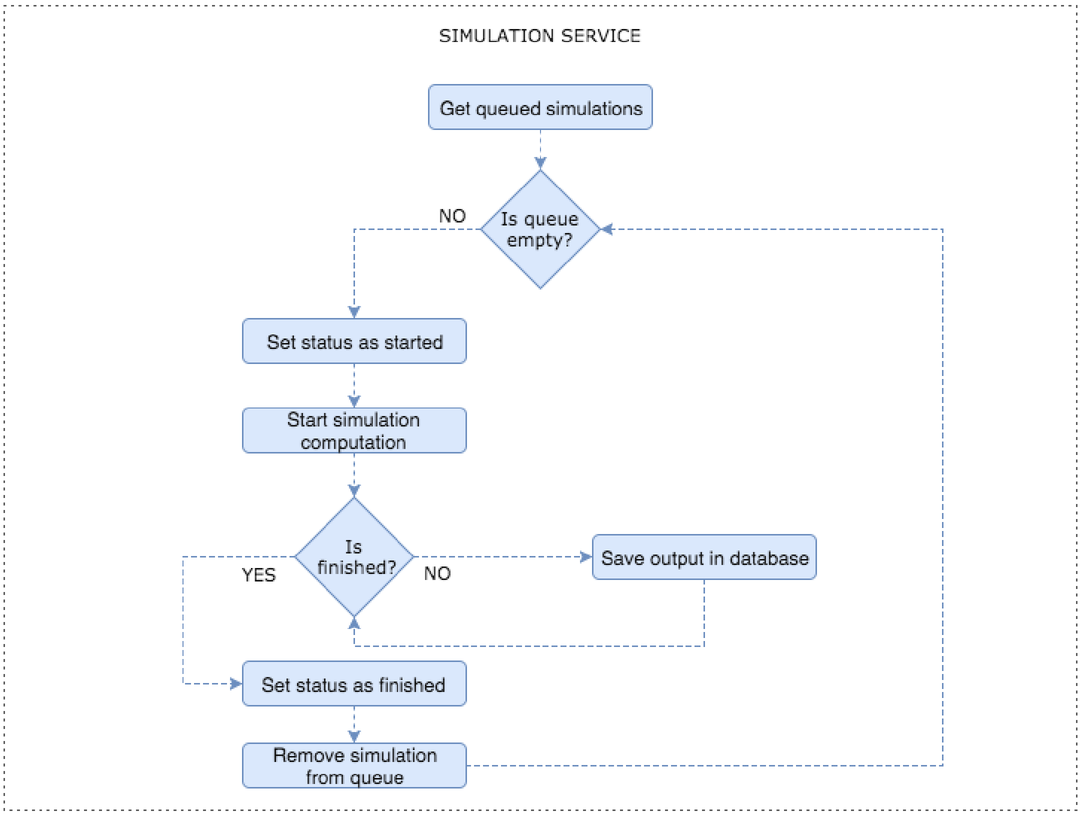


FIGURE 9.8: Generated simulations computing process workflow diagram.

Download simulation result XML file

After computing a set of simulations, next step can be downloading each simulation results. Remember that execution generates an resulting XML file with all calculated data. User has to select desired simulation and click on download file button. Workflow steps are (figure 9.9):

1. User selects workspace.
2. User selects simulation's set from listed sets.
3. User selects desired simulation from listed simulations.
4. If simulation's status is finished (i.e. status is not "not-started", "started" or "failed").
   4.1. Web application displays download results file button.
   4.2. User click on download results file button.
   4.3. Web application request to API Rest results file.
   4.4. Simulation service checks results file availability (it is possible that even execution was successfully finished, the results file could not be generated).
   4.5. If file is available, simulation service returns the file, otherwise returns an error page and web application shows a error page.

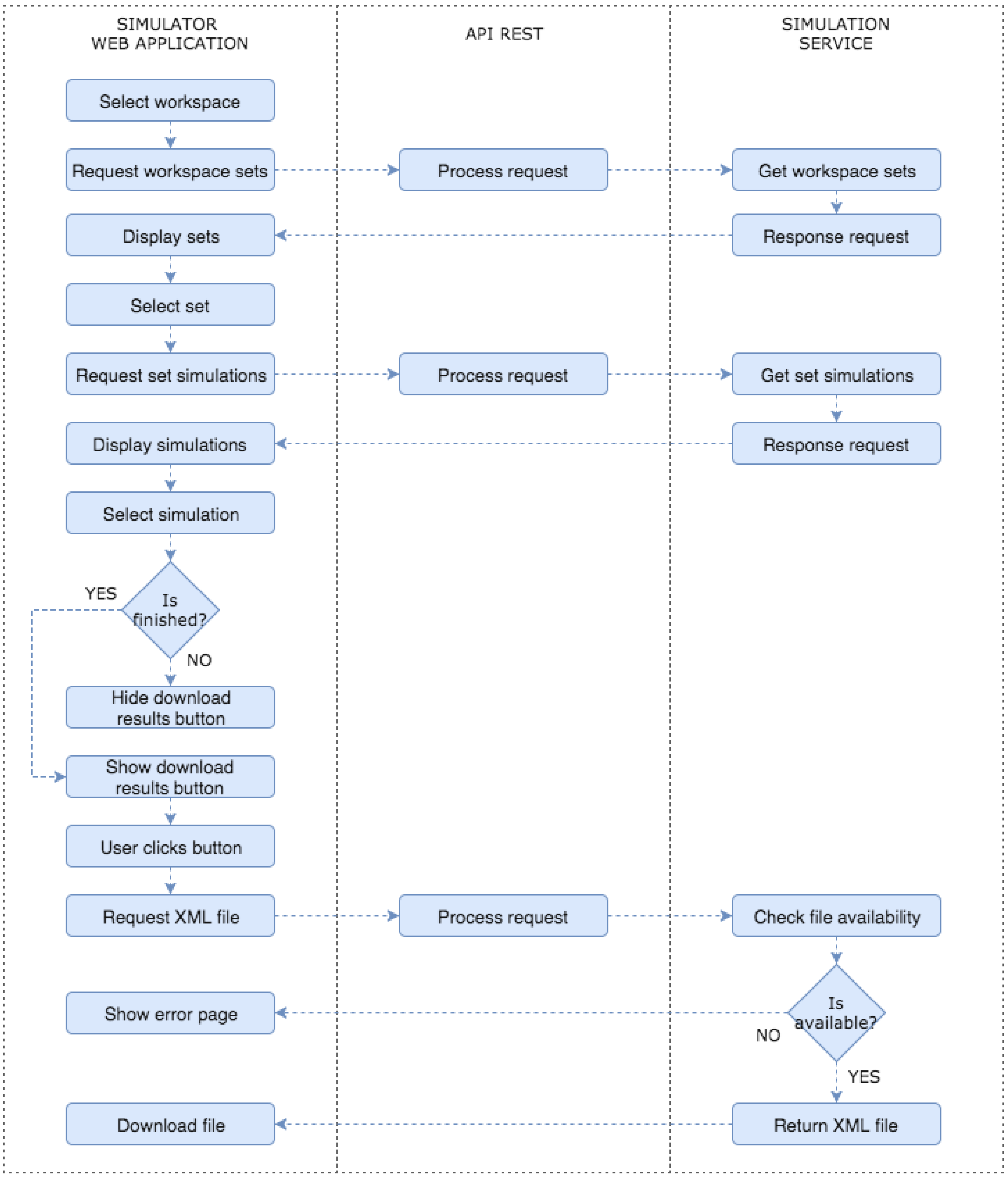


FIGURE 9.9: Download resulting XML file workflow diagram.

View generated attack in simulation with visualizer

This functionality user more than one tool, that is using both simulator and visualizer web applications in client-side and both simulation and graph services in server-side. At high level, workflow starts with user selecting desired simulation and clicking open visualizer button on it. Then, when visualizer has opened it has to select which generated attack combination wants to preview and visualizer will preview it on the network. Workflow steps are (figure 9.10):

1. User selects workspace.
2. User selects simulation's set from listed sets.
3. User selects desired simulation from listed simulations.

4. User clicks on open simulation in visualizer button.
5. Simulator web application opens a new instance of visualizer web application with parameters as the simulation id, the network used and others.
6. Visualizer web application gets the network from API Rest formatted in JSON by graph service.
7. Visualizer displays network.
8. User selects attack combination number (M) and number of elements attacked (P) to see on network.
9. Visualizer web application gets attack sequence requesting API Rest.
10. Simulation service responses request with desired attack combination sequence.
11. Visualizer receives sequence and prints it on network.

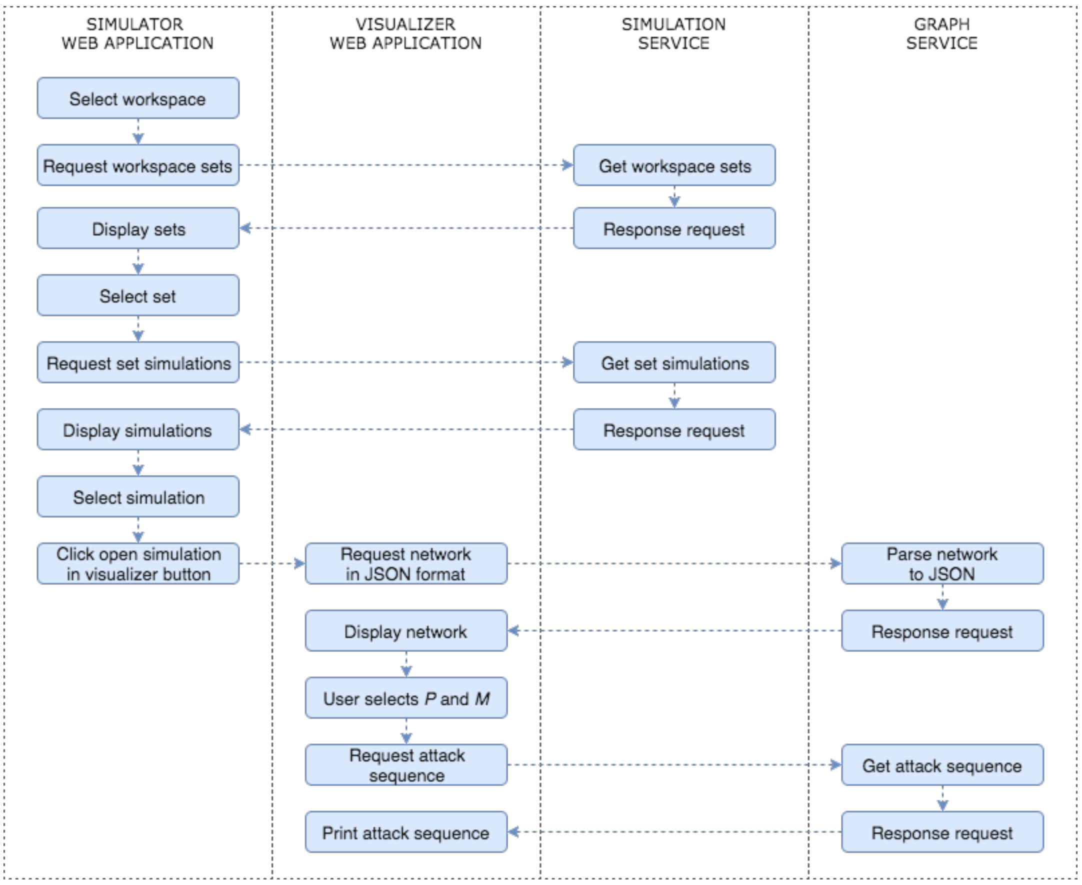


FIGURE 9.10: Preview a generated attack in a simulation using the visualization tool. API Rest is ignored to fit others but requests are handled by API Rest as seen in others functionalities workflows.

