Experiences with Synthetic Network Emulation for Complex IP based Networks

Stefano Cacciaguerra

Technical Report UBLCS-2005-04

March 2005

Department of Computer Science

University of Bologna

Mura Anteo Zamboni 7
40127 Bologna (Italy)

Dottorato di Ricerca in Informatica
Università di Bologna, Padova

Experiences with Synthetic Network Emulation for Complex IP based Networks

Stefano Cacciaguerra

March 2005

Coordinatore:	Tutore:
Prof. Özalp Babaoğlu	Prof. Marco Roccetti

Abstract

The development of communication technologies has allowed the proliferation of network applications that permit an increasing number of users, geographically distributed throughout the world, to access complex IP-based networks through multiple devices (for example, networked computers, smart phones and personal digital assistants). In the effort to test and evaluate the correctness and the performance of these new applications, many of them must be executed into complex IP-based networks (like in Internet). Complex IP-based networks integrate very different communication technologies and administrative domains supporting a high number of users. Hence, there is no typical network infrastructure but a range of possible ones. Therefore, it is difficult to build a controllable environment that supports the execution of several concurrent applications while being able, at the same time, to mimic a wide range of (IP-based) network configurations. To implement a scenario involving complex IP-based networks, three traditional approaches are possible: testbed, network simulation, or network emulation.

Testbed provides a field test over a real network using real applications. Unfortunately, it is linked to specific network conditions and traffic dynamics. Further, a testbed hardly supplies a fully controllable environment where these applications may be evaluated at an affordable cost. Synthetic Network Simulation, instead, provides closed, controllable environments and reproducible experiments. Unfortunately, it does not support a real implementation. As a consequence, a transcoding activity is necessary. Further, this code runs on simulated protocol module whose assumptions may not always be valid in real networks. Finally, Network Emulation uses the real implementation of protocols and applications to introduce live traffic into a simple, controllable and reproducible environment. Unfortunately, the typical capabilities of emulators are limited to simple packet manipulations. Further, network emulator must support real time constraints.

Our contribution is the design and the implementation of a “synthetic” emulative platform (called Interceptor) that reproduces complex IP-based Networks. Our emulative platform allows to integrate live traffic coming from real end systems into synthetic simulative environments. Hence, it is possible to support the execution of real applications, while simulating complex IP-based networks at a higher abstraction level. Therefore, Interceptor allows to arbitrarily interpose a number of simulated network nodes (i.e. routers) and links between (real) end systems. Finally, external workload traffic generators may be used to exercise both the end systems and the synthetic emulative networks in a manner which is representative of a user that runs an application.

Interceptor provides many advantages. It provides means for setting network topological configurations (nodes and links) varying from simple to sophisticated. It reproduces a wide range of network communication models from single link to general end-to-end communication. It supplies the possibility to change and update the simulation model without affecting the network interface. It provides the possibility to interact with nodes and links at run-time. It connects the end systems without working on a default gateway. It is transparent to the final user. It reports useful statistics about its performance.

In particular, to highlight the usefulness and the performance of our “synthetic” emulative platform three case studies and the comparison with related works are shown. The three case studies show the effectiveness of Interceptor to evaluate network applications on complex IP-based scenarios. Its comparison with the Dummynet network emulator and with the NS-2 network simulator shows its usefulness.

Acknowledgments

I would like to thank my tutors, Professor Marco Roccetti, for its support throughout my research efforts towards this thesis and their helpful discussions on the idea in the work. It has been a very important example for me.

I would like to thank Professor Giorgio Ventre and Dr. Alok Nadan for theirs appreciated comments and suggestions.

A special thank to my parents and my brother for the constant help.

Last but not least, many thank to my friends, Alessandro, Giovanni, Matteo, Paola, Stefano, Vittorio, and my girlfriend Teresa.

Contents

List of Tables

List of Figures

Chapter 1

Introduction

The Internet world is composed by several millions of computers, controlled by millions of individuals and organizations. The core of this inter-network is administered by thousands of competing operators, and it spans the whole globe, connected by fibers, satellites, radio-links, dial-up modems, and mobile infrastructures. The development of these large-scale communication infrastructures has enabled the proliferation of network applications that provide an increasing number of users, geographically dispersed throughout the world, with the access to a vast amount of informations. Those network applications can be implemented following different architectural approaches, such as, the client-server, the peer-to-peer (P2P) and fully distributed. Relevant examples of emerging services provided by those applications are: videotelephony, teleconferencing, instant multimedia messaging, teleworking, telemedicine, real time entertainment, interactive games and infotainment. This means that a lot of network applications have been recently developed almost flooding the Internet with their generated traffics. The different behaviors of all these different applications are able to influence the design and the implementation of both future infrastructures and provided services, in an endless cycle (Floyd *et al.*, 2001). An important example of it is represented by Napster. The planetary success of this distributed tool for music sharing has exponentially increased the amount of MP3-based traffic generated by its users (Coffman *et al.*, 2001). In essence distributed applications like Napster are radically accelerating the demand for larger bandwidth. This is only a prominent example beneath which several other network applications like that generate similar problems. For this reason, it emerges as mandatory to develop sophisticated tools that allow one to *evaluate* any kind of modern network applications prior to its deployment. This thesis provides a support for the study of the network applications trying to reproduce in a lab-based environment the main features of complex IP-based networks.

1.1 Problem Statement

To reproduce the behavior of complex IP-based networks, like the Internet, we need to understand their main characteristics. Floyd and Paxson say that the Internet is "an immense moving target" with several main properties that make it hard to reproduce (Floyd *et al.*, 2001). All these properties may be summarized as follows.

As a first fact, the Internet has an enormous capability to unify different network technologies and administrative domains. However, the main problem in reproducing an environment like the Internet is that the IP protocol guarantees the connectivity between different network technologies and different administrative domains, but it does not make provide them with similar behaviors (Floyd *et al.*, 2001). Secondly, the Internet has a very large size. This produces two main effects. The range of possible protocol heterogeneity is very large. This means that if only a little part of the Internet has a particular behavior, thousands of computers behave in that way. Further, the scalability of many network protocols and mechanisms are not guaranteed on a *priori* basis. Many of them tend to work fine with hundreds or thousands of computers, but they fail if the scale grows up. Internet changes continuously over time. The number of connections increases at an ever-accelerating rate (LBNL, 2001). This brings to a further problem, as new and different kinds of datalink technologies must be integrated. The most prominent example in this sense is the revolutionary advent of the wireless communication technologies such as Wi-Fi, Bluetooth, and UWB. These new technologies have a further important consequence: the traffic dynamics on the Internet may change due to the need of accommodating novel (unstable) communication environments (take for example the wireless environments).

Traditional approaches exploited to access the effectiveness as well as the performance of a network application in a synthetic environment typically follow three different ways: testbed development, network synthetic simulation and network emulation. The idea of developing a *testbed* is based on two basic assumptions. First, a real implementation of the protocol (or of the application) to be experimented is available, and, second, a real networked environment exists where the protocol (or the application) may be experimented under the full control of the experimenters. The main problem with such an experimental environment is that it may produce incomplete results, as it may be bound to some given network conditions (and the corresponding traffic dynamics) during the test. Further, a testbed hardly supplies a fully controllable environment where experiments may be reproduced on a large scale at an affordable cost. Finally, a testbed can be used only to explore the existing Internet, but it does not provide a support to explore the future of the Internet. *Network synthetic simulators* provide a controllable environment and reproduce experiments at an affordable cost. Network synthetic simulation is probably the best way to follow in order to study the behavior of an application when its real implementation does not exist. However, the typical problem with *synthetic* simulators is that they do not support real protocol implementation and their direct execution. In this case, the simulationists should decide which protocol characteristics can be included in their study and which can be ignored, thus creating a simulative prototype of a new network protocol (or application) of interest. This is essentially a transcoding activity based on the idea of scaling down the level of abstraction of simulation. Hence, the obtained simulative code may fail to reproduce the behavior of the protocol (or the application) due to either possible transcoding bugs or model simplifications that may lead to inconsistent results. In this respect, there is a seminal work by Brakmo and Peterson (Brakmo *et al.*, 1996) showing that even if an abstract specification of the TCP transcode can be mostly useful for rapid experimentation, however running the actual TCP code must be preferred if errors have to be avoided. *Network emulation*, instead, uses a real implementation of a protocol (of an application) to introduce its generated live traffic into a controllable and reproducible simple environment. The typical capabilities of emulators are generally limited to simple packet manipulations and do not cause interference from simulated cross traffic. Hence, the main disadvantages of pure emulation are three. First, with the emulation we need as many workstations as the instances of the protocol to be emulated. Second, it is not possible to emulate a host or a link, which is faster than its hardware. Third, the real time behavior of the software is tied to the available specific hardware.

All these considerations suggest that the two possible alternative solutions for protocol experimentation amount to either i) exploiting a network simulator equipped with an emulative interface or ii) expanding the capabilities of a network emulator by means of synthetic simulative environment. Whichever alternative solution is followed means providing a dynamic support for managing real protocol implementations, complex traffic patterns and complicated network topologies.

We have already commented on the difficulty to support real protocol implementations, as to the management of real traffic patterns, instead, it is worth mentioning that the production of a sort of traffic mix composed by a range of different applications is needed to simulate a real scenario. There is hope that a trace-driven approach may be the best way to capture the full range of traffic mix patterns if one could collect enough different traces. However, one of the main problems with this approach is that a lot of real protocol implementations use adaptive congestion control mechanism. As a result of it, the traffic modeling based on trace capturing may produce inaccurate results if these traces are used to simulate different contexts where adaptation mechanism does not come into scene. This does not mean that traffic modeling based on measurements is fruitless, but a trace-driven source level simulation must be preferred (Barford *et al.*, 1998). In simple words, we may conclude that traffic modeling based on measurements produce accurate results only when traces are used to simulate a given shape of traffic, which resembles, at the application (source) level, to a scenario similar to that one where traffic capturing is preferred. Hence, the critical point is that a reasonable solution to the reproduction of a traffic mix must consider a spectrum of scenarios, rather than one scenario in particular. Unfortunately, this last consideration increases the load of work required for a single simulation.

Finally, as to the reproduction of complex network topologies, we must recognize this is a very serious problem. In fact, there is no typical Internet infrastructure but a range of possible typical network topologies may exists. This fact has promoted in the past the development of customized network topology generators that, however, are typically able to explore only a single property, like locality or hierarchy for example, of the current structure of the Internet. Further, the network topology is administered by several competing entities. Each entity does not want to provide topological information to the others. This fact does not permit to have a global point of view of that network topology, in general.

As for the choice between a simulator with an emulative interface and an emulator with synthetic simulative environment, it is worth pointing out the following. In simple words, two minutes of simulation should take only two minutes of real time. It is possible to keep this constrain only if we control the performance of the framework. When simulators are equipped with an emulation interface the main problem is the difficulty to perform an accurate simulation while meeting the real time deadlines imposed by the external live traffic injection into simulative environment. Instead, when an emulator is equipped with synthetic simulative technologies, the main problem is that this kind of platform must embed a software tool able to generate simulated traffic, as well as to specify complex network topologies. A few attempts adopt a traffic generator to reproduce traffic patterns over arbitrarily complex topology scenario where the emulators run. Obviously this scenario requires some sophisticated solutions to orchestrate the traffic generation with the concurrent run of several distributed emulators.

1.2 Thesis Contributions

In response to above-mentioned problems, we have designed and developed an emulative platform (called *Interceptor*), which incorporates a synthetic simulative environment. Our emulative platform allows to integrate real end systems (and their applications) into the synthetic simulative environment. Interceptor interposes a coarse-grained synthetic environment as an intermediate among real sources of network traffic. Using this approach, it is possible to support the real execution of network applications on real end systems (on IP protocol), while simulating at a higher abstraction level complex IP-based networks (Roccetti et al, 02a). In particular, we are able to support the real execution of network applications, while maintaining the network communication interface unchanged. This means that we can confine in a lab-based environment such real end systems, while simulating a large-scale distributed setting with appropriate network parameters. In particular, Interceptor embodies a network topology manager, which may be used to interpose a set of arbitrarily connected network nodes (i.e. simulated routers) between the (real) source and its destination. Using our network topology manager one is able to specify an arbitrarily complex topological scenario that may be dynamically modified. Finally, Interceptor exploits an external workload traffic generator that we have developed to exercise both the end systems and the synthetic complex IP-based networks in a manner representative of more users that run the same application. In particular, Interceptor is composed by the four following software components:

- an emulative interface, that exchanges IP packets with real end systems,
- an event manager that schedules events (related to IP packets) based on a specified simulation model,
- a network topology manager,
- and a workload traffic generator.

The most noteworthy aspect of our contribution amounts to the fact that to the best of our knowledge, no emulator exists that is able to address altogether the issues mentioned above. Below we briefly discuss each single software component and emphasize its novelty.

The *Emulative Interface* is exploited to introduce live traffic into the synthetic environment and then, to inject the resulting traffic from the synthetic environment into the live network. The main characteristic of our emulative interface is that it manages traffic injection by expressly using the “standard filter channels” of the Linux operating system (2.4.0 Kernel Family). Indeed, our emulation interface has been developed by using IPTABLES over Netfilter framework, whose system-calls permit to fetch packets from the network level (Kernel-mode) to the simulation level (User-mode). In other words, a filter channel allows to create a tunnel that passes through the emulator and connects the source node to its destination without the need for a default gateway installed on a machine the Interceptor runs on. There are several advantages deriving from this approach. First, as only standard (OS-based) filter channels are exploited, neither the contents of IP packets need to be converted into a particular format, nor there is the necessity to pass through non standard-filter channels to reach the simulation environment. Second, as there is no need to work with the Interceptor in the role of default gateway, then emulative experiment can be conducted in a (public) laboratory without any interference on any other interconnected machine.

The *Event Manager* implements the event driven scheme to advance a simulation experiment. Four different types of event are managed, namely: *arrival*, *departure*, *simulated-arrival* and *simulated-departure*. The management of the arrival and departure event is concerned with the operation performed by the emulation interface. Instead, the management of the simulated-arrival and simulated-departure event is concerned with the operation performed by the Network Topology Manager. To properly manage the events, the event manager is able to embed different communication models that may be used to delay IP packets according to a predefined delay distribution. The embedded simulation models may be developed according to a coarse-grained approach. Coarse-grained communication models permit to abstract the simulation at a level, which provides a good trade off between the accuracy of the experiments and its ability to perform in a real time fashion. In this way, it is possible to reproduce a wide range of network communication models varying from simple to sophisticated according to the wanted accuracy. This is a very critical point when emulative platforms are developed, as the accuracy of the simulation results must be guaranteed while preserving the real time constrains imposed by the emulation. To better address this kind of problems, we have also developed a special mechanism devoted to perform a "lookahead strategy" for the prefetching of imminent events in the simulation queue. Finally, our event manager embodies a mechanism, which is able to generate internal IP background traffic. This mechanism simulates the effect of the presence of other packets inside a link without generating them. In this way, we pay no cost for the computation of these ones.

The *Network Topology Manager* provides a support to specify a modeling of complex network topology and, to decide which simulative communication model can be adopted in a given experiment. The Network Topology Manager may be used to specify the number and the type of each simulated node and link. The interaction between the

Network Topology Manager and the Event Manager are governed by means of the simulated-arrival and the simulated-departure event. The most prominent characteristic of our Network Topology Manager is that it gives users the possibility to manipulate both simulated nodes and links at run-time. At run-time, indeed, the Network Topology Manager permits any manipulation of links including the change of: i) the routing table of any node, ii) the simulative delay distribution between two nodes, and finally iii) the network topology (without the possibility to create new nodes). Obviously, new nodes may be added into the simulative scenario at the compile time. Once a given network topology has been specified the procedural behavior of each active component of the network may be specified in a rigorous way by embedding in the simulation model the software code that implements it. This has been made possible by integrating into the Interceptor a C++ based programming interface based on a multi agent system, called SPADES.

The *Workload Traffic Generator* has been developed to reproduce higher numbers of users running simultaneously the same application. The goal in developing our workload generator was to exercise final end systems (i.e. final destination for the generated application traffic) and synthetic simulative networks in a manner representative of more users that run the same application. As several machines replicate, each one could not run a copy of the same application (since it would not be practical). Therefore, we have resorted to the ideas suggested in (Barford *et al.*, 1998; Jin *et al.*, 2001). The SURGE (Scalable URL Reference Generator) (Barford *et al.*, 1998) generates references matching empirical measurements of server file size distribution, request size distribution, relative file popularity, embedded file references, temporal locality of reference, and idle periods of individual users. Moreover, GISMO (Generator of Internet Streaming Media Objects and workloads) (Jin *et al.*, 2001) enables the specification of a number of streaming media access characteristics, including object popularity, temporal correlation of request, seasonal access patterns, user session durations, user inter-activity times, and variable bit-rate self-similarity and marginal distributions. In essence, following these ideas, we developed a realistic workload generator which mimics a set of real users running the same application on the same machine, permitting to the Interceptor to move up to several tens of virtual users who simultaneously exploit the same (real) application on the same machine, in emulation mode.

Summarizing, our emulative platform is able to reproduce complex IP-based network scenarios based on a coarse-grained approach, where the performance of the simulation is favored without sacrificing the accuracy of modeling (Breslau et al, 00). Indeed we choose to specify the procedural/functional (e.g. network level protocol) behavior of only the most important network components of a given simulation experiment. Instead, for known-important components of the simulation only the duration of the corresponding protocol activity is described according to a predefined delay distributional model. As we are able to create synthetic network emulation, Interceptor accepts live traffic coming from real application and introduces it transparently in the synthetic simulative environment. A set of sophisticated mechanisms (e.g. prefecthing mechanism) have been devised to support the schedule of the accepted packet in an efficient way while meeting the emulation real time constrain. Finally, Interceptor is able to accommodate to several software instances equivalent to users who simultaneously run the same application, in emulation mode. Based on this reason, we feel that the Interceptor may be very effective to test the network applications over different complex IP-based networks in a lab-based environment.

1.3 Outline

The reminder of this work is organized as follows. Session II discusses issues about network simulation. Session III presents a discussion about the network emulation, as it aims at managing network simulation and emulative interface together. Session IV illustrates the design issues of the Interceptor project. The aim of Interceptor is to design a software architecture supporting synthetic network emulation for complex IP-based networks. Sessions V describes this system architecture, illustrating which are its software components, how they work and which is their implementation. Session VI presents three case studies obtained using the Interceptor. Session VII contrasts it against other popular tools like NS-2 and Dummynet. Session VIII concludes the thesis by summarizing the obtained results and by outlining future research.

Chapter 2

Network Simulation: Background

A simulation is the reproduction of a real process or system over time. Simulation involves the generation of an artificial history of a system and its observation to study the operating characteristics of the real system. Developing a simulation model promotes the learning of the behavior of a real system over time. In general, this model describes a set of assumptions about the dynamics of the real system. These assumptions may be expressed in different ways, like mathematical, logical and symbolic relationship, among the entities of the system. Once developed and validated this model, it is possible to explore the behaviors concerning the real system. In this way, it is possible to evaluate potential changes on the system to know their impact on system performance without realizing them. Simulation is very useful to study a system in the design phase. As a result, the simulation modeling can be used to predict the performance of new designing systems. Sometimes, a model can be developed in a simple way by adopting mathematical methods, differential calculus, probability theory, algebraic method and so on. However, many real-world systems are so complex that adopting the previous solutions is not enough. In this case, a better solution may be to adopt a computer based simulation that tries to imitate the behavior of the system over time. In this possible scenario, data are collected as if they were produced by the real system.

2.1 Advantages and Disadvantages of Simulation

The output data from a simulation should correspond to that obtained from the corresponding real system. Differently from the optimization models, simulation models are *run* rather than solved. Simulation has some advantages and some disadvantages (Banks *et al.*, 2001). The main advantages show that it is possible:

- to investigate new policies, decision rules, information flows, operating and organizational procedures without involving the real system,
- to experiment new hardware designs, physical layouts and transportation systems without spending money to build a prototype or to rent these resources,
- to induce certain phenomena to make assumptions about how or why they occur,
- to speed up or to slow down the phenomena under study by compressing and expanding the simulation time,
- to understand how or why the variables of the system interact and, also, their importance concerning the performance system,
- to find the bottleneck of the system showing where its operations are excessively delayed,
- to facilitate the comprehension about how the system operates and the answer about "what-if" questions.

Here below, we report the main disadvantages.

- Building a model is not so simple, because time and experience are necessary. In particular, if two skilled people build two models, these may have some similarities, but it is unlikely that they will be the same.
- Simulation results are difficult to interpret. It may be hard to determine if an observation is the result of a system interrelationship or randomness.

- Simulation processes can be time consuming and expansive. In particular, if an analytical approach is possible, less time consuming and less expansive, it is preferred to simulation.

2.2 General Issues about Simulation

The most important issue when performing simulation is accuracy of modeling. Simulation results have little meaning when the simulative model differs greatly from the actual system being evaluated. In most simulations, some simplifying assumptions are required, but the ramifications of any assumption should be thoroughly understood and evaluated. If possible, the design parameters should be made programmable to the user rather than fixed by some assumption that will limit the scope of the model accuracy. This point leads to the next issue. Programmability is important in simulations, which attempt to determine optimal parameters from a large design space. The user should be given easy access, e.g., via an input stream, to any parameters that are likely to vary. All non-essential input parameters should have default values so that the user needs not clutter up his input stream with parameters that are not of interest. Also, ease of programmability is very important. If the user has to fill in any input parameter change, the simulation exercise will be very inefficient.

Simulators can be built according to different approaches: time-driven or event-driven. In time-driven simulators, the system clock is incremented in identical discrete amounts, and for each increment, every element in the system is processed to determine if its state should be updated. For event-driven simulation, an ordered queue of events, where each event is timestamped, drives the simulation. During each simulation cycle, the head event is processed and removed from the queue. During the processing of an event, the system clock is updated to match the timestamp of the event, and other events may be generated and inserted in the queue. The trade-offs are immediately obvious. For applications in which a large percentage of the elements in the system change state in every time unit, a time-driven simulation should be used. In opposition, for applications where events sparsely populate the timeline, an event-driven simulator is appropriate. Hybrid forms are also possible. For instance, a time-driven simulation can use a list of active elements in conjunction with an event queue of inactive elements. For sake of completeness, we give you some definitions about three kinds of time: real executive time (or simply real time), simulative time and executive time. The real executive time (or simply real time) is the time of the real system we want to simulate. The simulative time is the representation of time in the simulative model. It is the wall clock of the simulation. Finally, the executive time is the run time of the simulator.

