Alma Mater Studiorum · Università di Bologna

CAMPUS DI CESENA
SCUOLA DI INGEGNERIA E ARCHITETTURA
Corso di Laurea Magistrale in Ingegneria e Scienze Informatiche

DESIGN, IMPLEMENTATION AND PERFORMANCE EVALUATION OF AN ANONYMOUS DISTRIBUTED SIMULATOR

Tesi di Laurea in Sicurezza delle Reti

Relatore:
GABRIELE D'ANGELO

Presentata da:
ANTONIO MAGNANI

Sessione III
Anno Accademico 2014-2015

Sommario (Italian)

La simulazione è definita come la rappresentazione del comportamento di un sistema o di un processo per mezzo del funzionamento di un altro o, alternativamente, dall'etimologia del verbo "simulare", come la riproduzione di qualcosa di fittizio, irreale, come se in realtà, lo fosse. Le origini della simulazione sono piuttosto recenti e, lo sviluppo di essa, ha seguito parallelamente il percorso di diffusione dell'informatica. La capacità di riprodurre modelli in grado di rappresentare in maniera estremamente minuziosa la realtà o il comportamento di un sistema immaginario, sono oggigiorno fondamentali in molteplici ambiti (e.g., dalle comuni previsioni meteo financhè a simulare l'attività cerebrale di un essere umano). Ma perché è così importante la simulazione? Innanzitutto ci permette di modellare la realtà ed esplorare soluzioni differenti e valutare sistemi che non possono essere realizzati per varie ragioni. Inoltre, come verrà esposto in questo lavoro, è fondamentale l'approccio simulativo per effettuare differenti valutazioni, dinamiche per quanto concerne la variabilità delle condizioni, con un risparmio temporale ed economico considerevole rispetto alla realizzazione di test reali.

I modelli di simulazione possono raggiungere un grado di espressività e realismo estremamente elevato, pertanto difficilmente un solo calcolatore potrà soddisfare in tempi accettabili i risultati attesi. Una possibile soluzione, estremamente attuale viste le tendenze tecnologiche dei nostri giorni, è incrementare la capacità computazionale tramite un'architettura distribuita (sfruttando, ad esempio, le possibilità offerte dal *cloud computing*). Questa tesi si concentrerà su questo ambito, correlandolo ad un altro argomento che

sta guadagnando, giorno dopo giorno, sempre più rilevanza: l'anonimato online. I recenti fatti di cronaca [1, 2, 3] hanno dimostrato quanto una rete pubblica, intrinsecamente insicura come l'attuale Internet, non sia adatta a mantenere il rispetto di *confidenzialità*, *integrità* ed, in alcuni, *disponibilità* degli *asset* da noi utilizzati: nell'ambito della distribuzione di risorse computazionali interagenti tra loro, non possiamo ignorare i concreti e molteplici rischi; in alcuni sensibili contesti di simulazione (e.g., simulazione militare, ricerca scientifica, etc.) non possiamo permetterci la diffusione non controllata dei nostri dati o, ancor peggio, la possibilità di subire un attacco alla disponibilità delle risorse coinvolte. Essere anonimi implica un aspetto estremamente rilevante: essere meno attaccabili, in quanto non identificabili.

Scopo di questo studio sarà dunque la progettazione, l'implementazione e la successiva valutazione di meccanismi di anonimato che rendano possibile l'esecuzione di simulazioni distribuite in cui gli host partecipanti non possano essere identificabili nemmeno dagli stessi partecipanti e, conseguentemente, difficilmente attaccabili da un agente esterno.

A tal fine verranno approfonditamente studiati e adottati gli strumenti ed i protocolli forniti dall'unico sistema di comunicazione anonima che ha, nel tempo, dimostrato la propria efficacia e, soprattutto, un sufficiente grado di diffusione (requisito fondamentalmente per l'ottenimento dell'anonimato): ***Tor*** [4].

Chiaramente la possibilità di essere anonimi ha un elevato costo dal punto di vista dei tempi di computazione, pertanto lo studio delle performance del simulatore anonimo realizzato sarà di fondamentale importanza per quantificare l'efficienza della soluzione anonima prodotta e dei relativi protocolli adottati.

Introduction

The Merriam-Webster dictionary defines simulation as *"the imitative representation of the functioning of one system or process by means of the functioning of another"* or the *"examination of a often not subject to direct experimentation by means of a simulating device"* [5]. These are just two possible interpretations of the notion of simulation, but both focus on the idea of the presence of an external entity (*a system or a simulating device*) capable of reproducing the behavior or the operation of a system. In the context of this thesis the "external entity" is a computer or, more precisely, a distributed set of computers.

The simulation has its roots back in 1777 with the Buffon's "Needle Experiment" [6], but its birth and subsequent diffusion will coincide with the formalization of the "Monte Carlo Method" proposed by Fermi, Ulam and Von Heumann, which will take place only in the mid-40s as part of the Manhattan Project [7]. With the advent of computer science, simulation would gradually gain more and more importance, first in the military field and then crosswise in any field of science, engineering, medicine, economic, entertainment (e.g., just think of simulation games) and so on. But why do we need simulation and why it is so important? Some of the main pros of using simulation than real test-bed are:

- Make an accurate depiction of reality and study new and different solutions;
- Evaluate systems that simply cannot be built for many reasons (e.g., poor feasibility like testing situation of disaster recovery);

- Performance evaluation under different conditions in a practical way (and in some cases faster) than real test-bed;
- Sometimes testing on an existing system can be very dangerous, or more simply, some stress testing are impossible to perform;
- Often many different solutions have to be evaluated in order to choose the best one;
- Money, most of the time perform a set of simulations is more economic and faster than the realization of real tests.

There is another advantage of using simulation, but in some cases this can be also a possible drawback: the level of detail that you can reach from it. In fact you can model simulations at a very high level of detail but it clearly has a price in terms of calculation and amount of time. A solution for these kind of large and complex models is to distribute the computing capability over a network instead of a single execution unit.
In this thesis, I will study, in particular, the concept of distributed simulation correlated with another relevant argument that is gaining importance day by day: anonymity. Suppose that we are using a network inherently insecure as Internet for our distributed simulations: how can we ensure that our data has not been altered or, at least, there has not been any kind of confidentiality violation? Also, in some domain with an exchange of sensible data or information, do we really want to share our identity (our IP adddress) over the Internet? In this work, I will try to give an answer to these questions using *Tor* [4], a free software and open network for enabling anonymous communication, and *ARTÍS: Advanced RTI System* [8, 9] a middleware for *Parallel and Distributed Simulation (PADS)* supporting massively populated models.

Furthermore, we must underline the importance to be anonymous for all of the distributed software. In the field of simulation, as we shall in the first chapter, the actual tendency is to exploit the computational capabilities offered by the new paradigms, like cloud computing. The possibility to take advantage of services which allow the dynamic and rapid increase or decrease

of the calculating capacity is fundamental for the realization of simulation models with a certain degree of complexity. However, the use of cloud platforms implies the entrustment of our simulation data to entities which are very distant from us, to the network which interconnects us with them (with relative risks) and, most of all, to the technologies and protocols actually used by the Internet which makes all of the participants identifiable. In some cases, we have no idea where or how the computation resources are located by the cloud providers: we only know that they will exchange information, and the basic assumption from which it is necessary to start for our evaluations is that these interactions will happen with a public network. The possibility to exchange data securely, respect to ***confidentiality*** and ***integrity***, it is also a fundamental requirement for the simulations, in a special way in the case of particularly sensitive contexts (e.g., military, medical and science research fields): it is always more evident how nowadays cyberwarfare is a concrete activity [2, 3], common and dangerous, from which it is better to protect ourselves in a preventive manner. Finally, being anonymous, implies a very important aspect: we are unattackable. A malicious subject, which we shall see in the second chapter, can not attack the ***availability*** of our assets because he does not know where they are located or how to reach them. Obviously, the instruments which we will study are not the *"silver-bullet"* capable of resolving every aspect relative to the computer security: the methods with which these are used is often more important than their effectiveness. Being fully anonymous is a constant activity which does not limit itself to the mere installation of a software; it is also important to underline the price to pay in terms of performance for being anonymous: in this thesis, after the implementation of the anonymity's mechanisms we will investigate and quantify this aspect.

In the first chapter, I will start with an introduction to computer simulation theory with a study of the state of the art of Parallel and Distributed Simulation, I will also introduce ARTÍS, its architecture and its features. After that, in the second chapter, I will explore the working principle of Tor

and I will describe the concept of anonymization. I will then implement the anonymity concepts provided by Tor in ARTÍS, and test the performance of distributed anonymized simulation to observe how much Tor may affect the computational time of a simulation run.

Contents

List of Figures

List of Tables

Chapter 1

Simulation Background

In this chapter, I will introduce the fundamental aspects concerning the notions of computer simulation. It is a very wide matter so I will focus on simulation paradigms, in particular Discrete Event Simulation (DES) and I will introduce this argument with a general definition, after that I will explore the concepts of sequential simulation and, above all, Parallel and Distributed Simulation (PADS), its advantages and some key aspects.

1.1 Computer simulation

A more useful definition for this work than the ones mentioned in the introduction of this document is the concept of computer simulation intended as *"a software program that models the evolution of some real or abstract system over time"* [10]. We have already seen some advantages of using simulation in the introduction of this thesis, but a question still remains: if we want to evaluate large and complex simulation models (*e.g., forecasting weather* or *simulate brain activity* [11]) how can this be achieved in a reasonable period of time? Or, at least, how can we reduce the necessary computational time (known as *Wall-Clock-Time, WCT*)? Or even, how long are we willing to wait for an outcome? Before attempting to answer these questions I must necessarily introduce some concepts.

1.2 Simulation paradigms

Computer simulation and simulation tools following various paradigms, as stated earlier in this work, I will focus on *Discrete Event Simulation (DES)* but I would also introduce the most important approaches presented in scientific literature over the years. Each one of these paradigms has its benefits and drawbacks and it must be emphasized that there are many ways of doing simulations: it is usually a case-by-case evaluation.

1.2.1 Monte Carlo simulation

The name of this simulation paradigm derives from the casino in the Principality of Monaco: this choice is not properly casual in fact the objective of this simulation is to model risk in an environment where the output is subject to probability. The method is used to derive estimates through simulations: it is based on an algorithm that generates a series of numbers uncorrelated with each other, following the probability distribution that is supposed to have the phenomenon to be investigated. It is considered the father of the simulation, with the needle experiment already mentioned in the introduction of this thesis [7]. The Monte Carlo method is used, for example, in financial services. With this method it is possible to model the future of investment portfolios: the stocks are the inputs of the simulation, each one has its own distribution of possible output in terms of share price in prospective. The result of the simulation is the equivalent amount of money (derived from the stock's values) of the portfolio at a certain time in the future [12].

1.2.2 System Dynamics (SD)

System dynamics is a continuous simulation paradigm that represents the environment as a set of stocks and flows. Stocks are an aggregation quantity (*e.g., money or items*) and flows regulate level of a stock with inflows increasing the stock and outflows decreasing it. Stocks change in a continuous

manner in response to the correspondence of the inflows and outflows from the stock [13, 12]. It is especially suited to investigating strategic issues: [14] gives examples that include modeling supply chains, forecasting energy consumptions and analysing business cycles.

1.2.3 Agent-Based Simulation (ABS)

The basic idea is to build systems with a bottom-up approach: the model is constituted by a set of active entities (*agents*), with an individual and independent behavior, that communicate particular events or reactions to received messages over time. The main aim of modeling systems in this way is to evaluate the evolution of the behaviors, the communication patterns and structures that emerge [15]. A possible example of agent based simulation was given by [16] where it is modeled the diffusion of an epidemic: that model is able to represent the explosion of an epidemic in a population with different billion of independent agents.

1.2.4 Discrete Event Simulation (DES)

A Discrete Event Simulation models the activities of a systems as a discrete progression of events in time, in other words the model evolution happens at discrete moments in time by means of *simulation events*. The simulation evolves through the creation, distribution and evaluation of events [10, 17] and marks a change of *state* in the system [12]. DES is a hugely flexible approach: almost any aspect can be represented by who develops the model, allowing to reach an incredible level of detail [18].

The key concept is the *"event"* which represents a change in the system *state* that has occurred at a particular moment in time (each event contains a timestamp, indicates when this change occurs in the actual system). Hence the computation of an event can alter some parts of the state and eventually lead to the creation of new events [9]. In Figure 1.1, it is possible to observe

a simple DES example.

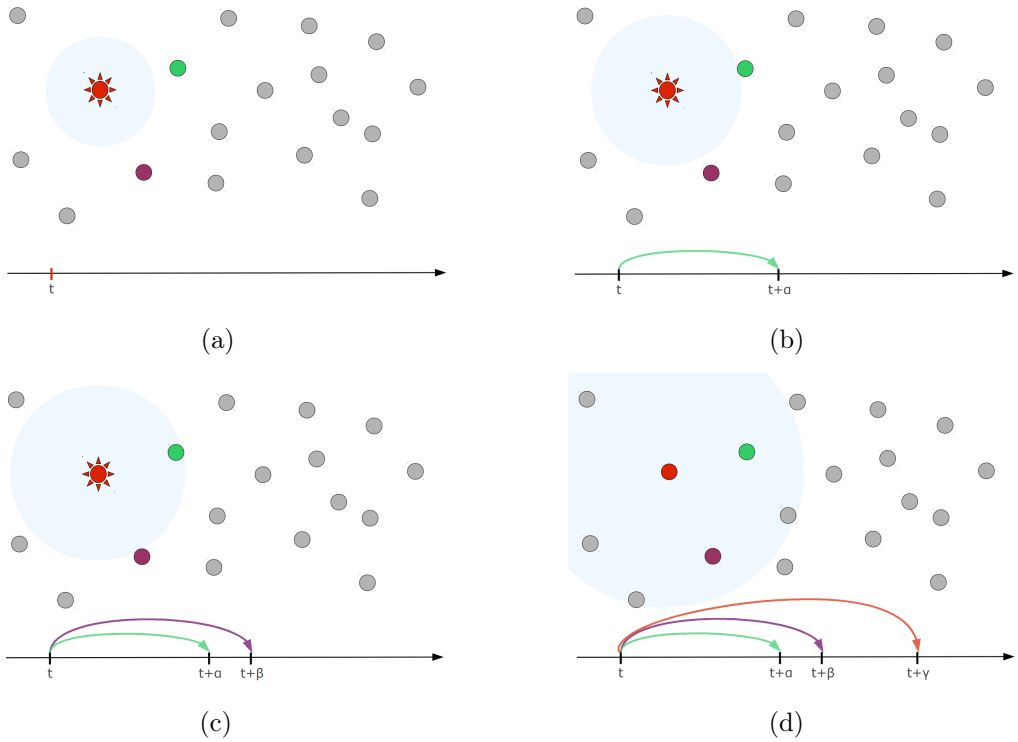

Figure 1.1: A simple example of DES representing a set of mobile wireless hosts. In **(a)** at time t the red node opens its transmission. In **(b)** at time $t+\alpha$ the green node starts receiving. In **(c)** at time $t+\beta$ the dark violet node starts receiving. In **(d)** at time $t+\gamma$ the red node concludes transmitting [19].

A simple implementation of a DES includes these *data structures* [20]:

- a **set of variables**: also called *state variables* that describe the state of the system (*e.g., in a classical example, to simulate an airport these variables can indicate the counts of the number of aircraft that are flying and the ones who are on the ground. We can also model the boolean state of the runways: busy - due to a take-off or a landing - or free*);
- an **event list**: containing events that are to occur some time in the simulated future (*e.g., in our airport, the list of arrivals and departure*

is a clearly example of an event list);

- a **global clock**: a global clock variable with the purpose of indicate the instant at which the simulation now resides (*e.g., the airport clock*).

We can consider a ***simulator*** like a collection of *handlers*, each one controls and responds to a different event type [19, 20] (e.g., an *Arrival Event* in our airport has its handler: in this case the handler increments the count of aircraft that are flying overhead and checks if the runway is free or not: if the runway is free the handler will *schedule* in the *event list* another event - *Landed Event* - at *"global clock time plus the necessary landing time"*). It is important to emphasize that events are produced on the basis of temporal chains of conditions and circumstances that generate them but they have to be executed in non-decreasing time order: this implies that the above mentioned *event list* is a **priority queue** usually implemented with heap-based solution.

Furthermore a simulator must respect two relevant constraints:

1. ***Reproducibility:*** each simulation must always be *reproducible* so that it is possible to repeat and analyze the sequence of events and the outcomes of the simulations [21];

2. ***Causal ordering*** of the events must be respected. Two events are defined in *causal order* if one of them depends on the other [22]. The execution of events in *non causal order* leads to ***causality errors***.

If all events and tasks are handled and accomplished by a single execution unit (*e.g., a single CPU core and some RAM)*, the simulator is then called a ***sequential simulator***, also know as *monolithic*. The simulation performed by these simulators is called *sequential*: in this kind of simulations we are sure that the events and updates of the state variables are processed by the execution unit in a "non-decreasing timestamp order". Therefore the *"event loop"* (also called *main loop*) of the simulator continuously removes the smallest event (in consideration of its time stamp) from the event list, and processes

that event. Thanks to the presence of a single execution unit this kind of simulator respects the two above mentioned constraints: in fact the reproducibility of a simulation is guaranteed by the presence of a single process, therefore if the inputs are the same the sequence of events will always be the same. Also the causal ordering is respected because all the events that happen in the simulation can be temporally placed without ambiguity, thanks to the presence of a unique global clock shared by all the entities of the system.

As you can imagine, over the *simulation clock*, even the state is unique and global. We can conclude that the main advantage of this approach is its simplicity but we have to consider some significant limitations: first of all, how fast is a single CPU? Are we sure that the simulation time is not a priority for us? In some cases outcomes have to be in real-time or even faster but large and complex models might generate a large number of events putting relevant workload on the CPU. Moreover, the RAM available on a single host may be not sufficient to store all state informations produced at some point of the simulation [23]: it is technical not possible to model some system and it is evident that this kind of simulation can not scale. In addition to these considerations, during the writing of this thesis, the *Semiconductor Industry Association (SIA)* has announced that the pursuit of *Moore's Law* [1] will definitely abandoned soon [24]: this conclusion is due to the extreme processors miniaturization that has now reached its limit. Even this aspect clearly has an obvious impact having regard to above-mentioned considerations: computing needs to be rethought, shifting the simulation paradigm to other solutions.

1.2.4.1 Parallel Discrete Event Simulation (PDES)

A first answer to try to improve Discrete Event Simulation performance is move to a parallel paradigm. *Parallel Discrete Event Simulation (PDES)*

[1] The Moore's Law says that the complexity of a micro-processor, measured in consideration of the number of transistors for chip, doubles every 18 months.

refers to the computation of a single DES program on a parallel computer, with the purpose of generate outcomes faster than the monolithic solution. With PDES it is possible to represent large and complex models using the hardware resources composed from several interconnected execution units (e.g., a single core), each unit has to handle a part of the simulation model and its local event list. The events produced by a single core may have to be transmitted to the other units and, clearly, all of this needs to be correctly partitioned and synchronized [25].

1.3 PADS: Parallel And Distributed Simulation

Parallel simulation and distributed simulation, refers to the methodologies that enable to perform a simulation on systems that contain multiple processor, such as a notebook or even a cluster of computers, interconnected by a wired or wireless network. More in general, we can consider a ***Parallel And Distributed Simulation (PADS)*** like any simulation in which several processors are utilized [26]. Hence, the common idea is to use multiple processors but it is appropriate to differentiate between ***parallel*** and ***distributed*** simulation: in this work I will intend the term *parallel simulation* as the use of architecture that adopts shared-memory processors or tightly coupled parallel machines. The term *Distributed simulation*, on the other hand, specifies simulation systems that use loosely coupled machines [26].