Chapter 10

Implementation

This chapter defines how logical modules defined in previous chapters are implemented identifying technologies used.

10.1 Server structure

Whole system is installed in a unique **Ubuntu 64 bits** production server which is hosted in a physicall server located in University of Girona, hence it is not in any cloud service. Also for developing we have a testing server similar to production one (with less resources).

10.1.1 Production and testing servers interaction

All development is done in testing server and after different tests changes are transferred to production server using version control system **Git** and service **Bitbucket**[1]. Development steps should be similar than detailed below (figure 10.1):

1. Make changes in code.
2. Test basic functionalities related to changes done.
3. Deep testing of all functionalities related to changes done. This step can be omitted when fixing bugs easy reproducible.
4. Commit all changes in testing environment.
5. When a stable tested version is reached, push changes to Git.
6. Pull changes in production server.
7. View and use changes in production server .

[1]https://bitbucket.org/

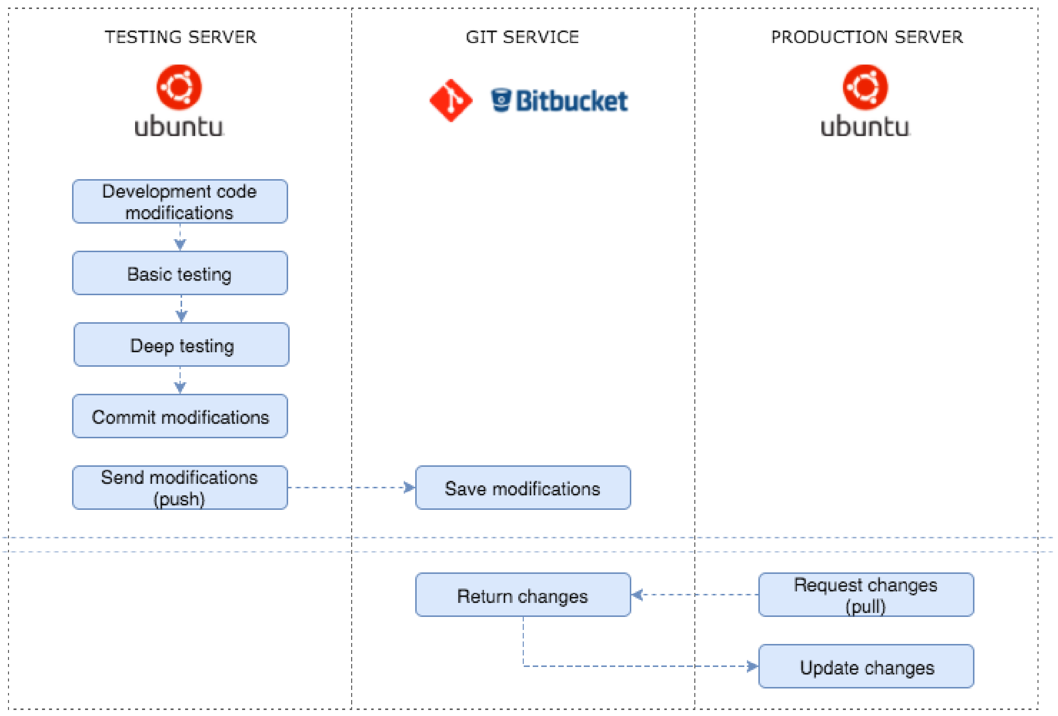
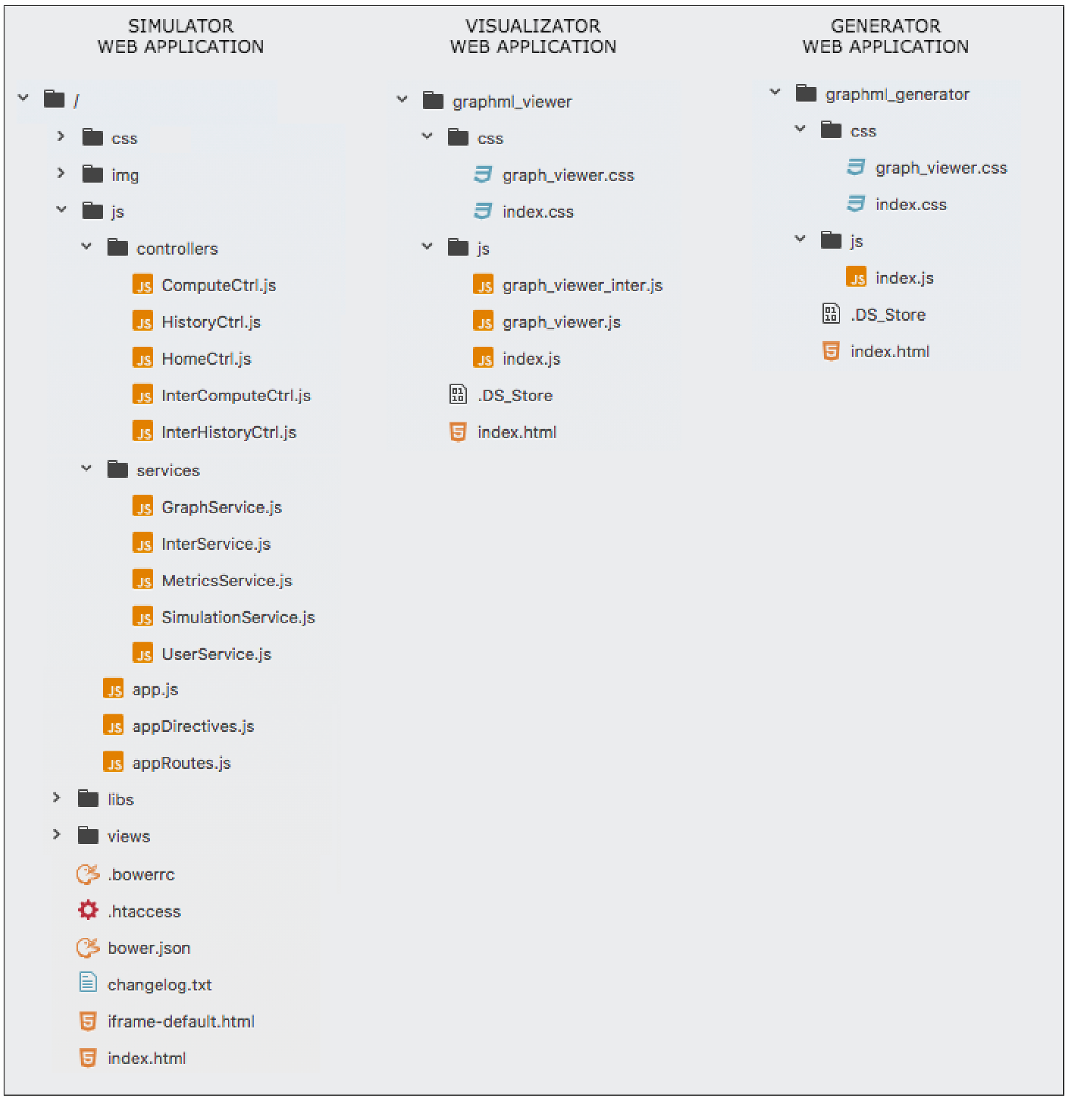


FIGURE 10.1: Development modifications interaction process using Git.

10.1.2 Client-side implementation

In previous chapter available web application logical modules in client-side are defined as simulator, visualizer and generator web applications. All three logical modules are implemented as single web page application, all of them available through an **Apache server**[2].

[2]https://httpd.apache.org/



```
SIMULATOR                         VISUALIZATOR                  GENERATOR
WEB APPLICATION                   WEB APPLICATION               WEB APPLICATION

/                                 graphml_viewer                graphml_generator
  css                               css                           css
  img                                 graph_viewer.css              graph_viewer.css
  js                                  index.css                     index.css
    controllers                     js                            js
      ComputeCtrl.js                  graph_viewer_inter.js         index.js
      HistoryCtrl.js                  graph_viewer.js             .DS_Store
      HomeCtrl.js                     index.js                    index.html
      InterComputeCtrl.js           .DS_Store
      InterHistoryCtrl.js           index.html
    services
      GraphService.js
      InterService.js
      MetricsService.js
      SimulationService.js
      UserService.js
    app.js
    appDirectives.js
    appRoutes.js
  libs
  views
  .bowerrc
  .htaccess
  bower.json
  changelog.txt
  iframe-default.html
  index.html
```

FIGURE 10.2: Client-side web tools implementation structure.

As seen in figure 10.2, both visualizer and generator tool have been implemented as a simple web page without any framework used and simulator tool has been implemented using frameworks as **AngularJS**[3], **Bootstrap**[4] and others.

On the one hand, visualizer and generator have a unique *index.html* customized with some css files and the core applications codes are in JavaScript files (.js). In case of visualizer, core JavaScript code is separated in three different JavaScript files, first related to main page functionalities as upload or import a network and two others related to the visualizer itself[5].

On the other hand, simulator uses a different structure based on AngularJS framework which, basically, loads dynamically the content and the interface of the single page. This structure has folders to maintain css files, images, all JavaScript core, necessary dependencies or libraries and all views to include dynamically. JavaScript

[3]https://angularjs.org/
[4]http://getbootstrap.com/
[5]When figure 10.2 was created, the interdependent networks visualizer mentioned in future work chapter was already started, hence it is visible in figure.

core is divided in main core files (*app.js, env.js, etc.*), controllers that control web page workflow and services which basically act as interconnection with API Rest and perform other useful functionalities. Extra necessaries libraries are defined in file *bower.json* and installed using **Bower**[6] in *libs* folder.