2.3 Network Simulation

Today, Internet is an integral part of our daily life. As result of the large-scale adoption of a new generation of network applications, major changes in the infrastructure of the Internet are being proposed. These changes are designed either to support the growing traffic or to supply the services required for the new generation of applications. It is extremely important to build tools for analyzing and comparing these future applications and these major changes before and after their development (Brakmo *et al.*, 1996). Since the Internet is a large set of networks, it is not practical to conduct controlled experiments directly on its infrastructures. This means that there is a need for simulative environments, which are powerful enough to mimic the behaviors of the Internet, as well as analysis tools to evaluate the results of these simulations. In this context network simulation was born.

We can define network simulation as the part of the simulation that mimics network systems. In literature, we can find the general principles of the simulative experiments that we should follow to ensure the validity of the results. Typically, these principles are presented in general terms. One of the main tasks of the simulationist is to apply them to its specific simulative experiments. In this paragraph, we want to show some practical guidelines about the network simulation that go beyond the general principles. Most of these guidelines should substantiate the general simulation principles involving well-known rules that contain realistic evidence of what happens when they are not followed.

2.3.1 Sensitivity to network parameters

In general, protocols and communication networks work with discrete entities, like bits and packets. Several protocol components behave in a discrete manner and several network parameters have discrete quantities. Since the nature of these complex systems is discrete, also small changes in the parameters that define a network experiment can impact in a drastic way the results of a simulation. One of the main guidelines to investigate is the sensitivity of a simulation to different settings of such discrete values inside the network parameters. The sensitivity analysis is the systematic investigation of the reaction of the simulation outputs to the settings of the discrete values of the model input or to the changes in the model structure. For example, what happens when a parameter doubles or what happens if a module changes (we do not discuss about marginal changes or fewer perturbations in the input values). Sensitivity analysis can also help to optimize and validate the simulative model. Based on experience and on observations of the real system, the model user and the model designer would probably know the behavior of the simulation when the input variables are increased or decreased. In most of the large-scale simulation models, there are many input variables and, thus, many sensible tests must occur. The model designer must attempt to find the most critical input variables for testing if varying all input variables is too expansive or time consuming. If the real system data are available for at least two settings of the input parameters, objective scientific sensitivity tests can be conducted using appropriate statistical techniques. In this case, the guideline claims to assign different values to the parameters of the simulation over the maximum range as possible; in operative mode, this means to: i) simulate the right cases, ii) perform sensitivity analysis, iii) give confidence intervals with your results.

2.3.2 Analyzing results

A simulationist should control the results of each simulation to find any instance in which these outcomes vary significantly from the average. These instances are useful to highlight possible problems in the simulative environments, in the applications, or, even with the implementation bugs. In particular, if we only consider the average as a significant result, missing important information may occur. In each simulation, it is important that a simulationist looks at the graphs, which allow him to see the results of all the experiments, not just the outcome of the average of them. Looking just at the average may be misleading. One of the important uses of a simulator is to observe the effects on the experiment tuning the parameters. These observations may help to both find problems and solutions. This approach is called output analysis and concerns the examination of the data generated by a simulation. The aim of the output analysis is to predict the performance of a system or to compare the performance of different system designs. In general, statistical output analysis is used to evaluate the data produced by the simulator. If the performance of the real system is measured by a parameter *P*, the result of a set of simulation experiments will be the estimator *P'* of *P*. The precision of the estimator *P'* can be measured by the standard error of *P'*. The purpose of the statistical analysis is to estimate this variance, or to determine the number of observations required to achieve a desired precision.

2.3.3 Realistic traffic sources

In general, traffic sources are a fundamental part of most network simulations and are commonly simulated in two ways, by means of Poisson sources, or having multiple "bulk transfer" connections. There are some limitations if you use Poisson sources. First, these sources are unable to accurately reproduce either LAN or WAN traffic. Further, Poisson sources are non-reactive; in fact, they behave the same way regardless of the network behavior. Differently from the Poisson sources, multiple bulk transfers are able to produce reactive traffic. The main problem is that the dynamics of this traffic are not adequate to imitate the behavior of the real one. If we compare the results of the traffic produced by this simulative approach and by the real system on a graph, the degree of dissimilarity is, in general, visible. Furthermore, some studies should include traffic sources that are more realistic than either Poisson or bulk transfers. Today, traffic sources show that the pattern has self-similarity property, implemented in general by means of a heavy-tailed distribution. Another important issue is the reproducibility of the traffic pattern. Hence, if we configure always a traffic generator with the same initial parameters, it should produce every time the same pattern, regardless of the other network components (like the underlying protocols, the number of nodes or the type of network). The reproducibility of the traffic pattern does not mean that a packet containing a number “n” of bytes is transmitted exactly at time “t” every run, but rather that the structural features of the traffic should be reproduced. These traffic sources allow us to investigate different effects of two different protocols. Summarizing, the fundamental principles are to adopt realistic reproducible traffic sources and provide reasons for any distributions used in each simulation.

2.3.4 Byte versus time transfers

The definition of throughput is the amount of data sent over the time necessary to send it. If we want to compare the difference in throughput between two protocols, one of which is usually faster than the other, there is the possibility of wrong measurements in certain cases. In a simulation, during a transfer period, the quantity of traffic may either increase or decrease. If the quantity increases, the protocol will be measured as slower than its real capability. This is due to sharing the communication channel with higher levels of traffic at the end of the test. Vice versa, if the quantity decreases, the protocol will be measured as faster than its real capability. An alternative way to avoid that the fluctuations of simulative traffic have different effects on the two comparing protocols and to measure the throughput is to exchange as much data as possible for a certain amount of time. In this way both protocols will compete with the same background traffic for the period of the transfer, but, once more, the measured differences will depend again on the characteristics of the traffic. Then, comparing the throughput of two different protocols, it is possible to measure either how much time it takes each to send the same amount of data, or how much data each can send in the same amount of time. The latter may be more useful to understand if the obtained measures of the two protocols in a certain time period were dependent from different traffic fluctuations.

2.3.5 Variability VS uniformity

In general, a simulation may loose the variability (randomness) that is intrinsic to the real world. This problem can be generated by an error in designing the simulator or in configuring simulative parameters. Loosing the variability means making uniform some characteristics of the real system reproduced inside the simulative environment. The problem is that the uniformity may be seen as the lack of variability, because it makes even the degree of randomness of a real system. An example of the uniformity is the start of all applications at the same time. This scenario is enough unlikely in the real world. If we do not care about this aspect, it may be easy to have applications that fire at the same time on all hosts of the simulated network. This lack of variability produces periodic bursts of data, for example, all hosts send the delayed acknowledgments or start retransmitting lost packets at the same time. In particular, the effect is equivalent to start multiple independent communications at the same time. Indeed, a similar burstiness is also possible when the applications in each node do not fire at the same time. In fact, if one uses a low number of realistic traffic sources involving multiple communications, periodical and synchronized behavior of the sources that are not present in the real world may occur. In this way each traffic source is responsible for a large percentage of the traffic. Note that this behavior may be not evident, but by means of the Fourier analysis finding unwanted periodic uniform behaviors is possible. One way to avoid this problem is to have a lot of traffic sources involving multiple communications. Hence, the basic principle is to adopt a large number of distributions on everything that needs it. Unfortunately, dealing with complex simulators, what it is necessary to have distributions in is not always so clear.

2.4 Issues about Network Simulator

When one wants to design a network system, three different issues must be taken into account: protocol interactions, traffic pattern generations, and complex network topologies. Then, in general, in a network simulator a protocol implementation, a traffic generator and a topology manager should be present. In the history of network simulation, the three issues are not always clearly separated and, in some cases, some of them may be not configurable or only a subset of possible solutions is treated. In this Section, we analyze the main features of modeling these three issues.

2.4.1 Protocol interactions

One of the main advantages of the network simulation is probably the possibility of studying the behavior of an application when its real implementation does not exist. This possibility allows to know *a priori* the behavior of a new protocol design (or a new application design) by building a simulative prototype and by testing it inside a network simulative environment. In fact, differently from the testbeds that need a real implementation of the testing protocol (or application) and that can be conducted only *a posteriori*, a network simulator can use a simplified prototype based on a simulative language. In this case, the simulationist that implements the simplified simulative prototype should decide which protocol (or application) characteristics can be included in its study and which can be ignored. This operation is essentially a transcoding activity based on the idea of scaling down the level of abstraction of simulation. Some studies show that the simplified version of the relevant Internet protocols (or applications) may work well, but there are several studies that are very sensitive to their details. At this point, the simulationists must face some hard choices about the transcoding activity. Then the simulative prototype may fail to reproduce the behavior of the protocol (or the application) due to either possible transcoding bugs or model simplifications that may lead to inconsistent results. In particular, Brakmo and Peterson (Brakmo *et al.*, 1996) show the difficulty of reproducing the TCP code and they claim that even if an abstract TCP transcode is preferred for a quickly simulation, however running the real implementation of TCP code avoids possible errors. In fact, if after the simulation trails of new protocol design we want to implement the tested simulative prototype, many problems may occur. The variations in the implementation of only a single feature, even if it conforms to the designing of the protocol (or the application) and if it is consistent with the simulative prototype used in the simulative environment can produce a different behavior (in the real world) that has a dramatic impact on its performance. The implementation of a protocol (or the application) abstraction may behave in unexpected ways when combined with other protocols and mechanisms inside a real complete network. Although the Internet is composed by a set of standard protocols (and standard application), actually each of them is implemented by many different developing communities, often with very different features. For example, in the (Floyd *et al.*, 2001) more than 400 different implementations and versions of the TCP protocol have been identified. These considerations show that the simulationist may be driven to wrong conclusions and to misunderstand the capability of the protocol (or application). A solution to avoid the transcoding activity may be the possibility to write a fully detailed implementation of the protocol (or application) in a simulator. Unfortunately, it would be very hard to implement a whole complex protocol inside a network simulator due to its architectural constraints. As a result, no simulator is used to re-implement one of the common protocols.

A more realizable solution claims that the simulator must facilitate working with actual protocols, rather than just abstract specifications of protocols. This means to support real protocol implementations and their direct execution. Then, a possible alternative to put into practice this last consideration is to explore the network emulation approach. This approach introduces the real protocol execution inside a simulative environment. The emulator runs real protocols; this means that the actual behavior like processing overheads needs not be simulated and a two minute simulation is really two minutes real execution. For a higher degree level of details see the third chapter.

2.4.2 Traffic generation

Modeling and simulating the Internet means also to reproduce a number of competing traffic sources. One of the main basic problems of the network simulation is how it is possible to introduce different traffic sources inside a simulative environment, reproducing a realistic traffic mix. A possible solution seems to be the collection of enough traces that capture the full heterogeneity of the Internet world. Many times, this method fails or results inaccurate. In fact, many Internet protocols adopt adaptive congestion control that causes these failures. If one of these protocols detects a sign that the network is under stress, it suddenly slows down the pace at which it sends packets. This fact does not produce in its corresponding trace a sign that reflects the unexpected change. In fact, these mechanisms slow down the rate of their communications if a congestion is detected anywhere along the end-to-end path between source and destination. In this way, a person that looks at the traces of an unloaded high-speed link is not able to understand that its packets might still send at a higher rate, if somewhere along the end-to-end path enough resources are available. This phenomenon makes the traffic traces shaped. The use of trace-driven simulation to incorporate the different real Internet effects is limited by the shaping effect of the rate adaptation from the end-to-end congestion control. This means that we cannot reuse a trace in another context, because the communication would not behave in the same way, and would produce inaccurate results. However, measuring traffic is not fruitless, if one focuses on trace-driven source-level simulation instead of fixing mind on trace-driven packet-level simulation. Hence, we do not focus on the volumes of data sent by endpoints, but on only the lower-level specifics of exactly which packets are sent when they are shaped. Unfortunately, not all sources can be reliably characterized by traffic traces. For example, the remote login user is very sensitive to the heavy congestion and may terminate in advance. Another more general class of examples contains the inherently adaptive applications. These applications have either their packet-level characteristics or their application-level behavior shaped by current traffic. Perhaps, another important issue in simulating the Internet is to explore a scenario with a range of congestion levels. In fact, all degrees of congestion (from no congestion to all congested) may be observed with non-negligible probability. In particular, a simulation that focuses only on congested networks is probably of limited interest without a range of comparison with other simulations that adopt different degrees of congestion. Predicting the evolution of the congestion in Internet is as hard as reproducing a realistic traffic mix. For these reasons, it is impossible to define a *typical* level of congestion or a *typical* Internet traffic mix. Hence, we must keep in mind that we must consider a spectrum of scenarios that can simulate *typical* levels of congestion or *typical* Internet traffic mixes.

As a conclusion of these considerations, we want to illustrate an interesting approach to reproduce traffic patterns. If we want to generate a traffic patterns we need a workload traffic generator. A workload generator is a traffic generator that simulates, at higher level, the behavior of the studied application, producing, at a lower level, a communication generating a specific kind of traffic pattern. The topics of the next Section are how to model a workload and how to characterize a workload for network systems. In particular, we will describe two well-known examples of workload traffic generators (SURGE, GISMO).

2.4.3 Workload modeling

The hardware and the software components of a system influence its performance as well as the load it must schedule. For this reason, the study of the workload is important to understand the performance indices of the target system (Calzarossa *et al.*, 2000). In particular, some of these indices depend directly on the workload. Reproducing the dynamics of a real workload is complex and difficult. Further, if a model may be built, it should mimic the activities of the real load in a compact, repeatable and accurate fashion. Even if the real workload emerges from the execution of different competing deterministic events, it is modeled as non-deterministic, by means of statistical techniques. This is due to the quantity of data to be measured, to many variables interacting together and to phenomena to be considered. Here below, the statistical techniques for building workload models are explained along different steps.

1. *Formulation*. The parameters to be used for the workload description are selected, as well as the characterization level and the workload basic component. A criterion for the assessment of the model representativeness is also identified.

2. *Collection of the parameters*. It is necessary to collect the different values of the workload parameters to build a model during the different runs.

3. *Statistical analyses of the collected data*. This step is divided into the following sub-steps:

a. *Preliminary analysis*. The collected data may be divided into different workload partitions. The detection of different populations allows to highlight smaller and more controllable portions of the original data set.

b. *Analysis of the parameter distributions*. This analysis is conducted by applying different transformations of the original values of the parameters or by identifying the outliers. If the outliers are detected, they are removed to avoid distortion of the classification process.

c. *Sampling*. To facilitate the scheduling of the amount of collected data with a realistic quantity of time and memory, one should consider a small number of components.

d. *Static analysis*. If the values of the parameters are scaled inside a common interval, it is possible to achieve a robust classification. In general, the classification and partitioning of the workload components is achieved by adopting statistical techniques useful for discovering homogeneous groups in a given data set. Then, from these groups, it is possible to select some representative elements in relation to the selection criteria adopted to build the workload models.

e. *Dynamic analysis*. If one should mimic the characteristics of the workload that change their values over time, it would be necessary to consider the properties of several time series. Possible characteristics of the workload may be: dynamics of the processing requests and the evolution of the combinations of the different components execution over time. Different techniques, such as numerical fitting, statistical analysis of non-stationary series of events and stochastic processes can represent the analyzed data sequences in a concise form (i.e. the arrival and departure patterns of components to the system). A possible approach to describe the dynamics of the workload is the technique of probabilistic graphs. In particular, it is possible to apply user behavior graphs (Calzarossa *et al.*, 1993) to highlight the dynamics of the workload by replicating the series of commands submitted by the users. Each user is represented as a graph and each node of this graph coincides with a specific type of command. A different number of edges are connected to each node and they represent the probability of execution of the next command immediately following the conclusion of the current one. The time spent for each node equals the time passed running the command, waiting for its conclusion and deciding about the next one.

4. *Representativeness*. Finally, the representativeness of the model must be evaluated. There are different criteria, in particular, there is a performance-oriented criterion based on a vector V that is composed by different indices of the workload performance. Hence, the representativeness of the model is measured by comparing the two vectors V_r and V_m, where the first is the set of performance indices of the real workload execution, while the second refers to the model performance. Some examples of possible performance indices are response time, throughput, and resource utilizations.

Here below, we show how it is possible to apply the workload characterization to network-based systems. In this context, network-based environments refer to both basic computer networks considered in isolation, or interconnected together (e.g., WAN, MAN and LAN). In general, the performance indices of this context are: throughput, channel utilization and various forms of delay expressed as a function of the traffic nature based on the user behavior, on the protocol characteristics and on the network infrastructures. Capturing the essence of the network-based systems is possible by conducting several sensitivity studies about their performances. These results are achieved by changing the load over time, the topology of the network under study and the connectivity capabilities. All these considerations justify the importance of discovering a set of parameters that are able to capture and reproduce the essence of traffic flowing in the networks. The main approach to the workload characterization of network-based environments (Calzarossa *et* *al.*, 1993) is based on a layered structure (see Figure 2.1) that divides the workload components into three basic groups: user terminals, processing nodes and communication subsystem. According to this approach, the load generated by the users may require services either from the local processing nodes or from the network (see the dashed lines in the Figure 2.1). This means that different workload components for different layers must be designed. To implement the load at each layer probabilistic graphs are adopted. This is due to their capability to reproduce the static and dynamic properties of the load. Layer 3 includes the user behavior graphs that are able to capture the dynamics of the workload by reproducing the sequences of commands run by the users. On layer 2, the user behavior graphs turn into system graphs that are based on the sequence of requests, set in arrival time order, coming either from users or the network. Finally, on layer 1, the system graphs turn into different traffic shapes on the communication subsystem, if the requests from the upper layer must be processed on remote nodes. This means that the traffic of layer 1 is described by network graphs that are the representation of the message flow on the communication subsystem. The workload of network-based environments can be modeled by considering the interactions among the different layers, summarized in the points below:

- Measurements of the arrival sequence of the commands submitted by each user and the building of its behavior graph (layer 3);
- Measurements of the resource consumptions and of the arrival times of all the requests processed by each node and the building of the system graph (layer 2);
- Measurements of arrival time, length, type and source/destination addresses for all the messages inside the real load flowing in the network and the building of a network graph and its corresponding traffic flow (layer 1).

The parameters that can be collected and then used to build a graph for the three layers and the corresponding modeling techniques are listed in Table 2.1.

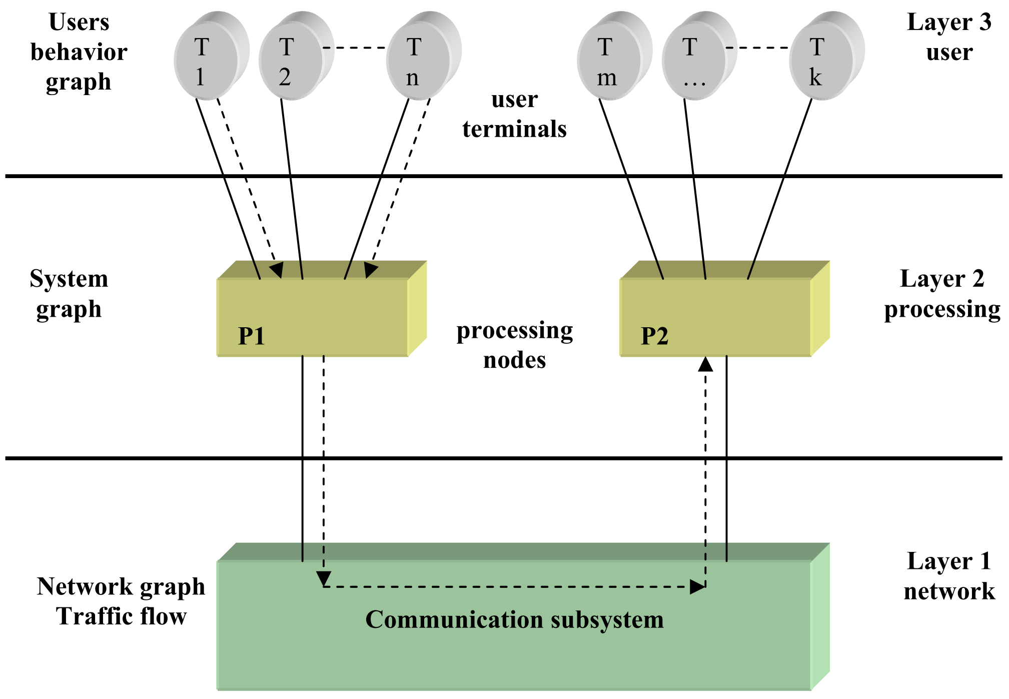


Figure 2.1: Layered structure for the workload characterization of network-based environments (Calzarossa *et al.*, 1993).

Table 2.1: Parameters and techniques adopted for network-based environments.