These two types of simulation can answer both questions introduced in this document at 1.1, in fact there are many benefits to executing a simulation program across multiple processors:

- *Reduced execution time*: the subdivision of a single complex execution into different sub-computations, and the concurrently execution of these smaller parts across N processors, make possible to reduce the total simulation time up to a factor of N.

- *Geographical distribution*: performing the simulation on a set of distributed machines permits the creation of virtual environments with many participants that are physically placed at distinct location. The most common example is *online gaming*: the players, from different countries around the world, are participating at a simulation and are interacting with the others one as if they were situated in the same place. In the case of multiplayer gaming, distributed simulation is not only convenient but required.

- *Fault tolerance*: Another important benefit of using multiple processors is the chance of gaining more tolerance to failures [26]. It is possible, at a certain time of the simulation, that one processor falls or communication to it collapse: the other machines that are participating at the computation can acquire its state (or even the event list). As a result of this recovery, the simulation can proceed regardless the failure. In a monolithic simulation a collapse of the unique execution unit implies the anticipated end of the simulation (we can only introduce temporal saving mechanism of the state, but this obviously generate a considerable overhead).

- *Scalability*: We are able to increase (or decrease) the number of processing units, this makes the simulator scalable in relation to the size of the simulated system, making possible simulations of large and complex systems composed by a very high number of entities. This would not be achievable with a monolithic simulator because of time limits [27].

- *Composability of models*: the capacity to choose and combine different components in various configurations into a valid simulator to accomplish specific user requirements [28]. Hence, for example, we can compose distinct models in a single simulator.

- *Interoperability*: it is a feature of a system to work with other components, or also entire systems, with minimal changes or even without

altering the implementation. We need to know only the appropriate interfaces and this property allows us, for example, to integrate between various simulators components.

In PADS the model is split into numerous execution units, each of them manages a subset of the state space and a portion of the events. In this context is used a simulator composed by many processes executed in concurrency on the same host or on several computers: each process, called **LP (Logical Process)**, has to handle the progression of a part of simulation and must communicates with the other processes in order to perform a global evolution; this interaction among processes implies several important considerations about the consistency of the simulation due to the needed synchronization and data distribution operation [26]. Each LP is executed by a different ***Physical Execution Unit (PEU)***[2], the PEUs can be placed on the same computer (case of multi-core or multi-CPU architecture) and in this case they typically use shared memory to communicate, or they can be located in several hosts and communicate through a network. It is more common to see heterogeneous systems that mix these communication types: the architecture is usually an "hybrid", for example a mix of local resources, that are using a LAN or even shared memory, and remote resources that are using network protocols like TCP/IP or Tor. In Figure 1.2, it is possible to observe a possible example of a real world architecture for PADS. In conclusion, the model is physical divided between all the PEUs, each PEU (through the run of the relative LP) has to manage the execution of the assigned part of the model. It is also evident that the method used for communication between LPs can strongly influence the simulator's performance.

Hence, we cannot rely on a shared global state and the separation of the simulation bring out anything but simple issues like the partitioning of the model, the synchronization of processes and the data distribution among them.

[2]A PEU is essentially a core in a multi-core architecture or a CPU in a single-core architecture.

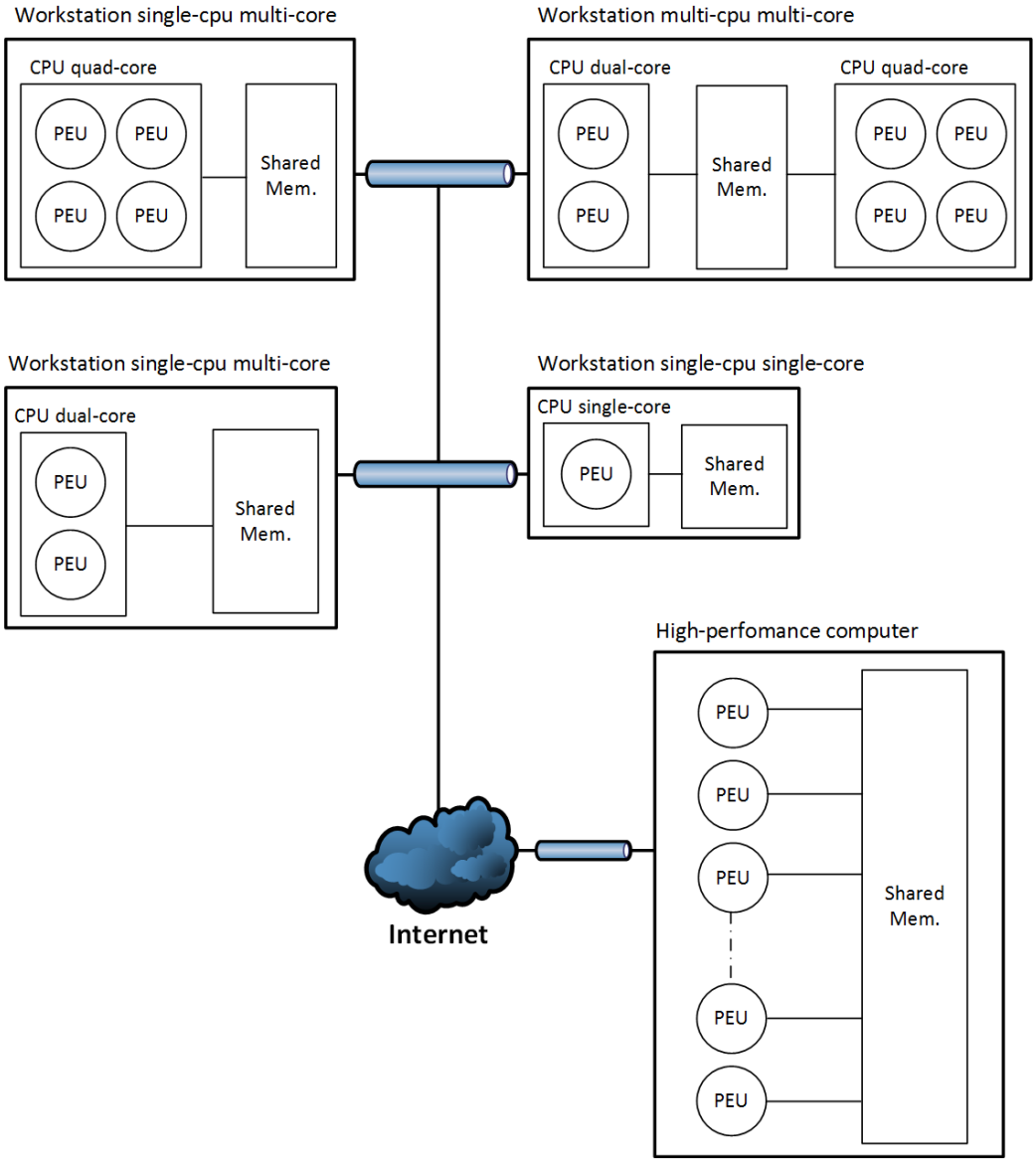

Figure 1.2: An example of possible PADS architecture with distributed and heterogeneous computers. Each PEU (substantially a core) will execute a LP. It should be noted that also the communication channels are heterogeneous (e.g., a LAN, shared memory, Internet).

1.3.1 Partitioning

As previously stated, the simulation model must be partitioned among the LPs. This can be very difficult work because often there is not an easy

way to identify a possible partition of the model. Sometimes the domain of the simulation allows a functional or semantic decomposition of the model but usually this is not the general case. Another point to consider is that the workload should be balanced across LPs (***load balancing***) and the communication between them should be minimized [29]. There are some relevant factors to consider to define the partitioning criteria, one of them is certainly the model of the simulation (e.g., maybe it is possible to distinguish classes of objects that interact among objects of the same group; in this case we have a natural model partitioning that respects as well the load balancing of communication), another one is the hardware architecture on which the simulation will be executed and, in the end, even the characteristics of the synchronization algorithm that is implemented can guide the partitioning of the model.

1.3.2 Data distribution

The model is partitioned so the principal consequence is that each component of the simulator will produce status updates that might be relevant to other entities: with the term ***data distribution*** we intend the dissemination of these updates all over the execution architectures. For overhead reasons, broadcast is not a good idea (the LPs are not interested in receiving updates of all the other LPs) hence only the necessary data has to be forwarded to the interested processes, this implies the application of criteria in order to balance data production and data consumption adopting "publish/subscribe" pattern [30]. Data distribution is closely related to synchronization.

1.3.3 Synchronization

In section 1.3 of this work, we have seen that each LP is executed by a different PEU, realistically at a different speed. Furthermore, we have also considered the presence of different kinds of networks which can be more or less congested: about that we assume that the network can introduce delays

but communication is reliable (e.g., TCP-Based).

Moreover, we have to introduce a definition of correctness to distinguish the validity of the result of a PADS: we consider the result correct only if the outcome is the same of the one deriving from a monolithic/sequential execution. This means that *causal ordering* (see section 1.2.4) must be respected also in PADS: this is much difficult in "PADS" than in sequential simulation, because the LPs have only a partial view of the global state and of the events that were happened in the model or that are pending on the event list of the others. To achieve result's correctness, we need some kind of synchronization among the several component of the simulation, this aim usually has a very relevant cost [19]. To solve this problem different solutions have been proposed, divided in three main families:

- *Time-stepped*;
- *Conservative*;
- *Optimistic*.

1.3.3.1 Time-Stepped

In this synchronization approach, the simulated time is partitioned in periods of a fixed-size called *time-steps*. Each LP can pass to the next time-step only when the current one is completed by all the other LPs. Hence, it is evident that the time is discretized (even if is clearly a continuous measure) and it comports a constant synchronization among all LPs: this is typically implemented with a barrier at the end of each simulation step.

The key aspect of this synchronization paradigm is the choice of an appropriate time-step size: small steps imply more synchronization points and this can have a strongly repercussion on performance, on the other hand large steps may influence the consistency of the simulation. This approach is very simple to implement and to understand but the main drawback is the unnatural representation of simulation time in some domain, furthermore a slow LP is a relevant bottle-neck for the simulation.

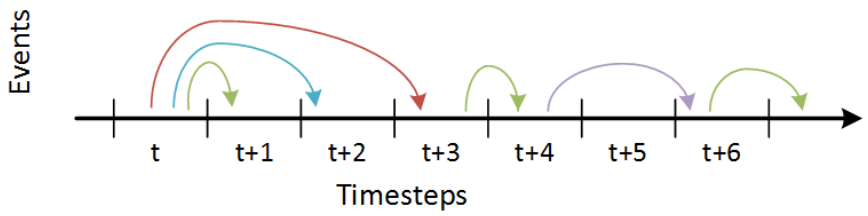

Figure 1.3: Discretization of time in time-stepped synchronization approach [19].

1.3.3.2 Conservative

The aim of the conservative (*or pessimistic*) synchronization is to avoid *causality errors* (see paragraph 1.2.4); in this approach we want to determine when the processing of an event is safe or not: we establish this when all events that might influence the interested event have been processed. Hence, the LPs have to determine if each event is safe before processing it: if in an event list we have an unprocessed event E_t with timestamp T_t (and no other event with an inferior timestamp), and we can establish that it is not possible to receive other events with an inferior timestamp (i.e., in consideration of timestamp T_t), then we can securely process E_t because it is guaranteed that doing so will not later lead to a violation of the causality constraint. To respect the causality constraint, this synchronization approach block the LPs that are containing an unsafe event until the event itself is not safe: it is clear that we have to take appropriate precautions to avoid the danger of deadlock situations [25].

One of the most used algorithm for pessimistic synchronization is the *Chandy-Misra-Bryant (CMB)*. In the CMB algorithm each LP needs to have an incoming message queue for each LP that interacts (i.e., sends event) with it. A LP produces events in a non-decreasing timestamp order, hence it is relatively simple to establish the succeeding safe event: it has only to check all the incoming queue. After that, the event with the lowest timestamp t is surely safe and can be processed. In CMB the deadlock is prevented using

NULL messages that allow to interrupt the "circular chain" (i.e., is one of the required condition to lead to deadlock's situations): these messages have not a particular semantic but they are necessary for sharing information about synchronization: each time an event is processed, a LP must send N NULL messages each of which to the N LPs to whom it is connected [25, 19, 31]. It is evident that NULL messages can be present in a very huge number, so it is legit to assume that CMB is *"communication oriented"* instead of *"computation oriented"* (e.g., like optimistic mechanism): this communication overhead may have a relevant effect on the total time of simulation and, above all, introduce so many message can be an evident problem for the purpose of this work.

1.3.3.3 Optimistic

The first assumption in this approach is that the processes are free to violate the causality constraint, indeed optimistic methods detect and recover from causality errors; they do not strictly avoid them. Each LP process all events in *receiving order* and, in contrast to conservative mechanisms, they are not interested if the processing of an event is safe or not: eventually they determine when an error has occurred, and invoke a procedure to recover a prior state that was correct. Doing a very simple parallel: we can consider this procedure like saving a game on a computer game, if something goes wrong, we can still restore the previous situation. This mechanism of recovery the internal state variables of the LP in which happened the violation is called ***roll back***. By definition, a process detects a causality error each time it receive an event message containing an inferior timestamp compared with the timestamp of the last event processed (i.e., this event is in "late"): literature define this event a *straggler* [25]. Hence, when a straggler is detected, the interested process must perform a roll back: its state is restored to a previous simulation time (respect to the straggler's timestamp), after that it must propagate the roll back to all other LPs (i.e., these restoring messages are called "anti-messages"): if the last event processed by an LP that has received

an "anti-message", has a timestamp inferior to the straggler's timestamp, then even this LP has to perform a roll back [32]. It is possible to generate "rollback's cascades": to deal with this eventuality is fundamental that each LP stores a list of the sent messages and also the local state data. It is evident that we need to spend a lot of memory and computational resources; the basic idea is to perform and propagate a consistent restore in order to make a rollback up to a global safe state.

This mechanism is called ***Jefferson's Time Warp***[32] and, as previously stated on the conservative approach section, it is "computation oriented": the proponents of this approach assert that computation is much faster than communication; on the other hand advocates of conservative approach point out that the Time Warp mechanism is much more complex to implement.

1.4 Cloud computing & PADS performance

We have seen the key aspects of PADS but what about performance? In particular related to new trends like *Cloud computing*[3]? Considered what was stated in the previous sections it is reasonable to think to exploit the recent evolutions of computing systems, like public cloud technologies, also in the PADS context. The most important advantage is surely the approach *pay-as-you-go* or rather you pay only for the rented resources and you can increase or decrease them dynamically but, a possible drawback, this kind of environment can be very dynamic, variable and heterogeneous. Moreover, the "market of cloud services" is composed of many competing vendors and the price of the same service is often the result of a market economy of supply and demand: the user usually must compose services from different providers to take advantage (i.e., in price terms) of the competition among vendors. The first consequence of this is a more heterogeneous and dynamic

[3]As Defined by NIST the Cloud is: *"A model for enabling ubiquitous, convenient, on-demand network access to a shared pool of configurable computing resources (e.g., networks, servers, storage, applications, and services) that can be rapidly provisioned and released with minimal management effort or service provider interaction"*[33].

architecture, where the lack of interoperability and the use of different APIs among vendors are a relevant problem.

It is also necessary to introduce other parameters to evaluate this approach: the most common metric to evaluate a simulation is the time needed to perform a simulation run (WCT) but in this context it may also not be the most relevant one; we have to take in consideration others factor like how much time we can wait for the outcome of a simulation or how much we want to pay the cloud providers for a simulation run? Typically an upper bound of time is defined and this guides the cost of the resources. Is this pricing model suitable for PADS?

In relation to what was observed about the mechanisms of synchronization it is possible to assert that both *optimistic* and *pessimistic* approaches are not well suited for a pay-as-you-go policy: indeed the *CMB algorithm* is communication-oriented (i.e., the user has also to pay the bandwidth and this communication overhead might be a problem for the WCT) and a lot of computation is wasted (e.g., an LP is stuck waiting to get information to decide if an event is safe), on the other hand, for example in a Jefferson's Time Warp mechanism, there is the risk of spend a considerable amount of time and computation due to roll-backs mechanism [31], furthermore the possible rollback cascade must be handled (i.e., this implies investments in bandwidth and, as we have seen, in volatile memory). In all mechanisms (time-stepped included as explained in 1.3.3) the slowest LP might affect global performances of the simulation: in a pessimistic approach all the LPs are constantly waiting for a NULL message from the slowest one, in an optimistic approach, it is clear that the slowest LP shares events that are always in late respect to the ones produced by the other LPs: the simulation is bottle-necked by it and its lateness will cause a lot of roll-backs.

Besides the advantage of not having to buy hardware, as we said in the introduction, cloud computing offers to applications the opportunity to require more resources at run-time: this translates into a significant benefit for many kind of domains. In our context, we can scale down the CPU consumption

of processor-bound execution by adding instances (*horizontal scaling*) or, on the other hand, we can request an upgrade of actual nodes where a higher number of processes are located (i.e., to reduce remote communications) in order to avoid CPU-bound bottleneck (*vertical scaling*). In the fourth chapter, we will see an example with Amazon Elastic Compute Cloud (*EC2*) [34] instances .

Related to the considerations about synchronization mechanisms, we must also consider the problem of partitioning: the distribution of cloud services, maybe provided from different vendors (it is our case as we will explain in the fourth chapter of this work), means a convoluted version of the partitioning issues described above. Our purpose is to reduce communications and, at the same time, balance the computation workload across the execution units: especially in the context of cloud computing it is necessary to realize a good partitioning strategy in order to pursue these achievements (*load balancing*). The communication pattern evolves during the execution of the simulation: it is usually frequent that, at some point, several processes open and close communication streams to other component, in response to the evolution of the simulation events. These dynamic changes have to be managed with an *adaptive* approach and it is clear that the static partitioning is not a performant solution [35] .

1.4.1 Adaptivity

The lack of adaptivity implies static partitioning and this leads to drawbacks like a poor load balancing or a weak interactions approach between partitions (e.g., the communications among different partitions are frequent but the processes inside the partition do not communicate with each other: this implies that they are not clustered properly).

An adaptive partitioning of the model is fundamental to resolve most of the complications described in this section. First of all, following the solution proposed in [31], it is necessary to further decompose the model in tiny components called ***Simulated Entities (SEs)***. The model behavior

is defined by the interaction, with a message exchange protocol, between all the SEs; a subset of the total SEs is contained in each LP (that is running on a specific PEU) but they are not statically bound to that LP: they can be moved in order to reduce the communications overhead and enhance the load balancing. In order to optimize the use of computational resources or even to introduce a real scalability mechanism, the simulator can creates or delete LPs during the simulation. In other words, the simulation is organized as a ***Multi Agent System (MAS)*** [36].

The simplest strategy to reduce communication overhead is to group the strongly interacting SEs within the same LP, respecting the load balancing constraint (i.e., having too many SEs in the same LP does not lead any benefit).

In Figure 1.5, it is possible to distinguish three different "*interaction groups*" (marked with different colors). An efficient allocation of SEs is the one depicted in Figure 1.5 (b), we can notice that the interaction groups have been correctly reallocated to the same LPs/PEUs. It is important to notice that, in this example, load balancing is respected (i.e., each LP has the same number of SEs before and after the migration, and the sizes of interaction groups are approximately the same). This procedure has its costs that include network transfer delay, serialization and de-serialization of state variables of the migrated SE, therefore it requires a constant evaluation of communication and load balancing of each LP, which allows to understand if and when it is convenient to perform a migration.

Furthermore, with regards to the observations made about the performances of the synchronization mechanisms, we can evidence particular delays due to slow LPs: the simulator can exploit these adaptivity mechanisms in order to stem this retards; for example, at a given simulated time, it could migrate some SEs from the slowest LP (decreasing its workload) to the most performant one.