10.1.3 Server-side implementation

Server-side logical modules seen in figure 9.5 are API Rest, simulation service, graph service, user service and database. When implementing web application back-end it was decided to use a **NodeJS** based structure for API Rest and all three services, that is creating a server running with this technology listening for request from web application client-side (logical API Rest) and after processing request redirect them to necessary service depending on request type. Then, we decided to use **MongoDB**[7] as a **NoSQL** database as detailed in analysis and decisions chapter to save whole application data thanks to middle-ware (or library) **mongoose**[8] which joins NodeJS server with MongoDB database.

All necessary components seen in figure 10.3 are those necessary to use the system successfully. Aside from NodeJS server and database we need to be able to execute R simulation process code which is done through a **bash script** which really acts as a research group member executing the R code.

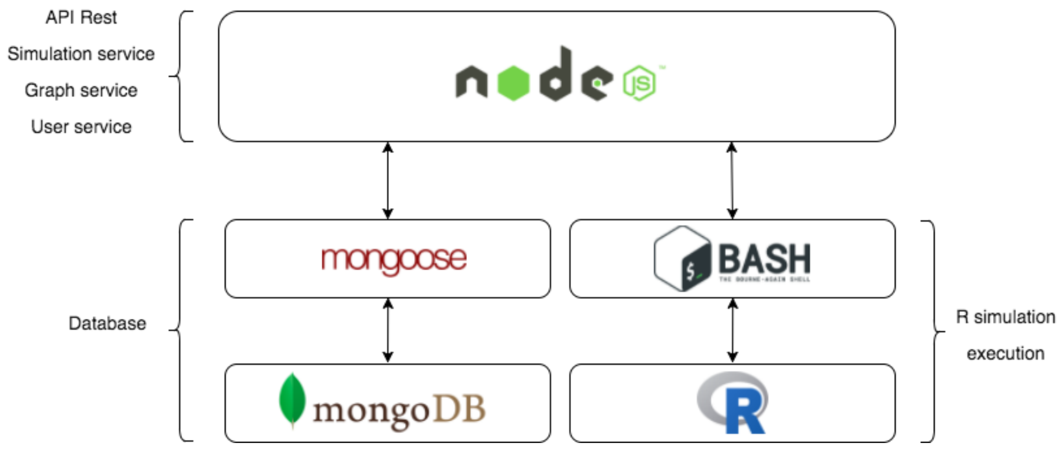


FIGURE 10.3: Server-side implementation components structure.

Server-side structure individual components are:

server.js – Is the NodeJS core server itself. It creates the server and starts listening requests, then load necessary packages, libraries and middle-ware as moongose or exec_child (necessary to execute bash scripts asynchronously). As the file name extension tells it is coded in JavaScript.

package.json – Is the file where necessary dependencies are defined. Before executing NodeJS server dependencies have to be installed using **npm**[9]. Npm will find

[6]https://bower.io/
[7]https://www.mongodb.com/
[8]http://mongoosejs.com/
[9]https://www.npmjs.com/

for dependencies in package.json file.

models* folder** – Contains all database entities defined as mongoose middle-ware defines. Each entity has each own *.js* file defining a class of type ***mongoose.Schema. With these files mongoose is able to know MongoDB database structure.

compute_simulation.sh – It is a bash script that executes the R code to compute a unique simulation. It is launched by NodeJS server and receives as parameter the full string line used to compute a simulation in R code.

analysis_script.sh – It is a bash script used when visualizer requests a live network analysis (e.g. nodes degree, network diameter, etc.). It is called by NodeJS server who has saved before in temporal directory the network to analyse.

***tmp* folder** – Contains temporally files as, for example, generated networks. It is emptied always by NodeJS when it is launched.

***R* folder** – Contains whole R code developed before this project.

10.2 Performance improvement

An important point when executing simulations is to achieve a optimum performance reducing the execution time so in this project a performance improvement has been done. In other chapters it is said that one requirement of application is not modifying existing R code but this improvement required changing this R code because it is the core of simulations executions.

10.2.1 Paralleling metrics calculation

Existing R code was totally sequential, that means there probably were a way to improve executions paralleling steps. We analysed every step in order to check if paralleling that step causes a remarkable performance improvement. Each step analysis were:

Importing network – It is a unique work, so no paralleling option can be done here.

Initials metrics computation – Computing metrics require a lot of computation time in some metrics. For that reason paralleling this step can save execution time but only when many metrics are selected, because for a less number of metrics it doesn't worth it. We decided to parallelize this step executing each metric in a different thread.

Attacks generation – This step can be paralleled but we decided not doing it because generating attacks usually spends no much time.

Metrics computation – As said in *initial metrics computation* step, this step is paralleled too, calculating each metrics in a thread. As seen in figure 10.4 all metrics are paralleled in every P number of elements attacked, that is calculating metrics for

$P = 1$ elements attacked is paralleled, then execution waits until all metrics executions have finished (i.e. all threads finished) and then follows with same procedure with $P = 2$, $P = 3$, etc.

Principal component analysis – This step is executed using external libraries, hence no paralleled improvement is done.

Robustness value and surface calculation – These steps are fast to execute so there is no need to parallelize them.

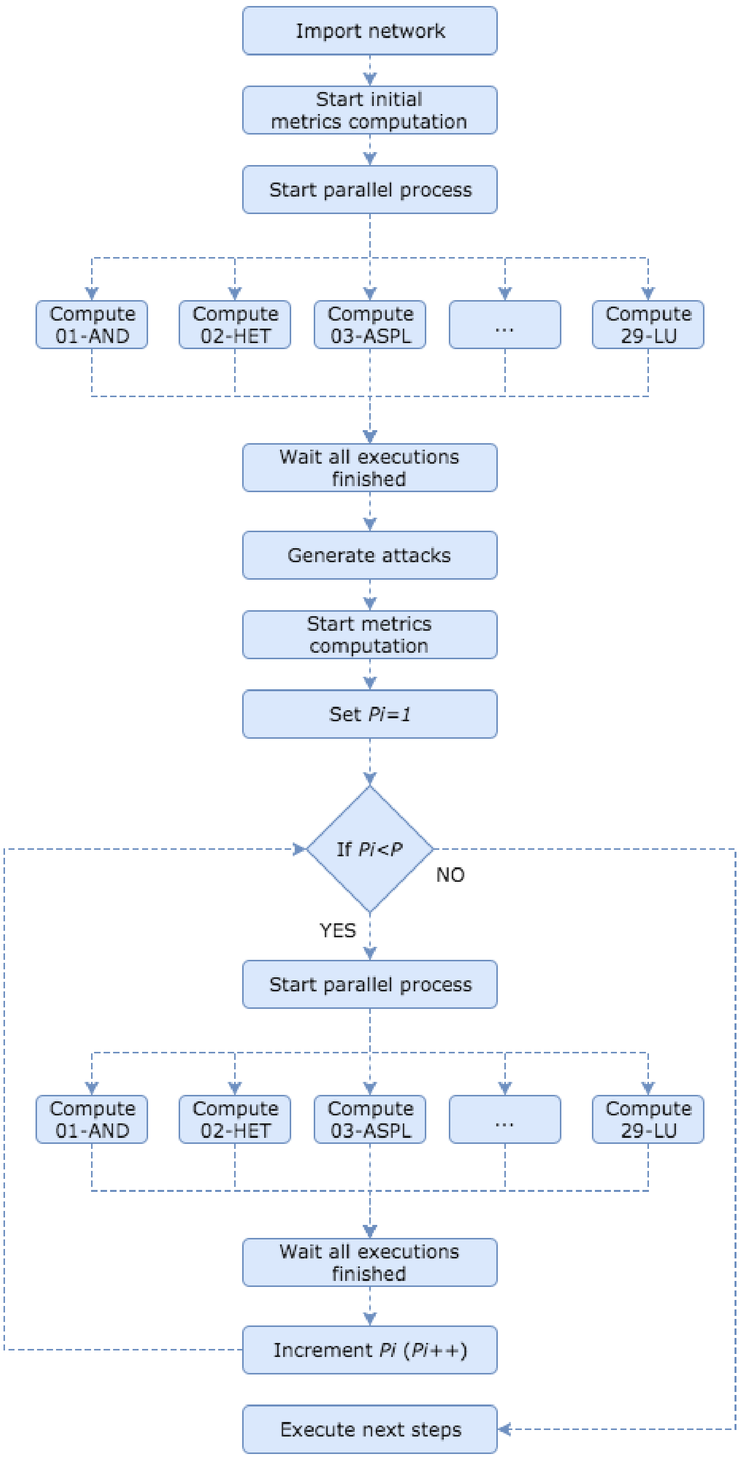


FIGURE 10.4: Metrics calculation paralleled process. P is the maximum number of elements attacked.

10.3 Collaborators code implementation

During this project some collaborations with others institutions working in network analysis have been done. Collaboration details are explained in next chapter but here we explain how their code has been added to our system.

External code is always set as a *black box* that our simulator makes request to it, that is important because that way we don't modify that code but collaborators do, so works is clearly separated.

10.3.1 Epidemic attacks

In collaboration with Kansas State University (KSU) we added their code to generate epidemic attacks over a network in our simulator. As they work too with R adding their code was not hard, we simply added in a folder their code which have a *public* function with some parameters that generates and returns a list of attacked elements. Execution and interaction process can be seen in figure 10.5.