Layer	Characterization level	Parameters / Techniques	
		Measured	**Derived**
3	User	arrival time command type	arrival rate interarrival time user behavior graph
2	processing node	arrival time request type HW/SW resource consumptions	arrival rate system graph
1	communication subsystem	arrival time message type message length source / destination nodes	generation rate interarrival time network graph traffic flow matrix

2.4.4 Network topology

The wide-spread development of the Internet has, as a side effect, a variety of inter-networking problems concerning routing, resource reservation, and administration. Solving these problems is possible by involving simulation and analysis that try to reproduce the actual topological structure of the Internet by means of models. In fact, interesting large networks are expensive and difficult to control; for those reasons they are rarely available for experimental trials. Hence, it is generally more efficient to conduct trials by adopting simulative scenarios that provides an abstraction of the real topology of the network. The current state of the art remarks that we can find a lot of randomly generated or trivial network topologies, but there are few studies about scaling the results and how these results vary from a topology model to another one (Zegura *et al.*, 1997). A fundamental question for simulation is what kind of topology we should use for reproducing a network-based environment. This topology should explain how the computers in the network are connected with each other and the properties of the links that implement the interconnection. In general, the largest part of simulative studies adopts specific topological properties or use synthetically-generated topologies. The correct way to evaluate the topology-related protocols can be pursued if these studies show an accurate reproduction of the Internet behavior. This means that the topology models must show typical properties or invariants of the real structure of the Internet. However, it is very difficult to pursue the correct reproduction of the Internet behavior without resolving problems like mapping the actual topology infrastructures, describing them, and building models that capture its typical characteristics (Medina *et al.*, 2001). In fact, the topology of the Internet is difficult to reproduce for two main reasons. The former is that the Internet topology is a target that is constantly evolving and the today's network is not tomorrow's (Floyd *et al.*, 2001). This evolution limits the possibility to design an accurate map of the structure of the Internet showing the difficulty to validate the "realism" of any topological model. The latter is that the Internet network infrastructures (differently from historical centralized public telephone networks) are controlled by a set of autonomous authorities that are not often willing to exchange low-level connectivity information (Paxson *et al.*, 1999). Thus, it is possible to design a scheme of general characteristics of the Internet, but it is not possible to construct an accurate topological model, due to its decentralized administration and to the lack of topological details. For these reasons, there are no models that reproduce a *typical* Internet topology. Hence, the simulation can explore protocols that are sensitive to topological structures, only specifying how the protocols perform over a given range of *typical* topologies. In spite of all these considerations, simulationists have begun to develop topology generators for Internet simulations. Several of them create networks with locality and hierarchy based on the current Internet. Unfortunately, the largest part of the considerations regarding the network behavior depends on other simulations and analyses that show us an approximation of the large-scale nature of network topologies. In spite of this consideration, these efforts are not fruitless and bring us three main benefits. First, we can design more efficient protocols that take advantage of its topological properties. Second, we can create more accurate artificial models for simulation purposes. Third, we can derive estimates of topological parameters that are useful for the analysis of protocols and for future speculations on the Internet topology. Usually the researchers in this area are divided into two groups. The first group consists of researchers (like us) that study the performance of network protocols and that exploit topology (generating or designing) tools to obtain excellent synthetic network topologies for their simulations. Instead, the second group includes researchers that study how to generate accurate synthetic topologies.

As already said, it possible to describe the general topological characteristics of the Internet, but it is not possible to model a typical network infrastructure. Here below, we illustrate the common representation of the network topology to implement the Internet (Calvert *et al.*, 1997). Designing the topology of a network means modeling the paths (i.e., sequence of nodes) along which communications flow among nodes. From this point of view, graphs represent the set of these paths, where nodes stand for switches or routers, and edges for links between switches or routers. The path may be a direct link or it may be any channel of communication between two nodes. In a real network more than a single path may connect two nodes; in general, only the best one according to a chosen metrics is reproduced inside the model. Again, the simulationists do not model individual hosts; it is more reasonable that a terminal node is a switch (i.e. a LAN). Further, modeling the link characteristics (e.g., bandwidth, latency, packet lost etc.) is not a task of network topology, Usually, these characteristics are described as overhead of the edges. Today, the general shape of real network topology may be influenced to some small degree by policies for assignment of IP addresses and government funding of inter-domain exchange points. From this point of view, the Internet (network) topology is a set of interconnected routing domains, where each domain is a group of nodes under a common administration that shares policy and routing information. Routing domains range in size from few nodes to thousands. The main characteristic of the network topology is the routing locality and the hierarchical structure. The routing locality is the graph between any two nodes in a domain that remains entirely inside the domain, while the hierarchical structure is the description of the structure with different levels of the network. From these considerations, each routing domain in a network infrastructure can be identified as either a stub domain or as a transit domain. A domain is a stub domain only if the path between any pair of nodes that belong to the domain stays within the domain. It is the implementation of the routing locality. In general, a stub domain stands for a group of interconnected LANs (for example a campus network). A transit domain connects different stub domains in an effective way; in fact each stub domain would be directly connected to each other by means of transit domains. For this reason, in a transit domain each backbone node connects to different stub domains (via gateway nodes in the stub) or to other transit domains. The integration of the transit and stub domains imposes a hierarchy on nodes. In general, a transit domain stands for wide or metropolitan area networks (i.e. it comprises a set of backbone nodes). Another difference between stub and transit domain is that the first one supports only the traffic that originates or terminates inside it, while the second has not this limitation. Furthermore, stub domains can be divided into two categories: single- or multi-homed. In the former the stub domain connects to a single transit domain; in the latter it connects to more than one transit domain. In particular, it is possible that some stub domains directly connect other stubs. Hence, from these considerations two general principles about the routing of the current Internet between two arbitrary nodes emerge. If the two nodes belong to the same stub domain, the path between them stays completely inside the domain. Instead, if they do not belong to the same stub domain, the path between them goes from the domain of the source node, passing through zero or more transit domains, to that of the destination node.

2.4.5 Synthetic network topology generation

Graphs are commonly used to model a topological structure, including the one to describe network systems. Graphs are very useful in order to face up problems ranging from routing to resource reservation. The literature (Zegura *et al.*, 1997) proposes different types of graphs like fixed topologies (for example rings, tree or stars), "well-known" topologies (for example the ARPAnet), and randomly generated topologies. Further, the literature shows also the limits of the different types, by highlighting how the use of a model affects the final results. Until now, the selection of a model has depended on the intuition and the experience of the simulationist, because investigation analyses on the differences and the similarities among the models have not been carried out. This is due to the fact that the problem of creating realistic topologies has not been solved yet.

Each of the above types has its intrinsic limitations. Regular topologies are a simplified overview of current or past real networks. Well-known topologies reproduce only specific real networks (of the current or past real networks). Random topologies may not reproduce any (past, present, or future) real network. In fact, most of simulationists are aware of the risk of drawing conclusions about real networks. For this reason their papers includes a disclaimer where they specify the topology model used and the parameters for drawing their conclusions. In (Zegura *et al.*, 1997) three examples, the network topology model is reported, as it really matters. Then, one must apply different criteria to model a network topology according to its utilization. For example, if the purpose is to test an algorithm in a dynamic scenario, the model should generate topologies with moderate node degree and many routing choices. If the purpose is to model a particular static network, then the model should accurately reflect this particular topology.

Here below, we consider three classes of generation methods: flat random graph, regular and hierarchical. Some of these methods are the direct implementation of the previous types of graphs. Flat random graph methods build networks as a set of nodes that are connected by edges with a given probability and without any preordained structure. Differently from the previous one, the regular generation methods generate networks with simple and rigid structure, like rings and meshes. The hierarchical generation methods create networks by assembling a structure from smaller network components. These last methods are a compromise between the flat random and the regular methods, because they impose some high-level structure and fill in details randomly.

The literature suggests different flat random methods used to model an Internet topology. All these methods are very similar: given a set of vertices distributed on a plane, an edge is added between each pair with some probability. The pure random methods adopt fixed probability. These methods are interesting for their simplicity and are commonly used, but they do not reflect the structure of real networks. One of the most famous is the Waxman method (Waxman, 1988), which adds edges with a probability distribution that is some function of the distance between each pair of nodes. The probability distribution of an edge from *u* to *v* is

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

where $0 < \alpha, \beta \leq 1$, d is the Euclidean distance from u to v and L is the maximum distance between any pair of nodes. Several variations on the Waxman method are proposed in literature. Other methods (Zegura *et al.*, 1997) try to capture locality by dividing edges based on length in classes, and by assigning a different probability for each equivalence class of edge lengths:

$$P(u,v) =\{\alpha,\ if\ d < r;\ \beta,\ if\ d \geq r\}$$

Each pair of vertices is treated in the same way (i.e. by means of the same function) with respect to the addition of edges. This implies that it may be possible to manage the quantity of edges, but not to configure the edges according to a preordained scheme. In fact, these methods generate large sparsely connected graphs. In particular, the probability that a graph is connected decreases as the number of nodes in the graph increases. This means that it is unlikely build a graph with a high quantity of nodes that have on average a low quantity of edges. To solve this problem one can first generate a partially connected graph and then remove and/or connect the edges to obtain a fully connected graph. Another possible solution is to create small sub-graphs and then connect them by means of a hierarchical structure (as the hierarchical methods do).

In general, analytic studies adopt the regular graphs to evaluate the performance of network protocols because their structure makes them manageable. The main shapes of the regular graphs are linear chains, rings, stars and meshes. The general structure is very simple and makes it possible to conduct a lot of trails in a non-complicated topology. It is important to highlight that differently from the flat random methods it is possible to control the number and the configuration of edges. Unfortunately, the regular graph methods are extremely rigid in their structure.

The previous two methods are not able to capture the hierarchy of the real networks. Here, two hierarchical methods for generating graphs with locality and hierarchy properties are presented. The locality is obtained by building small sub-graphs according to flat random graph methods and the hierarchy is obtained by assembling sub-graphs according to a specific structure.

The former, the N-level hierarchical method builds a graph by iteratively substituting each node with a sub-graph. Starting from a first connected graph, each single node in the graph is substituted by a connected sub-graph. The edges of the first graph are then connected to nodes belonging to sub-graphs that replace the first level of nodes. In particular, to build an N-level hierarchical graph, the top-level graph is created by adopting a scale parameter S_1 (in general, by adopting S_N, where N represents the level order). Then, each single node is subdivided again, according to the second-level scale parameter S_2. This operation may be repeated for N levels. At the end, the final graph is the combination of the scale parameters of each single level and the length between the two nodes is roughly determined by edge level. The main advantage of the N-level method is its simplicity because the number and the configuration of edges between each pair of nodes result in a good approximation, fostering domain locality. A limitation of this method is that only one type of nodes may be specified.

The latter method generates a hierarchical structure that connects together transit and stub domains by passing through different steps. The first step generates a connected random graph using one of the previous methods. At this step, each single node stands for an entire transit domain. The second step substitutes each node with a connected random generated graph that represents the backbone topology of a transit domain. The third step creates a number of connected random sub-graphs corresponding to each node of the transit domain. Further it interconnects these nodes by means of an edge to these sub-graphs that stand for stub domains. Finally the fourth step inserts extra edges between different nodes: from a transit to a stub or from a stub to another stub. The size of the graph and the distribution of nodes between transit and stub domains is controlled by the following parameters: the number of nodes, the average number of nodes for each transit domain, the average number of stub domains for each transit node, and the average number of nodes for each stub domain. These methods can create rather large graphs. Moreover, differently from the flat random graph, this method creates a graph with a high quantity of nodes that may have on average a low quantity of edges.

2.5 Examples of Network Simulators

Until now, a lot of Network Simulators have been created. These simulators present widely different characteristics. Some target a specific area of research interest such as a particular network type or protocol. Others (e.g. NS, REAL and INSANE) target a wider range of protocols. Some others, with a more general purpose, provide a simulation language with network protocol libraries (e.g. Maisie and OPNET). Finally, very focused simulators model details that are relevant to the final user. Concerning this last group, we highlight either available network topology or available traffic models/generators. Further, simulators focus on several complementary approaches taken to improve accuracy, performance or scaling. Some simulators speed up event by processing with analytic models of traffic flow or by adopting specific queuing behaviors for improving performance or accuracy. Other simulators improve performance implementing parallel and distributed systems. In fact, several simulators (e.g. Maisie and REAL) support multiprocessors or workstation networks. For sake of completeness, in this paragraph, we show you some well-known workload traffic generators and some topology generators.

2.5.1 VINT project and NS-2

One of the most important general-purpose network simulator is NS (Bajaj *et al.*, 1999). Since NS is widely used for network simulation, its modeling language is extensive. NS is a discrete event simulator targeted at networking research. NS provides substantial support for simulation of TCP, routing, and multicast protocols over wired and wireless (local and satellite) networks. NS began as a variant of the REAL network simulator. The VINT project implemented through the NS simulator and related software provides several innovations that have improved the way of evaluating network protocols: abstraction, emulation, scenario generation, visualization and extensibility. VINT/NS provides several levels of abstraction by scaling the simulation granularity both from detailed to high-level simulations. It also includes an emulation interface that exchanges network “live” traffic between real network nodes and the simulator. In VINT/NS a scenario generation supports the configuration of the simulative experiments. Furthermore, it provides tools for visualization (Nam) that allow simulationists to see the complex behavior in a network simulation. It is not so simple to analyze the behavior of a complex system involved in large scale networks providing only tables of summarized performance numbers or some plots. For this reason, visualization that shows the dynamics of the network system allows a deep understanding of protocol behaviors and debugging. One of the main innovations of VINT/NS is its extensibility. In fact, it is possible to insert new functionality, to investigate a variety of different scenarios and to evaluate new protocol designs. VINT/NS adopts two different levels of programming: the high level allows by means of OTcl scripts to configure predefined modules and the low level allows the C++ implementation of new modules. Below, we describe the first three features.

The study of the network protocols is conducted in different ways, either by evaluating only a single protocol or by aggregating several traffic flows coming from the interaction of different protocols. Computer resources limit the number of network components like nodes, links and protocol agents that can be used in an accurate simulation. To overcome these limitations it is possible to adopt three approaches that improve the performance of the simulator: tuning the implementation, removing superfluous detail or supporting parallelism. A distributed version of NS exists, but the efforts are focused on tuning the implementation and providing multiple levels of protocol abstraction. The idea is to remove unnecessary details without violating the validity of the model. VINT provides different levels of abstraction each of which sacrifices some details to preserve computer resources. A user can tune simulator performance versus accuracy by adjusting the simulation abstraction level. This means that one can increase the level of abstraction to improve the performance of the simulation or decrease it to supply a greater accuracy.

Most simulation experiments are conducted inside a single simulated world that is executed on a computer or on a cluster of. In this way the simulation exploits only the protocols developed for the simulative environment. Instead, emulation allows a simulator to interact with real network nodes by exploiting also all real protocol executions. Hence, VINT/NS includes an emulation interface useful to evaluate the dynamic behavior of protocols and their implementations in end-systems. By means of the emulation interface it is possible to construct a scenario by placing the simulator as an intermediate node (or end node) along an end-to-end network path. In this scenario, the simulator introduces live network traffic into its environment. Within this environment, the live traffic undergoes the dynamics of the simulated network. For this reason, the simulator exploits a real-time scheduler that supplies the integration between the simulative environment and the real end-systems. VINT/NS takes advantage from its emulation interface for evaluating both end-systems and network element behavior.

It is very difficult to conduct simulation experiments under an appropriate set of network conditions. VINT/NS supports the creation of general events, like link failure, or the generation of complex traffic patterns and network topologies to generate a scenario. It is possible to generate a scenario in two ways: by configuring each single network component or by adopting the automated generation of scenarios. In the first way one can configure several programmable components, like network topology, traffic models and dynamic events, according with its pre-defined scheme. The second provides the automated generation of scenarios that allows simulationists to explore a number of possible configurations greater than through a manual approach. According with these two possibilities, VINT/NS supports two types of network topologies: the “pre-defined” and “automatically generated” ones. In the first case, it is possible to configurate a network topology manually or to adopt a pre-defined library configuration. This case supplies the possibility to implement a network topology varying from simple examples to the reproduction of real complex networks. Instead, in the second case, it is possible to generate, in automatic way (by means of GT-ITM), random topologies according to different parameters such as node degree, levels of locality and of hierarchy. Once more, according to the two possibilities, VINT/NS provides traffic models, such as traffic generation support or traffic libraries, for synthetic application of workload models. These traffic models may include the application of traffic generation, call patterns, or multicast group membership dynamics and may be derived from empirical data, analytic models, or generated randomly. VINT/NS supplies a lot of source models either for unicast or for multicast transport protocols (e.g. several TCP variants, UDP, SRM, RTP, FTP, Telnet and HTTP). In particular, the framework of VINT/NS permits to include new traffic models easily. Finally, VINT/NS also provides test generation to analyze the conditions of a simulation experiment and to evaluate the correctness of a protocol. In particular, they developed a framework for Systematic Testing of Protocol Robustness by Evaluation of Synthesized Scenarios (STRESS) that should be useful to find quickly any possible problematic case in the behavior of a protocol by adopting a systematic synthesis of test scenarios.

The VINT/NS promoted the extensibility by building programmable and modular software architecture. For this reason, in this software architecture, the configuration of the simulation is a program rather than a static configuration file. In fact, a simulation of the VINT/NS is composed of dynamical objects. The interaction of these objects creates the simulation. By configuring simulations in this way, the simulationist can extend the simulator with new primitives or dynamic event handlers that interact with a running simulation to change its course as desired. To this aim, the VINT/NS takes advantage of a split programming model, where the simulation core is implemented in C++ language while simulative scenarios are configured by a program based on an OTcl scripting language. For this reason, the VINT/NS software architecture adopts two programming levels to provide flexibility and to maintain performances. In the former, tasks like low-level event processing or packet forwarding require high performance and are modified infrequently. At this level, the VINT/NS uses the compiled C++. In the latter, tasks like the dynamic configuration of protocol objects, the specification and placement of traffic sources are often iteratively refined and undergo frequent change. At this level, the VINT/NS uses a flexible and interactive scripting language like the Object Tcl (OTcl). The distinction between the simulation core and the simulation configuration level allows the simulationists to easily configure the simulation environment and it encourages the implementation of new simulative core objects. This means that, the VINT/NS architecture implements fine-grained simulation objects in C++ and it combines them with OTcl scripts to exploit higher-level "macro-objects".

2.5.2 OPNET simulator

OPNET (Optimized Network Engineering Tool) is a synthetic general-purpose object-oriented network simulation environment (Chang, 1999). It provides an environment for the specification, simulation and performance analysis of network communications. OPNET supports many different communication systems: from a single LAN to global satellite networks. It is a discrete event simulator. OPNET provides modeling and simulation designing tool, employs a hierarchical structure to modeling, proposes a detailed library model specialized in communication networks and offers facilities to debug, probe and analyze the data produced. A simulation is in general composed by five phases: the definition of the problem, the building of models, the execution of a simulation, the analysis of the output data and the decision making. OPNET supplies the three central phases. It uses a hierarchical structure to design different aspects of the model being simulated. Further, it supplies a detailed library with models of existing protocols that can be included with or without change in simulation experiments and it supports the development new models. Finally, the model can become an executable code that can be debugged or executed to obtain the output data. This sophisticated object-oriented network simulation environment has a lot of tools, which help the simulationists to specify fine-grained models, to identify the interesting elements in the model, to run the simulation and to analyze the produced output data. These tools are divided into three different categories: useful to hierarchical model building, running simulations and analyzing results.

The first category provides three (Network, Node and Process) editors to develop a representation of a system being modeled. Network Editor is the physical topology manager of a communications network. By means of this we can define the position and interconnection of communicating entities: node and link. The Node Editor specifies in the node domain the communication devices created and interconnected at the network level. The process editor builds the process models that describe the logic flows, the behavior of processors and the queue modules.

The second category provides the Simulation Editor. This editor specifies three steps: data collection, simulation construction and simulation execution. Before the data collection a simulationist should decide which information should be extracted from the simulation. In the second step, the simulationist builds a simulation program and fills in as an executable file. Finally, OPNET supports different options to run simulations and different attributes to influence the behavior of the simulation.

The third category provides a Probe Editor, Analysis Tool and Filter Tool. The Probe Editor provides several different probe types in order to capture different types of output data (Statistic Probe, Automatic Animation Probe, Custom Animation Probe and Coupled Statistic Probe). Simulations generate different forms of output that include several types of numerical data, animation, and detailed traces provided by the debugger. Furthermore, it supports an interface in C language. By using it, developers may generate proprietary forms of output. Analysis Tool is useful to view and manipulate the statistical data and to display information in form of tables or of graphs. The Filter Tool allows the user to extract data from simulation output files and to display this data in various forms.

2.5.3 GLOMOSIM

GloMoSim (Global Mobile system Simulator; Zeng *et al.*, 1998; GloMoSim, 2001) is a library supporting sequential and parallel simulation for wireless networks. This library contains a set of modules that simulate different wireless communications in the protocol stack. GloMoSim library is written in PARSEC (PARallel Simulation Environment for Complex Systems; Bagrodia *et al.*, 1997), a C-based language specific for parallel simulation, which supports a simulative environment based on Maisie simulator. For a better comprehension of GloMoSim, we will spend a few words about PARSEC. PARSEC implements a message-based approach to discrete-event simulation. In particular, real processes are simulated by entities and events are represented by transmission of time-stamped messages among corresponding entities. PARSEC has also a visual-programming environment named PAVE (PARSEC Visual Environment). Thanks to PAVE, it is possible to support the visual design of PARSEC programs or to visually configure simulation models using GloMoSim library components. The execution of a simulator implemented with GloMoSim may be either sequential or parallel. In particular, the parallel implementation can be executed by adopting different conservative synchronization protocols and it has been implemented on both shared memory and distributed memory computers. The GloMoSim exploits its parallel implementation in order to simulate networks with a very high number of mobile nodes. GloMoSim wants to maintain the requirements of modularity and scalability. Differently from other approaches that simulate networks with 100.000 mobile nodes, by allocating 100.000 entities (a one-to-one relation), GloMoSim divides the network into a number of partitions by adopting an entity as a single layer of the protocol stack for all the nodes of that partition. Hence, the main aim of the GloMoSim is to support the development of a modular simulation environment for protocol stacks that can scale up networks, also with thousands of heterogeneous nodes. To achieve the modularity, GloMoSim is extensible and is made of standardized units, which can be assembled in different ways. In fact, the communication protocol stack for wireless networks is divided into a set of layers; each layer has its own API. By means of these APIs the simulated protocols running on one layer interact with those at a lower or higher level. The modularity facilitates reliable comparison of different protocols at a given layer. In fact, if a GloMoSim model implementation uses correctly the constraints of the APIs defined at each layer, it will be possible to exchange protocol models at a certain layer without modifying the remaining layers in the stack. In particular, if we want to maintain the modularity in a parallel implementation the APIs must bind the interactions among the entities that are running on different processors or on different machines. These interactions may be messages exchanged, function calls, or entity parameters according to the implementation. The modularity in a parallel implementation is possible because a PARSEC library entity representing a layer of the protocol stack is largely self-contained. This method encapsulates the complexity of a specific network behavior inside an entity and makes it regardless of the other ones. As a solution to support scalability GloMoSim aggregates more nodes inside one entity reducing the total number of entities. This approach improves either sequential performance or parallel performance.