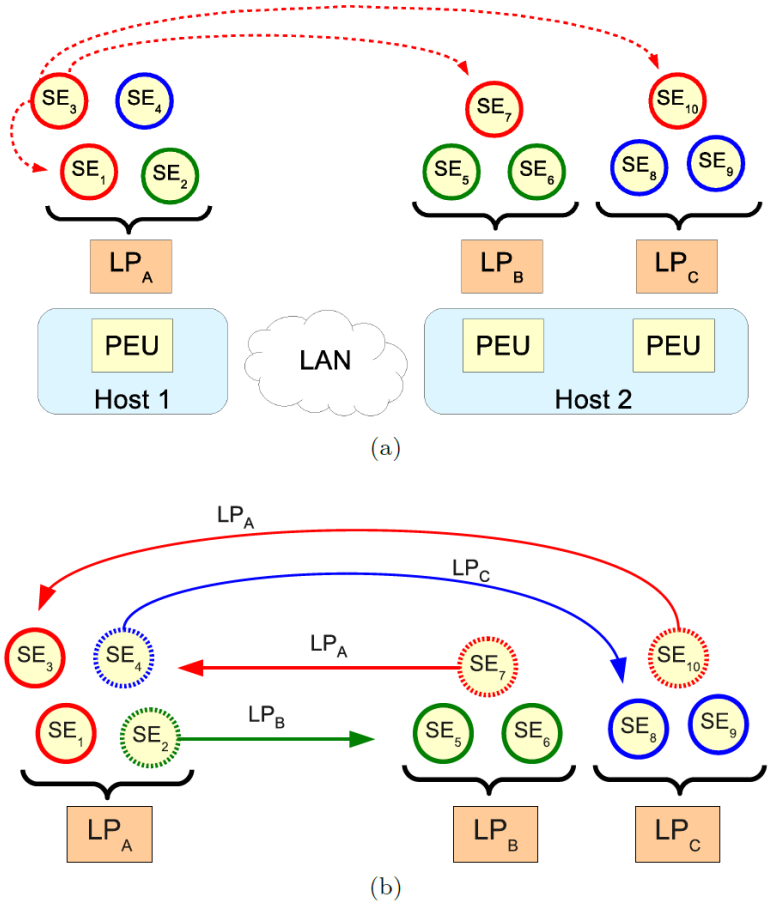

Figure 1.4: An example of PADS dynamic partitioning with two hosts connected by a LAN [31].

1.5 ARTÍS/GAIA

The Advanced RTI System (ARTÍS) [8, 9], the case study of this work, is a middleware that enables parallel and distributed simulation; it is specifically thought to support a high degree of model scalability and execution architectures composed of a huge number of Logical Process (and then PEUs). The design of the middleware is guided by the *High Level Architecture (IEEE1516 Standard)* [37].

One of the most important feature of ARTÍS is the implementation of

adaptive mechanisms to handle the different patterns of communication between LP, in order to reduce the communication and computation overheads above mentioned and, consequently, reduce the WCT of the simulation. ARTÍS provides an *Application Programming Interface (API)* that includes functions for message passing (e.g., send, receive, and so on), that are used indiscriminately by the LPs without them having to face the problem of how the underlying hardware architecture is structured: if through the ARTÍS API takes place a communication between two LP that are on the same host, this is carried out through shared memory; if instead the communication takes place between LP that are on different computers connected to the network, this is carried out by network protocols (e.g., TCP/IP). In this thesis, we will consider only the second case also for LPs located in the same computer. Furthermore ARTÍS offers the user the possibility to choose one of the mechanisms of synchronization described in section 1.3.3 of this work.

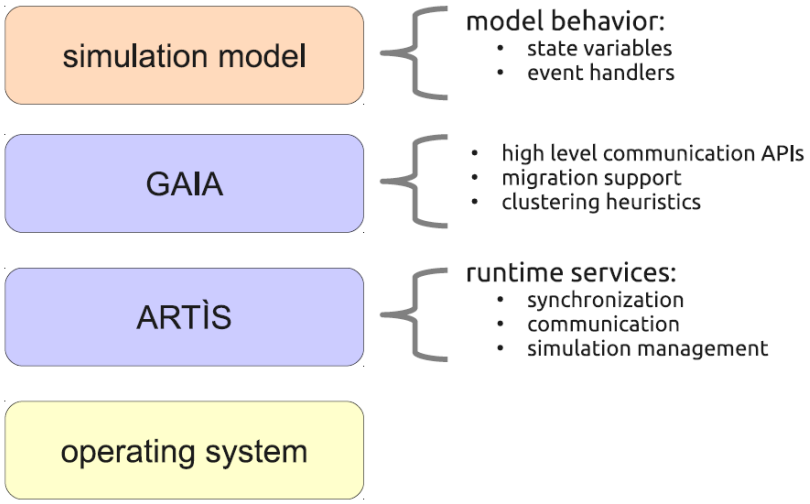

Figure 1.5: Structure of a simulator using ARTÍS middleware and GAIA framework [31].

The implementation architecture of ARTÍS is partially centralized: the LPs involved in the simulation have all the same role and they are all distributed over the network but there is an exception constituted by a central-

ized element called *SImulation MAnager (SIMA)*. SIMA covers various tasks including the management of synchronization barriers during execution, coordination of the LPs, the initialization and termination of the simulation. For the purposes of this thesis the role of SIMA is fundamental: it knows the number of LPs and the nature of communication channels that connect them and, moreover, it indicates to each LP how to enter into dialogue with others. I will explain in detail the behavior of the SIMA in chapter 3.

Generic Adaptive Interaction Architecture (GAIA) [8, 38] is a framework that relies on the middleware for parallel and distributed simulation ARTÍS. The approach described above in section 1.4.1 based on LPs viewed as containers of entity is made possible by the introduction of GAIA and its main features are: dynamic partitioning and adaptive allocation of model entities. The analysis of network conditions, the analysis of rates of communication between the various entities and implementation of migration are carried out by GAIA while running the simulation. In order to introduce an adaptive load balancing mechanism among LPs, *GAIA+* [39] was developed, an extension of GAIA that extends the framework with analysis of the workload of each LP and realize migration strategies with the purpose to balance distribution of computation.

Chapter 2

Anonymization background

This chapter will introduce the main concepts regarding the notions of privacy and anonymity, the principles behind anonymization networks and the functioning of one of the most popular softwares for enabling anonymous communications (Tor).

2.1 The importance of being anonymous

> *"Anonymity is a shield from the tyranny of the majority"*
> Supreme court of the United States of America [40].

It is granted to emphasize how connectivity and Internet usage have become pervasive today, as it is clear that in later times some rights, considered pivotal in the civil society, have lost value or, at least, have changed form. In the past, concepts such as privacy, anonymity, freedom of speech and expression have always been protected (at least in democratic countries): controversial voices have frequently been felt thanks to the cover of anonymity (e.g., many authors who might be afraid of being persecuted because of their thought have thus been able to freely express their opinions without fear of recrimination). Pseudonyms have always played a key role in politics and literature in general, so that even people marked by the opinions previously

expressed, or by becoming a member of some association or group, to express their ideas without the danger that these were distorted or denied a priori: many censored writers have continued to work thanks to the possibility of using a fictional name. However in the context of the Internet these rights have suffered, with the passing of time, a limitation and, in any case, have always received an ad-hoc treatment, as if the subtle difference between what should be an inalienable right have consisted of a means of communication. Recent news events have definitively established that the concept of privacy in the name of a supposed security, is now practically disappeared and, above all, it is impossible to be implemented in a context such as the Internet (if not by a sudden change of course from governments worldwide more than a technological change): any citizen of the world who has used the Internet and its services was, in all probability, observed, studied, watched by United States government entity (NSA[1])[1]. But is it really possible to give up rights and personal freedom in the name of a possible increase of national security? Are we really talking about our safety? Who really has access to our personal information and why? *"Quis custodiet ipsos custodes?"*[2] What advantages do we have from being truly anonymous on the net? And which drawbacks? The main reasons can be:

- **Freedom of speech**: in many countries around the world it is not possible to freely express our opinion. Just think about journalists and bloggers in some countries of the Middle-East or human rights activists that are using anonymous networks to report abuses from danger zones.

- **Location independence**: first of all, we might want to avoid sharing our position with the rest of the world, as a matter of personal safety (i.e. it is simpler than it seems to discover the city or even the street from where we are connected). Moreover, in some countries it may be forbidden to access certain services (e.g., like Google services or

[1] National Security Agency

[2] A Latin phrase of the Roman poet Juvenal. It is literally translated as *"Who will guard the guards themselves?"*

Facebook in China).

- **Privacy**: it may look the same but it is a direct consequence of anonymity. A possible definition from [41] is *"ensuring that individuals maintain the right to control what information is collected about them, how it is used, who has used it, who maintains it, and what purpose it is used for"*. In other words, we do something and we are free to decide whether we want to make it known to others or not. A simple example of everyday life concerns the sale of our Internet browsing data by Internet Service Providers (ISPs). There are many companies willing to pay for records of visited sites, for the text of performed searches on a search engine or even to know our username or information about our password. Also websites maintain their own logs that are containing a lot of useful and valuable information about their users.

- **Network security**: it may seem trivial but it is certainly one of the most important advantages. Using an anonymous network protects the user against a common (often malicious) technique called *traffic analysis*: it is the process of intercepting and studying messages with the purpose of reveal information from patterns in communication. The most dangerous aspect is that it can be performed even when the messages are encrypted: it is not necessary to violate the integrity of the data with decryption. Furthermore, knowing the source and destination of Internet traffic allows others to track your behavior and interests.

- **Do illegal things**: *All that glitters is not gold*, anonymity brings with it a series of problems concerning the execution of illegal activities (e.g., Silk Road marketplace [42]).

2.2 Online identity

The Internet is based upon a simple principle: transferring information from a terminal to another. In order to do this each terminal needs an iden-

tity which is determined by the IP address. An IP address is an unique identifiable information that is received by the other components of the network when any communications link is made over the network. This includes visiting web pages, sending or receiving e-mail, updating software, or using any network application. After the closing of a session (e.g., by turning off a device), when a terminal reconnects to the Internet will often gets a new IP address: this is not a certain rule for every provider. At some ISPs, the address assigned is bound to a connection for a period of time which can also be of several years and it becomes somewhat similar to a physical address (like a street address or even telephone number). A user can be easily identified by it. Our Internet identity is bound to the assigned IP also for a variable period of time after a particular Internet session: the ISPs that provide us connectivity must record and keep our personals data for months or even years due to respond to any request from the authorities. These data includes the name of the customer logged in, along with the IP address at the time of registration, as well as all, for example, the full record of websites the user has accessed on the Internet.

There are many services that allow to change IP address, but this is clearly not an effective solution: often what we do, the sites that we consult, or even what we search on a search engine, is sufficient to infer who we are. Moreover, I have already mentioned the traffic analysis attacks: a communication pattern is often almost as dangerous as the spread of an IP address. The only possible solution is to separate from the concept of identity (and IP address) and at the same time curbing the spread (voluntary or not) of our personal information.

But how can we actually do it? The basic idea is simple and may be summarized in two macro activities:

1. adding *intermediaries* between sender and receiver;

2. using *cryptography* (a lot of).

Prior to go into the details of the functioning of some protocols that guarantee anonymity, I can anticipate that the first activity consists, essentially, in the addition of some steps to the packet's path through the network. Why should we do this? To make it more complex, if not impossible, to determine the real source of the message. Clearly, this only activity serves no other purpose than to spread quickly our personal data, the application of encryption and decryption techniques will ensure the *confidentiality* and *integrity* of data[3].

2.3 The Tor Project

At the time of writing, the most famous and diffused anonymizing network is *The Onion Router* (Tor) [4]: the high number of users and the pervasiveness of the network, as we will see, is an indispensable characteristic for online anonymity: Tor has over 2 million users [43].

As often happens, also The Onion Router was born for military purpose: indeed the core of Tor (i.e., *Onion Routing technique*) was developed in 1997 by United States Naval Research Laboratory, with the purpose of assure, to the United States intelligence, a secure communication online [44] and the main goal of the creators was to limit a network's vulnerability to traffic analysis. The name *"onion"* is not accidental: it refers to the application of layers of encryption, equivalent to the layers of an onion. The onion data structure is formed by wrapping a message with layers of encryption, the number of layers is equal to the number of nodes that must be traversed before arriving at the destination. Each node can only decrypt its layer to obtain information regarding its successor (an example in Figure 2.1).

The first alpha version of Tor network was launched in 2002, in 2004 was published the second generation of Onion Router protocol [46], after this year

[3]**Data confidentiality**: private or confidential information is not made available or disclosed to unauthorized individuals. **Data integrity**: assures that information and programs are changed only in a specified and authorized manner [41].

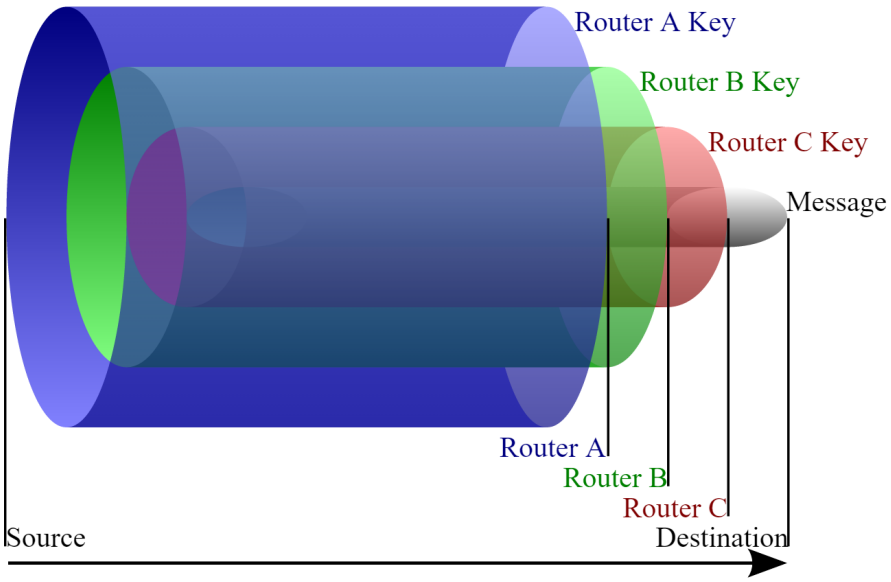

Figure 2.1: Example of an onion data structure: the source has wrapped the message in three layers of encryption, each one for each router in the network [45].

the growth of the Tor network will be constant up to the present day.

2.3.1 Introduction to Tor

The more general definition of Tor is: a circuit-based *low-latency*[4] anonymous communication service; onion routing is a distributed overlay network (i.e., built on top of Internet) designed to anonymize TCP-based applications (at the time of writing there is no support to others protocol like *User Datagram Protocol*). The main purpose of Tor is to discourage a possible attacker from linking communication partners, or from associating multiple communications to or from a single user. The main characteristics that drive the Tor design are:

[4]A network is *low-latency* if a human user can not notice delays between the processing of an input to the corresponding output. In other word, this kind of networks provide a real-time characteristics.

- **Deployability**: clearly the design must be used in the real world and it must not be expensive in term of resources (e.g., a reasonable amount of bandwidth);

- **Usability**: the system must be easy to use, a *difficult-to-use* system discourages the users and more users participate, more anonymity can be guaranteed. An easy to use system should require few configurations to the users and should be easily installed on all common operative systems;

- **Flexibility**: the protocol must be flexible and well-specified so that it can also be reused in other future researches;

- **Simple design**: the protocol's design must be as simple as possible; a complex system is difficult to manage from different points of view, and probably, more exposed to possible attacks. As often happens in the context of computer security: *"a system more is complex and more it is insecure"*.

2.3.2 Tor Design

The first operation accomplished by a client who is connecting to the Tor's network is the choice of a path and the building of a *circuit*, each node in this the circuit knows only its neighbors (i.e., the previous one and the next one): it does not know anything about the other nodes of the path. The nodes (also known as *relays*) that compose the overlay network can be distinguished in:

- ***Onion router (OR)***: each *OR* keep a *Transport Layer Security (TLS)* connection to every other onion router. There is no need for special privileges, the ORs are executed as user-level processes. The goal of the onion router is to connect clients to the requested destinations and relay data to the successor in the circuit. Each one maintains two different keys:

  - *Identity key*: is a long-term key used for different purposes. The most important are the signing of the TLS certificates and the signing of the *router descriptor* (a brief summary of its keys, address, bandwidth, entry and exit policy). The identity key is also used by *directory servers*[5] in order to sign directories.
  - *Onion key*: is a short-term key and is used to decrypt the requests coming from the users in order to build a circuit. The onion keys are changed periodically, to circumscribe the impact of key compromise.

- ***Onion proxy (OP)***: is a local software that each user must execute in order to participate to the Tor's network, it is needed to establish Tor's circuits and, most important for this work, to handle connections from applications.

The use of TLS connections implies that data are perfectly secrets, this can also prevent an attacker from altering the integrity or impersonating an OR. The unit of communication in Tor are fixed-size packets called *cells*: each packet is 512 bytes and it includes, as a standard TCP/IP packet, a header and a payload. The header is composed by a circuit identifier (*CircID*), that specifies the circuit in which the cell has to be routed, and a command (*CMD*) that defines how to use the cell's payload (*DATA*). Tor distinguishes two different kind of cells on the basis of their commands:

- *control cells*: these cells are always processed by the node. The commands for this kind of packet are:
  - *padding*: is used to communicate to keep-alive a circuit;
  - *create/created*: the first one is used to set up a new circuit, the second one is used as an *ack* to answer to a create request (i.e., to notify that a circuit has been created);

[5] A *directory server* is a trusted node that provides directories describing known routers. Users have to download them via HyperText Transfer Protocol (HTTP).

  - *destroy*: self-explanatory command, it destroys the circuit.

  The structure of a control cell is depicted in Figure 2.2 (a).

- *relay cells*: these cells are used to carry the stream data from the sender to the receiver. The header of this kind of cell includes other fields like a byte to identify relay cells, a stream identifier (*streamID*), a digest for *end-to-end integrity checking* and the length of the payload. The most important commands for these cells are:

  - *relay data*: the most common relay cell, it is used for data flowing;
  - *relay begin/end*: respectively used to open and close cleanly a stream;
  - *relay teardown*: this command is used to tear-down a broken stream;
  - *relay connected*: this cell passes through the circuit to communicate to an OP that a stream of data can start (i.e., the *begin cell* has been successfully received by the recipient);
  - *relay extend/extended*: operation used to extend a circuit incrementally, the second one is used as an *ack* (i.e., to notify that a circuit has been extended);
  - *relay truncate/truncated*: operation used to exclude a part of the circuit, the second one is used as an *ack* (i.e., to notify that a OR has been excluded);

  The structure of a relay cell is depicted in Figure 2.2 (b).

It is important to emphasize that all the relay cells are encrypted or decrypted together while they are moving through the circuit: Tor utilizes a 128-bit *Advanced Encryption Standard (AES)* cipher in order to generate a cipher stream. Now that I have introduced the different type of cells and relative commands, I can explain how Tor protocol works.

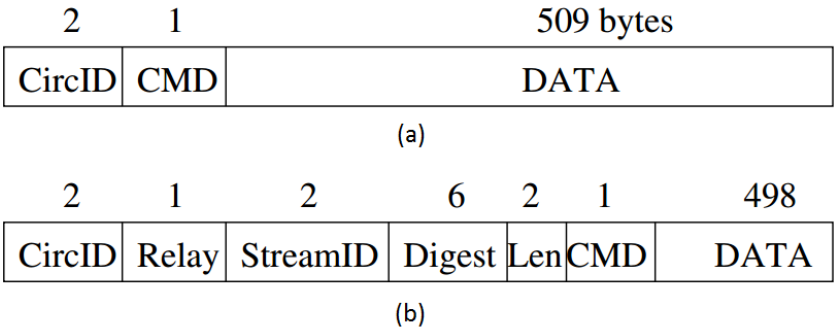

Figure 2.2: Tor Cells: structures of control (a) and relay (b) cells. [46]

.