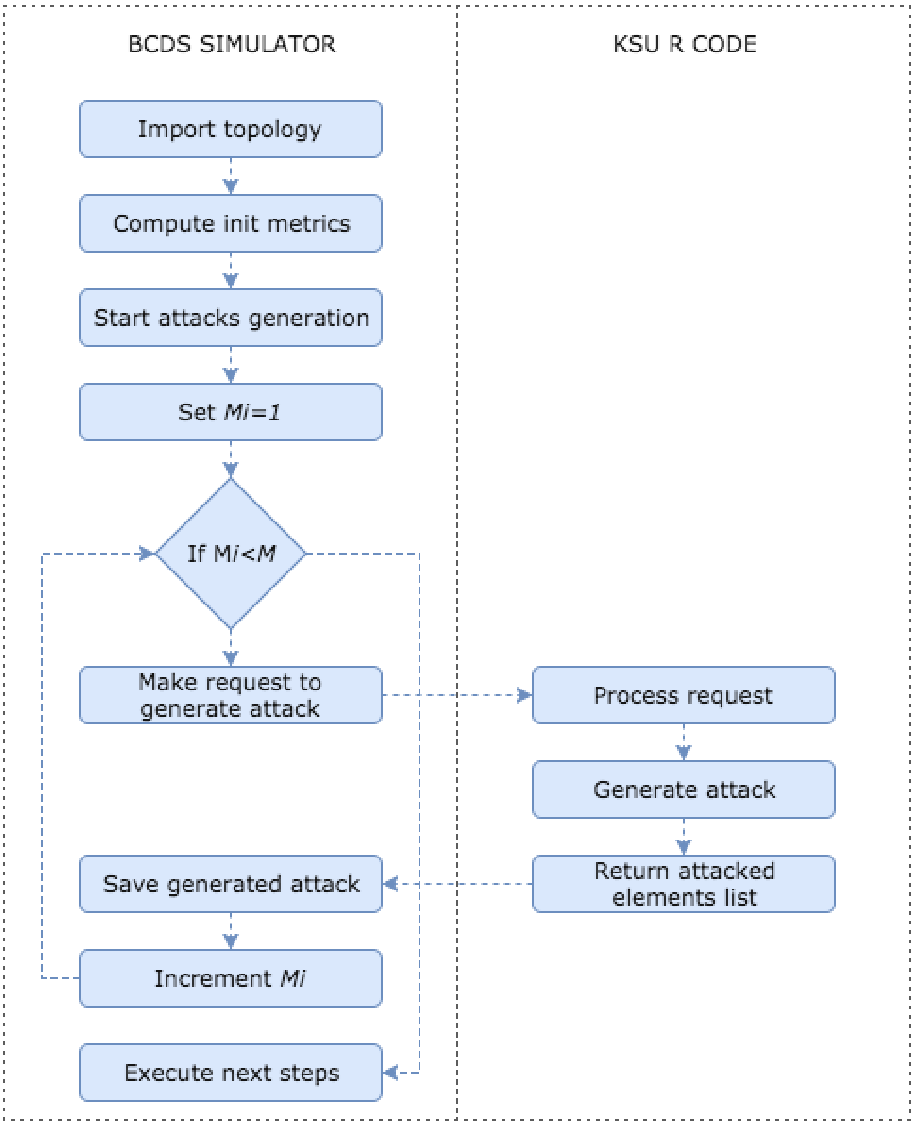


FIGURE 10.5: Simulator and KSU code interaction process. *M* is the number of combinations in experiment.

10.3.2 Throughput and link-usage metrics

In collaboration with Technical University of Munich (TUM) we added two metrics (28-TRH and 29-LU) to our system which are explained in next chapter. As TUM develop in Python a *middle box* written in Python had to be developed in order to communicate both simulator and TUM Python code. This middle box is written in Python and it aim is to receive requests from simulator (as a command python call from system) and then execute TUM Python API. This procedure is needed because there is no way to call directly in R a function defined in Python. Execution and interaction process can be seen in figure 10.6.

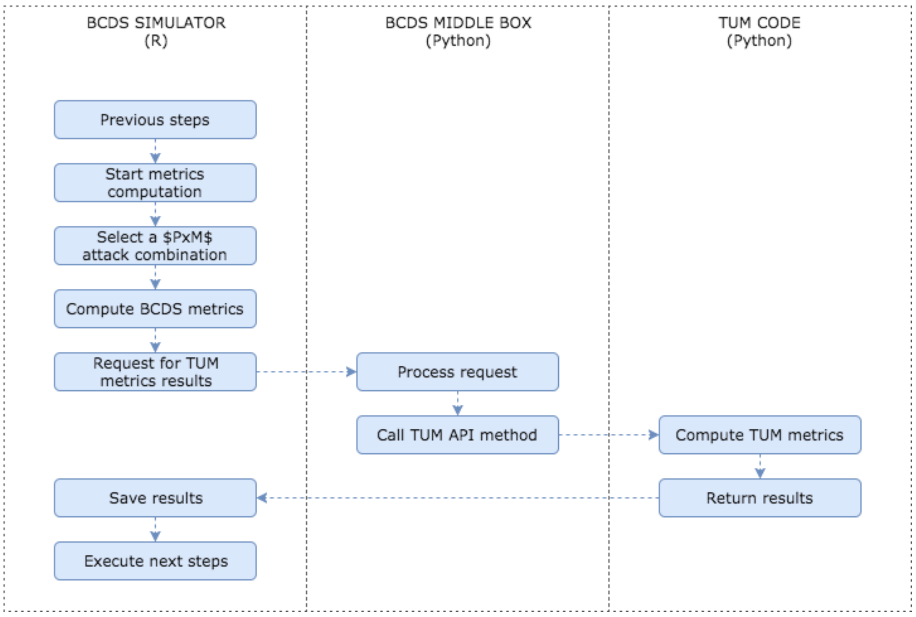


FIGURE 10.6: Simulator and TUM code interaction process.

Chapter 11

Results

This chapter's aim is to show the developed application when finished and to detail collaborations done with other institutions in this project. As a whole project result, the application has been developed successfully as can be seen below.

11.1 Developed application

Web application is available worldwide in research group servers[1] through login page with some social networks login options as seen in figure 11.1.

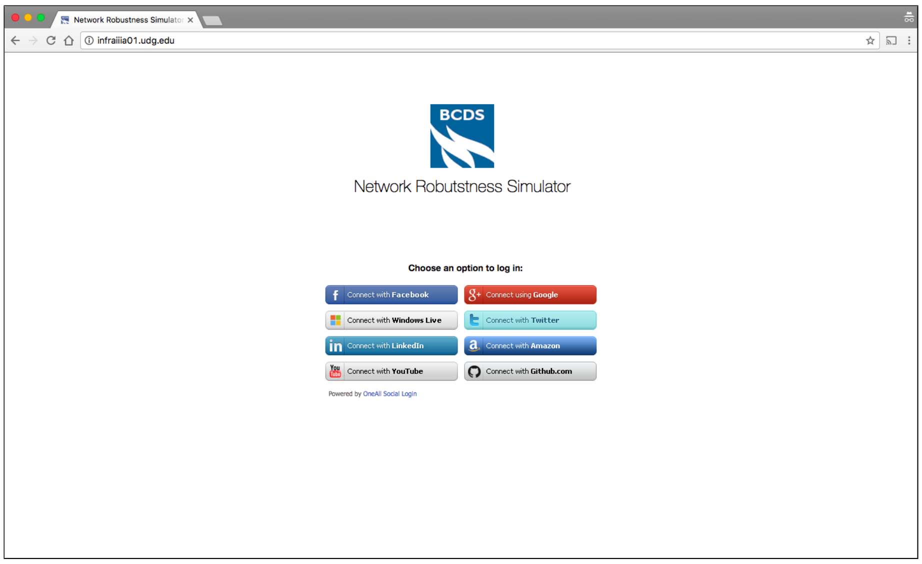
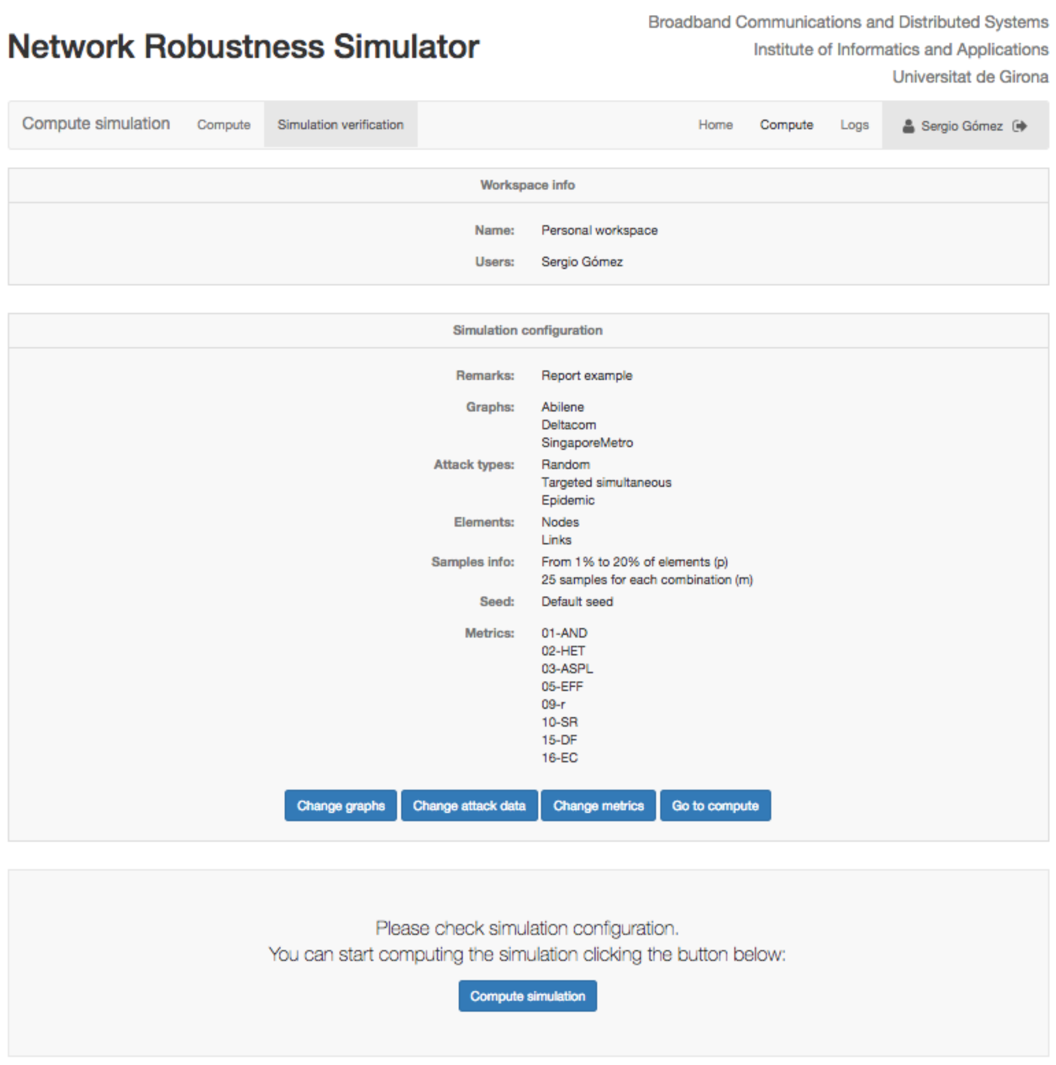


FIGURE 11.1: Application login page. User can register and login using different social networks.