2.5.4 Other network simulators

For sake of completeness, in this Sub-paragraph, we report, in concise form, some other famous and historical network simulators.

BONeS provides several building blocks, modeling capabilities, and analysis tools for the development and analysis of network products, protocols, and system architectures (BONeS, 1999). With its recent ATM Verification Environment (AVE), it is specifically targeted for ATM architectural exploration and hardware sizing.

COMNET III, a graphical, off-the-shelf package, permits simulationists to analyze and to predict the performance of networks ranging from simple LANs to complex network systems. Starting with a library of network objects with an object representing real world objects, the COMNET III object-oriented framework and its GUI gives user simulationist the flexibility to try different “what if” scenarios.

INSANE is a network simulator designed to test different IP-over-ATM algorithms with realistic traffic loads derived from empirical traffic measurements. Its protocol stack provides real-time guarantees to ATM virtual circuits by using Rate Controlled Static Priority (RCSP) queuing. A protocol similar to the Real-Time Channel Administration Protocol (RCAP) is implemented for ATM signaling. A graphical simulation monitor can provide an easy way to check the progress of multiple running simulation processes (INSANE Homepage).

Maisie is a C-based language for hierarchical simulation (Bagrodia *et al.*, 1994), or more specifically, a language for parallel discrete event simulation. A logical process is used to model one or more physical processes; the events in the real system are modeled by message exchanges among the corresponding logical processes in the model.

NetSim supplies a very detailed simulation of Ethernet, including realistic modeling of signal propagation, the effect of the relative positions of stations on network events, collision detection and handling process and transmission deferral mechanism (Lewis *et al.*, 1993).

REAL is a simulator to study the dynamic behavior of flow and the congestion control schemes in packet switch data networks. Network topology, protocols, data and control parameters are represented by Scenario*,* which are described using NetLanguage, an ASCII representation of the network. About 30 modules are provided which can simulate the behaviors of several well-known flow control protocols (Keshav, 1997).

2.6 Examples of Traffic Generators

The main task of a workload traffic generator is to understand how servers and networks respond to load variations. In general, generating representative communication traces is difficult, because the service workload may have a lot of different features.

First, empirical studies have shown that there is a high variability of demands that derives from CPU loads and a number of open connections. Then, it is important to focus on the properties of the communications that have high variability, such as request interarrivals and file sizes. Second, in general, the workload traffic generated by service requests in a network system may be self-similar. This means that this kind of traffic exhibits high variability over a wide range of scales. Self-similarity of the traffic forces the performance of a network system to be negative, thus it is important to reproduce this feature when a workload generator is designed and implemented. To develop a workload generator with these properties, it is possible to use a trace-based approach. Trace-based workload generation adopts prerecorded log files of past traffic and statistical techniques to reproduce workloads. Although this approach is easy to implement and reproduces the behavior of a real system, it treats the workload as a "black box". For this reason, it is very difficult to obtain an insight into the causes of a system behavior and to tune the workload to reproduce future conditions or varying demands.

There are many different kinds of workload traffic generator, but in this study we are interested in those describing Web applications and Multimedia traffic. Here below, we want to show you the two main examples: SURGE, GISMO. Finally, we want to introduce another interesting approach implemented as a distributable platform for traffic generation called D-ITG.

2.6.1 SURGE: workload generator for web traffic

The SURGE (Scalable URL Reference Generator) is a well-known workload traffic generator that exercises servers and networks by reproducing the behavior of users that execute web applications (Barford *et al.*, 1998). SURGE generates a high variability on the workload. The SURGE tool is composed of two basic parts: one concerning the idea of user equivalents and the other concerning a set of distributional models.

The idea of user equivalents asserts that the workload generated by this tool should reproduce the structure of the real traffic generated by different users. In particular, SURGE tool reproduces the web requests of these users (see the Figure 2.2). Hence, the intensity of the activity of different users that request the service generated by this tool should be expressed in user equivalents (UEs). Each user equivalent is a single process that alternates a time period for making requests of a service and another for lying idle, during an endless loop. Both the requested services and the idle periods must show the distributional and the correlation properties of the corresponding population of real users. Thus a UE is implemented as independent ON/OFF process where ON is referred to periods during which services are used and OFF to those during which nothing happens.

The distributional models describe the set of probability distribution used by each UE. Some of them are heavy-tailed, this means that the output of the distribution has a high variability. The set is composed by the following six statistical properties of Web streams: File Sizes, Request Sizes, Popularity, Embedded References, Temporal Locality and OFF Times (see the Table 2.2).

1) The “File Sizes” distributional model describes the set of files stored on the server matching with empirical measurements. This probability distribution is a Lognormal joined with a Pareto and properly exercises the file system of the web server.

2) The “Request Sizes” distributional model describes the collection of files downloaded according to empirical size measurements. This probability distribution is a Pareto and affects appropriately the network. In general, this distribution is different from the first, because it indicates which single stored file should be downloaded and how many times (i.e. more or equal than zero times).

3) The “Popularity” distributional model describes the number of requests made to each single file on the server. This probability distribution follows Zipf’s Law and is related to the previous two. The distribution of popularity exercises the caches properly, since popular files typically stay there.

4) The “Embedded References” distributional model describes the number of embedded references in a web object. This probability distribution is a Pareto and describes how many objects (images, texts, sounds, etc.) are embedded in each web page.

5) The “Temporal Locality” distributional model describes the probability that a requested file will be requested again in the near future. This probability distribution is lognormal and exercises the caches properly.

6) The “inactive OFF Times” distributional model describes the bursty nature of requests of a user while the “active OFF Times” distributional model reproduces the time to transfer web objects. The “inactive OFF Times” distribution is a Weibull while the “active OFF Times” distribution is a Pareto.

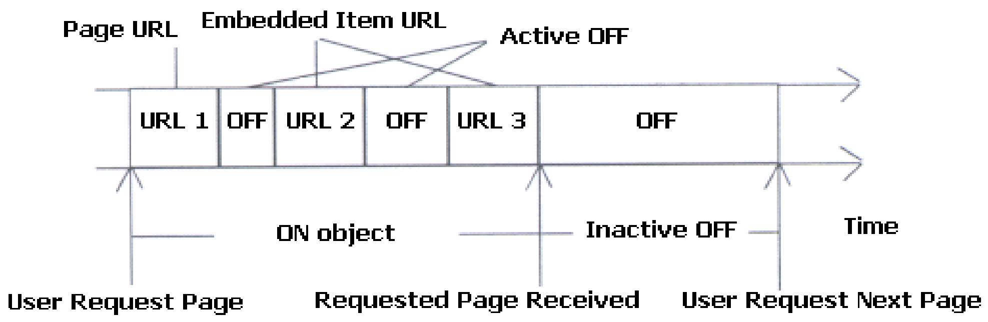


Figure 2.2: Schema of a request from (Barford *et al.*, 1998)

Table 2.2: Distributions used in the workload generator from (Barford *et al.*, 1998)

Component	Model	Probability Density Function	Parameters
File Sizes – Body	Lognormal	$p(x) = \frac{1}{x\sigma\sqrt{2\pi}} e^{-(ln x-\mu)^2/2\sigma^2}$	$\mu = 9.357$; $\sigma = 1.318$
File Sizes – Tail	Pareto	$p(x) = \alpha k^\alpha x^{-(\alpha+1)}$	$k = 133K$; $\alpha = 1.1$
Popularity	Zipf		
Temporal Locality	Lognormal	$p(x) = \frac{1}{x\sigma\sqrt{2\pi}} e^{-(ln x-\mu)^2/2\sigma^2}$	$\mu = 1.5$; $\sigma = 0.80$
Request Sizes	Pareto	$p(x) = \alpha k^\alpha x^{-(\alpha+1)}$	$k = 1000$; $\alpha = 1.0$
Active OFF Times	Weibull	$p(x) = \frac{bx^{b-1}}{a^b} e^{-(x/a)^b}$	$a = 1.46$; $b = 0.382$
Inactive OFF Times	Pareto	$p(x) = \alpha k^\alpha x^{-(\alpha+1)}$	$k = 1$; $\alpha = 1.5$
Embedded References	Pareto	$p(x) = \alpha k^\alpha x^{-(\alpha+1)}$	$k = 1$; $\alpha = 2.43$

2.6.2 GISMO: workload generator for streaming media access

GISMO (Generator of Internet Streaming Media Objects and workloads) is the well-known workload traffic generator that reproduces streaming media access (Jin *et al.*, 2001). GISMO exercises servers and networks in a manner representative of a user requesting streaming media access. This streaming media access is composed by different characteristics: object popularity, temporal correlation of request, seasonal access patterns, user session durations, user inter-activity times, variable bit-rate (VBR) self-similarity and marginal distributions (see the Table 2.3). Thanks to these characteristics, GISMO generates realistic and scalable streams that are useful to study Internet streaming media delivery techniques. To reproduce a realistic synthetic streaming access workload GISMO implements an access model (see the Figure 2.3 and Veloso *et al.*, 2002). In GISMO, a session is the time of service that begins with a download request and ends up with the abortion of an ongoing transfer or with the end of this transfer. The workload exercises servers and networks as the result of two main aspects: the session arrivals and the properties of individual sessions.

The former aspect, the session inter-arrival time is the time spent between the arrivals of two consecutive sessions or of two sessions requesting the same object. The session arrivals are described by three characteristics implemented as distributional model: popularity distribution, temporal correlation, and seasonal access frequency.

1) The "Popularity" distributional model describes the relative number of streaming object made on the server. The distribution of streaming object popularity follows Zipf's Law.

2) The "Temporal Correlation" distributional model describes the time passed, once a file has been requested and it will be requested again in the near future. In particular, the streaming media accesses requested by different sessions to the same object may be not completely independent and they may exhibit temporal correlations. To reproduce this behavior, GISMO assumes that some request arrivals are correlated, while some others are independent. The distribution of the Temporal Correlation is a Pareto.

3) The "Seasonal Access Frequency" distributional model describes the seasonal patterns of the access frequency. Such patterns depend on the type of running application and on different aspects of location and time. GISMO does not make any assumptions about this characteristic because it asserts that the distribution of access frequency at different times should be supplied by its users.

The latter aspect, the properties of individual sessions show the model used for specifying each single user session. An individual session is described by three characteristics implemented as distributional model: object size distribution, user interactivity times, and object encoding.

1) The "Object Size" distributional model describes the streaming object sizes. This characteristic is a main cause for session duration; this is due to the fact that if the download object is larger, the duration of the transfer will be longer. The distribution of the Object Size is a Lognormal.

2) The "User Inter-activity Times" distributional model describes the moment of the user activities. These are two kinds: the "stop" and "fast forward and rewind" activities. GISMO considers the "stop" activity as a partial access and models it as an aborted session. Hence, the distribution of the "stop" activity is a Pareto. Instead, it considers the "fast forward and rewind" activities as intra-session jumps and models them as another Pareto.

3) The "Object Encoding" distributional model describes the autocorrelation of the variable bit rate needed to transfer this object in real-time. Multimedia objects have self-similar characteristics. GISMO models the VBR auto-correlation of a streaming object using a self-similar process and uses a heavy-tailed marginal distribution to specify the level of burstiness of the bit-rate.

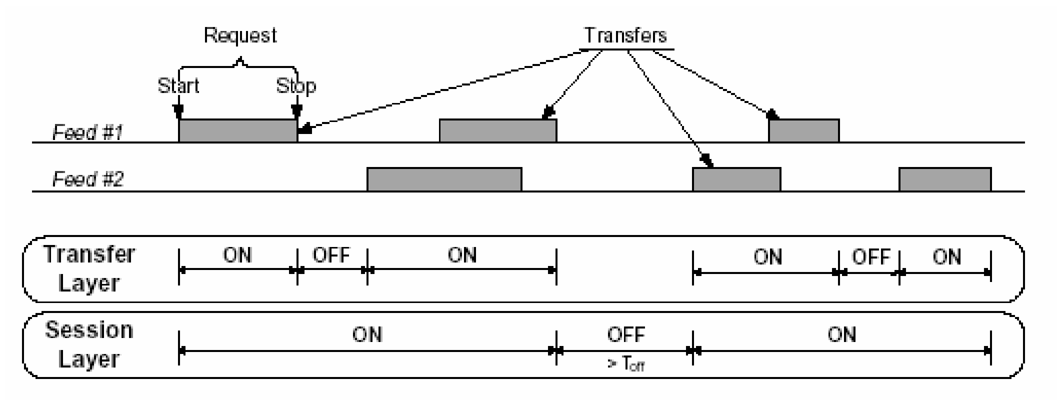


Figure 2.3: Relationship between client activities and ON/OFF times at the session and transfer layers from (Veloso *et al.*, 2002)

Table 2.3: Distributions used in the workload generator from (Jin *et al.*, 2001)

Component	Model	PDF	Parameters
Popularity	Zipf-like	$f(x) \sim \frac{1}{x^{\alpha}}, x = 1, 2, ..., N$	α, N
Temporal Correlation	Pareto	$f(x) = \alpha \frac{k^{\alpha}}{1-k^{\alpha}} x^{-\alpha-1}, k < x < 1$	α, k
Seasonal Access Frequency	User-specified		
Object Size	Lognormal	$f(x) = \frac{e^{-(\ln x - \mu)^2/2\sigma^2}}{x\sigma\sqrt{2\pi}}, x > 0$	μ, σ
User Inter-activities	Pareto	$f(x) = \alpha \frac{k^{\alpha}}{1-k^{\alpha}} x^{-\alpha-1}, k < x < 1,$	α, k
VBR Auto-correlation	Self-similarity		H
VBR Marginal Distribution(body)	Lognormal	$f(x) = \frac{e^{-(\ln x - \mu)^2/2\sigma^2}}{x\sigma\sqrt{2\pi}}, 0 < x < C$	μ, σ
VBR Marginal Distribution(tail)	Pareto	$f(x) = \alpha k^{\alpha} x^{-\alpha-1}, x \geq C$	α, k

2.6.3 D-ITG: a distributable platform for traffic generation

The Distributed Internet Traffic Generator (D-ITG) is a platform able to generate typical traffic patterns at network, transport or application level (i.e. over the IP stack) according to the Inter-Departure Time (IDP) and the Packet Size (PS) assessments of real packet flows (Emma *et al.*, 2004). Along with this approach, a typical traffic flow is identified by the time between the transmission of two successive packets and the amount of bytes transmitted with a packet. The values of the two variables may be modeled by stochastic functions as Cauchy, Exponential, Gamma, Normal, Pareto and Uniform. Further, the D-ITG supports the setting of the generation seed to run reproducible experiments too. The runs of D-ITG with the same seed produce the same traffic patterns.

D-ITG works on different operating systems: Linux, Windows and Linux Familiar. It is implemented as a multithread and as a multitask system and supports TCP, UDP, ICMP, DNS, Telnet and VoIP protocols. D-ITG is able also to monitor and to log the performance of the computers involved in its runs. In particular, to measure the performance it adopts two different modes implemented by One-Way-Delay Meter (OWDM) and Round-Trip-Time Meter (RTTM) (Avallone *et al.*, 2004a). OWDM enables to send packets to a specific host and measure the transmission time of each packet. RTTM enables to transmit packets to a specific host, which sends them back to the sender. D-ITG promotes to store information both on the receiver and on the sender side. In addition, it permits the sender and the receiver to delegate the logging operation to a remote server. This is very useful when the receiver is a PDA or a Palmtop that has a limited storage capacity.

The system architecture (Avallone *et al.*, 2004b) of D-ITG is composed by: an Internet Traffic Generator Sender (ITGSend), Internet Traffic Generator Receiver (ITGRecv), Internet Traffic Generator Log Server (ITGLog) and finally an ITGSend Manager (ITGManager). In particular, ITGSend and ITGRecv are synchronized by means of a Traffic Specification Protocol (TSP). A similar protocol is used between ITGSend and ITGManager. The TSP Protocol creates a connection between ITGSend and ITGRecv, authenticates ITGRecv, communicates the information on the generation of a flow, terminates the connection between ITGSend and ITGRecv. ITGSend can generate traffic according to single flow mode, multiple flow mode and daemon mode. This last mode involves ITGManager. In this case the ITGManager implements the workload generator of the D-ITG platform. For this reason, when ITGSend runs in daemon mode, it stays idle waiting commands coming from ITGManager. The message, sent by ITGManager, to request the generation of a specific traffic flow has the same format of that used from command line. In particular, an ITGManager can control more than one remote ITGSend. Further, ITGRecv runs by waiting for a TSP connection on the 9000 port. ITGRecv writes log files that reports the performance of each flow at packet level. The log file can be stored locally or remotely adopting the ITGLog. Moreover, ITGRecv adopts an authentication phase that prevents to use ITGSend as a Denial of Service (DoS) attacker. In this distributed architecture, DoS attacks may be possible only if the attacker controls the computer where ITGRecv is running. Finally, ITGLog is the server that may receive remotely the log files both of ITGSend and ITGRecv. In particular, it can be used in critical situation as for example in a wide area distributed scenario and when device with limited storage capacity are involved.

2.7 Examples of Topology Generators

As already said, there are different kinds of topology generators. Some generates random topologies (Waxman, 1988), others imitate the hierarchical properties of the Internet (Doar, 1996; Calvert *et al.*, 1997), and some others (Jin *et al.*, 2000; Aiello *et al.*, 2000; Medina *et al.*, 2001) reproduce the connectivity properties of the Internet. Another set of topologies, for which special generators are not required, are regular topologies (such as the mesh, star, tree, ring, lattice, etc). Selecting one of the topology generators depends on several factors: the nature of the study to be performed, the size of the required generated topology, the characteristics that the generated topologies may have, etc. In general, available topology generators can be classified into two categories (Medina *et al.*, 2001). In the first category there are ad-hoc models based on some considerations: Waxman models (Waxman, 1988), GT-ITM (Calvert *et al.*, 1997) and Tiers (Doar, 1996). Instead, in the second category there are models based on measurements. This last category can be further divided into two sub-categories: causality-oblivious and causality-aware models. In the former, there are causality-oblivious models that reproduce the power-law distribution (Faloutsos *et al.*, 1999; Jin *et al.*, 2000; Aiello *et al.*, 2000). These models reproduce well, for example, the outdegree distribution of Internet topologies, but do not provide any insight concerning why such properties arise in the Internet. In the latter, there are causality-aware models that try to model basic causes of the insights of a topology. Example of the possible insights may be why the network grows incrementally, and new nodes trying to connect to higher-degree nodes (Barabasi *et al.*, 1999; Medina *et al.*, 2000) (Medina *et al.*, 2000) may belong to a causality–aware sub-category because its main model tries to reproduce the origin of the power laws in Internet topologies. Finally, we note that not all existing topology generation models have been implemented, like the "small-world" model of (Watts, 1999). A brief description of the main topology generators follows.

2.7.1 Waxman

Waxman (Waxman, 1988) is one of the first topology generators. This generator produces random graphs and it includes also network specific characteristics. The Waxman model adopts a probability function to interconnect two nodes, which is parameterized by the distance that separates nodes in the plane. It has several drawbacks. First, it does not reproduce significantly the topological maps of real networks. This means that there is no clear hierarchical structure and the distribution of the links across the network is improbable. Second, it does not guarantee any connected network. Once it you generated a network, you must control the connectivity. If this propriety is absent, you must discard or modify the generated network. Third, in the network the number of nodes and links grows in a similar way. This is not likely in real networks because new links are added for redundancy and not because more nodes joined the network.

2.7.2 GT-ITM

GT-ITM (Calvert *et al.*, 1997) is one of the most famous packages to generate and analyze graph models of inter-networks. The main component of GT-ITM supplies the Transit-Stub (TS) model (Zegura *et al.*, 1996) that is useful to replicate the hierarchical structure of the Internet. The Transit-Stub model does not support the representation of host systems, but all nodes are routers. As already said in the previous section, first, the implementation of the TS model generates a connected random graph using a variation of Waxman method. Second, it produces connected sub-graphs by repeatedly expanding each node of the first graph and checking the graph for connectivity. If the graphs are not connected, it discards them. In this way, the final graph is taken at random from all possible connected ones. Generating a final connected graph may take a long time if the number of edges is small. Extra edges are added randomly from stub domains to transit or from stub to stub. Further, this package equips nodes and edges with different types of information. In fact, each node has a label that shows if it belongs to a transit or stub node, toact as an identifier to indicate the domain to which it belongs, and to find it inside the domain. In addition, stub nodes show, the default gateway for the transit domain. Each edge has a weight that can be used to route packets according to an adequate routing policy. The GT-ITM release is written in C and includes the necessary libraries from the Stanford GraphBase (Knuth, 1994).

2.7.3 Tiers

The Tiers (Doar, 1996) is another generator for the reproduction of the Internet topology. Tiers builds network topologies based on a three-level hierarchy, called tiers. The three levels of hierarchy are referred to as Wide-Area, Metropolitan-Area and Local-Area networks. The WAN, MAN and LAN levels correspond respectively to the transit domains, stub domains and basic domains. The total number of the transit domains is one, so this tool is able to describe just a single WAN. Tiers generate each single connected sub-graphs (corresponding to single domains) by means of the minimum spanning tree that joins its nodes. The minimum spanning tree is used in general as the basis for laying out large networks. In this tool, we have two levels of redundancy: intra-network and inter-network redundancy. Intra-network redundancy means that extra edges are added to the closest nodes in the network. Instead, Inter-network redundancy means that extra edges are added to the closest nodes in the next higher domain. These two levels of redundancy should guarantee some degrees of local connectivity within geographic constraints. It is possible to represent a lot of connected hosts. In particular, when a node is a gateway node, the node has two instances: a LAN instance node and a MAN instance node. The information exchange between the two instances of nodes should reflect the processing delay and bandwidth constraints within the host. Tiers is written in C++.