Construct a circuit

The construction of a circuit is an incremental operation: an OP determines a path (actually, is usually composed by at least three relays) and, after that, it will negotiate a symmetric key with each relay on the circuit, hop by hop. Hence the OP (e.g., Alice) send the first *control cell* to the first OR (*OR1*) in the path: its command is *create* and the *CircID* is assigned *ex novo*. The payload of this first message includes the first half of *Diffie-Hellman*[6] handshake (g^x) encrypted with OR1's onion key. The answer is a *created control cell* that include in the payload g^y (second half of D-H) with a hash of the agreeded key $K = g^{xy}$. The first hop of the path is completed, now *Alice* sends the first *relay extend cell* to OR1, specifying in the payload: the address of the next relay (*OR2*) and an encrypted g^{x2}. OR1 has to copy the D-H half-handshake into a new *control create cell* destined to the new recipient to extend the circuit (this implies the choice by OR1 of a new CircID, Alice is not interested to know this CircID). After that, OR2 responds with a *created cell* to OR1, OR1 forward the payload to Alice with a *relay extended* cell. The circuit is now extended with OR2, Alice and OR2 share

[6] *Diffie-Hellman key exchange (D-H)* is a method of securely exchanging cryptographic keys over a public channel. The goal of this method is to establish a common secret between two different subject, this secret will be used for a secure communication for exchanging data.

a common key $K_2 = g^{x2y2}$. To conclude the circuit with a third relay, Alice and the other ORs proceed as above. It is important to notice that Alice never uses public key thus remaining anonymous. In Figure 2.3 it is depicted the procedure above.

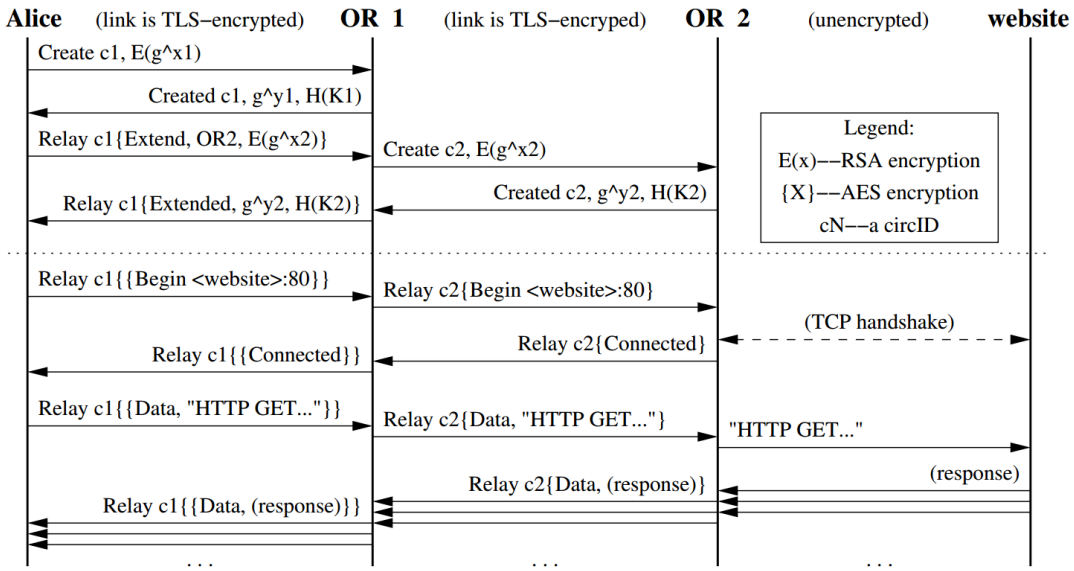

Figure 2.3: Constructions of a two-hop circuit (procedure above the dotted line) and fetching a web page in Tor. [46]

The circuit has been fully established, now the OP can send relay cells. Every time an OR receives a relay cell, it looks up the *CircID* and decrypts the header and payload with the session key for that path. The OR always checks the integrity using the digest, although it is not valid it sends the relay cell to the next OR in the circuit: if the last OR of the circuit receives an unidentified cell, the circuit is automatically closed. When an OR wants to reply to an OP with a relay cell, it encrypts the cell with the key it shares with the OP and sends back the relay cell to the OP. The successive ORs in the circuit add layers of encryption. When an OP finally receives a relay cell it iteratively decrypts the content with the session keys of each OR of the specified *circID* (from the nearest to farthest), at any step of this unwrapping

procedure it checks the integrity thanks to the digest of each cell.

Opening streams

When an user's application needs a TCP connection to a specified address and port, it requests the OP to prepare the connection. This demand is performed by *Socket Secure (SOCKS)* protocol (in the third chapter we will see how this protocol works). After the creation of the circuit, the OP sends a *relay begin* cell (with a new random *streamID*) to the exit node to open the stream. Once the exit node is connected to the remote host (e.g., a website), it answers back with an ack (*relay connected* cell). After having received the ack, the OP notifies to SOCKS the opening of the stream. Now the communication consists of a series of *relay data* cells. The relay cell exchange message protocol is depicted in 2.3 below the dotted line.

2.3.3 Rendezvous Points and Hidden Services

One of the main features of Tor is the possibility for users to offer services (e.g., web publishing, instant messaging server, etc.) while their location is totally hidden (clearly the IP will never be revealed). This anonymity is extremely important while an OP is offering a service because it protects against *Distributed Denial of Service (DDoS)*[7] attacks: the attackers do not know the OP's IP address, they are forced to attack the onion routing network. The explanation of Tor's Hidden Service Protocol is fundamental to understand how it is possible to make a simulator anonymous.

First of all an OP (e.g., *Bob*, our hidden service in this case) needs to create a long-term public key pair to identify its service. The next step is to create some circuits towards different ORs, called *introduction points (IP)* (Figure 2.4(a)). Once the circuits are created, Bob creates a *hidden service descriptor* composed by his public key and a list of the IPs: he signs this

[7] A DDoS is an attempt to make a resource unavailable to its intended users, typically by flooding the target of requests

advertise with his *private key* and uploads it to a distributed hash table (Figure 2.4(b)). Another OP (e.g., *Alice*, the client) is interested in Bob's hidden service: she needs to know the *onion address* of the service (maybe Bob told her or she found it in a search engine or website). She requests more info about Bob's hidden service from the database and subsequently chooses a new relay, called *rendezvous point (RP)*, that it will be used for her connection to Bob (Figure 2.4(c)).

The rendezvous point is ready, the hidden service is up and the descriptor is present: Alice must prepare a *"presentation letter"* called *introduce message* (encrypted with the Bob's public key) composed by a *one-time secret* and the address of the RP: she contacts one of Bob's IPs and asks to deliver the introduce message to Bob (Figure 2.4(d)). The message also contains the first part of the D-H handshake, as described in the procedure for opening of a circuit. Bob decrypts Alice's introduction message: now he knows the RP's address and creates a circuit to it; after, he sends a *rendezvous message* that includes the second half of D-H handshake and an one-time secret (Figure 2.4(e)). Now the RP links Alice to Bob, but it can not identify neither the sender nor the recipient, or even the information transmitted through the circuit. Alice starts the communication with the sending of the usual *relay begin* cell along the circuit (Figure 2.4(f)).

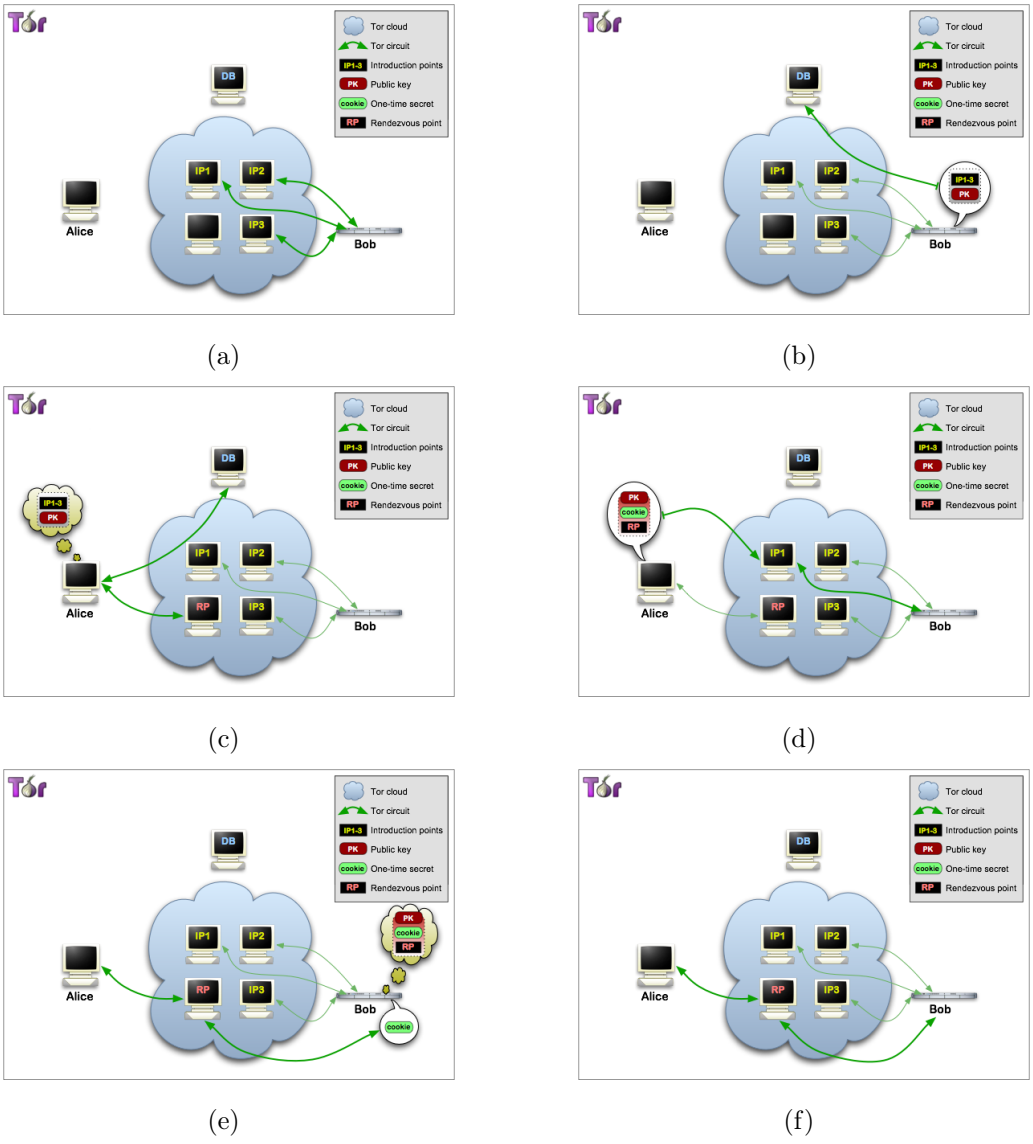

Figure 2.4: Tor's Hidden Service Protocol [4].

Chapter 3

An anonymous simulator

In the first two chapters, I have introduced the theoretical foundation to implement a prototype of an anonymous simulator. In this chapter, we will see the design of *ARTÍS/GAIA*, the proposed changes in the implementation needed to anonymize the communication streams and the LPs participants and the creation of the hidden services (see section 2.3.3).

The anonymized version of the simulator is a global prototype feature, this implies that is applied to all the Logical Processes at compilation time. Hence, it is not possible to have some anonymous LP and others that are not anonymous: in any case all the LPs should use Tor, anonymous ones to preserve their anonymity and to communicate with other LPs in the same condition, non-anonymous to translate the *onion address*. Moreover, the hypothesis of not anonymizing some LPs does not make much sense associated with the purposes of this thesis.

3.1 ARTÍS logical architecture

As previously said in section 1.5, ARTÍS is a middleware for parallel and distributed simulation, it is inspired by a Runtime Infrastructure provided by *High Level Architecture (HLA)*[37]. HLA is the point of reference for distributed computer simulation systems, it guarantees reusability and in-

teraction with other computer simulations, independently of the computing platforms. This standard does not cover the implementation of a distributed simulation, which depends on who develops the simulator, but it defines an architecture and a set of specifications that a simulation must respect, also to ensure the appropriate interoperability. A HLA-compliant simulation is composed by a set of *federates* (like our LPs) that interact with each other to perform a simulation (i.e., *a federation*). The possible interactions between federates are defined by the *Federation Object Model (FOM)*, the FOMs have to follow the structures defined in the *Object Model Template (OMT)*. In the end, the most important element for us, and in general for HLA, is the *Runtime Infrastructure (RTI)* which represents the implementation of the distributed system underlying the simulation and that makes it possible to execute the federates and their interactions. A possible drawback of HLA standard is the absence of a paradigm based on the migration of the simulated entities, the framework ARTÍS/GAIA was developed to solve this problem [31].

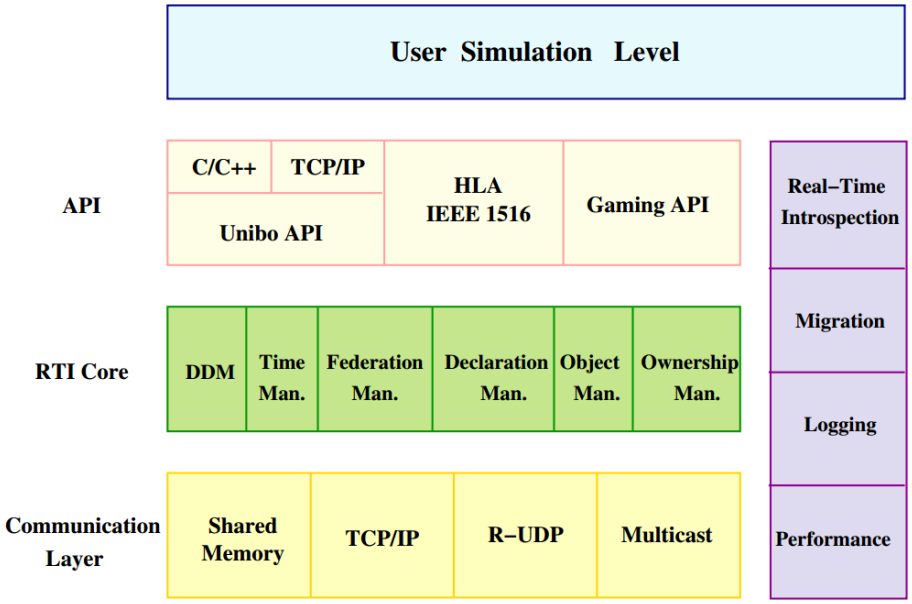

Figure 3.1: Logical architecture of ARTÍS [47].

The implementation of ARTÍS follows a component-based design pattern: this translates into an easy to extend set of different modules. In Figure 3.1 we can observe the layered logical structure of ARTÍS, where the different logical modules are organized in a stack-based architecture.

The communication layer (and associated API) is probably one of the most important and it is definitely the one that is central to the purposes of this thesis. Considering the communication cost of a distributed simulation, ARTÍS tries to improve the performances selecting adaptively the most appropriate communication module: for example, in multi-core or multi-processor architecture, ARTÍS takes advantage of shared memory to reduce the communication overhead; in the same way, if several hosts are located in the same Local Area Network (LAN), hence without shared memory, ARTÍS adaptively uses Reliable User Datagram Protocol (RUDP). In our context this is not possible, as stated in section 2.3, Tor can use only TCP as a transport layer protocol. Therefore, the anonymous communication feature must be positioned on the top of the TCP/IP module.

3.2 Communication architecture

As we have seen in section 1.5, ARTÍS provides a central element in the architecture: the SImulation MAnager (SIMA). This element is fundamental, especially in an anonymous simulator: I can affirm that the partially centralized architecture of ARTÍS is predisposed to anonymization thanks to the presence of the SIMA, without it I would have had to introduce a similar entity. Indeed, in the "initialization phase", the SIMA knows the number of LPs and the nature of channels which connect them, each LP needs to contact the SIMA in order to receive information about the other participants at the simulation. It is also a relevant point of synchronization: the simulation can start execution only once the transmission channels between the LPs are effectively initialized.

Figure 3.2, depicts the communication architecture. The first step is

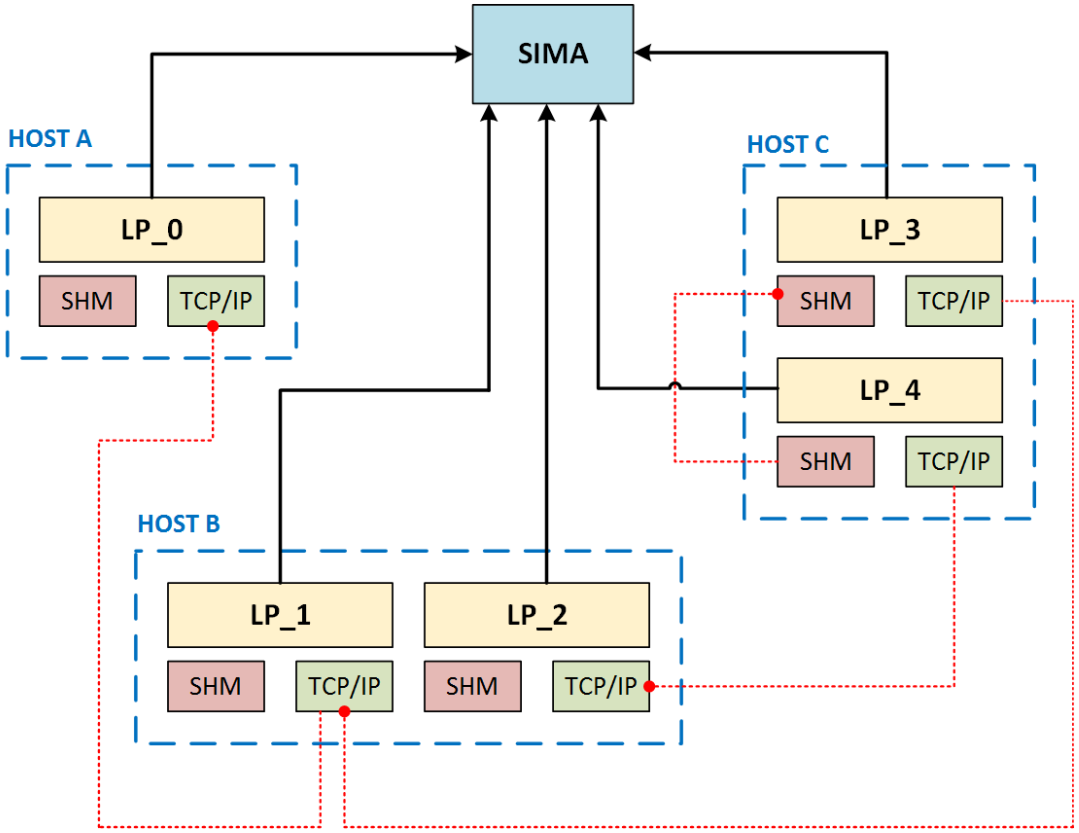

Figure 3.2: Communication architecture of ARTÍS.

the initialization of SIMA, the LPs contact it in order to send information about themselves (in Figure 3.2 this phase is represented by the black lines). Because the SIMA knows the number of LPs, it waits until every single LP has sent his information. The simulation will begin only when each LP receives information about all the other LPs from the SIMA.

Subsequently the nodes start exchanging data in an adaptive way (in Figure 3.2 the red dotted lines). In the next sections I will go into more detail about this communication protocol, at this point it is important to notice that each LP needs to previously know how to contact the simulation manager. This forces us to determine in advance the identity (or better, the *fake identity*) of our SIMA.