[1]http://infraiiia01.udg.edu



Workspace info

Name:	Personal workspace
Users:	Sergio Gómez

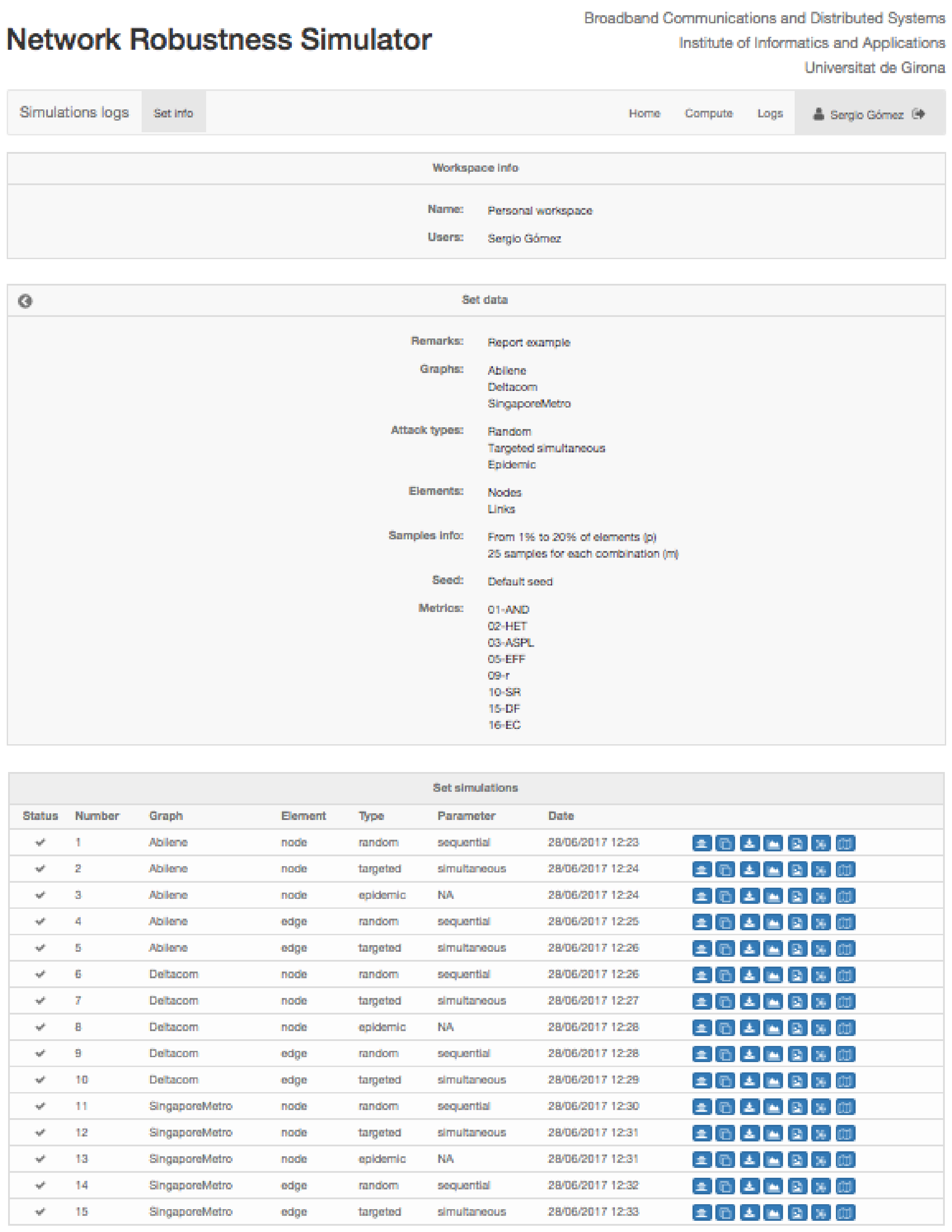
Simulation configuration

Remarks:	Report example
Graphs:	Abilene Deltacom SingaporeMetro
Attack types:	Random Targeted simultaneous Epidemic
Elements:	Nodes Links
Samples info:	From 1% to 20% of elements (p) 25 samples for each combination (m)
Seed:	Default seed
Metrics:	01-AND 02-HET 03-ASPL 05-EFF 09-r 10-SR 15-DF 16-EC

Please check simulation configuration.
You can start computing the simulation clicking the button below:

FIGURE 11.2: Last step before computing a new experiment. User can check all selected parameters before starting computation.

Then user is able to select a workspace and select parameters to compute a new experiment. This can be seen as an example in figure 11.2 where user is in last step previous to start computation. It can be seen all parameters selected in order to check them before executing the experiment. The experiment launched is done over Abilene, Deltacom and Singapore metro networks against random, targeted and epidemic generated attacks over both nodes and edges. Also from 1% to 20% of elements are attacked (P) and 25 different attack combinations samples (M) are used.



Network Robustness Simulator

Broadband Communications and Distributed Systems
Institute of Informatics and Applications
Universitat de Girona

Workspace info

Name:	Personal workspace
Users:	Sergio Gómez

Set data

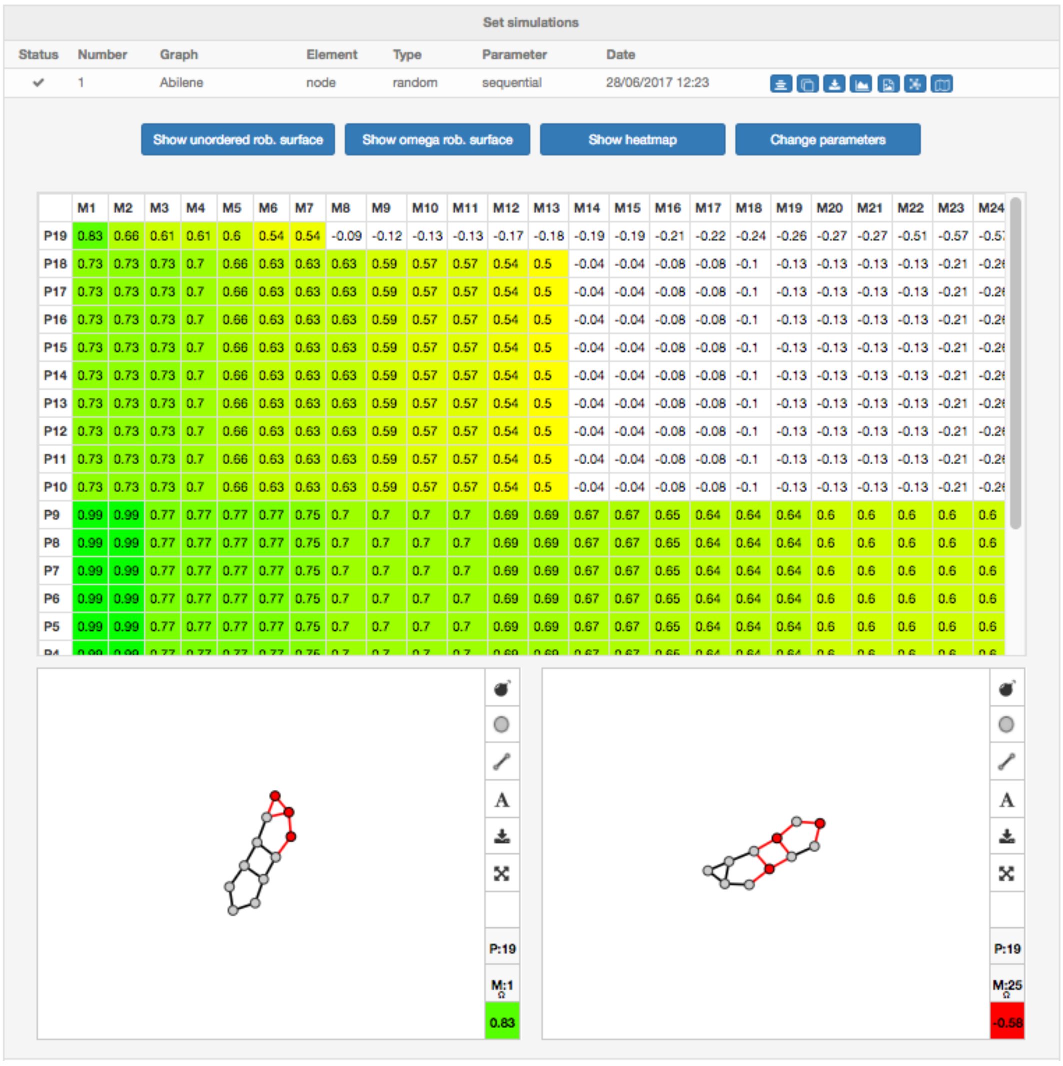
Remarks:	Report example
Graphs:	Abilene Deltacom SingaporeMetro
Attack types:	Random Targeted simultaneous Epidemic
Elements:	Nodes Links
Samples info:	From 1% to 20% of elements (p) 25 samples for each combination (m)
Seed:	Default seed
Metrics:	01-AND 02-HET 03-ASPL 05-EFF 09-r 10-SR 15-DF 16-EC

Set simulations

Status	Number	Graph	Element	Type	Parameter	Date
✓	1	Abilene	node	random	sequential	28/06/2017 12:23
✓	2	Abilene	node	targeted	simultaneous	28/06/2017 12:24
✓	3	Abilene	node	epidemic	NA	28/06/2017 12:24
✓	4	Abilene	edge	random	sequential	28/06/2017 12:25
✓	5	Abilene	edge	targeted	simultaneous	28/06/2017 12:26
✓	6	Deltacom	node	random	sequential	28/06/2017 12:26
✓	7	Deltacom	node	targeted	simultaneous	28/06/2017 12:27
✓	8	Deltacom	node	epidemic	NA	28/06/2017 12:28
✓	9	Deltacom	edge	random	sequential	28/06/2017 12:28
✓	10	Deltacom	edge	targeted	simultaneous	28/06/2017 12:29
✓	11	SingaporeMetro	node	random	sequential	28/06/2017 12:30
✓	12	SingaporeMetro	node	targeted	simultaneous	28/06/2017 12:31
✓	13	SingaporeMetro	node	epidemic	NA	28/06/2017 12:31
✓	14	SingaporeMetro	edge	random	sequential	28/06/2017 12:32
✓	15	SingaporeMetro	edge	targeted	simultaneous	28/06/2017 12:33

FIGURE 11.3: Experiment results page overview. All experiment parameters are shown and all individual simulation are listed.

When execution finished, we can see an experiment overview page with all simulations launched and their results as seen in figure 11.3. All previous selected parameters are shown and also a list with experiment simulations appears with ability to click on each of them to get details.