2.7.4 Inet and PLRG

Inet (Jin *et al.*, 2000) and PLRG (Aiello *et al.*, 2000) are two generators that try to reproduce the connectivity of Internet topologies as reported in (Faloutsos *et al.*, 1999). At the beginning, these generators use a power-law distribution to compute the node degrees and, afterwards, interconnect nodes adopting different rules. After the first phase Inet controls if the resulting topology is connected, it joins nodes with a degree greater than two by means of a spanning tree, it attaches nodes with a degree one to the generated spanning tree and, finally, deals with the remaining nodes. After the generation of the node degrees, PLRG clones each node with *d* degree *d* times and, finally, the resulting nodes are randomly joined.

2.7.5 BRITE

The state of the art presents many specific topology generators. This may result disadvantageous for many reasons. In this way, a simulationist is forced to learn the differences of any topology generator. This drawback forces, usually, the simulationists to use the most popular, the best supported or the easiest one. In the literature, we find different topology generators that adopt different contexts and have different goals. This fact implies that the simulationists should use different generators. In particular, the high number of different generators available makes comparative analyses more difficult. Another drawback is that developing a new generator is difficult and available generators are not easily extensible. The real challenge can be described into two issues. The former is how it is possible to develop a generation tool that represents an interface between the Internet research and the topology generation research. The latter is how it is possible to design this tool to facilitate topology generation research. BRITE 1.0 (Medina *et al.*, 2001) is a universal generation tool that tries to answer to these two questions. BRITE was born as an alternative to model-oriented topology generators that are very specific. All other generators described above fall in this category. Instead, a universal generator should not adopt only a specific set of models, but it should be extended to permit the addition of new models in an easy way. This means that a universal topology generator should be flexible and adaptable and should generate representative topologies that should be employed in different simulation scenarios. Further, it should be as efficient as possible without losing robustness, it should integrate existing network topology models and be user-friendly. BRITE is designed as a flexible topology generator that is not constrained to support any particular topology generation model. BRITE supports multiple generation models by implementing a model with different degrees of freedom concerning the layout of the nodes and the method to join them in a connected graph. It also provides the capability to import topologies generated by GT-ITM, Inet, Tiers, and BRITE itself, or topological data collected from real networks. In particular BRITE imports topological data from itself to combine them with other models or other imported formats. This tool can generate output in its own file format or as the input for the settings of simulators, like, NS (Fall,1999) and the Scalable Simulation Framework (SSF, 2004). Furthermore, BRITE includes an analysis engine, called BRIANA. BRIANA supplies a set of analysis routines that should be useful to evaluate any topology model imported into BRITE. The Architecture of BRITE contains a Model and a Graph as input, a set of exporting methods as output and function members. The Model data member can be implemented with multiple specific generation models. It is possible to specify in a topology the use of one or more instances of the available generation models. The Graph data member can be implemented with the minimal functionality required by the generation models. It is possible to extend or re-implement new or more complex functionalities without rewriting the remaining code. Finally, the set of export methods helps us show the output BRITE topologies into specific formats.

The setting of a topology depends on specific generation models being used. In general, in BRITE, the generation process of a topology is divided into a four phases. In the first, the layout of the nodes is set up. Then, the nodes are connected. The third phase assigns the value to the properties of topological components. Finally, the topology is shown in a request format.

Chapter 3

Network Emulation: Background

A possible alternative to the synthetic simulation approach is emulation. We cannot consider emulation as an alternative to simulation or test development, but rather as a complementary approach. This approach exploits workstations connected by means of real networks that run modified operating system to simulate slower links and larger propagation delays. In general, during an emulation, simulated time and real time are very close (i.e. five minutes of simulated time takes five minutes to run). The main advantage of this approach is that an emulator runs real protocols. This means that the behavior of a real protocol and all its implications (like processing overheads for example) must not be simulated. Unfortunately, this approach shows some limitations due to the hardware on which emulation runs. In particular, these disadvantages highlight that: i) to emulate a network with ten hosts one needs ten workstations, ii) it is not possible to emulate a host or a link which is faster than its hardware and iii) the real time scheduling of the software is constrained to the frequency of the specific used hardware. This last consideration draws attention to the available time resolution, which can affect the accuracy of the emulation (Brakmo *et al.*, 1996). In fact, if an emulator has a time accuracy of 10 ms, it cannot emulate the exact delay involved in sending packets smaller than 20 Kb on a 2Mb/s link, because it takes exactly 10 ms to send this amount of bits.

As to network emulation, it promotes the integration of the real network into an emulative environment, leading to an emulative scenario. Special interfaces called emulation interfaces are provided to introduce live network traffic into the emulative environment and to inject simulated traffic from the emulative environment into the live network. The role of the network emulator can be different depending on it appearing to the end systems as a router or as another end system. In the first case, live traffic can pass through the emulator (transparently to end systems) and it is affected by objects (for examples different kinds of queues, simulated nodes if the emulation exploits a single computer or different instances of the emulators if the emulation is fully distributed) within the emulative environment, or by other traffic on the live network. In the second case, the emulator is viewed as an end system, so it can generate or receive live traffic by itself. This means that it can include traffic sources for the generation of live network traffic as a source host or it can receive the communications from real-world entities as a destination host. The former approach is currently more adopted than the latter (NS manual, 2003), as the simulationists prefer to separate the generation of the traffic (due to the protocol under study) from the emulation of the communication system. In essence, there are two different modes to manage the packets captured from the real network:

- *Opaque mode*, where live data are treated as opaque packets, and
- *Protocol mode*, where live data are interpreted or generated directly by the emulator.

In *opaque mode*, the emulator does not interpret the network packets. This means that the protocol fields of the real packets are not manipulated by the emulator. Hence, live data packets may be dropped, delayed, re-ordered, or duplicated, but it is not possible to process specific protocol traffic manipulation that changes one or more values of the fields of the packet headers (e.g. it is not possible to drop the IP packet with sequence number 1123). Instead, in the *protocol mode*, it is possible to manipulate the values of the fields of the packet headers by interpreting or by generating new arbitrary field assignments.

In the following Sections, we provide an insight of the proprieties of the network emulation and examples of the network emulators.

3.1 Emulation Abstraction Layer

The main aim of any network emulator is to reproduce the behavior of a network scenario in order to study the performance of the integration between this scenario and the software communications over it. Therefore, each emulator needs a *network model* to mimic a specific network scenario and to characterize its properties. The *emulation interface* is a combination of both software and hardware that provides an entry point to the protocols (or applications) under study within the implemented network model. The emulation interface is in charge of integrating the emulated network and its live traffic coming from protocols (or applications) under test, operating either in opaque or in protocol mode. These protocols (or applications) can be tested at different layers and then the emulation interface must supply services at different layers, named the *emulation abstraction layers* (see the Figure 3.1). The choice of an emulator for a specific scenario depends on the emulation abstraction layer it supplies for the study of the protocols (or applications) under test. Providing an adequate emulation abstraction may be made at the following layers.

• *Transport Layer*, here emulation reproduces the features of the communication channel at the transport level (of the stack ISO-OSI), e.g. a TCP or an UDP. The entry point of this kind of emulators is set at transport layer, so they promote the measurement of the performances of the applications on their emulated communication channels.

• *Network Layer*, here emulation mimics the end-to-end behavior of a network connecting hosts at the network level, by reproducing the performance of the network protocol due to packet delays, congestion losses, etc. The entry point of this kind of emulators is set at network layer, so they promote the measurement of the performances of the transport protocols and the applications on their emulated end-to-end behavior of a network.

• *Link Layer*, here emulation replicates the behavior of a single network links at the data-link level due to limited bandwidth, frame delay, etc. The entry point of this kind of emulators is set at Data-Link layer, in addition to the measurement of the performances of the network protocols, they promote the transport protocols and the applications on their emulated links.

The major research issue of every network emulation approach is how to determine the maximum possible conformance of the network model implemented by the emulation abstract layer to the behavior of a real network. Each aspect of the network model should reproduce physical effects, hardware design, and protocol issues of a real network. Hence, these considerations show that it is more difficult to realize realistic emulation on a higher abstraction layer than on a lower. This is due to a great number of factors that one should consider. For example, network layer emulation should reproduce the effects of the network level, like routing, queuing, discarding etc. Instead, link layer emulation approach can exploit the actual implementations of network protocols to create these effects. From this point of view it emerges that lower emulation abstraction layer will carry out more realistic results. The authors of (Herrscher *et al.*, 2002b) say that emulation on link layer is the lowest possible emulation abstraction without using special hardware.

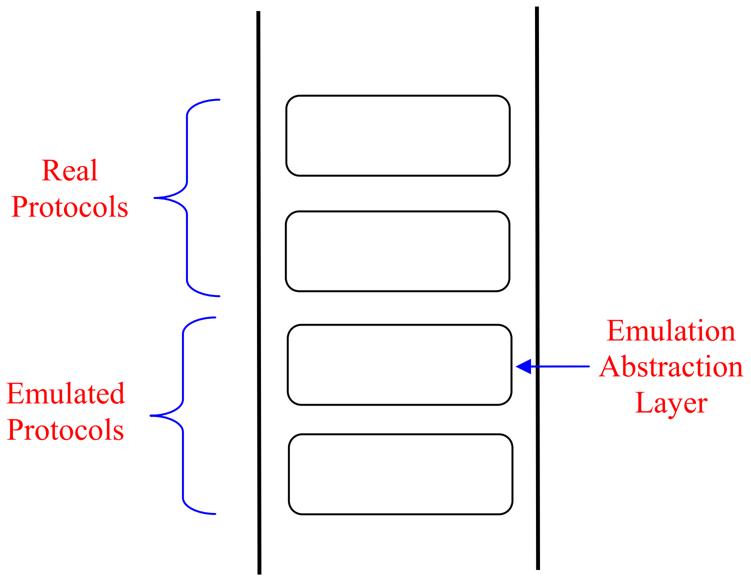


Figure 3.1: Real and Emulated Protocols

3.2 Categories of Emulators

There are three possible categories of emulators: central-control, simulation-combined and trace-based network.

Central-control emulators abstract the network infrastructures to a model with a set of parameters. This kind of emulators exploits these parameters to reproduce the behavior of the features of the network infrastructures by creating, for each packet that passes through it specific network, conditions and traffic dynamics. An emulation experiment is conducted by connecting hosts to the central-control emulator. Central-control based emulation may be designed for a particular network such as IP or ATM for example, or alternatively it may be built to capture some general characteristics. In this latter case, we have so the called several general purpose emulators. Some well-known genera- purpose network emulators are ONE (Allman, 1997), Dummynet (Rizzo, 1997), and NIST NET (Carson, 1997; NIST NET, 1999). Typically this kind of emulators work at network layer, hence the parameters supported include packet delay distribution, packet drop pattern, bandwidth, and queues. This approach achieves good results by reproducing wired network infrastructures even though it does not manage to emulate dynamic network conditions well. Therefore, central-control emulators may be not able to support the prerequisites of mobile wireless networks. The main advantage of central-control emulators is the easy configuration that allows one to analyze the performance of end-to-end network protocols by tuning emulative parameters. However, this approach is not able, in general, to supply an emulative environment for the evaluation of topology-related protocols such as for example multi-hop ad hoc routing.

Simulation-combined emulators expand their capabilities by means of simulation technologies. Hence, according to this approach it is necessary to adopt, during an emulation experiment, the abundant resources available in a network simulation. In 1999, Fall equipped the well-known VINT/NS simulator with an emulative interface and a real time scheduler. As VINT/NS (Fall, 1999) is an extensible and powerful event-driven network simulator exploiting protocol modules, algorithms, traffic generator, topology manager and visualization tools, it was showed that it was possible to combine simulation and emulation. The main advantage of this approach is the possibility to exploit a large amount of simulation resources in the central emulator. Differently from the central control approach that supplies a limited number of network parameters available, a simulation-combined emulator can support an environment for the evaluation of topology-related protocols. However, this kind of emulators inherits the intrinsic drawbacks of network simulation.

The last category shows the *trace-based network* based emulators that exploit three different phases for network modeling: data collection phase, trace distillation phase, and modulation phase (Satyanarayanan *et al.*, 1997). The first phase collects traces from the network infrastructures. The second phase builds a network model by means of collected traces. The final phase mimics the traced network effect in a laboratory. Differently from the other two approaches that can only try to approximate the network effects, the trace-based network emulators are interesting because they mimic these effects derived from real traces. However, the main disadvantage is that this approach can provide a reproducible environment for the performance evaluation of end-to-end protocols (and applications) only of the collected network traces.

3.3 Parameters

The parameters of a network model depend on the granularity of the emulation environment and on the emulation abstraction layer (Dam *et al.*, 1998; Herrscher *et al.*, 2002a). The main purpose of different emulators is to reproduce networks consisting of several links, routers, hosts, etc. This means that the parameters of a network model may be for example (corresponding to the previous components) delay variation, router queuing, packet reordering. In particular, the parameters concerning the modeling of a point-to-point or a multipoint link are:

- *delay:* fixed delay (ms), variable delay (stochastic function leading to a ms value)
- *bandwidth limitation:* maximum bandwidth (bit/s), queue length (bytes or packets), type of bandwidth sharing (simplex, half-duplex, or duplex)
- *packet loss:* probability value, additional burst loss model

The *delay* of a packet between two adjacent hosts consists of different possible components: propagation delay, medium access delay and serialization delay. The propagation delay is due to the type and the length of the channel between the two adjacent hosts. Further, it may be comprehensive of additional delays due to the presence of special devices, like repeaters. If the communication between the two hosts does not depend on the mobility (i.e. emulation of wireless network) the propagation delay will be constant. Other possible additional delays may come from particular link layer protocols that require waiting before starting to send the bits. The medium access delay may be described by means of stochastic models. A limited bandwidth may introduce the serialization delay that represents the time spent to send and to receive a packet. This component is a function of the bandwidth and the respective packet size.

Bandwidth has the generally means how much information can be carried during a specific time period (usually a second) over a communications link. For example, a link with a broad bandwidth is one that may be able to carry enough information to support the succession of images in a video presentation. More technically, bandwidth is the width of the range of frequencies that an electronic signal occupies on a given transmission medium. The bandwidth is expressed in bits (of data) per second (bps), (i.e. a modem that works at 57600 bps has twice the bandwidth as one that works at 28800 bps). We remember that a real communication path usually consists of a succession of links, each with its own bandwidth. If one of these is much slower than the rest, it is said to be a bandwidth bottleneck. Hence, the maximum bandwidth of a connection can affect link performance. The behavior of the bandwidth limitation may be described by a "leaky bucket". The bucket has a hole that permits only to the maximum bandwidth bps to transit into the link. Further, the bucket also has the capacity represented by a FIFO queue where it can store the surplus traffic. If the queue is full then the incoming packets are dropped. In particular, the bandwidth limitation in a link can be different depending on the direction of the connection. This means that the bandwidth can be limited for each direction separately. In particular for each direction the bandwidth might be shared by all communications of a multipoint connection.

Two are the main causes for *packet loss*: link congestion and transmission errors. The first cause, as explained before, is due to the behavior of the bandwidth limitation that drops the surplus packets when link congestion is detected. The second cause is due to possible transmission errors. This highly depends on the medium of the link. The first cause is more likely than the second. The packet loss probability is a very dynamic parameter. As a consequence, a realistic emulation model has to provide this flexibility.

Before concluding this section, it is important to highlight that each parameter partially depends on the other parameters. In fact, the serialization delay depends on the bandwidth limitation as well as the packet loss due to the link congestion. Further, the queue delay depends on the propagation delay of the FIFO queue.

3.4 Routed Emulation VS Intermediate Emulation

As to those emulators that belong to the central-control category and adopt the network layer as entry point, they must allow the simulationists to integrate real computing objects (such as network workstations) into the emulative environment. These network objects may be confined into a lab-based environment and may support the run of complex distributed applications, while the emulator simulates the effects of their execution in wired or wireless networks. To exploit their integration capability, the live traffic produced by the real objects should arrive to or depart from the emulator. In particular, for IP based network emulator, this is possible by making an emulator able to route IP packets exchanged between source and destination host as an emulative stream. As already said, the IP protocol guarantees the connectivity between different network technologies and different administrative domains. Hence, each network emulator should be able to route IP packets as typical routers or default gateways do. This approach is named Routed Emulation (RE). A possible alternative approach is the ability to maintain consistent networked communications on the emulative scenario. The idea is to send packets for a certain destination to a virtual address of the emulator. The emulator, aware of the fact that these packages are not directed to it, introduces them into the emulative environment, and afterwards it sends them to the final destination. This other approach is named Intermediate Emulation (IE) and the computer that exploits this ability is called Intermediate Emulation Server (IES). In general running in RE mode is a stronger condition than in IC, because the first mode implies the second. Really, the RE is not essential and in some cases it may represent a constraint. In this way, the emulator becomes the default gateway of all computers involved in the scenario; this is not good because if the emulative experiment is conducted in a (public) laboratory it can interfere with any other interconnected machine that does not participate in the experiment. In general a lot of network emulators work in RE mode.

Here below, two methods that allow emulation consistency or routing are described: virtual host and virtual router method (Bradford *et al.*, 2000).

The *virtual host method* supports the use of virtual IP number to re-address the hosts that belongs to the emulative scenario. Suppose A and B hosts with respectively 192.168.1.2 and 192.168.1.3 IP addresses want to communicate by means of the E emulator with 192.168.1.1 IP address. Further, suppose that A wants to communicate with B without adopting the 192.168.1.3 IP number, but rather by using a virtual address of an unused network, like the not routable 10.0.0.0 class. In particular, each real IP address should have a corresponding virtual address. In this way A can contact B sending packets addressed to the virtual host 10.0.1.3. At this point, the emulator must listen to the network to intercept all packets addressed to 10.0.0.0 class and after reading them for its own use it must route to the final real host. Inside the emulator, each packet can be manipulated (i.e. a protocol mode is adopted) before delivering it back to B destination. In fact, in our example, when each packet enters the emulator, the real IP source address is converted into the virtual 10.0.1.2. Then, just before being put back on the interface destined for B, the virtual destination address 10.0.1.3 is converted into the real 192.168.1.3. It is important to highlight that the emulator can perform these operations only if it is able to respond to ARP requests for the virtual addresses. This means that the default gateway (192.168.1.254) must route the packets from A to B addressing the 10.0.0.0 virtual network to E. If the emulator is not able to respond to these ARP requests, the default gateway will send these packets to the wider Internet. The address of the default gateway (192.168.1.254) must be otherwise unused on this network to prevent other hosts mistakenly trying to route these packets. This configuration permits to connect directly A with B (without passing through E) by using the default routes for the real address of B. Moreover, this configuration prevents any kind of interference between the users of the emulator and the others. The main disadvantage of this method is that it fails to apply protocols where IP addresses are carried as data rather than information to route a packet. Examples of such protocols include Quake network applications and IRC. This could be resolved by using protocol specific mapping routines in the emulator that can rewrite data segments. This method allows the network emulator to behave only as IES because it is in charge of the consistency of the emulative stream, but not of routing.

The *virtual router method* uses the real IP addresses of the host involved in the emulative scenario. Suppose that A and B hosts stay on the same network; in this way they can directly send packets to each other without passing through any other computers. This means that the packets exchanged between A and B does not exploit the emulation facility. We can prevent this problem by configuring the A route to B via E emulator. Vice versa, we can configure similarly the B route to A. Hence, the proposed configuration allows packets to pass through the emulator as required. In particular, the E address must be a virtual address reachable by ARP request and able to answer by RARP response. In fact, if the real IP address of the emulator were used, most likely the top host operating system on which emulator runs could not route correctly the packets avoiding the emulation environment and, could even duplicate them. The main disadvantage of this method is that a route must be configured on each machine for each real destination within emulative scenario. An appropriate use of the netmask technique can attenuate the problem above. However, this method forces all packets from A to B (and vice versa) to pass through the emulator. This means that this method interferes with the other users on A and B submitting their packets to the emulation. This method imposes the network emulator to behave as default gateway or IES.

To conclude this Section, we show a method to simulate internal hosts that may appear "real" to real hosts outside. These simulated hosts, like real ones, respond to pings, and show up on traceroute through an emulator and may offer other different services. It is possible to implement small mechanisms working on the IP protocol. The *internal host method* forces the emulator to respond to ARP requests on behalf of the internal hosts. In this way the real hosts can send packets to internal ones. The IP address of the internal hosts can belong either to the same network of the real hosts or to other virtual networks.

3.5 Lab-Based Environment

Networks vary widely in scale and performance, from simple local networks connecting few machines to those spanning the globe, connecting millions of machines. It is clear, therefore, that there is no single type of network suitable for all users. In order to try to provide a satisfactory compromise, networks are commonly interconnected. The interconnection of networks provides additional complications due to the wide range of heterogeneous network types that may require connection. Similarly, there are also large variations in the scale of distributed applications that are connected by such networks. Applications may be thought of as being widely distributed if they involve many machines or machines separated by large distances. Typically, such applications must tolerate reduced network performance in comparison to those communicating over a purely local network. The aim of network emulation focuses on the possibility to reproduce in a laboratory environment, through an emulator, large-scaled scenarios like wide area networks or wireless networks. This means emulating wide-area or wireless network characteristics on one or more interconnected local area networks (LAN). To reach this aim we must describe the main differences between networks on: delay, packet loss and different bandwidth. Hence, the emulator running on a LAN should mimic accurately the performance of wide-area and wireless networks to be effective. Regarding delay, there is at least an order of magnitude of difference in favor of LANs. Apart from this obvious difference, delays remain approximately constant in a LAN. Instead they are very variable over time (for the same path) in a wide-area or wireless network. In particular, delays between two hosts in the previous two network types depend on causes like network load, currently used route etc. Differently from a wide-area or wireless networks that lose packets, on a LAN they are rarely dropped. This makes a substantial difference in protocol performance, because in the reliable LAN the duplication of the packets due to their loss never occurs. The last difference depends on the fact that wide-area or wireless networks consist of heterogeneous hardware or moving devices, so the value of the bandwidth is due to the different bandwidths over different paths. To sum up, the performances of a LAN are better than those of wide-area or wireless networks; so it is possible to use a LAN to reproduce a wide-area or wireless scenario reducing the LAN performances. In fact we can manage the best characteristics of a LAN to reproduce the behavior of the less ones of wide-area or wireless network. In this way it is possible to reproduce complicated and large-scaled scenarios in a laboratory environment.