3.3 Proxy server

In the second chapter, we have seen that Tor uses a SOCKS proxy interface to allow the support of TCP-based programs. It is our case, the goal is to pass each TCP communication inside a Tor's tunnel and we need to use SOCKS to route network packets between a client and a hidden service. Many applications use a simple remote proxy with the promise of providing anonymity between sender and receiver, but typically these applications cannot provide any anonymity against an eavesdropper who can observe all messages from and to the proxy [48]; through the use of a local SOCKS server, and thanks to encryption offered by Tor, the possibility of eavesdropping is averted. The functioning of SOCKS, related to Tor, is quite simple and we can summarize it in the following steps:

1. The client (e.g., a LP) connects to the local SOCKS server instead of the destination's hidden service (e.g., the SIMA);

2. The client sends a *connect request* to the receiver;

3. The hidden service replies to the SOCKS server of the client with an *answer code*: if the request is accepted by the destination then the SOCKS server tries to establish a connection:

   (a) if it has not been possible to establish the connection, the SOCKS server sends to the client a negative answer and the connection is turned down;

   (b) if the connection is established with success, the SOCKS server starts to transfer data in a bidirectional way between the client and hidden service.

While I was checking the documentation of the Tor project, I noticed that the use of some versions of SOCKS is highly discouraged due to probable leaks of information caused by hostname resolution (the documentation

refers in particular to DNS[1] queries that are not encrypted, it is possible for an attacker to see what onion address a user is connecting to). For more information check [49].

3.3.1 SOCKS4a Protocol

The best version of SOCKS according to the documentation of the Tor Project is the 4a which uses only hostnames resolution and not IP resolution. In the SOCKS 4a protocol a client can specify a destination domain name (in our case an *onion address*) rather than an IP address (this is not possible in the base 4 version).

The typical structure of a connection request in SOCKS 4a looks like the one represented in Figure 3.3, the fields depicted have this interpretation:

- **VN**: version number, in our case *0x04*;
- **CD**: this field represents the type of command of a request (*0x01* to establish a TCP stream connection, *0x02* to establish TCP port binding):
- **DSTPORT**: the port of the receiver in network byte order;
- **USERID**: the sender can communicate to the receiver an identifier (e.g., in our case we use the onion address of the sender);
- **HOSTNAME**: this is the name of the host we want to contact, in our case is an onion address.

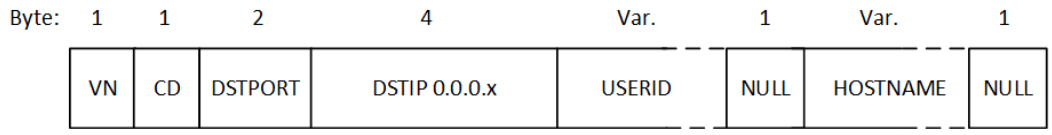

Figure 3.3: Structure of a SOCKS4a Connection Request

[1]Domain Name System is a decentralized naming system for hosts, services, or resources connected to the Internet or a private network.

On the other hand, the hidden service answers to the SOCKS server with a response message structured like in the Figure 3.4.

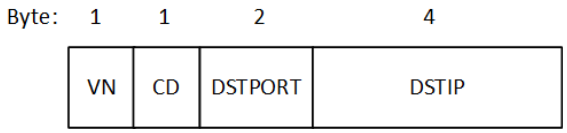

Figure 3.4: Structure of a SOCKS4a Connection Response

In the case of a connection response (our main case), the last two fields are ignored. The CD field is the most important, indeed it contains the result code of the request: **0x5A** (i.e., *90 dec*) means request accepted, **0x5B** (i.e.,*91 dec*) means request refused or failed. In the next sections, I will explain the implementation of this protocol in ARTÍS.

3.4 Interaction in ARTÍS

As previously described (refer to Figure 3.2), at the startup of the simulation we have an initialization phase: the main goal for the LPs is to communicate their existence to the SIMA and then know how to contact the other LPs. First of all, each LP needs to know the onion address and the TCP port of the hidden service of SIMA, the solution I have adopted was to place this information in a configuration file that contains other specifications of the simulation model of the respective LP.

Not only the SIMA but also the LPs expose hidden services in order to have their own onion address; about that, another relevant aspect to consider is the binding of the port used by these hidden services: which port should we use? For the moment we assume that the port is assigned to the LP in the startup phase, I will explain this aspect in the section 3.5.1.

Once a LP has bound the appropriate TCP port, it creates the first step of "*Tor tunnel*" using the necessary local SOCKS server (which is listening for a TCP connection on port 9050 by default): therefore proceeds creating a connection to `127.0.0.1:9050` and subsequently applying the SOCKS

protocol described above. Due to an experienced instability of Tor services at startup, LP may have to repeat several times the sending of a SOCKS connect request (typically the first responses are all with CD field equal to 0x5B or rather "Request refused").

When the SOCKS response is finally positive, a LP has its own connection (in our case a *network socket*) to communicate in a bidirectional way with SIMA: the first message sent includes all the LP info (for us the most important are surely the onion address and the TCP port) that are collected by the SIMA, the answer is a unique identificator for the LP.

Once finished the process of collecting the information about the LPs, the SIMA first sends to all a *table header* (that contains information like the number of nodes) and after sends the entire table. An LP uses the table header to allocate memory required to store the messages subsequently received, relating to other participating nodes to the simulation. The sequence diagram depicted in Figure 3.5 summarizes the interaction of this important initialization phase.

After this first step, each LP needs to establish a TCP connection to the rest of LPs: in other words we need to build a *complete (and anonymous) graph*[2] among the participating nodes to the simulation. We have already bound the hidden service port on every node, so the first LP (with the lower identificator) is accepting connection on its port. The other nodes, make a SOCKS request to each LP (i.e., hidden services) with a lower identificator. This procedure is incremental and the related communication topology is represented in Figure 4.1, the arrows indicate only the destination of SOCKS requests.

[2] A *complete graph* is a graph in which every pair of distinct node is connected by a unique edge. In our case the nodes are the LPs and the edges are a bidirectional TCP connections

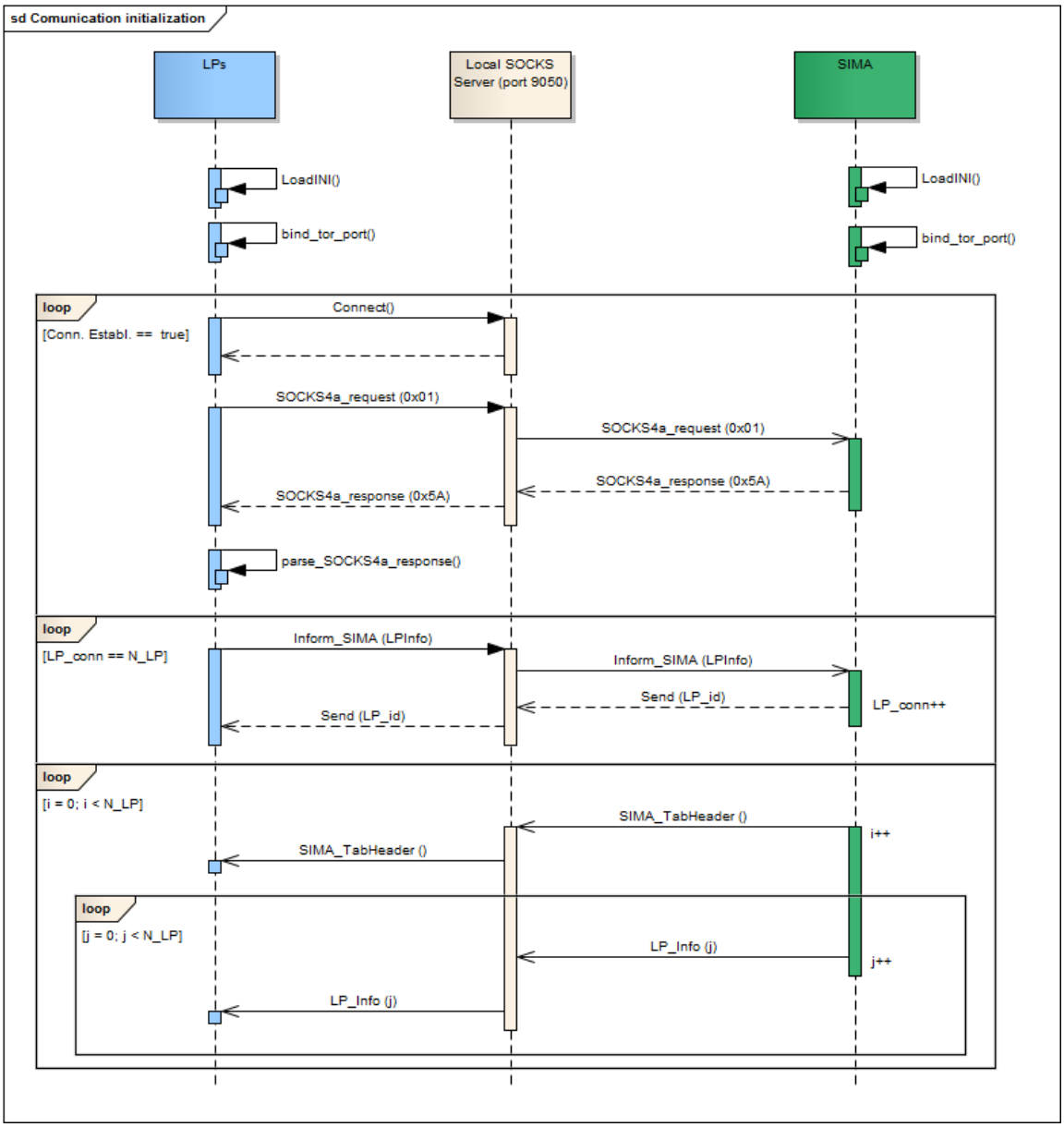

Figure 3.5: Sequence diagram of communication initialization using ARTÍS API. This UML diagram might not be the most appropriate choice to represent the interactions among entities of a distributed system, however the purpose of this figure is to clarify the initialization phase and the diagram is expressive enough for this goal. Furthermore, it is possible that the three entities are located on the same host.

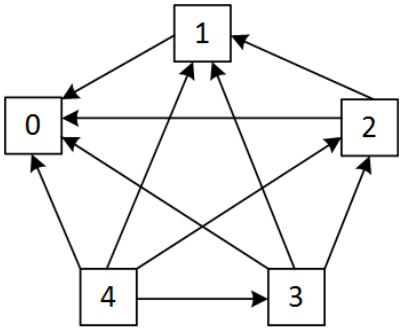

Figure 3.6: Communication topology between 5 LPs, the arrows indicate the SOCKS request destination. LPs recipients are listening for connections on TCP port associated with each relative hidden service.

3.5 Implementation

The middleware ARTÍS/GAIA is, for performance reasons, entirely developed in C programming language, but it also provides to users the Java language bindings to take advantage of the APIs. From the various features offered, I can cite the possibility of choosing the method of communication (e.g., shared memory or TCP), the possibility of using *Message Passing Interface (MPI)* or the choice of the synchronization approach [8]. Most of the features offered by ARTÍS/GAIA are defined at compile time through the file `compilation_config.mak`: we want that the possibility for the nodes to be anonymous is added to this list, therefore it will be necessary to modify the above mentioned file and all the *makefiles*[3] interested by the presence of this functionality (in particular those related to the Message Passing, GAIA and different synchronization approach directories). The modification takes shape with the addition of an instruction (`USE_TOR=YES/NO`) within the compilation configuration file: when this instruction is enabled all of the makefiles will include `USE_TOR` to the *macros* of the compiled files. Evidently this choice has a fairly pervasive impact in the code: based on the definition of the

[3] A *makefile* is basically a script that guides the make utility to choose the appropriate program files that are to be compiled and linked together.

above mentioned macro it is necessary to circumscribe certain portions of code and avoid others (this takes place utilizing the preprocessor statements like `#ifdef` or `#ifndef`) and in some cases this is not so simple.

Activation of the anonymous function also impacts on the start-up scripts of the simulation: we shall see in the section 3.5.1, the presence of Tor implies the configuration of the hidden services which promotes the passage of a series of parameters not present in the standard execution script. In regards to this, the data structure which describes each LP has to be modified in order to maintain its relative onion address (see Code 3.1).

```
typedef struct
{
    int     type;          /* Channel Type                */
    int     protocol;      /* Communication Protocol      */
    int     nodeid;        /* LP Identifier               */
    int     gnodeid;       /* Group Identifier (SHM and MPI) */
    int     port;          /* TCP Port                    */
    int     sockfd;        /* File Descriptor             */
    double  lookahead;     /* Outgoing Lookahead          */
    char    hostname[MAX_HOSTNAMELEN]; /* LP Hostname     */
    char    cname[MAX_HOSTNAMELEN];  /* LP Canonical Name */
    char    onion_address[ONION_ADDRESS_LEN]; /* LP Hidden Service Onion Address */
}
LPInfo;
```

Code 3.1: The data structure of the LPs.

The implementation of anonymity is mainly realized via the creation of Tor's tunnels in the initialization phases of communications previously described. All of the TCP connections will have to go through both SOCKS local server and Tor's circuits, this has made it necessary to realize the protocol of request-response in SOCKS4a and relative response parsing (designed in Figure 3.5 and implemented in Code 3.2).

```
SOCKET create_socks_socket(char *hname_snd, char *hname_rcv, uint16_t port )
```

```
{
    SOCKET sock = NO_SOCKET;
    int len = -1;
    char *req = NULL;
    char *reply_buf[RESPONSE_SOCKS4_LEN];
    int connected = 0;
    int try = 0, maxtries = 100;
    int nw , nr;

    for (try = 0; try < maxtries; try++)
    {
        //1. Create connection to local SOCKS server
        sock = do_connect_tor_socks();
        if (sock < 0)
        {
            SOCKET_CLOSE(sock);
            sock = NO_SOCKET;
        }
        else
        {
            //2. Create SOCKS Connection Request message
            len = build_socks_connect_req(&req, hname_snd, port,hname_rcv);
            //3. Send SOCKS Conn. Req. to the onion address of the rcv
            nw = writen(sock, req, len );
            ASSERT( nw == len, ("!") );
            //4. Read the SOCKS Response
            nr = readn( sock, reply_buf, RESPONSE_SOCKS4_LEN);
            //5. Parse SOCKS Response message
            connected = parse_socks4a_connect_resp(reply_buf,RESPONSE_SOCKS4_LEN);
            if (connected == 1) break;
            printRTI("COMM_____", "...retrying connect()\n");
            SOCKET_CLOSE(sock); sock = NO_SOCKET;
            sleep(3);
        }
    }
    return sock;
}
```

Code 3.2: Implementation of SOCKS4a protocol. The parameters passed to this function are the onion addresses of sender and receiver, and the port of the receiver's hidden service.

The realization of TCP connections in C is done utilizing the APIs and the protocol defined by Berkeley Socket [50]: without having to explain in detail the programming of these sockets, it is important to underline certain modifications of the behavior defined in the absence of anonymity. Firstly, as stated above, we obviously cannot rely on IP addresses, this has made it necessary to modify or exclude all the components which use or store this information.

Another relevant change regards the search and the exclusion of the possible calls to API functions `gethostbyname()` and `gethostbyaddr()`: we have observed that the resolution of the hostname is one of the most critical points in maintaining anonymity. The purpose of these functions is to resolve a hostname (e.g., a web site or also an onion address) in an IP address (in the case of `gethostbyname()`) and viceversa (in the case of `gethostbyaddr()`). Regarding `gethostbyaddr()` we are fortunate because the framework does not use this function, however there are many `gethostbyname()` invocations: we are clearly not interested with this resolution and we must avoid it and circumscribing these portions of code in the only case in which the proposed connectivity is not anonymous. When we are in anonymous mode, the calls will be avoided and the resolution of the hostname (hence of the onion addresses) will be directly delegated to Tor. All of the present hostnames will be substituted with the relative onion addresses.

3.5.1 Hidden services creation

We have already described the notion of hidden service in paragraph 2.3.3, clearly to seek benefit from and to configure the hidden services it is necessary that the hosts have Tor installed. The creation procedure is not one of great complexity, however in our case we must consider different aspects, but we shall proceed with order.

To create a hidden service and obtain an onion address it is necessary to modify Tor's configuration file (in the Unix-like systems, those supported

by ARTÍS/GAIA, it is located in `/etc/tor/torrc`) inserting the following instructions:

```
HiddenServiceDir /directory/path/hidden_service/
HiddenServicePort fake_port 127.0.0.1:real_port
```

The first row defines the directory (which must be accessible in reading and writing to the user that is using Tor) in which two important files are created: *private_key* and *hostname*. At the start-up of Tor's services, a new pair of public and private keys is created for the hidden service linked to the specified directory: private_key is located into that directory and it must not be shared with anyone, otherwise he would be able to impersonate the hidden service. The second file contains a brief summary of the public key and can be shared with the rest of the world: it is our onion address and it is similar to something like this:

```
zqktlwi4fecvo6ri.onion
```

In the second row, *fake_port* corresponds to the virtual port that the clients are thinking of using while the *real_port* corresponds to the port which the hidden service is actually listening to, the connections towards the fake_port are redirected to the real_port: these ports can coincide.

I deliberately wanted to leave suspended a query about the choice of ports: which ports should the LPs use? The initial solution foresaw the binding of a random free port at execution time, exactly before the connection procedure to the SOCKS local server. However, this choice presented numerous disadvantages such as the management of concurrent access to the file *torrc* on behalf of many LPs (plausible hypothesis regarding the cases of, for example, multi-core hosts) and, moreover, the introduction of considerable overhead due to bootstrapping of Tor's services to every present LP in that host (necessary procedure to render effective all of the modification in the file *torrc*).

As previously stated, the port is assigned during the start-up phase of the simulation and this permits us to resolve both problems: there is no longer a necessity to manage the concurrent access to the file and, in the case of several LPs in the same machine, the bootstrapping of Tor's services will be executed once.

The execution script creates the hidden services (if necessary, they may have already been created) equal to the number of the LPs present in a specific host: the port is determined in a predefined range (from 10200 to 10300), in an incremental manner. Subsequently Tor's services are reloaded, the procedure requires only an arbitrary number of seconds invested in the creation of necessary circuits; every LP will then be started with variable parameters on the base of the simulation model with the addition of onion address and relative port. In Code 3.3, we can observe the passage of the parameters to the created LPs.

```
# LPs execution
X=0
Y=$(($LP-1))
PORT=10200

while [ $X -le $Y ]
do
        ONION_ADDRESS=$(./read_onion_address $X)
        if [ $X -lt $Y ]
        then
                # foreground execution
                ./LP_model $((PORT+X)) $ONION_ADDRESS &
        else
                # background execution
                ./LP_model $((PORT+X)) $ONION_ADDRESS
        fi
        X=$((X+1))
done
```

Code 3.3: A piece of an execution script: in this example the LPs receive as parameters just their onion address and relative port.

The script `./read_onion_address` (Code 3.4) is very simple: it is used to access the file `HOSTNAME` in order to get the onion address of the LP `"X"`. This script must run only when the Tor's services have been reloaded after the execution of the hidden services creation script.

```
#Read_onion_address
HSN=$1 #Hidden Service Number
SIM_DIR="$(pwd)"
HOSTNAME=$SIM_DIR/"HS_"$HSN"/hidden_service/hostname"

sudo chmod 777 $HOSTNAME
echo $(<$HOSTNAME)
sudo chmod 700 $HOSTNAME
```

Code 3.4: Read onion address script: each *hostname file* resides in a directory *HS_N* where the *N* is the progressive number of the relative LP.