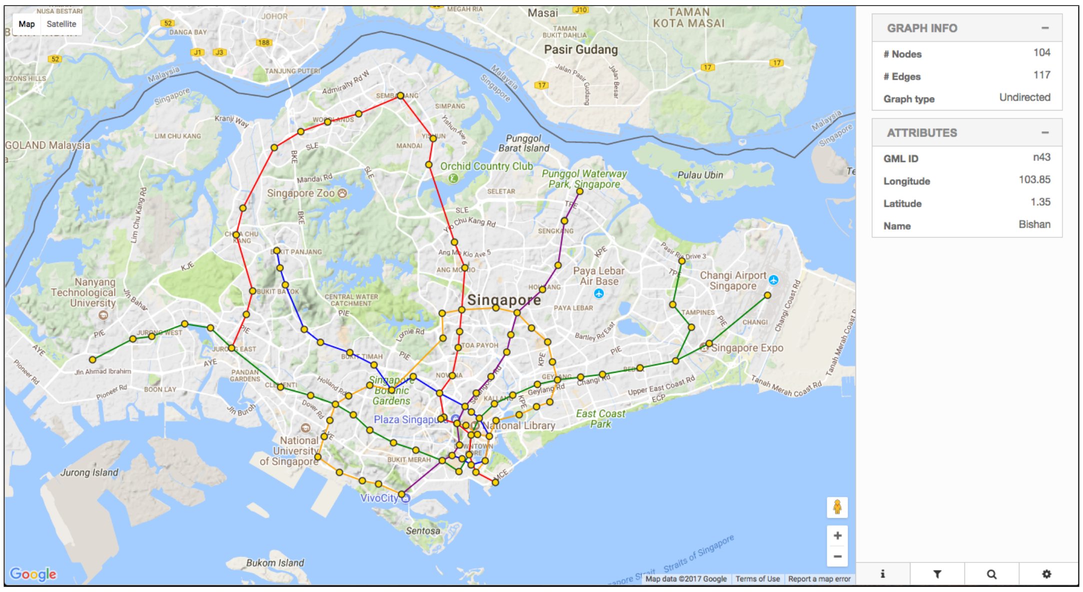


Status	Number	Graph	Element	Type	Parameter	Date
✓	1	Abilene	node	random	sequential	28/06/2017 12:23

	M1	M2	M3	M4	M5	M6	M7	M8	M9	M10	M11	M12	M13	M14	M15	M16	M17	M18	M19	M20	M21	M22	M23	M24
P19	0.83	0.66	0.61	0.61	0.6	0.54	0.54	-0.09	-0.12	-0.13	-0.13	-0.17	-0.18	-0.19	-0.19	-0.21	-0.22	-0.24	-0.26	-0.27	-0.27	-0.51	-0.57	-0.5
P18	0.73	0.73	0.73	0.7	0.66	0.63	0.63	0.63	0.59	0.57	0.57	0.54	0.5	-0.04	-0.04	-0.08	-0.08	-0.1	-0.13	-0.13	-0.13	-0.13	-0.21	-0.2
P17	0.73	0.73	0.73	0.7	0.66	0.63	0.63	0.63	0.59	0.57	0.57	0.54	0.5	-0.04	-0.04	-0.08	-0.08	-0.1	-0.13	-0.13	-0.13	-0.13	-0.21	-0.2
P16	0.73	0.73	0.73	0.7	0.66	0.63	0.63	0.63	0.59	0.57	0.57	0.54	0.5	-0.04	-0.04	-0.08	-0.08	-0.1	-0.13	-0.13	-0.13	-0.13	-0.21	-0.2
P15	0.73	0.73	0.73	0.7	0.66	0.63	0.63	0.63	0.59	0.57	0.57	0.54	0.5	-0.04	-0.04	-0.08	-0.08	-0.1	-0.13	-0.13	-0.13	-0.13	-0.21	-0.2
P14	0.73	0.73	0.73	0.7	0.66	0.63	0.63	0.63	0.59	0.57	0.57	0.54	0.5	-0.04	-0.04	-0.08	-0.08	-0.1	-0.13	-0.13	-0.13	-0.13	-0.21	-0.2
P13	0.73	0.73	0.73	0.7	0.66	0.63	0.63	0.63	0.59	0.57	0.57	0.54	0.5	-0.04	-0.04	-0.08	-0.08	-0.1	-0.13	-0.13	-0.13	-0.13	-0.21	-0.2
P12	0.73	0.73	0.73	0.7	0.66	0.63	0.63	0.63	0.59	0.57	0.57	0.54	0.5	-0.04	-0.04	-0.08	-0.08	-0.1	-0.13	-0.13	-0.13	-0.13	-0.21	-0.2
P11	0.73	0.73	0.73	0.7	0.66	0.63	0.63	0.63	0.59	0.57	0.57	0.54	0.5	-0.04	-0.04	-0.08	-0.08	-0.1	-0.13	-0.13	-0.13	-0.13	-0.21	-0.2
P10	0.73	0.73	0.73	0.7	0.66	0.63	0.63	0.63	0.59	0.57	0.57	0.54	0.5	-0.04	-0.04	-0.08	-0.08	-0.1	-0.13	-0.13	-0.13	-0.13	-0.21	-0.2
P9	0.99	0.99	0.77	0.77	0.77	0.77	0.75	0.7	0.7	0.7	0.7	0.69	0.69	0.67	0.67	0.65	0.64	0.64	0.64	0.6	0.6	0.6	0.6	0.6
P8	0.99	0.99	0.77	0.77	0.77	0.77	0.75	0.7	0.7	0.7	0.7	0.69	0.69	0.67	0.67	0.65	0.64	0.64	0.64	0.6	0.6	0.6	0.6	0.6
P7	0.99	0.99	0.77	0.77	0.77	0.77	0.75	0.7	0.7	0.7	0.7	0.69	0.69	0.67	0.67	0.65	0.64	0.64	0.64	0.6	0.6	0.6	0.6	0.6
P6	0.99	0.99	0.77	0.77	0.77	0.77	0.75	0.7	0.7	0.7	0.7	0.69	0.69	0.67	0.67	0.65	0.64	0.64	0.64	0.6	0.6	0.6	0.6	0.6
P5	0.99	0.99	0.77	0.77	0.77	0.77	0.75	0.7	0.7	0.7	0.7	0.69	0.69	0.67	0.67	0.65	0.64	0.64	0.64	0.6	0.6	0.6	0.6	0.6
P4	0.99	0.99	0.77	0.77	0.77	0.77	0.75	0.7	0.7	0.7	0.7	0.69	0.69	0.67	0.67	0.65	0.64	0.64	0.64	0.6	0.6	0.6	0.6	0.6

FIGURE 11.4: Single execution attacks comparison using visualizer.

Each simulation has its own buttons to see results in different ways as downloading XML files, seeing them in visualizer or seeing execution output. In figure 11.4 a screenshot of a comparison between two different generated attacks can be seen. Left side visualization shows a low damage attack (robustness remains at 0.83) and in right side visualization a high damage attack is shown (robustness value is less than 0), that is with same number of elements attacked (3) generated effect can differ considerably. That happens because in left side attack elements selected are not central but in right side one are.



GRAPH INFO

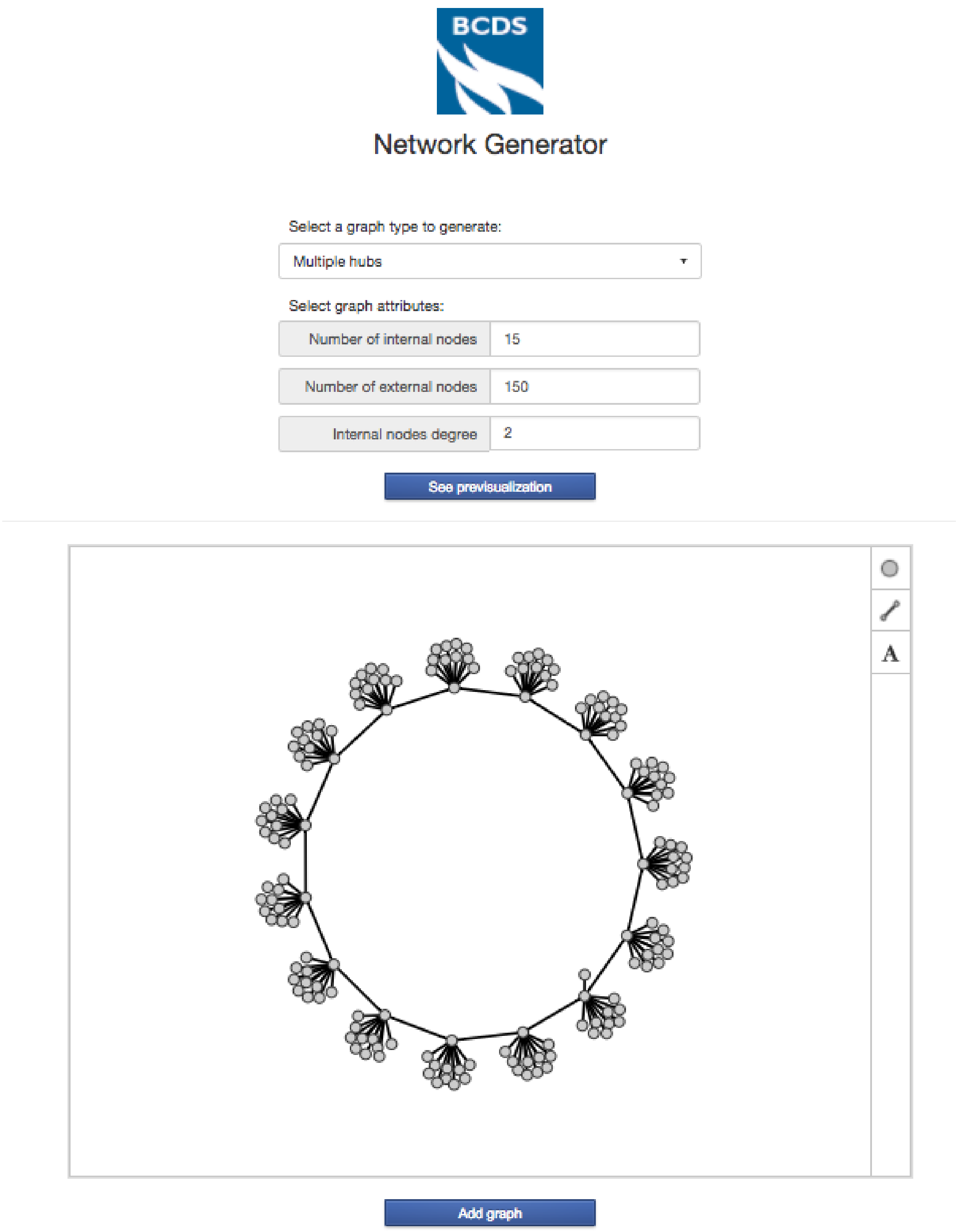
# Nodes	104
# Edges	117
Graph type	Undirected

ATTRIBUTES

GML ID	n43
Longitude	103.85
Latitude	1.35
Name	Bishan

FIGURE 11.5: Visualization of a network over a map. Network is related to Singapore metro network (colourised as real metro lines colours).