3.6 Goals

After the explanation about the network emulation and which are its main features, we want to specify which are the goals that must be fulfilled (Dam *et al.*, 1998).

- The design of the emulator should not prevent the use of any specific operating system or hardware in the emulative scenario.

- The operation of the emulator should be transparent to real protocol implementation under study. This means that a user should not have to rewrite his/her code nor have to recompile the operating system on his/her computer to take advantage from the emulative environment.

- If delay, bandwidth and packet loss are the characteristics that describe the network emulation, they should be flexible enough to reproduce a wide range of network conditions (we are considering the network layer emulation).

- The emulator should be easily configurable by means of a Graphical User Interface. Further, by exploiting this GUI it should be possible to update continuously the emulation over time.

- The emulator should support the management of log files where useful statistical information is reported.

- The emulator should support a range of possible settings varying from simple to sophisticated.

- The installation of the emulator should be simple and not invasive. This means that it does not involve any changes to a typical laboratory environment. In particular the emulator should not run on the default gateway machine.

- Again, the emulator should be able to accept and manage adequately packets sent from end-systems belonging to different LANs. The LANs may be far.

3.7 Examples of Network Emulators

The literature on network emulation shows efforts by various groups. Modeling the properties of computer networks is the first step for both simulation and emulation. Until now, a lot of network emulators have been created. These emulators present widely varying characteristics. There is a large number of central control approaches. These include Dummynet, NISTNet, ONE. Some emulators improve performance implementing parallel and distributed systems. This includes ModelNet, IP-TNE, VSI and NetBed, NetWire, NETShaper. Further, some of them target a specific area of research interest such as a particular network or protocol like EMWIN (wireless communication). Others may use a very specific approach, like Delayline that works at the application level rewriting the socket system calls, or BBN that is a hardware emulator.

3.7.1 DUMMYNET

One of the most well-known network emulators is Dummynet (Rizzo, 1997). Dummynet is a simple, flexible and accurate network emulator that was built with modifications to the communication protocol stack. Dummynet runs on a standalone system by intercepting communications of the protocol layer under test and by introducing them into an environment where the effects of finite queues, bandwidth limitations, communication delays and packet losses are exploited. In simple words, this tool suggests an approach to insert a model that reproduces a simplified network in an operational protocol stack and to conduct trails on a standalone system. Dummynet enjoys the advantages of both simulation and testbed: control over parameters of emulation, simplicity, capacity to use real traffic generators and real protocol implementations. This tool allows to carry out experiments with network protocols by running a set of unmodified real world applications like FTP, SSH and HTTP on a workstation.

Dummynet is inserted between the interfaces of the network layer and the transport layer of the protocol stack of the computer on which is executed (see the Figure 3.2). Therefore, it reproduces the network infrastructures between two host including two different types of elements in the stream of the communication: routers and links. In order to simulate their presence, respectively a router is represented as limited queue and adopted queuing policy, while links are represented as bandwidth limitations, communication delays and packet loss. The basic configuration of a trail consists of one or two routers and one link. These elements are modeled by means of a couple of queues (i.e. router-queue and link-queue) that intercepts the communications between the protocol layer under observation and its lower layer. The couples may be two, namely one for each direction of the communication. The router-queue is characterized by a maximum size, while the link-queue by a bandwidth and a communication delay. When each communication direction is used, packets are inserted in the router-queue until the maximum size is reached. At this point, it is possible to adopt a queuing policy and eventually to reorder the packets randomly. After this point, packets are shifted from router-queue to link-queue according to the bandwidth limitation. The link-queue adopts a FIFO policy. Packets stay in this queue for a number of seconds. After this period, they are removed and sent to the protocol layer under observation. At this point random packet losses may occur. In particular, the shifting from router-queue to link-queue and the departure of the packets to the next protocol layer are executed according with a T periodic task (where T is a submultiple of the communication delay).

Dummynet like the other network simulators and emulators show some limitations to reproduce the behavior a real system. The main limitations come from the granularity and the precision of the system clock. The granularity limits the resolution in all timing related measurements. This is due to the modeling of high-speed networks where the packet delay becomes comparable with the time quantum of the system clock.

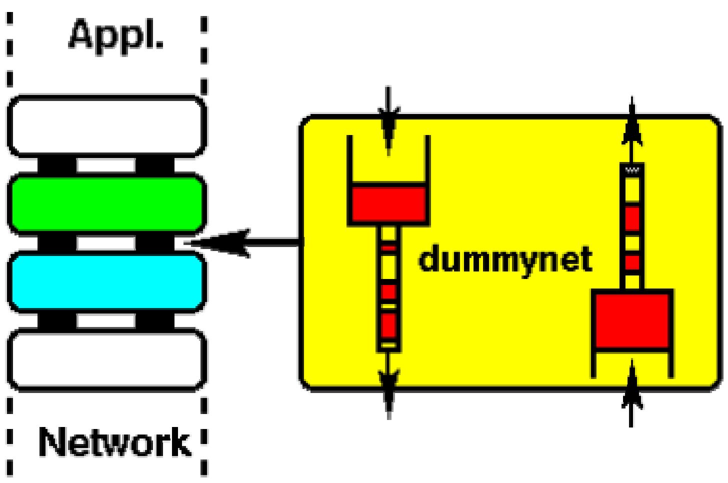


Figure 3.2: Architecture of Dummynet reported from (Rizzo, 1997)

Another limitation is that the overload of a time-driven system might deliberately delay some packets or might miss one or more timer ticks. Dummynet works adopting a time-driven approach with T period and its emulative environment is synchronized with this period. This kind of synchronization never occurs in real-world networks and might hide or amplify some phenomena. Finally, we conclude by highlighting that only a prototype implementing this approach between IP and TCP exists.

3.7.2 NETBED

NetBed (Lepreau *et al.*, 1999; White *et al.*, 2002a-b) is the improvement of a network experimentation platform that focused on efficient setup and control over emulated topologies, named Emulab. The main aim of Emulab was to make facility available to any simulationist and easy to use. Hence, it makes the automation of testbed configuration possible and it allows both interactive and programmatic exploration of different experimental conditions. NetBed introduces simulated and distributed nodes into the original platform and allows their simultaneous use alongside emulated nodes in mixed, virtual topologies. Reaching this aim is possible for NetBed integrating emulation, simulation, and live network experimentation into a common framework. The idea behind NetBed allows to create a virtual topology configuring a set of distributed, emulated, or simulated nodes, by means of either graphical tools or via command-line scripts. For this reason, an experiment is defined by its configuration and any run-time dynamics (e.g., traffic generation) specified via general-purpose interface (for example: NS/Tcl scripts). The automated facility of NetBed includes different settings at different levels of configuration. In fact, it is possible to manage experimenter accounts, set emulated link characteristics, and map the virtual nodes. Furthermore, it is possible to link to physical resources, downloading clean disk images on emulated nodes, setting up network interfaces and IP addresses. Once an experiment is configured, a simulationist can interactively log into emulated or distributed nodes. This is possible also because a NetBed experiment may last from a few minutes to many weeks. All aspects of the experiment can be controlled via web interface, giving simulationists the possibility to make multiple runs, to change their parameters or to collect long-term data. Web interface of NetBed allows experimenters to create, pause, and terminate experiments remotely. NetBed provides also batch experiment system that permits to the simulationist to submit a configuration script over the web. When enough hardware resources become available to run the experiment, it notifies to user that the experiment has started.

One of the main advantages of NetBed is to support heterogeneous resources. In fact, a complex scenario inclusive of links and nodes may be emulated by traffic shaping, like Dummynet, may be simulated via NS, or may be realized by exploiting real wide-area links. In particular, NetBed exploits the emulation facility of NS (NSE) to transfer packets passing through the simulative environment to the live networks. One of the most important features of NetBed is the consistent interface that provides to control nodes and links regardless of their implementation (distributed, emulated, or simulated). The integration of heterogeneous resources is largely enabled by a database. The database stores the heterogeneous resources and serves as a level of indirection between front-end, general-purpose tools, interfaces, back-end and domain-specific implementations. It presents a consistent abstraction of heterogeneous resources to higher layers of NetBed and to simulationists.

Finally, the infrastructure of NetBed may be sufficiently flexible to incorporate wireless mobile virtual nodes and different link types. This possibility will bring important practical benefits to experimentation in wireless and mobile domain, including:

automated and efficient creation of virtual topologies, efficient use of resources through time- and space-sharing, increased fault-tolerance through resource virtualization, capability to leverage existing tools and easier validation across experimental techniques.

Easiness of use and automated configuration are necessary to enable qualitatively new approaches. These permit a speed up of the time needed to launch again a NetBed experiment. In particular, experiment setup cost is even higher in wireless and mobile domains, which require careful measuring of interference effects and "walk-through" experiments.

Efficient use of scarce and expensive infrastructure is also important and a sophisticated testbed system can markedly improve utilization. This approach tries to minimize the value of the time-sharing (i.e., "swapping out" idle experiments) and space sharing (i.e., isolating multiple active experiments). The resource efficiency is more important for wireless and mobile devices since they are less prevalent than PCs. NetBed virtualizes node names and IP addresses so that nodes and links form equivalence classes. Any nodes with the same properties and interconnection characteristics are suitable candidates for a specific class. While virtual nodes may be explicitly bound to specific physical hosts, the flexibility to allocate from an equivalence class adds a measure of fault tolerance. This approach is useful in that infrastructure where node or link failures are anticipated like large-scale clusters, wide-area networks, or unstable wireless environments. Incorporating wireless and mobile devices under the NetBed infrastructure brings a mature set of tools and features to these domains. The NetBed extends consistent control and specification of traffic generators across all experimental methodologies and allows one to control applications running on wireless nodes by means of the current event system and standard commands.

The common NS interface makes it easier to compare experimental environments, thereby facilitating and encouraging validation. Extending this capability to wireless and mobile nodes will provide an automatic way to compare simulated radio propagation models with real devices.

This new hybrid approach of experimentation, incorporating resources from multiple experimental environments, can simultaneously leverage the particular strengths of each. In particular, a simulationist can leverage the greater scalability of simulation without surrendering confidence in the accuracy of the results. According to this approach it is possible to replace a small “island” of simulated nodes with physical nodes, using live wireless communication amongst them. Moving or removing this “island” of nodes from the experiment will yield similar results if the simulative model and the real instantiation of this model are equivalent. The integrations of such close interactions between live traffic and simulation require careful synchronization between simulated and real time.

3.7.3 MODELNET

ModelNet (Vahdat *et al.*, 2002) is a scalable and comprehensive large-scale network emulation environment. In ModelNet, real applications run on edge nodes configured to route all their packets through a scalable core cluster, physically interconnected by gigabit links (see the Figure 3.3). This core is responsible for emulating the characteristics of a specified target topology on a link-by-link basis. In ModelNet, simulationists execute a configurable number of instances of target applications on Edge Nodes within a dedicated server cluster. Each instance is a virtual edge node (VN) with a unique IP address and corresponding location in the emulated topology. Edge nodes can run any OS, any IP network stack and they may execute unmodified application binaries. To the edge nodes, an accurate emulation is indistinguishable from direct execution on the emulated network. ModelNet emulation runs in real time, so packets pass through the emulated network with the same rates, delays, and losses as a real network. Standard administrative commands configure edge nodes to route their network traffic through a separate set of servers acting as Core Routers. The core nodes are equipped with high performance hardware and modified FreeBSD kernels that enable them to emulate the behavior of a configured target network under the offered traffic load. The core of the emulation (i.e. the set of core nodes implementing the emulative environment) routes traffic through a network of emulated pipes, each with an associated packet queue and queuing discipline. Packets move through the pipes and queues only by reference. In this way, a core node never copies packet data. Packets enter the queues coming from VNs or exit from upstream pipes and go on through other pipes according to their specified bandwidth, latency, and loss rates. Each queue buffers a maximum capacity that results in packet drops when it is overflowed. When a packet exits from the last pipe, the core transmits the packet to the edge node hosting the destination VN.

ModelNet passes through five phases before running an experiment: Create, Distillation, Assignment, Binding and Run. The first phase, Create, generates a network topology, by means of a graph. The edges represent network links and the nodes represent clients, stub, or transit domains. ModelNet writes the graph using the Graph Modeling Language (GML). The Distillation phase converts the GML to a pipe topology that reproduce the network infrastructure. In this phase it is possible to simplify the network, trading accuracy for reduced emulation cost. The Assignment phase implements the network topology, mapping the topology of the previous phase to the core nodes. This map partitions the pipe graph to distribute emulation load across the core nodes. One of the hard problem, the ideal assignment of pipes to cores, depends on routing, link properties, and traffic load offered by the applications (it is an NP-complete problem). The Binding phase allocates VNs to edge nodes and configures them to execute applications. ModelNet allocates one or more VNs onto each real edge node that binds to a single core. In this phase different sets of configuration scripts are generated for each node (i.e. either core or edge nodes). In the core nodes, the scripts install sets of pipes according to the distilled topology and routing tables between all pairs of VNs. In the edge nodes, the scripts set edge nodes with emulation-based IP addresses for each VN. The last phase starts the emulation by running application code on the edge nodes. A single command script automates the execution of thousands instances of a particular application across a cluster. In particular, VNs running application instances must bind IP endpoints to their assigned IP addresses in the emulated network rather than the default IP address of real node hosting those (VNs). This is fundamental to guarantee the emulation consistency. In fact, the emulation consistency is obtained by rewriting the source and destination fields of the outgoing packets with the correct (emulation-based) values. In this way it is possible to route through the core and to their final destination correctly.

Here below, it is explained how the core works and, in particular, the packet processing steps are illustrated. The core is configured to intercept packets entering the emulated network based on IP address by means of the IP firewall (ipfw). All VNs bind to IP addresses of the not routable 10.0.0.0 A class. When the packet is intercepted, the core kernel module looks up the route for the given source and destination. In the Binding phase, ModelNet builds a routing table in each core node. This table is filled up with the pre-computed shortest-path routes among all pairs of VNs obtained in the distilled topology. Each route is an ordered set of pipes that must be crossed to send a packet from source to destination. In this way it is possible to show the set of pipes that must be passed through in the emulated topology. At this point the core creates the descriptor that references the buffered packet. This descriptor will be scheduled, instead of the buffered packet, on the set of pipes specified by its route. As this set of pipes are shared by different competing flows and each has a differently configurable queue, the core is able to reproduce the packet drop and network congestion effect, due to an application-specific communication patterns.

The descriptor scheduling adopts a heap of pipes that are sorted by the imminent timeout. In particular, the timeout of a pipe is the exit time of the first packet in its queue. The scheduler has a time frequency configured at 10Khz and runs at the highest priority level. The scheduler checks up on the heap for all pipe timeouts that are later than the current emulation time. When one or more timeouts expire, all descriptors at the head of the late queues are removed and are inserted into the tail of the next queue that represents the next pipe on the route of these packets. If a descriptor on the last pipe and its timeout is expired, it is delivered to the real link toward the final destination. Then, the scheduler calculates the new timeout for all pipes that had expiration during the current emulation time. After this computation, it reinserts these pipes into the heap according to this new timeout expiration.

To respect the emulation accuracy a coordination of kernel components is necessary. To achieve this aim the core performs two principal tasks with different priorities. The first task is managing hardware interrupts to intercept (ingoing) packets from the network interface. The second aim is to periodically insert packets from a pipe to another pipe or from a pipe to the final destination. Hence, the second task has a strictly higher kernel priority than the first. In this way, the core prefers to emulate carefully the delay of the buffered packets rather than to service ingoing packets. As a result, the CPU saturation of the core produces dropped (ingoing) packets rather than inaccurate emulation.

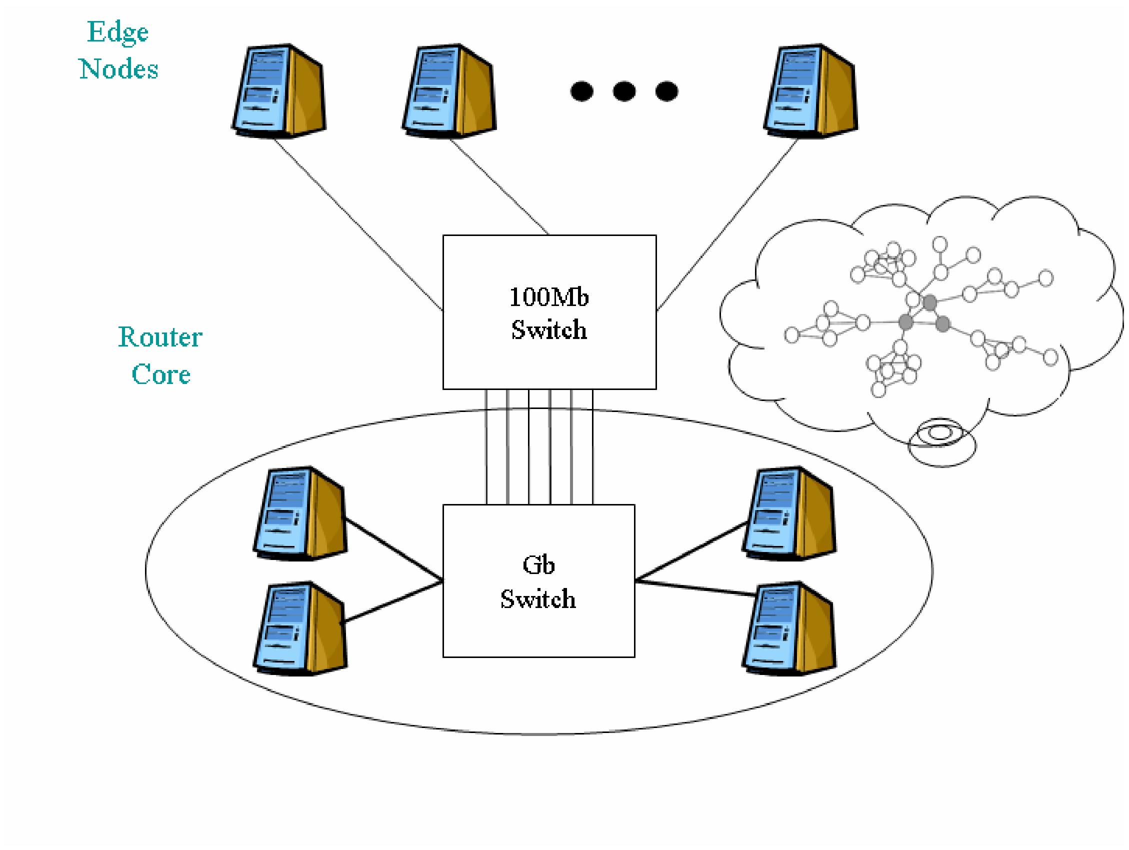


Figure 3.3: Architecture of Modelnet reported from (Vahdat *et al.*, 2002)

Hence, this approach may result in a critical situation where a lot of packets are dropped at the physical level and the accuracy becomes relative. The core implementation of ModelNet is based on Dummynet, with extensions to improve accuracy and to support multi-hop and multi-core emulation.

3.7.4 Other network emulators

For sake of conciseness, in this Sub-Section, we report some other examples of network emulators.

Delayline (Ingham *et al.*, 1994) can introduce propagation delay and allows to specify a packet loss probability of an unreliable communication channel before passing the packets to the actual sockets provided by the operating system. Delayline provides transport layer emulation through a custom socket implementation. A major limitation of this tool is that it is designed only for UNIX systems and can only be used for applications, which use Berkeley sockets for communication. To make use of the tool, an application has to be filled in to call the Delayline system calls. Therefore, the tool is not transparent to applications.

NISTNet (Carson, 1999) has a fixed queue size, but adds delay variation, packet reordering, and packet duplication. NISTNet is a tool working within the Linux kernel. As Dummynet, it can introduce bandwidth limitation, delay, and packet loss, and then it adds delay variation, packet reordering, and packet duplication. NISTNet operates at network emulation layer and therefore can only be configured on the basis of IP layer addresses. By using up a number of connected nodes running an emulation tool, a comprehensive network scenario can be set up.

NetWire (Carniani *et al.*, 2001) is a distributed architecture designed for educational and research purposes, which provides a synthetic and realistic network environment that may be used to teach and learn parallel algorithms (or parallel operating systems) as well as to research and develop new distributed algorithms. NetWire is an architecture based on a client/server derivation scheme: each client can interact with one or more servers emulating one or more networks by the NOEL protocol (Network Oriented Emulation Language), which is an extension of TCL over TCP/IP specifically designed for NetWire. The user can thus control all the physical parameters of each network or part of it (communication channels, hubs, network adapters and so on). Furthermore, the NetWire API library interfaces the synthetic network environment to real software applications with ease, hiding the whole architecture behind the appearance of a network device driver, fully compatible with the operating system the applications run on. Moreover, NetWire already provides a featured Xwindows interface, and because of the integrated TCL language and the interactions between NOEL and TK, it is possible to quickly build up new and powerful GUI based programs.

ONE (the Ohio Network Emulator) (Allman, 1997) is a tool that enables researchers to emulate a network between a pair of interfaces on a single Solaris-based workstation. The user can control various features of the network, such as propagation delay, queuing characteristics and bandwidth. In addition, the tool interfaces with Satellite ToolKit (STK) to emulate variable propagation delays based on the orbits of satellites. ONE was initially developed by Ohio University's Internetworking Research Group. The program is now maintained in conjunction with the NASA Glenn Research Centre's Satellite Networks and Architectures Branch.

IP-TNE (Bradford *et al.*, 2000) (the Internet Protocol Traffic and Network Emulator) is a network emulator that uses a fast parallel discrete event simulation (PDES) kernel to model interactions between real IP packets and IP packets generated within a network emulator. IP-TNE uses the Taskit PDES kernel that has been modified in order to operate as a real-time system. Taskit is a simulation kernel that implements CCT (the Critical Channel Traversing) algorithm.