To modify the *torrc* file and adding, for each LP, the above described instructions, we use the script `./hs_creator` (Code 3.5). This script receives as parameters two values: the first one (`LP`) is the progressive number of the current LP and it is used in order to create the appropriate directory, the second one (`PORT`) is the progressive number of port assigned to that hidden service. The script checks if there is already a directory with the same path on the *torrc* file, in this case the hidden service for that directory has been previously created: thanks to the coupling between the progressive number (*LP*) and the port number, it is possible to re-utilize the previously created hidden services. For example, the progressive *LP=0* will have associated the directory *HS_0* and the port will be the *10200*; the progressive *LP=1* will have associated the directory *HS_1* and the port *10201*, and so on.

```
#LPs hidden service creation script
######
LP=$1
PORT=$2
SIM_DIR="$(pwd)"
```

```
HS_DIR=$SIM_DIR/"HS_"$LP"/hidden_service"
CONFIG_TORRC_FILE=/etc/tor/torrc
######
if [ ! -d "$HS_DIR" ]; then
  mkdir -p $HS_DIR
fi
  sudo chmod 700 $HS_DIR

if grep -q 'HiddenServiceDir '$HS_DIR "$CONFIG_TORRC_FILE"; then
   echo 'Hidden Service already exists'
else
   echo 'HiddenServiceDir '$HS_DIR >> $CONFIG_TORRC_FILE
   echo 'HiddenServicePort '$PORT' 127.0.0.1:'$PORT >> $CONFIG_TORRC_FILE
fi
```

Code 3.5: Hidden service creation script: this script receives two parameters, the progressive number of the LP and the assigned port.

The creation of the hidden services relative to the LPs is therefore dynamically executed, moreover the same procedure cannot be applied for the SIMA (we must remember that the onion address and the TCP port of the SIMA must be known and included in the specified configuration file) whose onion address should be previously created and shared with all LPs.

Chapter 4

Performance evaluation

4.1 Simulation model

The simulation model used belongs to the class of discrete models, hence the state variables which characterize the simulation will change in well defined instant of time. Moreover, the used model is dynamic in that it represents the evolution in time of a system; an additional characteristic is the presence of aleatory which classifies the simulation model in the stochastic class. We underline how this classification (*discrete, dynamic, aleatory*) defines, in broad terms, the peculiarities of the paradigm used by ARTÍS/-GAIA, or DES which we saw in section 1.2.4 of this work.

The model, contained in the directory `/MODELS/MIGRATION-WIRELESS/` in the last distribution of the middleware, it represents a set of mobile hosts which move within a specific area; their movement is random and is determined by a common mobility model: the *Random Waypoint (RWP)* model [51]. The area is defined by a bi-dimensional toroidal space, this implies that once the limits of the defined area have being reached by a simulated entity, this will not stop but will continue its course from the opposite perimetral point. The entities, should they be provided by a wireless device (customizable parameter), interact between each other via these devices within a cer-

tain proximity limits.

Certain parameters useful to the model are defined in the already cited configuration file: specifically the dimension of the toroidal space, the communication radius of the wireless devices and the speed of the nodes. As mentioned in 3.4, within the same file are contained the information to contact the SIMA (Onion address/IP address and the port). The number of simulated hosts (*SMH*) is initially equally distributed on all the participants logical processes: this parameter is passed to the LPs by the simulation execution script, together with the total number of processes (*NLP*) and, in the case of the activation of the anonymity mode, with the port used by the hidden service and the relative onion address.

```
./wireless  $NLP $SMH $PORT $ONION_ADDRESS
```

The implementation of the functionality of anonymity is therefore correctly transparent for the simulation model, the only shrewdness is the allocation of two variables act on storing the two input added parameters.

The synchronization approach utilized is Time Stepped (see section 1.3.3) and the model finds in its *macros* wide margins of configuration; among its most interesting parameters we have the duration of the simulation (defined in simulation steps by the `END_CLOCK` macro, default 1000), the flight time of ping messages between the node (expressed in terms of time step needed, `FLIGHT_TIME`, default 1.0), potential payloads added to simulate different network conditions (e.g., a particularly congested network), the probability of an entity to interact in a determined step with another device within its communication radius via a ping message (`PERC_PING`, default 0.2) and finally the percentage of hosts provided with a wireless device (`PERC_COMM`, default 1.0).

The nodes can interact exchanging three different types of messages:

- *MoveMsg*: used to update the position of a SMH;
- *CommMsg*: used to turn ON or turn OFF the wireless device of a SMH;

- *PingMsg*: used to send a simple ping message.

If GAIA is active, there is a fourth class of message called *MigrMsg*: they are used to communicate the migration of a SMH from a LP to another one (which will be effectively realized with the transfer of the SMH's state to another LP by the function `GAIA_Migrate`).

Each SMH is characterized by the following data structure:

```
typedef struct hash_data_t
{
    int           key;          // SMH identifier
    int           lp;           // LP in which the SMH is run
    float         posX;         // current position X-axis
    float         posY;         // current position Y-axis
    float         targX;        // target position X-axis
    float         targY;        // target position Y-axis
    char          changed;      // used by simulation visualizer
    char          mobile;       // static or dynamic simulated entity?
    char          comm_device;  // wireless enabled or not?
    TSeed         Seed;         // seed used for the random generator
    unsigned char rwp_state;    // random way point mobility state
}
hash_data_t;
```

Code 4.1: The data structure of the SMH.

Each LP, after loading the input parameters, proceeds initializing the communication following what we have said in the third chapter. This operation concludes with the assignment of an identifier (`LP_ID`) to each LP. Subsequently, the length of each time step is defined: this happen reading a parameter defined in another file (*channels.txt*, default 1.0). This value has to be less than the `FLIGHT_TIME` previously described (i.e., it is not possible to correctly model the messages if these should reach the destination before the completion of a step, clearly this would result in an inconsistency).

On the basis of the assigned `LP_ID`, multiplied by the number of simulated entities, the identifier of the initial SMH is defined for the relative LP.

Once this is completed, the model predisposes the configuration of certain settings relative to GAIA, particularly useful for the evaluations that we will perform: `GAIA_SetMigration` permits the activation and disactivation of the migrations mechanisms (i.e., according to what is described in 1.4.1, the simulated entities that compose the same interaction group are clustered in the same LP), `GAIA_SetMF` allows the specification of the *Migration Factor* that is a float value used to define the migration threshold of the heuristic, finally the function `GAIA_SetLoadBalancing` permits the activation of the mechanisms of load balancing, offered by GAIA+, among LPs (i.e., utilized in order to respect the balancing from a computational viewpoint).

The simulation output statistics are produced, step-by-step, in an extensive manner by what we define *"LP Master"*, by default the LP with identifier equal to zero. The other LPs will produce reduced outputs containing the WCT measured at the end of each step, the effective step performed, the number of entities located in a LP and the number of the migrated entities towards another LP in a determined step. The simulation for all of its duration (corresponding to the definition of the `END_CLOCK` value) is composed by a sort of *event loop* which will only effect the dispatch of the various incoming messages towards the appropriate *event handlers*. Some handlers will be directly activated by the messages originating from the various SEs (in this case we are referring to *Simulated Model Events*), others will be executed in response to the events produced by GAIA framework (in this case we refer to *GAIA Related Events*).

All the operations of receiving messages, as well as the respective sending functions, are clearly managed by the API provided by the ARTÍS/GAIA framework (i.e., specifically via the function `GAIA_Receive`, which holds in itself the function `TS_ReceiveV` provided by ARTÍS and utilized by the time-stepped approach for the reception of events from other LPs).

If a specific LP should not receive anymore events, from either model layer or other LPs, it proceeds with the termination handler of the current

step, if the whole simulation is not completed there will be a series of procedures useful for the evolution of the model (e.g., the movements or the possible migration of the SMHs) and the production of statistics: the *LP Master* will have to print the extensive outputs, which includes unique and global information like the intra-LP communications (i.e., the local communications) and the inter-LP communications (i.e., the remote communications).

Our expectation, in performance evaluation phase, is that the GAIA mechanisms allow to maximize the intra-LP communications, breaking down the elevated overhead introduced by the remote communication via Tor. For more information about the above cited function, refers to [52].

4.2 Simulation architecture

The architecture used for the performance evaluation is composed by three different hosts distributed throughout Europe. All of the computational resources is supplied by two different cloud providers: as previously stated, from a computer security viewpoint, we are entrusting the communication between the components of our simulation to a public network, inherently insecure. In this context, our objective are to preserve the confidentiality and integrity of the data in transit through the various physical execution units and to protect the availability of the assets, hiding their online identity. Regarding the availability, the idea of anonymizing the used instances allow us to consider our simulator unattackable in each of its components (e.g., a DDoS, for what stated in 2.3.3, can not take place: a possible offender is unable to attack a host without knowing its relative IP address).

Going back to the architecture: the communication will follow what is reproduced in Figure 3.2, however, in contrast to what it is represented, only one LP will be used for each machine. This decision will convey the means of communications, which are obviously be constituted only by TCP

connections.

Three hosts geographically distant from each other will be used in order to effect a concrete distribution of resources, one of this will cover simultaneously also the role of SIMA in the initialization phase of the simulation.

For the test, two instances offered by Amazon EC2 and one offered by Okeanos have been used: the two EC2 machines are instances of the type *ec2.micro* located in Ireland (named *EC2.dublin*) and in Germany (named *EC2.frankfurt*), the host originating from *Okeanos* is physically placed in Greece. For more information regarding the services and the architectures offered by the cloud providers Amazon and Okeanos, see respectively [34] and [53]. The hardware used in the test is therefore composed by:

- *ec2-instaces*: 1 vCPUs, 2.5 GHz, Intel Xeon Family, 1 GiB memory, Network performances declared: low to moderate.
- *okeanos*: 4 vCPUS, 2.1 GHz, AMD-V Family, 4 GiB memory, Network performances not specified.

In Figure 4.1, we can observe the geographical distribution of the nodes. In each host we can see the IP address (in red) and the onion address (in purple). Our SIMA is *EC2.dublin*, it is characterized by two different onion addresses: one for the SIMA and the other for its LP. The port used by the SIMA is static for both executions, as well as the hidden services ports used by the LPs. We are able to determine the paths followed by the IP packets (e.g., in Figure 4.1, they are depicted in red and we can define those paths by using applications like *traceroute*) but we can not make any assumptions for the circuits used by Tor's Onion Proxies.

4.3 Introduction to Tor's performances

Before proceeding with the simulation tests, it is worth spending a few words regarding Tor's performances and the overhead introduced by its protocols and therefore by the hops traversed in the network. Obviously we

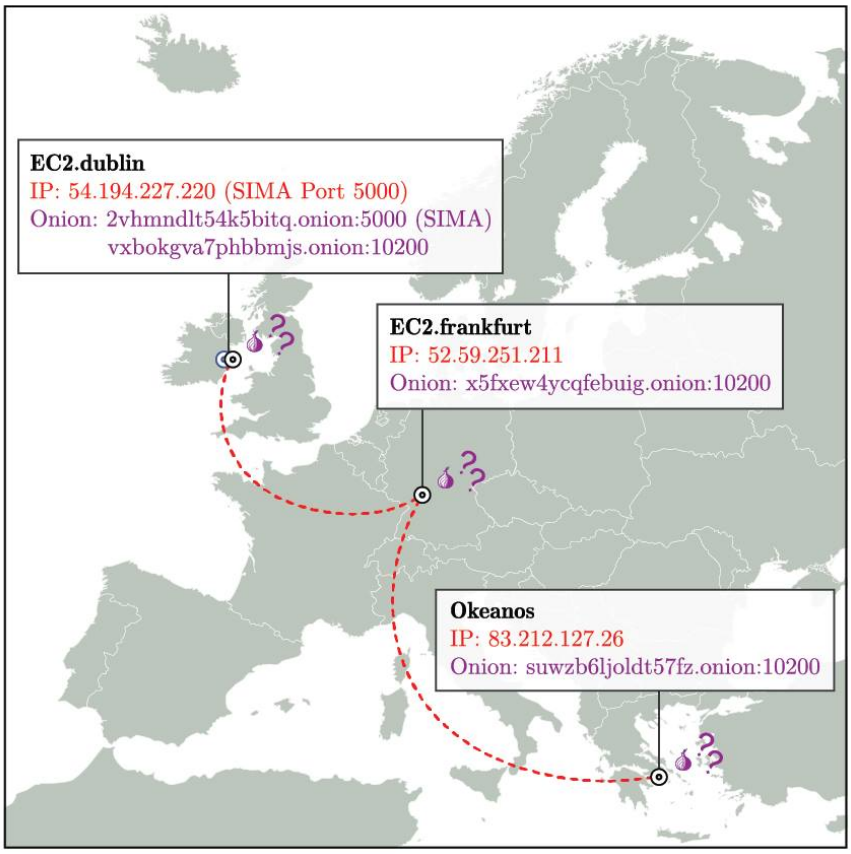

Figure 4.1: Simulation distribution of hosts used for the performance evaluation.

expect a decrease in response times with a general increment of the *RTT*, however it is necessary to quantify this and other aspects, for example the stability of Tor's network (we must remember that is formed by volunteers relay nodes, very often private hosts that could disconnect at any moment). Unfortunately in this evaluation of Tor's network we can not rely on the renowned application *"ping"* offered by the *Internet Control Message Protocol (ICMP)*[1]; however, we can test performances developing a simple software

[1] ***ICMP*** is a service protocol used in packet-switched network. It takes care to transmit information regarding failures (caused by the firsts 8 byte of IP's datagram), control information or messages between the nodes of a network. ICMP is directly encapsulated in IP, hence it is a network layer protocol and therefore it is not guaranteed the delivery

client-server which sends TCP packets to the onion address of the server, it will respond with a simple ACK. In order to have a term of comparison, I used the the *tcpping* [54] software: its purpose is to replicated the behavior of the ping applicative but at a superior layer: the transport one used by TCP.

The comparison was based on the sending of 200 packets, at intervals of three seconds, between the three hosts that compose our network.

	TCP/IP		**Tor**	
	$\overline{x}$ (ms)	σ (ms)	$\overline{x}$ (ms)	σ (ms)
EC2.dublin - Okeanos	92.91	0.75	326.42	278.52
Okeanos - EC2.frankfurt	67.14	0.16	282.74	104.83
EC2.frankfurt - EC2.dublin	20.84	0.20	540.74	54.5

Table 4.1: Performances of standard TCP/IP and Tor communications: the results are obtained with two hundred different *ping*, one every three seconds, between our cloud instances. $\overline{x}$ are the mean values instead σ are the standard deviations.

From the Table 4.1, it is possible to note difference of RTT among the various nodes. In the case of the classic TCP/IP communication, we can observe that the average time respects the geographical distance between hosts and a extremely low variability. Regarding the circuits used by Tor the situation is more articulated: we repeat it, we do not know anything about the paths that our packets will take and the geographical distance can not considered a useful parameter (e.g., it must be noted that the fastest channel via TCP/IP is the slowest with the corresponding Tor channel), we only know that the operations of encryption and decryption have a cost a we can now estimate them.

The standard deviation results must be noted: such an elevated values, with respect to the corresponding TCP/IP, are surely the result of the vari-

of packets.

ability of which Tor's network is subjected. This variability is justified by many factors: the nature of the relays which compose the network, the positions of the ORs in a specified circuit, and, extremely relevant, the fact the circuits are subjected to changing. This last factor can have considerable consequences and it implies the reconstruction of a new circuit (has we saw in 2.3.2, this entails a closing protocol that involves all of the nodes and the subsequent opening of a new circuit which implicates the exchanging D-H keys phase and so on).

To illustrate this variability, in the Figures 4.2, 4.3, 4.4, are represented the distribution of the RTT frequencies of the two hundred ping packets: it is important to underline that the collected data and the related figure have the only purpose to offer a general vision of Tor's network performances, they can not be an empirical evaluation.

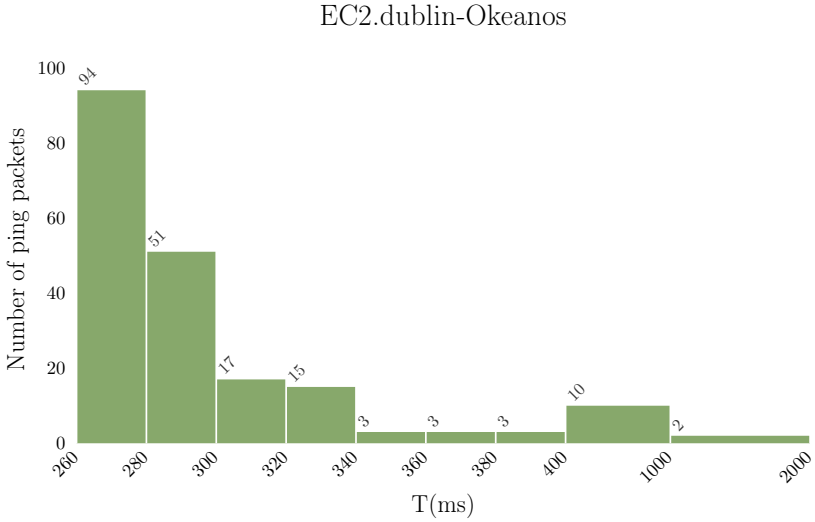

Figure 4.2: Distribution of frequency of 200 RTT of Tor's packets between EC2.dublin instance and Okeanos instance. The mean value ($\overline{x}$) is *326.42 ms* and the standard deviation (σ) is *278.52 ms*

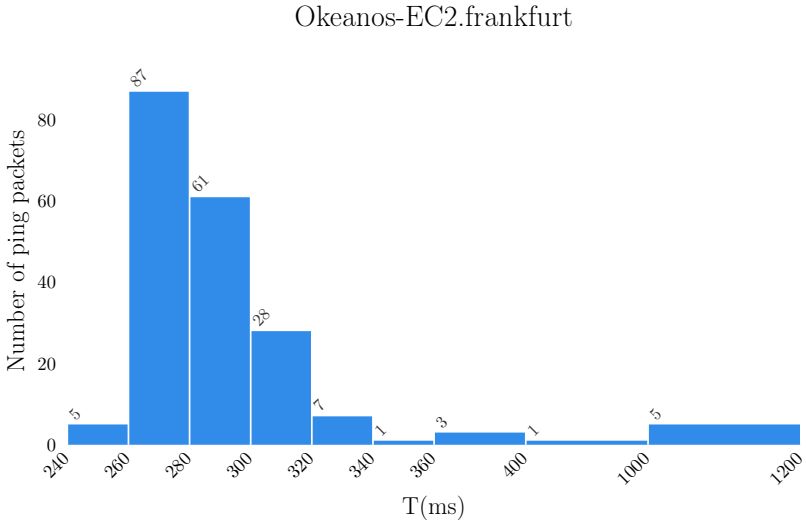

Figure 4.3: Distribution of frequency of 200 RTT of Tor's packets between Okeanos and EC2.frankfurt. The mean value $\overline{x}$ is *282.75 ms* and the standard deviation σ is *104.83 ms*.

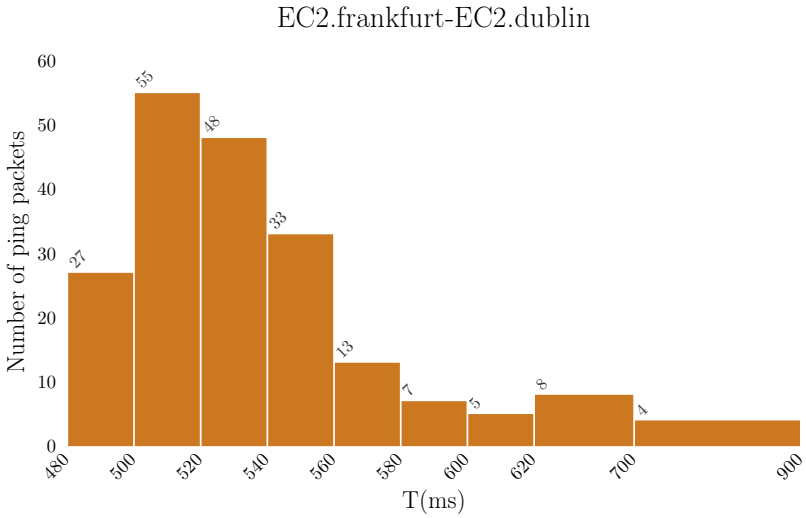

Figure 4.4: Distribution of frequency of 200 RTT of Tor's packets between EC2.frankfurt and EC2.dublin. The mean value $\overline{x}$ is *540.74 ms* and the *standard deviation* is *54.50 ms*.