Application as detailed in this reports has other functionalities than executing simulations. Users can visualize networks using the visualizer tool in both synthetic mode or over a map. In figure 11.5 real Singapore metro network is displayed over a map using Google Maps. Also users have the ability to generate networks and import them to the simulator. In figure 11.6



Select a graph type to generate:

Multiple hubs

Select graph attributes:

Number of internal nodes	15
Number of external nodes	150
Internal nodes degree	2

FIGURE 11.6: A multiple hub network with 15 internal nodes, 150 external nodes and internal nodes degree equal to 0 generation.

11.2 Collaborations

The main goal of this project was to achieve a useful tool but also it was important to find other research groups who want to use this tool in order to add their studies on it as collaborators. Other groups can use the application as a tool to visualize their complex networks or dynamical computations over them. Existing collaborations are detailed below:

11.2.1 Technical University of Munich (TUM) collaboration

The Department of Electrical and Computer Engineering of TUM has been working in routing algorithms using demand matrices through a network. They are able to find optimal paths for an existing network and demand matrix in order to maximize throughput and reduce link usage. TUM's work is based on SNDLib[2] networks and demand data.

This algorithm has been added to our simulator tool as two new metrics (28-TRH and 29-LU). These two metrics can be categorized as functional metrics and define the amount of traffic lost after a failure in the network, that is removing an element and computing again the algorithm, checking if resulting throughput value remains equal than before (using different paths) or has a lower value (some traffic can not be handled). Detailed algorithm explanation can be seen below:

Optimization algorithm

A network is described as a graph $G = (V, E)$, where V denotes the nodes of the network that can generate or switch the traffic, and E represents the communication links. The links (i,j) have a limited bandwidth C_{ij}. The flows in the traffic demand are described as $d = (s_d, t_d, c_d)$, where s_d and t_d denote the source and destination of the traffic demand, and c_d denotes the data rate of the flow. Binary variables $z_{ij,d}$ indicate if the edge (i, j) carries flow d, and x_d indicate if the flow d is accepted.

The objective is to maximize the throughput of the survived flows, while minimizing the sum of each demand's path length. Small weight is assigned to the number of used links, to ensure shortest possible paths $\epsilon \ll 1$.

$$\max \sum_{d \in D} x_d c_d - \epsilon \sum_{ij \in E} \sum_{d \in D} z_{ij,d} c_d$$

Capacity and flow conservation constraints have to hold.

$$\sum_{d \in D} z_{ij,d} c_d \leq C_{ij}; \forall (i,j) \in E;$$

$$z_{ij,d} \leq x_d; \forall (i,j) \in E; \forall d \in D;$$

$$\sum_{ij \in E; i=n} z_{ij,d} - \sum_{ij \in E; j=n} z_{ij,d} = s_{n,d} x_d - t_{n,d} x_d; \forall n \in V; \forall d \in D;$$

We also prevent the flow split, by allowing at most one incoming and one outgoing link to be used by the same flow:

$$\sum_{ij \in E; j=n} z_{ij,d} \leq 1; \forall n \in V; \forall d \in D;$$

$$\sum_{ij \in E; i=n} z_{ij,d} \leq 1; \forall n \in V; \forall d \in D;$$

Where s_d and t_d are helper functions defined as:

[2]SNDLib is a library of test instances for Survivable fixed telecommunications Network Design. Official website: http://sndlib.zib.de/

$$s_d = \begin{cases} 1, \text{if } n \text{ is a source of flow } d \\ 0, \text{otherwise} \end{cases}$$

$$t_d = \begin{cases} 1, \text{if } n \text{ is a destination of flow } d \\ 0, \text{otherwise} \end{cases}$$

Throughput

When optimization algorithm finishes, it is possible to calculate the network throughput metric value that is defined as the sum of all accepted flows:

$$Throughput = \sum_{d \in D} x_d c_d$$

Throughput metric represents numerically the whole quantity of data carried through the network.

Link usage

Link usage average metric value can be calculated from optimization result. It is defined as the sum of the throughput carried by every edge divided by the sum of edges:

$$LinkUtilization = \sum_{d \in D} \sum_{ij \in E} z_{ij,d} c_d / \sum_{ij \in E} C_{i,j}$$

Average link usage represents the usage percentage of edges, saying if network near to its maximum capacity or if it is not in use.

11.2.2 Kansas State University (KSU) collaboration

Kansas State University research group has been working in epidemic models in order to understand how an infection or failure can spread over a network. KSU has elaborated an algorithm to generate attack sequences using these epidemics models and thanks to our collaboration we have added their code in our simulator to let it have the ability to generate dynamically epidemic attacks.

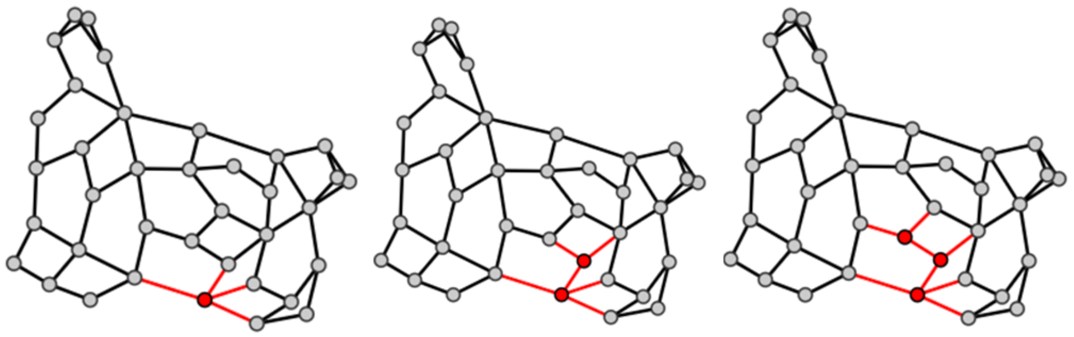


FIGURE 11.7: Epidemic attack spread example (from left to right). Next selected node has to be connected to an infected node.

11.2.3 Institut Català de Recerca de l'Aigua (ICRA) collaboration

ICRA is the most important research institution related to water in Catalonia. They work in topics about water distribution networks and sewerage system. In this case, we have not added any code in our simulator but they use the visualizer in order to visualize their big networks and to compute analysis over them. They shared with us the Girona's sewerage network and after some exhausting fixing, filtering and conversion done by us it is now accessible through the visualizer (figure 11.8).

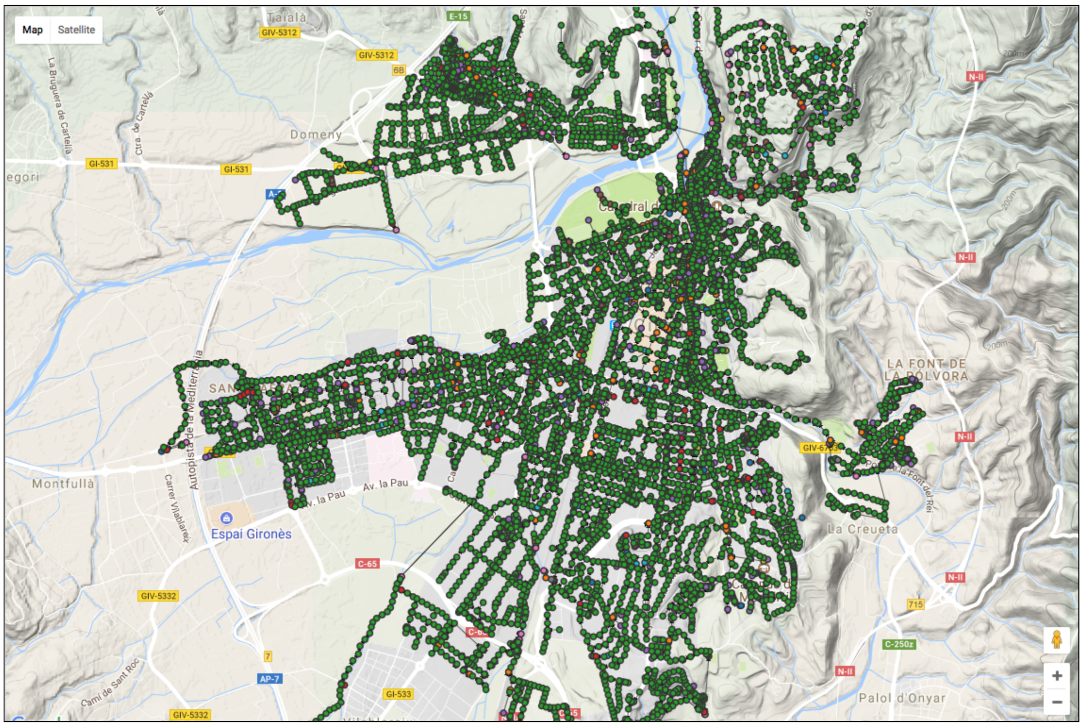


FIGURE 11.8: Girona's sewerage system network visualized over a map using the visualizer tool. Wells are painted depending on type.

Aside from visualizing network ICRA asked us for computing some analysis on this network as for example grouping pipelines depending on some criteria. We helped them adding this functionality to the simulator. We can see an example in figure 11.9.