EMWIN/EMPOWER (Zheng *et al.*, 2002a-c) is a new IP-based mobile wireless emulation. In EMWIN, a target mobile wireless network is precisely mapped to an emulation configuration in a laboratory private network consisting of several emulator nodes, each of which is able to emulate multiple mobile hosts. Moreover, the emulator node can be flexibly configured so that predefined network conditions and traffic dynamics can be generated in an automatic manner. With EMWIN, the mobility of the target mobile wireless network with a number of mobile nodes can be faithfully emulated in a wired network environment.

NETShaper (Herrscher *et al.*, 2002a-b) (Network Emulation Testbed) provides emulation on link layer that is completely transparent for applications and even protocols down to the network layer. Therefore, NETShaper can be used to evaluate e.g. routing protocols in their original form (i.e. without code modifications). NETShaper uses a bandwidth throttling approach that exactly mimics the actual behavior of saturated network devices, and therefore works with queuing algorithms provided by the network layer. Further emulation parameters include fixed delay, variable delay, and frame loss. Due to its modular architecture, it can be inserted and removed without rebooting systems. As a trade-off between centralized and distributed emulation, NETShaper emulative tool can run on several nodes forming a comprehensive scenario, and is controlled in real-time by a central dynamic network model.

VSI (Xu *et al.*, 2001) (Virtual Socket interface) is a socket-based interface to compose hybrid component networks with an existing parallel network simulation library called GloMoSim (Zeng *et al.*, 1998). This work implements a technique to exploit real time lookahead and shows a detailed analysis of the real time lookahead on the performance of hybrid component networks working with parallel execution of simulated components. In this particular case, the lookahead is the ability of simulated components to foresee the interactions with physical components. VSI uses the real time lookahead approach to manage the interactions between operational TCP applications and simulated TCP protocol models for the parallel simulation. VSI is portable and supports most of the TCP applications.

Finally we can see an example of hardwar- based emulator. **BBN** (Milliken, 1993) offers a hardware-based link emulator that emulates two or more unidirectional optical links. It can be used for delay emulation of up to 200 ms programmable in increments of 1 microsecond resolution. The emulator itself is essentially a SUN Sparc station with special hardware and software installed on it. It also provides certain error patterns and error rates. However BBN is far too expensive to be used by general-purpose protocol developers.

3.8 Simulation Resources: Traffic Generator and Topology Manager

In this Section, we show some simulation combined based network emulators that exploit workload traffic generators and some topology generators.

3.8.1 Traffic generators

Workload traffic generators are used to evaluate the performance of different servers and thus to find their optimizations. The crucial point of this study should show how these servers perform in a wide area network, like Internet. In particular, different works focus on understanding the effect of wide area conditions on web servers (Nahum *et al.*, 2001; NS). In fact, in the real scenario clients geographically far from these servers generate requests that exploit the wide network infrastructure to reach them. At the beginning, this scenario was reproduced by using a workload traffic generator, a servers and a LAN. The workload traffic generator makes real requests to the server interconnected by a high speed LAN. As already said before, the problem is that a LAN provides excellent network connectivity with high bandwidth and, in general, no packet loss or re-ordering.

These performance characteristics are very different from those of a real large-scale scenario. Therefore, these evaluations are not realistic since the scenario adopting a LAN interconnection does not capture the wide-area characteristics like modem communications, high packet loss and large packet delay. Then, the solution is to connected clients to servers via network emulator. Obviously, the emulator is in charge of introducing factors as delay and packet loss inside a controllable and reproducible environment. Hence the workload traffic generators produce the interarrival requests typical of different clients, while the emulator reproduces the network communication characteristics. The main difference between this approach and the synthetic simulation regards the generation of real traffic with real time constrains. An interesting example of this approach is showed in (Nahum *et al.*, 2001). This paper shows how the reproduced WAN characteristics affect the performance of a Web server. In particular, the authors of this paper adopt Dummynet as network emulator and their adaptation of SURGE as workload traffic generators.

3.8.2 Network topology manager

The creation of complex network topologies in the network emulation is very difficult, because, in general, they must be physically constructed. This represents one of the main drawbacks of the network emulation. In general, network emulator includes very simple topology that may be very specific. In particular, most of them support a host end-to-end scenario, where the emulator is in charge of transferring directly the packet from the source computer to the destination. Hence, in the network emulation, trying to create general network topologies is approached in three different way: combining different instances of an emulator with VLANs or in a high speed cluster, utilizing an host adjacency matrix to route the packet (specific for the emulation of Ad Hoc networks), and integrating emulation, simulation and live network experimentation into a common framework. The first ways to build a complex scenario for network emulation can be include the combination of network hardware infrastructure and instances of the same emulator. An example of this approach uses a cluster of computers connected to a high performance switch and a separate control network (Herrscher *et al.*, 2002b). If the switch supports the virtual LAN technique, VLANs can be used to configure any virtual topology. In this way each of the virtual network devices has a separate instance of the emulator. The combination of different instances of an emulator with VLANs permits to build complex scenarios for network emulation. Another example similar to the previous one is ModelNet (Vahdat *et al.*, 2002). ModelNet is a scalable and comprehensive large-scale network emulation environment. In ModelNet, real applications run on edge nodes configured to route all their packets through a scalable core cluster, physically interconnected by gigabit links. This core is responsible for emulating the characteristics of a specified target topology on a link-by-link basis. The core routes traffic through a network of emulated pipes, each with an associated packet queue and queuing discipline. In particular, ModelNet maps pieces of topology to different core nodes and distributes traffic by means of pipes among them. A second way to build a complex scenario for network emulation is to use a host adjacency matrix to route the packet. The example is specific for the emulation of Ad Hoc networks. Mobile hosts in a MANET can change the connectivity of the network at any time. In this scenario the dynamic topology switch is built (Lin *et al.*, 2002). It can emulate a wireless mobile ad hoc network using standard ethernet physical connection. The connectivity function is computed on the basis of time and relative position of each mobile host. In this way, it is possible to map the coordinates of each pair of hosts evaluating the connectivity between them. The dynamic topology switch is implemented as a central computer in a star network that controls the connectivity of each pair of hosts (in this network). The hosts can be any devices, directly connected to the central computer. In particular, the dynamic topology switch is able to switch traffic between any pair of hosts, based on its local connectivity table. The table can be dynamically updated so that the dynamic topology switch can emulate dynamic mobile ad hoc network topologies using fixed hosts in a wired network. Finally, the third way to reproduce a topology scenario is by integrating emulation, simulation, and live network experimentation into a common framework. NetBed (White *et al.*, 2002) realizes a virtual topology configuring a set of distributed, emulated, or simulated nodes, by means of graphical tools and command-line scripts. The automated facility of NetBed includes different settings at different level of configuration. In fact, it is possible to manage experimenter accounts, set emulated link characteristics, and map the virtual nodes. One of the main advantages of NetBed is to support heterogeneous resources. In particular a complex scenario may be emulated by traffic-shaping Dummynet (Rizzo, 1997) nodes, may be simulated via NS (Fall, 1999), or may be realized by wide-area links. In particular, NetBed exploits the emulation facility of NS (NSE) to transfer packets crossing the simulative environment and the live network boundary. One of the most important features of NetBed is the consistent interface it provides to control nodes and links regardless of their implementation (distributed, emulated, or simulated).

Chapter 4

Interceptor: Design Issues

The models adopted to reproduce network traffic communications may follow two different schemes, namely: either a fine-grained approach or a coarse-grained approach.

In the *fine-grained* approach (Breslau *et al.*, 2000), the model of network traffic communications incorporates meticulously the specification of each protocol layer. The most important feature of this approach is the model accuracy. Hence, according to this approach, separate simulation models may be defined, e.g., for the data link level in point-to-point links and LANs, and for the network level in IP based routers and hosts. However, hardware resource limitations, such as processing time and available memory, limit the accuracy level of any model specifying how many network nodes, links and protocols may be modeled. For this reason, the fine-grained approach is typically used in the networked simulated setting where the simulated time can be a compression or an expansion of the real executive time. The compression allows the simulator to spent little time in the reproduction of the least significant part of the model, while the expansion permits the simulator to use for a longer time the hardware resources in the reproduction of the most significant part of the model. Again, if the resources are not enough and we want to maintain a certain level of accuracy the network simulation exploits the hardware power of cluster of computers.

Moreover, the problem is exacerbated if we want to develop a network emulator (not a simulator) based on a fine-grained approach. This task may be very difficult to achieve if too many details must be taken into account during the emulation. This is due to the fact that an emulator is a real time tool, which uses a real implementation of a running application (or a protocol) under study. In fact, in a network emulator the simulated time should be very close to the real executive time (i.e. $t_s \approx t_r$). In this way, it is very difficult to manage the simulated time in favor of the model accuracy, as the simulation does with an expansion.

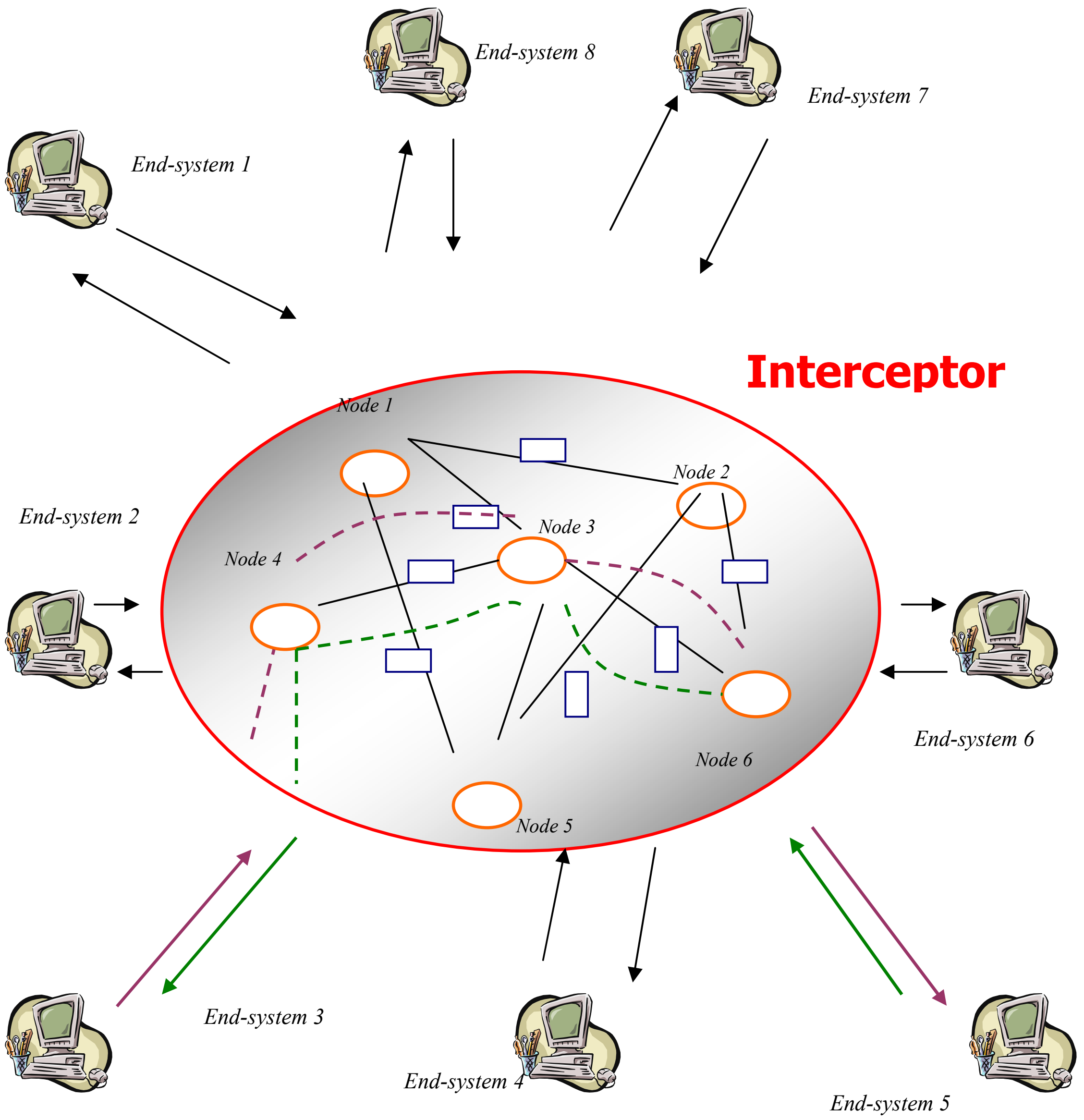


Figure 4.1: *Interceptor*: a synthetic emulative platform

Another main consideration about the possibility to adopt this approach in the network emulation is related to scalability problems deriving from the need to integrate the network communication model with the traffic coming from hundreds of real network hosts.
Further, in this case, when the emulator collects such external traffic and passes it through the network communication model, performance degradation in the processing time may occur. In simple words, the higher is the accuracy of the model reproducing the network infrastructure inside the simulation environment, the lower is the emulative performance.

Alternatively, the *coarse-grained* approach consists of designing network settings and sacrificing some model details to avoid performance degradation. This approach prefers performance to model accuracy, without sacrificing its validity. In general, adopting this approach may be the best solution for network emulation. In fact, if the aim is to maintain the possibility of integrating external real traffic respecting the time constraint of the predefined network communication models, the time spent to calculate these constraints must be as reduced as possible. This means that the time to compute one of these constraints must be lower than that the (real) traffic must wait inside the emulative environment. The time to make these computations depends on the number of details that must be calculated according to network communication model.

Interceptor (the tool we have developed) adopts this latter approach. In essence it is an emulative platform, which incorporates a coarse-grained (synthetic) simulative environment. Our emulative platform allows the simulationist to integrate real end systems (and their running protocols) into its synthetic environment. This is possible by exploiting either the opaque mode that treats the network data as non-interpreted packets, or the protocol mode that interprets or generates arbitrary field assignments of live network traffic. In particular, Interceptor typically works in the opaque mode, but if one wants to set up the communications between several hosts without enforcing the emulator to perform as the default gateway, the protocol mode is used.

To implement both the two modes, Interceptor exploits a centralized intermediate emulation platform that reproduces the behavior of a complex IP-based network connecting real end systems (developing a network layer emulation). Using this approach, it is possible to intercept IP packets exchanged between a couple of real end systems, and at the same time simulate packet delays and losses according to a predefined comprehensive end-to-end traffic model that abstracts from complex simulation details. Interceptor allows simulationists to use real IP-based devices (as sketched in Figure 4.1) and to introduce real protocol traffic (in form of IP packets) into its synthetic environment, while maintaining unchanged the network communication interface. Hence, it allows one to confine in a lab-based environment such real network IP-based devices, while simulating a complex network infrastructure with appropriate parameters. The simulative experiments may be conducted by adopting an event-driven scheme that advances the evolution of this system under real time constraints.

In particular, differently from the typical emulator, Interceptor incorporates two simulative technologies suitable for manipulating the synthetic environment, namely a network topology manager and a workload traffic generator. The network topology manager permits one to specify any network topology model consisting of a set of arbitrarily simulated network nodes (i.e. routers) between the real source and the real destination (i.e. real end systems). This manager permits to interpose an arbitrarily number of these simulated nodes (and links) among the (real) end systems. Further, an external workload traffic generator is used to exercise both the end systems and the synthetic environment in a manner representative of a user that runs a real application. In particular, if we want to expand the capabilities of a network emulator by means of these simulative technologies trying to avoid the degradation of the performance of the entire system, a coarse-grained approach must be adopted. Therefore, we define *synthetic emulative platform* a network emulator that include a simulative environment capable of mimicking complex IP-base networks adopting a coarse-grained approach.

As to the categories of the emulator we have discussed in the Section 3.2, we wish to point out that Interceptor exhibits characteristics which derive from all of them (central-control emulator, simulation-combined emulator and trace-based network emulator). In fact, Interceptor is designed and implemented to reproduce the network mechanisms and traffic dynamics with the trace modulation, to connect real hosts as central-control and to exploit technologies typically available in network simulators.

- Interceptor is able to reproduce some network effects by applying typical network mechanisms and traffic dynamics to each packet passing through its synthetic environment. Therefore, all real hosts that want to make use of this platform in order to participate in an Interceptor-based experiment must be connected to it as a *central-control* emulation does.

- The aim of the Interceptor approach is to combine network emulation with coarse-grained network simulative technologies so that these resources can be incorporated in an emulative experiment. Then, from this point of view, we add a synthetic environment to an emulator like the *simulation-combined* emulation does.

- It is not possible to simulate the behavior of real networks without reproducing the traffic dynamics of different protocols that run into them. From this point of view, Interceptor adopts trace modulation to create a representation of these dynamics as the *trace-based network* emulation does.

Finally, most networks show a very complex behavior as a result of three basic factors: protocol interactions, traffic patterns generation, and network topology management. All these factors have been addressed separately within the Interceptor according to the following design issues.

- The implementation of a protocol is not needed. As an emulator, Interceptor is able to introduce in its synthetic environment live traffic coming from real applications and to reproduce typical communication patterns.

- An external traffic generator working on real hosts is adopted. This is able to exercise the final (real) end systems and the synthetic network environment in a manner representative of multiple users that run the same application.

- A network topology manager that implements a support for network topology modeling and communications among simulated nodes inside the modeled topology is used.

In essence, Interceptor is comprehensive of an emulation interface, an communication model manager, a workload traffic generator and a network topology manager. We will discuss these software components separately.

4.1 Emulative Interface

The introduction of live traffic into the synthetic environment and the insertion of the resulting traffic from the synthetic environment into the live network are made possible by means of an emulative interface. This component allows the Interceptor:

- To manage packets either in opaque or in protocol mode, and
- To route packets according to an intermediate emulation or a routed emulation approach.

Both the opaque and protocol mode are made possible by implementing an emulative interface that uses the "standard filter channels" of the Linux operating system (2.4.0 Kernel Family). In fact, the emulation interface has been built over the Netfilter framework that implements the "standard filter channels", by adopting the Netfilter library. This library allows one to fetch packets from the network level (Kernel-mode) to the simulation level (User-mode) by means of system-calls and vice versa. In this way, it is possible to create a tunnel that connects directly the source node to its destination (passing through the synthetic environment). There are several advantages deriving from the integration of our platform inside the Netfilter framework. First, this framework provides a "*userspace*" application to receive/send datagram to the Data Link layer. For this reason, it is possible to generate, to modify, to interpret or to simply copy the content of a datagram (i.e. from the header IP to the payload of the application layer) depending on the adopted mode. Second, being the platform executed inside the Netfilter framework, neither the contents of IP packets need to be converted into a particular format, nor there is the need to pass through non standard-filter channels to reach the synthetic environment. This avoids additional computational costs due to the conversion or the packet exchange among standard and non-standard channels.

As to packet routing within the Interceptor, two possible alternatives may be implemented: intermediate emulation and routed emulation approach. The combination of IP aliasing technique and of Natting technique makes it possible to exploit the emulative interface without forcing it to run on the default gateway implementing an intermediate emulation. This fact allows to carry out the emulative experiment in a (public) laboratory without any interference on any other interconnected machine. Instead, by editing the routing tables of the workstations that participate in the experiment, we can decide that the computer on which Interceptor runs is their gateway, implementing a routed emulation. In this way, any communication that should not take advantage of the emulative environment is delayed like those belonging to the emulative experiment without any good reason.

4.2 Communication Model Management

This component is able to manage the communication model between two real end system and a simulated node or between two simulated nodes based on the link that is simulated. In essence, this model is able to simulate IP packet delays and losses according to a network communication model that abstracts from complex simulation details. If an IP packet is inserted in a simulated link of the synthetic environment (Arrival or S-Arrival events), this component calculates the time to pass through it. When this timeout expires the packet is delivered to the next simulated node (S-Departure event) or to its real destination (Departure event). This component reproduces a wide range of network communication models varying from single link (defined by latency, bandwidth, packet loss parameters) to more general end-to-end communication (related to empirical distribution, analytical law and stocastical model). The communication model manager allows a simulationist to tune the level of the simulation granularity, to support cross traffic and to exploit a mechanism that generates internal IP background traffic.

It is possible to calculate this timeout generated by a communication model by tuning the level of the simulation granularity. This means that it is possible to calculate the timeout as resulting from the three main basic characteristics of a single link or as resulting from an end-to-end communication model that mimics a path including different links and routers. In particular, in the former case the basic characteristics used to describe a single link are transmission delay, bandwidth delay and packet loss. Instead, in the latter, a single model reproducing the communication delays of a path comprehensive of different links and routers is based on a trace-driven approach that takes into account the network effects derived from real traces. In this way, it is possible to choose the level of abstraction of the emulation, which provides a good trade-off between the accuracy of the experiments and it ability to perform in real time.

It is possible that different source computers send packets towards the same destination. This means that packets belonging to two different sources might or might not pass through the same link toward the same destination; both cases are supported. To support both cases, Interceptor exploits a bandwidth controller, in the first case, and the management of an end-to-end statistical model, in the latter. In the first case, Interceptor needs to simulate the existence of only one path between the sources and the destination. In essence, it is necessary to manage cross traffic and the order of arrival of single packets is very important. If a packet arrived before the other at the emulative interface, the first has to reach the final destination (i.e. the order must be respected). Interceptor supplies this result adopting a bandwidth controller that is in charge of respecting the order by inserting packet in a FIFO queue. Instead, if it is possible that more paths may reach the same destination from the same sources, then it should be possible for a packet to arrive after another reaches the destination (i.e. the order might not be respected). Interceptor obtains this result by adopting end-to-end communication models that derive from a statistical distribution. This statistical distribution reproduces the communication delays necessary to describe the time spent in these different paths by adopting a trace-based approach that takes into account the network effects derived from real traces. In this way, if the inter-arrival of two packets coming from different sources is low, the statistical distribution may delay the first packet with a higher time.

This component also embodies a mechanism, which is able to generate internal IP background traffic. This mechanism generates internal background traffic that inserts virtual packets inside a simulated link. The main advantage of this mechanism is that it is able to simulate the effect due to the presence of these (virtual) packets coming from other daily internet applications. Every time the mechanism knows how many real and virtual packets are inside a link, just taking into account how many of them have already been delivered. In this way, it is always able to compute how many bytes are present in that link. By means of an equation that computes the quantity of filled-up bandwidth of that link, the mechanism knows how much time the real packet should wait in queue.