It is evident that our simulations will be affected be this unpredictability, however Tor offers certain personalization of the *torrc* configuration file,

which allows us to partially contain this problem: it is possible to define a list of *entry node* (i.e., the first hop of the circuit) and a list of *exit node* (i.e., the last hop before the destination) which the OP will have to use in order to contact the hidden service recipient. It is also possible to specify only a country code in order to reduce the distance between sender and receiver. In order to have a major degree of anonymity (specifying a country code could not be the best choice in this case) we will not apply this modification, leaving unchanged the configuration file.

4.4 Execution script

To have sufficient samples to execute an exhaustive analysis, and considering the duration of certain practice run performed in phase of developing/debugging, it became necessary to automate the startup of the simulations. For this purpose a script was utilized which allowed us to define the parameters of the models in order to render them common to all the nodes, avoiding the update of every single configuration file and which would subsequently carry out multiple runs, communicating via *Secure Shell (SSH)*[2] the start of the initialization phase to SIMA and LPs. In regards to this, it is worth specifying that during the performance evaluation phase also the LPs (as well as the SIMA) maintain the same onion address (as explained in Figure 4.1): this does not affect the maintainment of anonymity but saves us the restart procedure of Tor services in each node. In Code 4.2, we can see the script utilized for the starting of N runs.

[2] ***SSH*** is a network layer protocol that permits to establish an encrypted remote session with another host of a network [55].

```
#!/bin/bash

############
RESULTS_FILE="results_tor.txt"
# LPs
LPS=3
# Total number of Simulated Entities
SE=3000
# Migration Heuristic (0 = Migration Disabled)
MIGR_HEU=3
# Speed of Simulated Entities
SPEED=10
# Communication Range of Simulated Entities
RADIUS=250.0
# Number of Runs
RUNS=10
# Migration Factor (If MIGR_HEU=0 this parameter will be ignored by models)
MF=3.00
############
# Cleaning up old results
rm -f $RESULTS_FILE

for (( run=1; run<=RUNS; run=run+1 ))
do
        echo "Running Tor: mf=$MF run=$run"
    # Launching single run
        nice -n +20 ./run_tor.sh $SE $SPEED $MIGR_HEU $MF $RADIUS $run
done
# The averager executable calculates the average WCT of the runs
./averager $RESULTS_FILE $RUNS >> $RESULTS_FILE
mv $RESULTS_FILE $RESULTS_FILE-$SE-$RUNS
# Cleaning up the old results
rm -f $RESULTS_FILE
```

Code 4.2: Example of an execution script of multiple Tor's simulation run. We can see the configuration parameters constituted by the number of Simulated Entities (SE), the Migration Heuristic used (MIGR_HEU), the speed of the simulated wireless devices (SPEED) and their radius of action (RADIUS), the number of runs (RUNS) and, finally, the migration factor (MF).

The script `run_tor.sh` (Appendix A) starts the simulations creating SSH connections towards the nodes (the terminal that executes the run knows the IP addresses of all the participants of the simulation) and it communicates to them the parameters of the model, after that it collects the statistics produced by LP Master (which as we saw in the section 4.1, maintains an extensive description of the whole simulation, step by step) via the use of *Secure Copy Protocol (SCP)* [3].

4.5 Results

Utilizing as initial reference the results and the considerations produce in [57], the initial expectations were very low. Considering what we analyzed in 4.3, in particular related to the poor stability of Tor's network we expected that a large part of the simulation runs would fail following the possible unreachability of some nodes: it is sufficient that only one host does not participate actively to fail the whole simulation.

We must remember that we are using a time-stepped approach, consequently the slowest host (or the slowest communication circuit) affects the simulation time. Following the considerations in 4.3, we can affirm that probably utilizing a time-stepped approach is the most convenient choice in the case of an anonymous simulation. The idea of using a pessimistic approach, which involves the constant sending of NULL messages, will probably bring to a degeneration of the Wall-Clock-Time; on the other hand, the use of an optimistic approach will be extremely dangerous: in our specific case we are paying for the computational resources and for the bandwidth offered by cloud providers, are we really sure that we want incur in an extremely probable *roll-back cascade*? The hypothesis that a LP must lose a lot of time to rebuild Tor's circuits is concrete, thrown away the work produced by the other two LPs is not a risk that we want to take.

[3] ***SCP*** is a protocol whose purpose is to transfer files between two hosts (both remote or one local and one remote) in a secure manner. It uses Secure Shell protocol [56].

The tests are classified in this manner:

- ***TCP/IP-Tor ALL_OFF***: the migrations performed by GAIA and the load balancing offered by GAIA+ are disabled. We are considering this simulation as *"pure"*: every node manages the same number of SE equal to SE/LPs; the simulation is therefore balanced (only numerically) in the initialization phase. We expect elevated WCT, in particular in regards to Tor's case: hypothesize, for example, that a set of distributed SE in *EC2.dublin* communicates frequently with other SE collocated in *Okeanos* forming an *interaction group*, considering the experimented RTT, this situation will bring to an explosion of the overall WCT.

- ***TCP/IP-Tor ALL_ON***: the migrations effected by GAIA are active, as well as the load balancing of GAIA+. It is legitimate to expect an improvement in performances; as stated above this improvement should be, proportionally, superior in the simulation executed with Tor, however we can not make any assumptions because of the unpredictability of the channel of communication. The SEs of the previous example will be migrated from one of the two considered LPs, always in respect of the load balancing, obtaining the clustering of the interaction groups. Regarding the context in which this performance evaluation is collocated, we will use the same *migration heuristic (Mig_Heu)* and the *migration factor (MF)* for all of the tests: we want to demonstrate that GAIA/GAIA+ are in general effective also in the case of using Tor, for the moment we are not interested in an analysis bound to the variation of that heuristic and migration factors.

  In our case, we used a *Mig_Heu* equal to 3.0 (i.e., MIGR_E3) in which the migration heuristic is evaluated only if the simulated entity has sent at least N events (where it is by default equals to 30 but it can be changed via `GAIA_SetCountDown`). This heuristic is particularly

recommended in the case of an elevate number of SEs and where the interaction pattern can rapidly change. With regards to the MF, we used a default value equal to 3.0: this value defines the relocation threshold of the heuristic, a low value increments the number of migrations while a high value reduced that number.

The results shown in the following pages were gathered through 10 different runs for each single configuration. The configurations, other than the classifications ALL_OFF e ALL_ON, are defined by a fixed number of steps (1000) and by an incremental number of SEs. In the tables indicated below are represented, in seconds, mean WCT ($\overline{x}$), relative standard deviation (σ), experimental minimum and maximum WCT and the confidence interval (CI) relative to each mean WCT (this is calculated with a level of confidence of 90%).

Configuration	$\overline{x}$ (s)	σ (s)	Min (s)	Max (s)	CI (s)
TCP/IP ALL_OFF	130.46	1.44	128.05	132.48	0.75
TCP/IP ALL_ON	106.94	3.58	102.62	114.87	1.86
Tor ALL_OFF	923.58	414.36	508.37	1,683.73	215.53
Tor ALL_ON	528.90	100.29	425.07	709.10	52.17

Table 4.2: The Wall-Clock-Times taken for the executions of the simulations with 3000 SEs

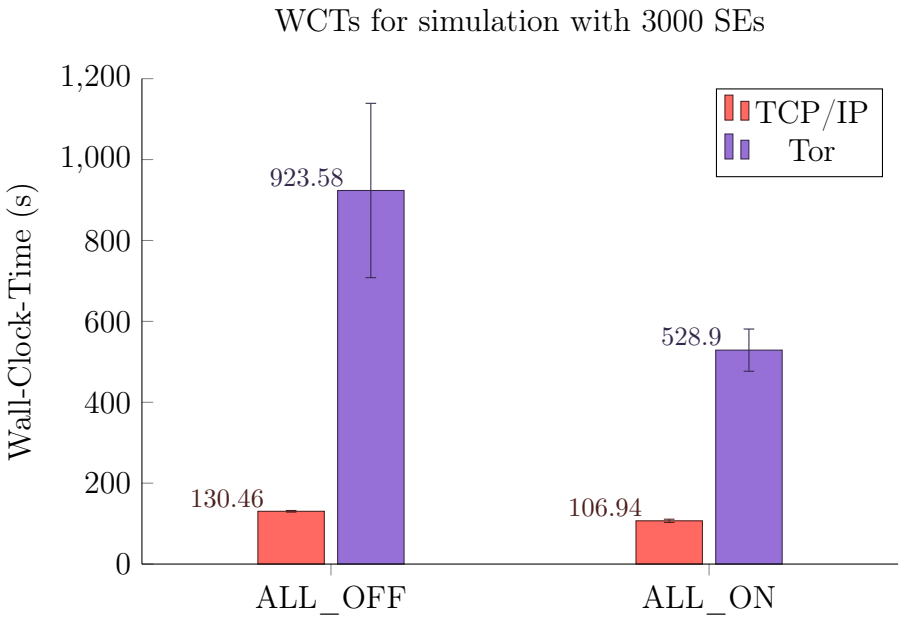

Figure 4.5: Average times to complete the simulation model with 3000 SEs and relative confidence intervals.

As we can see from the data in the Table 4.2, the anonymous simulations (without the GAIA mechanisms active) employ, in average, seven times those obtained by the "standard" simulations. This does not surprise us and, once more, what we must underline is the variability of the results acquired with Tor. The scissor between WCT minimum and maximum and the indication offered by the standard deviation confirms what we have previously stated: during the 10 sample runs, the circuits will have changed a various number

of times, decreasing (or increasing) the performance offered by the runs.

The portion of the report of the simulation shown in Figure 4.6 wants to demonstrate what happened during a run Tor ALL_OFF. In the second column we have the current WCT, in the third the relative step and in the fourth the number of SEs located in a determined LP. Observing the second one, it is implicit to understand the values of the standard deviation: certain steps of the simulation last only a few seconds, other some tens of seconds.

If we take a few a moments we can understand what happened to the LPs (probably to the slowest) during the simulation steps: we can see that the first four steps had an execution time in the order of a few seconds, subsequently we can observe that the step *557* (in line #4), was executed *52* seconds after the previous step, the following step was even slower (close to a minute). What we can speculate, considering the WCT from step *559* onwards, is that one or more LPs had a few problems of reachability: we can assume this is due to the reconstruction of the Tor's circuits.
par

```
#0          [343.92]   [553.00000]    1000
#1          [348.31]   [554.00000]    1000
#2          [349.13]   [555.00000]    1000
#3          [358.25]   [556.00000]    1000
#4          [420.33]   [557.00000]    1000
#5          [486.04]   [558.00000]    1000
#6          [500.62]   [559.00000]    1000
#7          [500.86]   [560.00000]    1000
#8          [501.51]   [561.00000]    1000
#9          [501.94]   [562.00000]    1000
```

Figure 4.6: Example of simulation results report.

The positives for the anonymous version emerge with the activation of

GAIA: the *speedup*[4] obtained, calculated as the ratio between $\overline{x}$ ALL_OFF and $\overline{x}$ ALL_ON, is equal to *1.75*. In Figure 4.5, it is visibly tangible both the reduction of the duration and, more importantly, a greater stability of the performance of the anonymous version. This is clearly motivated by the reduction of the number of messages exchanged between the various LPs which allows us a greater predictability of the time range of the simulations (one must note the considerable shortening of the interval of confidence).

However, to gather the 10 sample runs of the anonymous simulation with GAIA activated, it was necessary to execute a higher number of runs: 3 times a LP manifested an error in relation to the lack of an arbitrary number of SEs to migrate. Presently we are unable to explain the causes of these errors which are correlated to the use of GAIA in Tor's network (in no other circumstances was a simulation aborted).

Regarding the average number of global migrations and the migration ratio, we can state that the presence of Tor (as we might expect) does not have any influence (the average number of migrations with Tor was *7,582.40*, with TCP/IP *7,677.00*).

[4] ***Speedup*** is a parameter for the measuring of the performance's improvement between two systems or architecture that are processing the same problem.

Configuration	$\overline{x}$ (s)	σ (s)	Min (s)	Max (s)	CI (s)
TCP/IP ALL_OFF	527.73	26.38	465.81	554.17	13.72
TCP/IP ALL_ON	167.05	13.35	157.23	198.39	6.95
Tor ALL_OFF	1,585.78	825.88	939.30	3,403.19	429.58
Tor ALL_ON	1,021.91	202.94	735.73	1,336.46	105.56

Table 4.3: The Wall-Clock-Times taken for the executions of the simulations with 6000 SEs

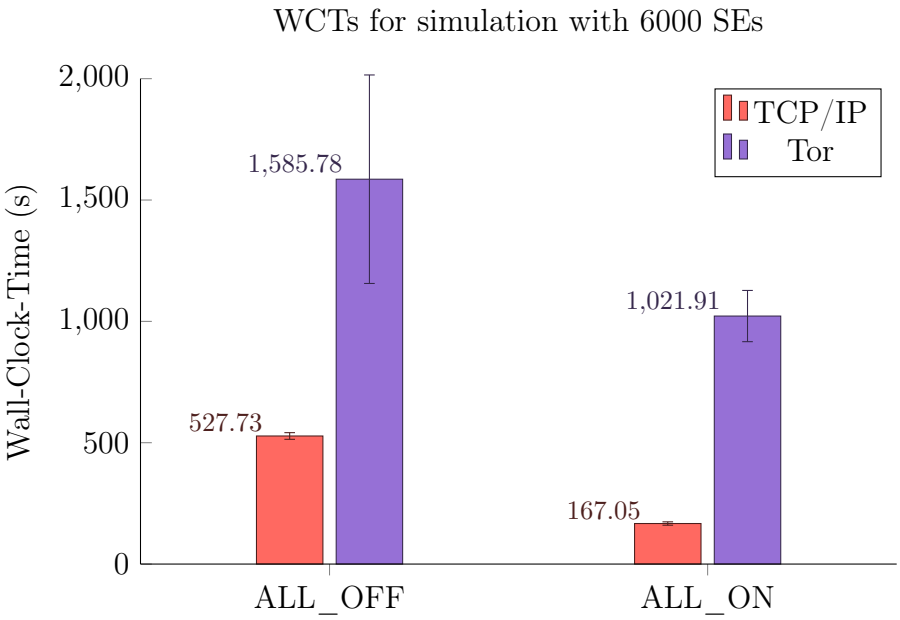

Figure 4.7: Average times to complete the simulation model with 6000 SEs and relative confidence intervals.

Even doubling the number of entities simulated, the considerations made for the set of previous runs does not change. Observing the Figure 4.7, it is evident a considerable reduction of running times for the version TCP/IP: the speedup is equal to 3.15 unlike the anonymous version which has value of 1.55. How can this reduction of the speedup comparing to the previous set of runs of 3000 SEs be explained? We can suppose that the increase of entities produces a greater interactivity in the LPs which can not be com-

pensated by clustering without damaging the load balancing active between the nodes. Consequently the increase of the use of Tor's circuits increases the computational times of the simulations. It is legitimate to believe that a further increment of the number of SEs will allow a greater reduction of the speedup.

Also in this case (anonymous communication and GAIA active) two runs were aborted for errors regarding the migration of entities.

Configuration	$\overline{x}$ (s)	σ (s)	Min (s)	Max (s)	CI (s)
TCP/IP ALL_OFF	1,155.99	57.68	1,076.49	1,220.34	30.00
TCP/IP ALL_ON	396.65	68.52	343.66	505.29	35.64
Tor ALL_OFF	2,274.58	1,010.47	1,144.28	3,821.52	525.59
Tor ALL_ON	1,720.24	279.10	1,232.30	2,072.04	145.17

Table 4.4: The Wall-Clock-Times taken for the executions of the simulations with 9000 SEs

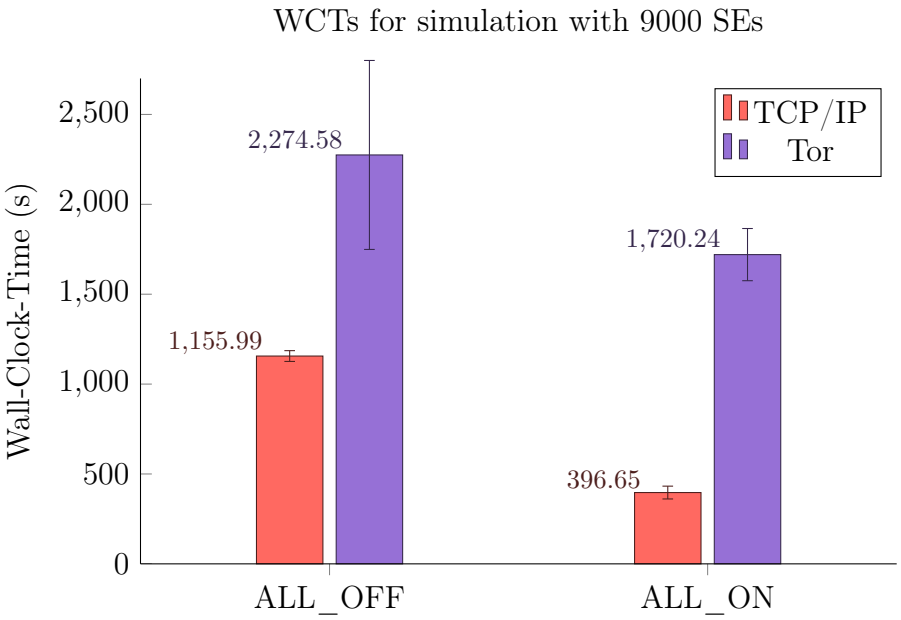

Figure 4.8: Average times to complete the simulation model with 9000 SEs and relative confidence intervals.

With this last case, we confirm our above considerations. As foreseen the speedup for the anonymous version has furthermore been reduced to *1.32*, whilst for the TCP/IP version it is *2.91*.

Even in this last set of runs two extra simulations have been performed to compensate for the problems regarding the lack of SEs migrations.

Conclusions

The main purpose of this work was to verify the possibility of realizing a distributed simulation with a total degree of anonymity, and assess its effectiveness and subsequently its efficiency. We can consider the implementation of the anonymous feature via Tor a success: the participating hosts, after a superficial packet sniffing analysis (using Wireshark software [58]), resulted anonymous, unlikely to be attacked by an external agent.

The critical point which lead this work from the beginning was the performance and the possibility of seeing the simulation times explode. Furthermore, previous studies like [57] (that work involves a higher number of nodes), highlighted the possibility of a low failure resistance: initially this lead us to believe that the stage of collecting data should use greater number of runs, with lower success rates. In this sense, we can affirm that we are surprised of the performance of ARTÍS in its anonymous version which, however variable in its WCT, managed to complete 100% of the runs. The introduction of the GAIA mechanisms bought about a percentage failure of nearly 20%: it is necessary to go into more depth on the reasons for these failures, however a possible solution is already in development and evaluation. The implementation of the software layer *GAIA-FT* provides, via replications in the various LPs, our simulations a fault tolerance: the SEs are replicated in what are known as virtual SEs. The number of replicas guarantees a grade of fault tolerance, this has an evident cost both in terms of computational resources, also in terms of WCT.

Currently it is essential to use Tor to obtain anonymous software, it is evidently the only alternative sufficiently widespread to guarantee anonymity. A short while before writing this thesis, a group of researchers published a new version of Tors protocol, operating in the network layer, which guarantees greater performance and could be the future generation of onion routing (i.e., the third): this version of Tor is called *High-speed Onion Routing NETwork (HORNET)* [59, 60] and its development and diffusion will be followed with interest in order to improve the performances analyzed in the fourth chapter.