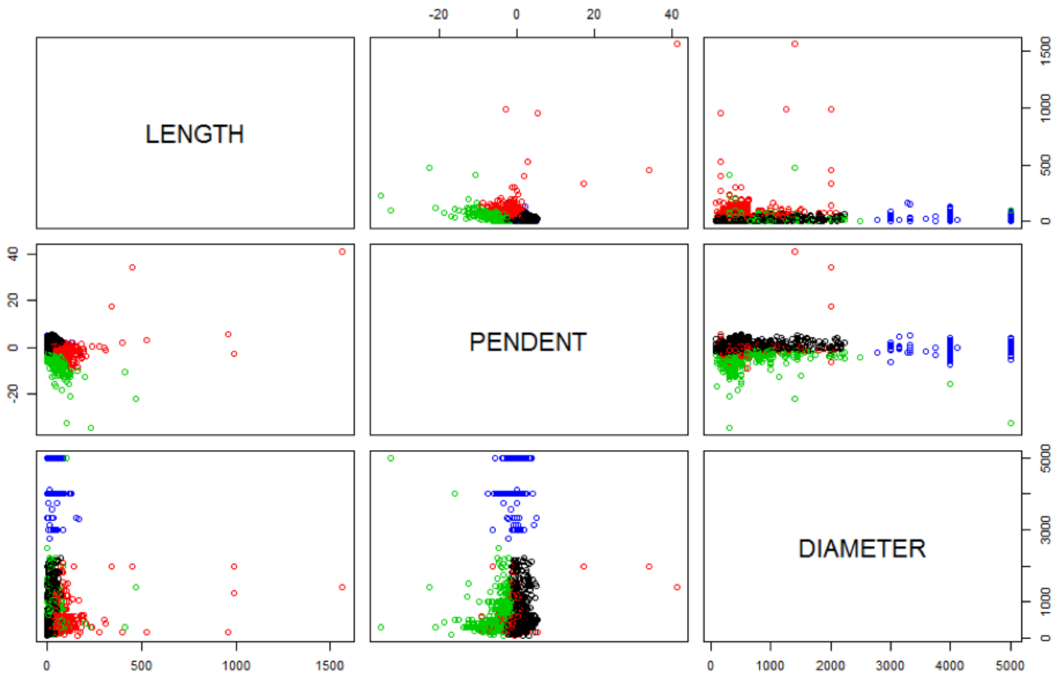
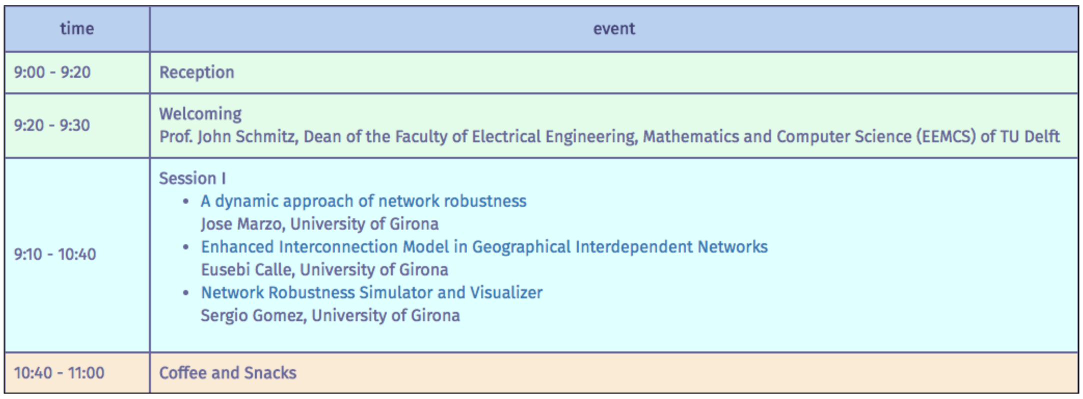


FIGURE 11.9: Girona's sewerage system network pipelines clustering depending on their length, pending and diameter attributes values.

11.3 The 3rd Delft-Girona Workshop of robustness of complex networks

In the background of the 3rd Delft-Girona Workshop[3] related to complex networks robustness that took place in Delft University of Technology (The Netherlands) on 11th July, 2017 a depth demonstration of the whole application with a great acceptance from present public in workshop. A whole session was needed to explain all simulator functionalities (theoretical and practical demonstration) as can be seen in workshop programme (figure 11.10).



time	event
9:00 - 9:20	Reception
9:20 - 9:30	Welcoming Prof. John Schmitz, Dean of the Faculty of Electrical Engineering, Mathematics and Computer Science (EEMCS) of TU Delft
9:10 - 10:40	Session I • A dynamic approach of network robustness Jose Marzo, University of Girona • Enhanced Interconnection Model in Geographical Interdependent Networks Eusebi Calle, University of Girona • Network Robustness Simulator and Visualizer Sergio Gomez, University of Girona
10:40 - 11:00	Coffee and Snacks

FIGURE 11.10: Delft-Girona workshop programme showing Universitat de Girona's dedicated session to simulator.

[3]More info at: https://www.nas.ewi.tudelft.nl/rocn/index.html

Chapter 12

Conclusions

Having in mind the objectives of this project, these are the general conclusions:

- It has been **successfully developed the simulator** in a web application including its three tools as simulator, visualizer and generator. The application is able to compute experiments in a user friendly way and it is accessible worldwide.
- The previous existing R code has been changed to new web based system with **many functionalities added** that make the whole system more powerful. All functionalities are now accessible easily through a navigator which is helpful to researchers with no computing knowledge.
- Other **institutions and/or collaborators are interested in using this application** in order to use it for their experiments. They see the application as a very helpful one thanks to many features available on it that they don't have in their systems.
- Aside from using synthetic networks, during the project textbfsome real networks has been used in order to analyse them using the application, which indicates the application **can be used in many kind of real networks**.

Personally, I believe this project has resulted in a very useful application which help BCDS research group members and others to execute and understand their experiments easily. The most important personal conclusions are:

- This project was the start on introducing myself in research topics. I started this project with the aim to learn how research is and to improve on it. I am really satisfied in this point, being examples the collaboration with Technical University of Munich with a two weeks stay in it or the assistance as speaker in the 3rd Delft-Girona Workshop of robustness of complex networks. This gave me much experience in order to improve with my personal trajectory.
- An important aspect in a final project is to use whatever you learned during the master. In this project I have used a lot of knowledge learned during the master. It is important to understand that this project is a join of two different topics in computer engineering which are web developing and research investigation.

Finally, I am proud of making most of the whole application with no external help than my previous knowledge in web developing and forums available in internet. This includes setting up the server, developing the application and testing it.

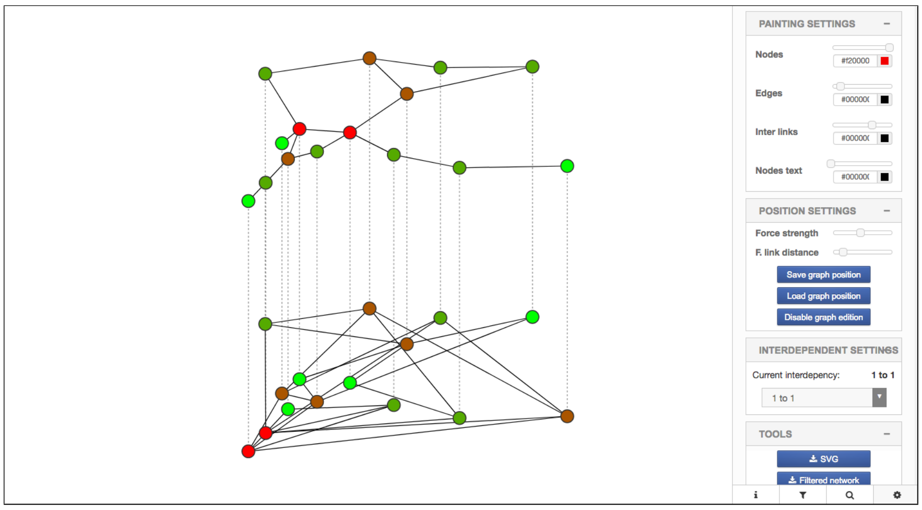
Chapter 13

Future work

As the resulting application is powerful, possible future work can be defined as ***any functionality available in a network analysis tool can be added to application***. That means once a network is imported to the system, any analysis can be done on it and see its results interactively. If desired functionality is not yet available, it can be added with no much time cost.

Our next steps scheduled to improve the system are:

- **New user interface:** As some collaborators gave us their feedback, we decided to implement a easier to use user interface. It includes a depth redesigning of the interface making it resizeable to display in mobile devices and adding tips to let collaborators understand better how application works without our help.
- **Interdependent networks:** BCDS research group is also working in interdependent networks (i.e. two networks interconnected in some way). Next step is to let the application compute their experiments and visualize results on it as in *single network mode*. It is still in development, we can see a *beta version* capture in figure 13.1.



Current interdepency: 1 to 1

1 to 1

FIGURE 13.1: Interdependent networks mode beta version capture. Capture has been taken while development phase.

- **New collaborations:** We are working on achieving new collaborators to add more metrics and algorithms to our system. With more collaborations we have, more powerful the application is.

Bibliography

[AJB00] Réka Albert, Hawoong Jeong, and Albert-László Barabási. Error and attack tolerance of complex networks. *nature*, 406(6794):378–382, 2000.

[Cas11] Marc Manzano Castro. *Metrics to evaluate network robustness in telecommunication networks*. PhD thesis, Universitat de Girona, 2011.

[CLMR03] Paolo Crucitti, Vito Latora, Massimo Marchiori, and Andrea Rapisarda. Efficiency of scale-free networks: error and attack tolerance. *Physica A: Statistical Mechanics and its Applications*, 320:622–642, 2003.

[DC04] Anthony H Dekker and Bernard D Colbert. Network robustness and graph topology. In *Proceedings of the 27th Australasian conference on Computer science-Volume 26*, pages 359–368. Australian Computer Society, Inc., 2004.

[ESVM$^+$11] W Ellens, FM Spieksma, P Van Mieghem, A Jamakovic, and RE Kooij. Effective graph resistance. *Linear algebra and its applications*, 435(10):2491–2506, 2011.

[MCSS$^+$14] Marc Manzano Castro, Faryad Sahneh, Caterina Scoglio, Eusebi Calle Ortega, and Josep Lluís Marzo i Lázaro. Robustness surfaces of complex networks. *Scientific Reports, 2014, núm. 4, P. 6133*, 2014.

[MCTP$^+$13] M. Manzano, E. Calle, V. Torres-Padrosa, J. Segovia, and D. Harle. Endurance: A new robustness measure for complex networks under multiple failure scenarios. *Computer Networks*, 57(17):3641 – 3653, 2013.

[MKF$^+$06] Priya Mahadevan, Dmitri Krioukov, Marina Fomenkov, Xenofontas Dimitropoulos, Amin Vahdat, et al. The internet as-level topology: three data sources and one definitive metric. *ACM SIGCOMM Computer Communication Review*, 36(1):17–26, 2006.

[SHe$^+$10] James P.G. Sterbenz, David Hutchison, Egemen K. etinkaya, Abdul Jabbar, Justin P. Rohrer, Marcus Scher, and Paul Smith. Resilience and survivability in communication networks: Strategies, principles, and survey of disciplines. *Computer Networks*, 54(8):1245 – 1265, 2010. Resilient and Survivable networks.

[SSYS10] Ali Sydney, Caterina Scoglio, Mina Youssef, and Phillip Schumm. Characterising the robustness of complex networks. *International Journal of Internet Technology and Secured Transactions*, 2(3-4):291–320, 2010.

[TMHWVM13] Stojan Trajanovski, Javier Martín-Hernández, Wynand Winterbach, and Piet Van Mieghem. Robustness envelopes of networks. *Journal of Complex Networks*, 1(1):44–62, 2013.

[WBTD09] Jun Wu, Mauricio Barahona, Yuejin Tan, and Hongzhong Deng. Robustness of regular graphs based on natural connectivity. *arXiv preprint arXiv:0912.2144*, 2009.