Appendix A

Distributed execution script

In order to better understand the startup procedure of the remote simulations, illustrated below is the script initially produced by the supervisor of this MsC thesis and modified for the performance evaluation phase of this work.

```
#!/bin/bash

#######################################
# Simulation execution parameters
#######################################
#
SESTATE=32
PINGSIZE=1024
SIM_DIR=$(pwd)
#
#      Allocation of LPs on hosts
HOSTS_FILE="hosts-distributed_tor.txt"
#
#      Filename for results
RESULTS_FILE="results_tor.txt"
#
rm -f wireless_tor.ini
ln -s wireless_tor.ini.distributed wireless_tor.ini
#
#######################################
ESC="\033["
```

```
TOT_SMH=$1
SPEED=$2
MIGRATION_HEURISTIC=$3
MIGRATION_MF=$4
RADIUS=$5
SEED_POS=$6
SIM_TIME=$7

#####################################
# Cleaning up
function cleanup {
        I=0
        for h in $HOST_LIST; do
                echo "...cleaning: $h"
                ssh -p ${SSH[$I]} $h "pkill -9 sima; pkill -9 wireless"
                I=$((I+1))
        done
}
#####################################
if [ "$#" != "7" ]; then
        echo "          Incorrect syntax...         "
        echo "USAGE: $0 [#SMH] [#SPEED] [#MIGRATION_HEURISTIC] [#MIGRATION_MF] [#RADIUS
            ] [#SEED_POS] [#SIM_TIME]"
        echo ""
        exit
fi

echo -e "${ESC}29;39;1mWIRELESS SIMULATION ... ${ESC}0m"
if [ ! -f $RESULTS_FILE ]; then
        # Preparing the new results file
        touch $RESULTS_FILE

        echo "# ART\'IS/GAIA+ simulator - MIGRATION-WIRELESS model" >> $RESULTS_FILE
        echo "#" >> $RESULTS_FILE
        echo "# Running parameters: SMH=$TOT_SMH SPEED=$SPEED MIGRATION_HEURISTIC=
            $MIGRATION_HEURISTIC RADIUS=$RADIUS SEED_POS=$SEED_POS" >> $RESULTS_FILE
        echo "#" >> $RESULTS_FILE
        echo "# Columns:" >> $RESULTS_FILE
        echo "# 1 - Number of LPs" >> $RESULTS_FILE
        echo "# 2 - Migration Factor (MF)" >> $RESULTS_FILE
```

```
        echo "# 3 - Random waypoint MAX=MIN speed" >> $RESULTS_FILE
        echo "# 4 - Total number of migrations (global value)" >> $RESULTS_FILE
        echo "# 5 - Final migration ratio" >> $RESULTS_FILE
        echo "# 6 - Final average local communication ratio" >> $RESULTS_FILE
        echo "# 7 - Final average local communication ratio gain" >> $RESULTS_FILE
        echo "# 8 - Final clustering efficiency" >> $RESULTS_FILE
        echo "# 9 - Wall Clock Time" >> $RESULTS_FILE
        echo "#10 - SE state size" >> $RESULTS_FILE
        echo "#11 - PING state size" >> $RESULTS_FILE
        echo "#" >> $RESULTS_FILE
fi

echo "...parsing the LP-to-host allocation file"
POS=0
TOT_LPS=-1
while IFS='' read -r line || [[ -n "$line" ]]; do
        HOSTNAME=`echo $line | cut -f1 -d':'`
        SSHPORT=`echo $line | cut -f2 -d':'`
        PRT=`echo $line | cut -f3 -d':'`
        LP=`echo $line | cut -f4 -d':'`
        PTH=`echo $line | cut -f5 -d':'`
        ONION_ADDR=`echo $line | cut -f6 -d':'`
        HOST[POS]=$HOSTNAME
        SSH[POS]=$SSHPORT
        PORT[POS]=$PRT
        LPS[POS]=$LP
        EXE_PATH[POS]=$PTH
        ONION[POS]=$ONION_ADDR
        TOT_LPS=$((TOT_LPS+LP))
        POS=$((POS+1))
done < "$HOSTS_FILE"

HOST_LIST=`echo "${HOST[@]}" | tr ' ' '\n' | tr '\n' ' '`
SSH_PORT_LIST=`echo "${SSH[@]}" | tr ' ' '\n' | tr '\n' ' '`

# Intercept ctrl-c then cleaning
trap cleanup SIGINT

# Cleaning before starting
cleanup
```

```
# Partitioning the #SMH among the available LPs
SMH=$((TOT_SMH/TOT_LPS))

SIMA_HOST=${HOST[0]}
SIMA_SSH=${SSH[0]}
SIMA_PORT=${PORT[0]}
SIMA_PATH=${EXE_PATH[0]}

unset HOST[0]
unset SSH[0]
unset PORT[0]
unset LPS[0]
unset EXE_PATH[0]
unset ONION[0]

echo "...running SIMA on $SIMA_HOST:$SIMA_PORT (waiting for $TOT_LPS LPs)"
# SImulation MAnager (SIMA) execution
ssh -p $SIMA_SSH -n -f $SIMA_HOST "sh -c 'cd $SIMA_PATH; nohup ./sima $TOT_LPS > sima-
    $SIMA_HOST.log 2>&1 &'"

for i in `seq 1 ${#HOST[@]} `;
do
       for y in `seq 1 ${LPS[$i]}`;
       do
              echo starting "#LP:" $y/${LPS[$i]} on ${HOST[$i]}
              echo "... ${ONION[$i]} : ${PORT[$i]}"
              ssh -p ${SSH[$i]} ${HOST[$i]} "sh -c 'cd ${EXE_PATH[$i]}; nohup ./
                  wireless $TOT_LPS $SMH $SEED_POS $MIGRATION_HEURISTIC $MIGRATION_MF
                   $SPEED $RADIUS ${PORT[$i]} ${ONION[$i]} 2> $i.err'" &
       sleep 5
       done
done
echo "...waiting for LPs termination"
wait
echo "...collecting statistics"
for i in `seq 1 ${#HOST[@]} `;
do
       if ssh -p ${SSH[$i]} ${HOST[$i]} stat ${EXE_PATH[$i]}/0.out \> /dev/null 2\>\&1
              then
                  scp -P ${SSH[$i]} ${HOST[$i]}:${EXE_PATH[$i]}/0.out $SIM_DIR/tmp-0.
                      out
```

```
                ssh -p ${SSH[$i]} ${HOST[$i]} "sh -c 'rm ${EXE_PATH[$i]}/0.out'"
            else
                ssh -p ${SSH[$i]} ${HOST[$i]} "sh -c 'rm ${EXE_PATH[$i]}/*.out'"
        fi
done
# Collecting statistics
MIGRATIONS=`cat tmp-0.out | grep "Total number of migrations" | cut -d":" -f 2`
MIGRATION_RATIO=`cat tmp-0.out | grep "Final migration ratio:" | cut -d":" -f 2`
LCR=`cat tmp-0.out | grep "Final average local communication ratio:" | cut -d":" -f 2`
LCR_GAIN=`cat tmp-0.out | grep "Final average local communication ratio gain:" | cut -
    d":" -f 2`
CLR_EFF=`cat tmp-0.out | grep "Final clustering efficiency:" | cut -d":" -f 2`
WCT=`cat tmp-0.out | grep "Elapsed time" | cut -d":" -f 2`

echo -ne "$TOT_LPS\t" >> $RESULTS_FILE
echo -ne "$MIGRATION_MF\t" >> $RESULTS_FILE
echo -ne "$SPEED\t" >> $RESULTS_FILE
echo -ne "$MIGRATIONS\t" >> $RESULTS_FILE
echo -ne "$MIGRATION_RATIO\t" >> $RESULTS_FILE
echo -ne "$LCR\t" >> $RESULTS_FILE
echo -ne "$LCR_GAIN\t" >> $RESULTS_FILE
echo -ne "$CLR_EFF\t" >> $RESULTS_FILE
echo -ne "$WCT\t" >> $RESULTS_FILE
echo -ne "$SESTATE\t" >> $RESULTS_FILE
echo "$PINGSIZE" >> $RESULTS_FILE

rm $SIM_DIR/tmp-0.out
echo "...simulation done"
```

Bibliography

[1] Steven Levy. How the nsa almost killed the internet, 2014. URL http://www.wired.com/2014/01/how-the-us-almost-killed-the-internet/. [Online; accessed 25-January-2016].

[2] Sharon Weinberger. Computer security: Is this the start of cyberwarfare?, 2011. URL http://www.nature.com/news/2011/110608/full/474142a.html. [Online; accessed 10-January-2016.

[3] RT.com. Snowden: Cyber war more damaging to us than any other nation, 2015. URL https://www.rt.com/usa/221031-snowden-cyber-warfare-threat-usa/. [Online; accessed 10-January-2016.

[4] The Tor Project Inc. Tor: anonymity online, 2006. URL https://www.torproject.org/.

[5] Merriam-webster.com. Definition of simulation, 2016. URL "http://www.merriam-webster.com/dictionary/simulation".

[6] Wikipedia. Buffon's needle — wikipedia, the free encyclopedia, 2015. URL https://en.wikipedia.org/w/index.php?title=Buffon%27s_needle&oldid=671661324. [Online; accessed 18-January-2016].

[7] David Goldsman, Richard E. Nance, and James R. Wilson. A brief history of simulation revisited. In *Proceedings of the Winter Simulation Conference*, WSC '10, pages 567–574. Winter Simulation Conference,

2010. ISBN 978-1-4244-9864-2. URL http://dl.acm.org/citation.cfm?id=2433508.2433574.

[8] L. Bononi M. Bracuto L. Donatiello, G. D'Angelo. Pads: Parallel and distributed simulation research group, 2005. URL http://pads.cs.unibo.it/. University of Bologna.

[9] Luciano Bononi, Michele Bracuto, Gabriele D'Angelo, and Lorenzo Donatiello. Scalable and efficient parallel and distributed simulation of complex, dynamic and mobile systems. In *Proceedings of the 2005 Workshop on Techniques, Methodologies and Tools for Performance Evaluation of Complex Systems*, Washington, DC, USA, 2005. IEEE Computer Society. ISBN 0-7695-2447-8. doi: 10.1109/FIRB-PERF.2005.17. URL http://portal.acm.org/citation.cfm?id=1114282.1114476.

[10] W David Kelton and Averill M Law. *Simulation modeling and analysis*. McGraw Hill Boston, 2000.

[11] Markus Diesmann. The road to brain-scale simulations on K. *BioSupercomputing Newsletter*, 8:8, 2013. URL http://www.csrp.riken.jp/BSNewsLetters/BSNvol8-1303/EN/report03.html.

[12] Stewart Robinson. *Simulation: the practice of model development and use*. Palgrave Macmillan, 2014.

[13] Robert Geoffrey Coyle. *System dynamics modelling: a practical approach*, volume 1. CRC Press, 1996.

[14] John D Sterman. *Business dynamics: systems thinking and modeling for a complex world*, volume 19. Irwin/McGraw-Hill Boston, 2000.

[15] Charles M Macal and Michael J North. Tutorial on agent-based modelling and simulation. *Journal of simulation*, 4(3):151–162, 2010.

[16] Jon Parker and Joshua M Epstein. A distributed platform for global-scale agent-based models of disease transmission. *ACM Transactions on Modeling and Computer Simulation (TOMACS)*, 22(1):2, 2011.

[17] J Banks, JS Carson, and BL Nelson. *DM Nicol, Discrete-Event System Simulation*. Prentice hall Englewood Cliffs, NJ, USA, 2000.

[18] Susan K Heath, Arnold Buss, Sally C Brailsford, and Charles M Macal. Cross-paradigm simulation modeling: challenges and successes. In *Proceedings of the Winter Simulation Conference*, pages 2788–2802. Winter Simulation Conference, 2011.

[19] Gabriele D'Angelo. Parallel and distributed simulation from many cores to the public cloud. In *High Performance Computing and Simulation (HPCS), 2011 International Conference on*, pages 14–23. IEEE, 2011.

[20] Richard M Fujimoto. *Parallel and distributed simulation systems*, volume 300. Wiley New York, 2000.

[21] O. Dalle. On reproducibility and traceability of simulations. In *Simulation Conference (WSC), Proceedings of the 2012 Winter*, pages 1–12, Dec 2012. doi: 10.1109/WSC.2012.6465284.

[22] Leslie Lamport. Time, clocks, and the ordering of events in a distributed system. *Commun. ACM*, 21(7):558–565, July 1978. ISSN 0001-0782. doi: 10.1145/359545.359563. URL `http://doi.acm.org/10.1145/359545.359563`.

[23] Esteban Egea López. Simulation scalability issues in wireless sensor networks. 2006.

[24] M. Mitchell Waldrop. The chips are down for moore's law.

[25] Richard M. Fujimoto. Parallel discrete event simulation. *Commun. ACM*, 33(10):30–53, October 1990. ISSN 0001-0782. doi: 10.1145/84537.84545. URL `http://doi.acm.org/10.1145/84537.84545`.

[26] Kalyan S. Perumalla. Parallel and distributed simulation: Traditional techniques and recent advances. In *Proceedings of the 38th Conference on Winter Simulation*, WSC '06, pages 84–95. Winter Simulation Conference, 2006. ISBN 1-4244-0501-7. URL http://dl.acm.org/citation.cfm?id=1218112.1218132.

[27] M. Hybinette and R.M. Fujimoto. Scalability of parallel simulation cloning. In *Simulation Symposium, 2002. Proceedings. 35th Annual*, pages 275–282, April 2002. doi: 10.1109/SIMSYM.2002.1000164.

[28] Mikel D Petty and Eric W Weisel. A composability lexicon.

[29] R.E. De Grande and A. Boukerche. Dynamic partitioning of distributed virtual simulations for reducing communication load. In *Haptic Audio visual Environments and Games, 2009. HAVE 2009. IEEE International Workshop on*, pages 176–181, Nov 2009. doi: 10.1109/HAVE.2009.5356113.

[30] Yu Jun, Come Raczy, and Gary Tan. Evaluation of a sort-based matching algorithm for ddm. In *Proceedings of the sixteenth workshop on Parallel and distributed simulation*, pages 68–75. IEEE Computer Society, 2002.

[31] Gabriele D'Angelo and Moreno Marzolla. New trends in parallel and distributed simulation: From many-cores to cloud computing. *Simulation Modelling Practice and Theory*, 49:320–335, 2014.

[32] David R. Jefferson. Virtual time. *ACM Trans. Program. Lang. Syst.*, 7 (3):404–425, July 1985. ISSN 0164-0925. doi: 10.1145/3916.3988. URL http://doi.acm.org/10.1145/3916.3988.

[33] Peter Mell and Tim Grance. The nist definition of cloud computing. 2011.

[34] Amazon. Elastic cloud compute. URL https://aws.amazon.com/ec2.

[35] David Nicol and Richard Fujimoto. Parallel simulation today. *Annals of Operations Research*, 53(1):249–285, 1994.

[36] Michael Wooldridge. *An introduction to multiagent systems*. John Wiley & Sons, 2009.

[37] IEEE Standard for Modeling and Simulation (M&S) High Level Architecture (HLA) - Framework and Rules. *IEEE Std. 1516-2000*, pages i –22, 2000. doi: 10.1109/IEEESTD.2000.92296.

[38] Gabriele D'Angelo and Michele Bracuto. Distributed simulation of large-scale and detailed models. *International Journal of Simulation and Process Modelling*, 5(2):120–131, 2009.

[39] Luciano Bononi, Michele Bracuto, Gabriele D'Angelo, and Lorenzo Donatiello. *Frontiers of High Performance Computing and Networking – ISPA 2006 Workshops: ISPA 2006 International Workshops, FHPCN, XHPC, S-GRACE, GridGIS, HPC-GTP, PDCE, ParDMCom, WOMP, ISDF, and UPWN, Sorrento, Italy, December 4-7, 2006. Proceedings*, chapter An Adaptive Load Balancing Middleware for Distributed Simulation, pages 873–883. Springer Berlin Heidelberg, Berlin, Heidelberg, 2006. ISBN 978-3-540-49862-9. doi: 10.1007/11942634_89. URL http://dx.doi.org/10.1007/11942634_89.

[40] Supreme Court of United States. Mcintyre v. ohio elections comm'n (93-986), 514 u.s. 334, 1995. URL https://www.law.cornell.edu/supct/html/93-986.ZO.html. [Online; accessed 25-January-2016].

[41] Stallings William. *Computer Security: Principles And Practice*. Pearson Education, 2011.

[42] Wikipedia. Silk road (marketplace) — wikipedia, the free encyclopedia, 2016. URL \url{https://en.wikipedia.org/w/index.php?title=Silk_Road_(marketplace)&oldid=702803157}. [Online; accessed 25-January-2016].

[43] The Tor Project Inc. Tor metrics - direct users by country, 2016. URL https://metrics.torproject.org/userstats-relay-country.html.

[44] David M. Goldschlag, Michael G. Reed, and Paul F. Syverson. Hiding routing information. In *Proceedings of the First International Workshop on Information Hiding*, pages 137–150, London, UK, UK, 1996. Springer-Verlag. ISBN 3-540-61996-8. URL http://dl.acm.org/citation.cfm?id=647594.731526.

[45] Wikipedia. Onion routing - wikipedia, 2016. URL \url{https://en.wikipedia.org/w/index.php?title=Onion_routing&oldid=699545059}. [Online; accessed 28-January-2016].

[46] Roger Dingledine, Nick Mathewson, and Paul Syverson. Tor: The second-generation onion router. In *Proceedings of the 13th Conference on USENIX Security Symposium - Volume 13*, SSYM'04, pages 21–21, Berkeley, CA, USA, 2004. USENIX Association. URL http://dl.acm.org/citation.cfm?id=1251375.1251396.

[47] Luciano Bononi, Michele Bracuto, Gabriele D'Angelo, and Lorenzo Donatiello. Artis: a parallel and distributed simulation middleware for performance evaluation. In *Computer and Information Sciences-ISCIS 2004*, pages 627–637. Springer, 2004.

[48] Souvik Ray. *Design and analysis of anonymous communications for emerging applications*. ProQuest, 2008.

[49] The Tor Project Inc. Tor project faq: Socks and dns information leaks. should i worry?, 2006. URL https://www.torproject.org/docs/faq.html.en#WarningsAboutSOCKSandDNSInformationLeaks.

[50] Michael J Donahoo and Kenneth L Calvert. *TCP/IP sockets in C: practical guide for programmers*. Morgan Kaufmann, 2009.

[51] Wikipedia. Random waypoint model — wikipedia, the free encyclopedia, 2015. URL https://en.wikipedia.org/w/index.php?title=Random_waypoint_model. [Online; accessed 14-February-2016].

[52] Parallel and Distributed Simulation Research Group. Gaia apis, 2011. URL http://pads.cs.unibo.it/doku.php?id=pads:gaia-apis. [Online; accessed 6-February-2016].

[53] Okeanos. Iaas service. URL https://okeanos.grnet.gr/home/.

[54] Richard van den Berg <richard@vdberg.org>. tcpping: test response times using tcp syn packets. URL http://www.vdberg.org/~richard/tcpping.

[55] Wikipedia. Secure shell (ssh), 2016. URL https://en.wikipedia.org/wiki/Secure_Shell.

[56] Wikipedia. Secure copy protocol (scp), 2016. URL https://en.wikipedia.org/wiki/Secure_copy.

[57] Michele Amati. Design and implementation of an anonymous peer-to-peer iaas cloud, 2015. URL http://amslaurea.unibo.it/8426/1/amati_michele_tesi.pdf.

[58] The Wireshark team. Wireshark, 2015. URL https://www.wireshark.org/.

[59] Chen Chen, Daniele E Asoni, David Barrera, George Danezis, and Adrain Perrig. Hornet: high-speed onion routing at the network layer. In *Proceedings of the 22nd ACM SIGSAC Conference on Computer and Communications Security*, pages 1441–1454. ACM, 2016.

[60] Sean Gallagher. Researchers claim they've developed a better, faster tor, 2015. URL http://arstechnica.com/information-technology/2015/07/researchers-claim-theyve-developed-a-better-faster-tor/. [Online; accessed 16-February-2016].