4.3 Workload Traffic Generation

In our opinion, a synthetic emulative platform can emulate effectively large scale (IP) scenarios if and only if it is supported by a workload traffic generator that generates traffic in accord with the application under study. In particular, if we think of our tool as the manager of the interaction between an inter-arrival queue and a service queue (i.e. as a typical example of simulation of queuing systems), the workload traffic generator will be in charge of generating the inter-arrivals of IP packets. The goal in developing workload generator is to be able to exercise the final end systems and the synthetic network environment in a manner representative of multiple users running the same application. In particular, the effects of high variability in the workload are reproduced by considering the network traffic as "self-similar". This means that network traffic can show significant variability over a wide range of scales and has a specific impact on network performance. Adopting this approach, a workload traffic generator that reproduces multiple users running simultaneously the same application has been designed. As the replication of several machines, each running a copy of the same application, is not feasible, we have resorted to the idea suggested in (Barford *et al.*, 1998). In essence, following this idea, we developed a realistic workload generator, called M^{2}G (MultiMedia workload traffic Generator), which mimics a set of real users running the same application on the same machine. This permits to Interceptor to move up to several tens of virtual users who simultaneously run the same application on the same machine, in emulation mode. M²G adopts particular workload characteristics to implement these generators, as SURGE does. These characteristics can be divided into two main categories: the idea called User Equivalents (UEs) and the set of distributional models. The definition of UEs is about a single process in an endless loop that alternates between making requests for remote files, and lying idle. Distributional models are a set of probability distributions needed by each UE that specifies file sizes, request sizes, popularity, temporal locality, session length and inactive off-times characteristics to indicate how requests must be done. In the following subsections the design of UEs and distributional models of M²G are shown.

4.3.1 User equivalents

M²G extends the concept of UEs to other kinds of applications different from the web browsing. Also GISMO (another important workload traffic generator) adopts this approach building a software tool that mimics a configurable set of real users while they are generating streaming media accesses, conforming to the different distributions. There are many different kinds of traffic, but we are interested in those describing the requests for downloading, coming from Web and Multimedia application and, in particular, generated by mobile devices (such as PDA, Tablet PC and Laptop). For this reason, a downloading multimedia user equivalent has been developed in Interceptor.

M²G implements the user equivalents in SURGE way. SURGE creates a number of users specified by the simulationist. These users download the files from a web server during their session time period within the default period of time (30 minutes). When the session period of an UE expires, the running process is killed. In this way, during the entire default period of time, SURGE generates a traffic coming from a decreasing number of sources. To monitor the performances of multimedia application, it is important to maintain a constant number of users during the default period of time to avoid different effects (deriving from the decreasing) on the study of an application (or a protocol). In particular, if during the study of the performance the number of users working on that application decreases, the ending workload will be different from the beginning. According to these considerations, M²G has a mechanism that manages a fixed number of users during the default period of time. In particular, the number of the users during the default period is constant. In fact, in M²G, each user requests a number of files depending on the session length. When an UE ends its session period, it notifies to M²G and its process is terminated; then a new UE is generated. In this way, M²G is able to generate a traffic coming from a constant number of sources during the entire default period of time.

4.3.2 Set of distributional models

The distributional models describe the set of probability distribution used by each UE. This set is used to generate references matching empirical measurements of server file size distribution, request for size distribution, relative file popularity, temporal locality of reference and idle periods of individual users that are required in the reproduction of a representative workload. M²G set of distributional models includes some distributions derived from SURGE, from GISMO and some others that we have deduced. The distributions describe: File Sizes, Request Sizes, Popularity, Temporal Locality, Session Length and inactive OFF Times.

1) The “File Sizes” distributional model describes the set of files stored on the server agreeing with empirical traces. This probability distribution is a Lognormal and properly influences the server file system. The implementation of this distribution is like the corresponding distribution in GISMO. The M²G provides also with a module that simulates a multimedia users downloading through a PDA. This kind of devices has a limited storage capacity, so, in this case, this distribution is adapted to its limitations.

2) The “Request Sizes” distributional model describes the collection of transferred files and match them with empirical traces. This probability distribution is a Pareto and influences the network properly. The implementation of this distribution is that of SURGE.

3) The “Popularity” distributional model shows the number of requests made to each file stored on the server. This probability distribution follows Zipf’s Law. The distribution of popularity affects appropriately the caches, since popular files will stay in caches. The implementation of this distribution is that of GISMO.

4) The “Temporal Locality” distributional model describes the probability that a requested file will be requested again in the next future. This probability distribution is Lognormal and exercises the caches properly. The implementation of this distribution is that of SURGE.

5) The “Session Length” distributional model describes the time period of a session. In particular, it defines the number of file requested by a user during its session. This probability distribution is Inverse Gaussian. The implementation of this distribution is that of SURGE.

6) The “inactive OFF Times” distributional model describes the bursty nature of an user requests. In the implementation of the multimedia application the user does not request one file surfing from a link to another, but it generates a list. Then, in this case, the “User Think Time” is equal to zero. This means that the implementation of “inactive OFF Times” distributional model returns to zero. For sake of completeness and for possible future works we chose to maintain this feature.

4.4 Network Topology Management

Using an emulator and a workload traffic generator may be not enough to support the evaluation of some specific topology-related applications or protocols. This evaluation may be effective only by exploiting a network topology model that inserted a set of (predefined) interconnected network nodes (i.e. routers) between real sources and destinations. Interceptor uses a support for network topology modelling. The network simulation and emulation adopt different solutions.

Typical simulative solutions suggest model-oriented generators designed and implemented to reproduce specific characteristics of the network. In this situation a simulationist, in order to understand the correctness and the performance of an application is forced to repeat the same set of experiments on different model-oriented generators. One of the main problems of this approach is to make comparative analyses of these different model-oriented generators.

Instead, the emulative solutions propose very simple specific topology or complicated solutions involving a lot of high performance switches and powerful computers that should be administered in a correct way at the same time. In this second case, these high performance switches and powerful computers are used to implement the core of the emulation (this core is responsible for emulating the characteristics of a specified target topology on a link-by-link basis). In these solutions changing topology model means reconfiguring a part of the computers that worked previously. One of the main problems is to make comparative analyses of these different topology models implemented in different way.

Our idea is to integrate the previous components (i.e. an emulative interface, a communication model manager and a workload traffic generetor) with a support that describes the topology of a network by tuning all its possible characteristics, without involving a lot of high performance computers. This support should be implemented by a tool, called Network Topology Manager (NTM) that describes complex network topology by offering the possibility to specify the single behaviour of each network object as node and edge. NTM should be a universal topology manager that is able to support any topology model just by changing the number and the behaviour of each single network object. Similarly to the idea of universal topology generation of (Medina *et al.*, 2001), this tool should provide an environment where making comparative analyses of the same topology-related application (or protocol) over different topology models may be possible. In fact, the NTM should allow one to extend a network topology model in an easy way. The possibility of extending a model makes the NTM flexible, adaptable and permits to generate representative topologies to be used in different simulative/emulative scenarios.

Furthermore, our NTM does not involve dedicated high performance computers to run the core of emulation, but just re-uses the same instances of the Interceptor components passing specific parameters as an input for computing the communication patterns between two nodes. For this reason, as it takes advantage of its simulative network objects, it does not need to administer all computers dedicated to the core of emulation in a correct way. To understand how our proposal can represent a useful tool to evaluate real applications, it is necessary to specify which ones are the main network objects of a complex network topology.

The topological structure of a network is designed as a graph, with nodes representing routers or general gateways, and edges representing direct connections (transmission links or networks) among nodes. In this situation, each host (i.e. the end-systems running the real application) can be represented as a leaf connected to another node (i.e. a router or a general gateway) of the graph. Additional information about the network can be added to the topological structure by associating information with the nodes and edges. For example, nodes might be associated to numbers representing the buffer capacity or the computational power. An edge might have values of various types, including costs, such as the propagation delay on the link, and constraints, such as the bandwidth capacity of the link and such as the packet loss on the link. The nodes in the network may be placed at random points or according to a specific scenario. Links (represented by edges between nodes) are added to the network by considering all possible pairs of nodes and then deciding if a link should exist according to a specific scenario. In particular, reproducing a network topology means to build a model of a complex system that mimics the behaviour of each single node and link. In general, the behaviour of each node (i.e. router or gateway) results from passing through different phases in an endless loop: sensing, thinking and acting. This means that a node (i.e. router or gateway) is anything that perceives its environment through its sensors (sensing phase), takes a decision (thinking phase) and acts in the environment through the actuators (acting phase). In other words, a node checks its communication channels and the information received from other nodes, adopts a policy behaviour and, finally, delivers its packets on the chosen channels or discarding it (implementing the taken choice). Further, each node can be independent or belong to a group (for example, autonomous system). In the former case, a node can choose the best possible action at each time step, perceiving the environment through its sensors without interacting with other nodes. It tries to put into practice the best action chosen by its thought. In the latter case, it is coordinated with the other members of the group, adopting a form of communication. This implies that in the sense phase it also receives this communication, in the thinking phase it tries to select the best action taking into consideration the previous communication with its partners and in the acting phase it does put in practice.

4.4.1 The design of a network topology manager based on agents

An interesting approach to design a network topology manager that is in charge of managing and reproducing a specific scenario like the one described above, is to adopt a multi-agent system. An agent is a computer system that is situated in some environment, and that is capable of independent action in order to meet its design objectives. A multi-agent system consists of a number of agents, which interact among them. In order to successfully work in their environment, agents have three capabilities: Reactivity, Proactiveness and Interactivity. The first means that agents are able to perceive their environment and to respond in a timely way to changes in order to satisfy their design objectives. This capability generates something similar to the above-described sense phase. The second means that agents are able to exhibit goal-directed behaviour by taking the initiative in order to satisfy their design objectives. This capability generates something similar to the above-described thinking phase and acting phase. Finally, the third means that agents are able of interacting with other agents in order to satisfy their design objectives. This capability generates something similar to the above-described group cooperation.

If we implement a Network Topology Manager as a multi-agent system, the management mechanism of each node could be represented by an agent and each edge could be represented by an interaction between two agents. In this approach, each leaf is an end-system (in particular, in our case an end-system is a real host). Each node should be capable of independent action and should be able to understand the boundary conditions to decide its behaviour, as said before and as an the agent does. Our idea is to build a Network Topology Manager implemented by a multi-agent system, called SPADES. This multi-agent system should be composed by programmable agents and by an environment where the agents live. In our idea, the mechanism to manage each node is a programmable agent (so the management mechanism is programmable). In this way, inside the same environment we can change at run-time the behaviour of each node. This possibility should allow one to make the comparative analyses of different topology models in the same environment. In fact, the NTM should allow one to extend a topology model adding new policy or specifying different behaviours to each single node. The possibility of extending the behaviour of a node makes the NTM flexible and the capability of each node to be reactive makes the NTM adaptable.

4.4.2 On integrating a network topology manager

We will now explain how it is possible to integrate the network topology manager to our platform. The communication model manager is able to reproduce communication patterns according to a simulative model that establishes the time needed to transit a packet from source to destination. To integrate a network topology manager, an incoming packet should be rebound (from a real source node) among a series of simulated nodes until it reaches the last simulated one. This last simulated node will be in charge of delivering this packet to the final real destination host. This means that the communication model manager should be able to communicate to a node, inside the NTM, the arrival of a packet and its final destination. Vice versa, the node inside the NTM should be able to communicate to the communication model manager. In particular, NTM specifies who the next node is and which model should be adopted to calculate the timeout for communicating to the next node (inside the NTM) that a packet is arrived.

To make the integration possible, the communication model manager that handles all the events (arrival and departure: to interact with the emulative interface) has been improved to support simulated arrival (S-arrival) and simulated departure (S-departure). The S-arrival event generated by NTM communicates to the communication model manager that different packets should wait for the their timeout to expire according to an appropriate communication model. Vice versa, the S-departure event is generated by the communication model manager to communicate to NTM that packets arrived to some nodes.

The main issue in the integration of NTM is that the total time spent inside the emulative platform should not violate the real time constrain. This means that the time needed to exchange messages with NTM about each single packet should be not longer than the time that has been computed for this packet by the predefined communication model.

Reaching an integration that does not violate the real time constrain is possible only adopting a coarse-grained approach. Adopting this approach means that we prefer the performance of emulation instead of reproducing an accurate complicated network topology model. This should be done without sacrificing the validity of the network topology model. For this reason, when a simulationist builds an example of topology, it does not model accurately all nodes, but only the most important for its experiments. This means that if between two nodes there are other (lesser important) nodes, an end-to-end model to compute the communication time is adopted. The other (lesser important) nodes are reproduced as additional delay inside the end-to-end communication model. This method decreases the number of interactions between the communication model manager and NTM, so it diminishes the computation time. From this point of view, our approach is able to represent complex network topology without reproducing all details. In particular, our aim is to obtain a good synthetic network environment for our synthetic emulations where it is possible to investigate real applications or protocols (i.e. not to study how to generate accurate network topologies).

The actual version of NTM allows any configuration of a network topology at a defined time by composing nodes and links according to a predefined model. Further, there is also the possibility to manipulate both nodes and links at a run-time. In fact, it is possible at a run-time to change: i) the routing table of any node, ii) the communication delay model between two nodes, and iii) the network topology (without the possibility to create new nodes). Once a given network topology has been specified the procedural behavior of each active object of the network may be specified in a rigorous way by embedding in the simulation model the software code that implements it.

4.5 Main Advantages

To conclude this Session we mention the main advantages of our approach:

- the possibility to support the real execution of network applications on real end systems, while it interposes a synthetic complex IP-based networked scenario
- the platform provides means for setting network topological configurations (i.e. nodes and links) varying from simple to sophisticated
- the platform is flexible in emulating a wide range of network communication models (at the network layer emulation), from single link (defined by latency, bandwidth, packet loss parameters) to more general end-to-end communication (related to empirical distribution, analytical law and stocastical model)
- the possibility to change and update the simulation model without affecting the emulation interface
- the platform provides the possibility to interact with nodes and links at run-time
- the platform reports useful statistical information about Interceptor
- the operation is transparent to the user. This means that the user does not have to recompile application or transport code nor recompile/re-install the operating system on his end-systems
- the installation of the platform is simple and does not require any major changes to a typical laboratory environment such as working on a default gateway
- the platform is easily configurable by editing a text file

Chapter 5

Interceptor: System Architecture

Interceptor is an emulative platform, which incorporates a coarse-grained (synthetic) simulative environment that supports the simulationists when they study network applications or protocols. This platform adopts typical resources of network simulation (as network topology management and traffic generation) plus the typical advantage of network emulation (like the direct use of the real application during the execution of an experiment). Interceptor implements a synthetic emulative environment for the reproduction of complex IP-based networks where real applications (or protocols) run over several real hosts and simulated nodes that communicate through emulated and simulated links. Interceptor is located inside a laboratory. It allows a simulationist to experiment real applications by reproducing complex IP-based scenarios inside its synthetic environment. Interceptor includes the four following software components (see the Figure 5.1):

- An *Emulative Interface* (EI), integrated in the “Netfilter” framework, configurable by means of “IPTABLES” command, that intercepts and manages the IP traffic coming from or going to real devices,

- An *Event Manager* (EM) that implements the event-driven scheme to advance the synthetic emulative experiment; in particular EM manages the communication models for each experiment,

- A *Network Topology Manager* (NTM) that allows to describe and manage a specific IP-based network topology, and

- A *Workload Traffic Generator* (WTG) that is able to generate workload traffic reproducing communication patterns typical of more instances of the same application (or protocol).

Interceptor incorporates an *Emulative Interface* developed to insert inside its synthetic environment the real traffic coming from external network hosts. The EI is able to intercept and exchange real IP packets, while, at a higher abstraction level, Interceptor simulates complex IP-based networks.

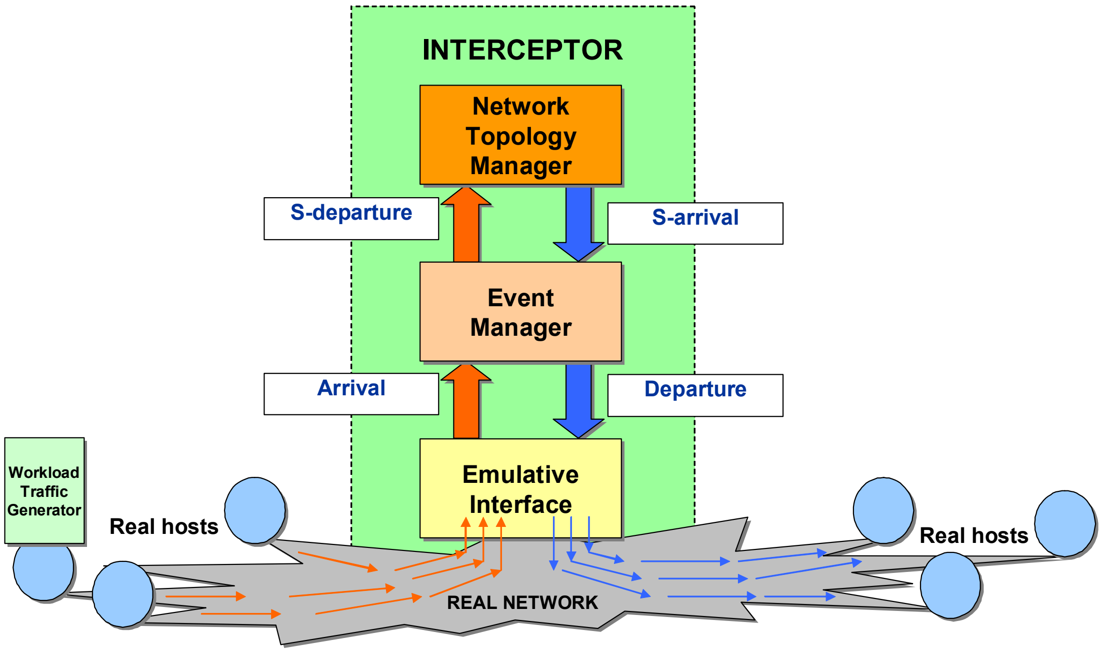


Figure 5.1: System Architecture of Interceptor

The *Event Manager* implements the event-driven scheme that advances the synthetic emulation. The types of events are: arrival, departure, simulated arrival and simulated departure (called, simply, S-arrival and S-departure). The EM schedules the arrival and the departure of the IP packets based on the information regarding the real hosts that have generated those packets or will receive them. The EM also schedules the simulated arrival and the simulated departure of the IP packets based on the information regarding the simulated nodes that have received them and will forward to the next ones according to the reproduced network topology. In particular, the EM manages the communication models.

The *Network Topology Manager* manages a network topology specified by means of a configuration file. The network topology is implemented as a series of simulated nodes that communicates through appropriate links. The IP packets must pass through this simulated topology to reach the final destination.

The external *Workload Traffic Generator* generates workload traffic that in form of IP packets reaches the Interceptor exercising the synthetic network scenario and the final (destination) hosts as if multiple users were simultaneously running the same application. Therefore, the WTG allows (real) hosts of the experiments to move up to several tens of users who simultaneously run the same application, in emulation mode.

The following example shows how Interceptor works. As in each simulation of queuing systems, an experiment within Interceptor has an inter-arrival queue and a service queue (Banks *et al.*, 2001). The WTG or more real applications (controlled by different users) inject IP packets in the lab network. The EI intercepts these IP packets and stores in the inter-arrival queue. After, the EI passes them (Arrival event) to the EM that implements the service queue of synthetic emulative platform. Once an incoming packet has arrived to the EM, it loops among the stages below until has not passed through the "synthetic" network.

- The EM computes the amount of (service) time that the packet spends (in the service queue) before being delivered to the next (simulated) hop adopting an appropriate communication model that reproduces the communication patterns between (real) host and (simulated) node or between two (simulated) nodes.

- As soon as this time expires, the EM notifies the next (simulated) node that the IP packet is arrived (S-Departure event).

- The simulated node (inside the NTM) routes the packet toward the next hop through a simulated link by indicating its (next) communication model (S-Arrival event).

Finally, when the IP packet has passed through the synthetic network, EM forwards it to the final host through EI (Departure event). This host will receive this packet in a transparent way, without knowing anything about Interceptor.

5.1 Emulative Interface

The Emulative Interface (EI) receives IP packets coming from real hosts (where WTGs or real applications have generated them). Therefore, the EI allows hosts to interconnect themselves by means of a multi-homed computer, running the Linux operating system. The multi-homed computer may play the role of a default gateway or of an intermediate emulation platform for each internetworked hosts, implementing respectively the routed emulation or the intermediate emulation. Since EI may perform over a multi-homed computer among all hosts, sophisticated techniques have been implemented to guarantee the correct behavior of packet flows through the synthetic emulative scenario.

When IP packets arrive, the EI stores them into the inter-arrival queue by marking its arrival time and by generating an arrival event. After the generation of this event, the EI passes them to the EM that implements the service queue, while at a higher level Interceptor simulates complex IP-based networks. It is worth mentioning that the application traffic entering Interceptor must exit under real-time constraints (managed by the EM). In essence, the EM computes the simulation time that the packet must wait inside the service queue. When this timeout expires (i.e. a departure event is generated), EM passes the corresponding IP packet to EI that is in charge of sending out this packet towards its external final destination.

5.1.1 libipq API

The EI adopts the *libipq API* of the *IPTABLES module* that works over the *Netfilter framework* (Russell, 2002). Netfilter is a framework compiled into the standard Linux kernel 2.4.0 family, which provides diverse hooks into the Internet Protocol stack over which loadable modules (like IPTABLES) can mangle packets outside the normal Berkeley Socket Interface. Hooks are specific points inside the protocol stack. Each protocol can exploit the five hooks (see the squares of the Figure 5.2). Modules can listen to the diverse hooks for each protocol. Hence, if a packet crosses the IP stack, the Netfilter framework checks if anyone has been listening for that protocol at that hook.

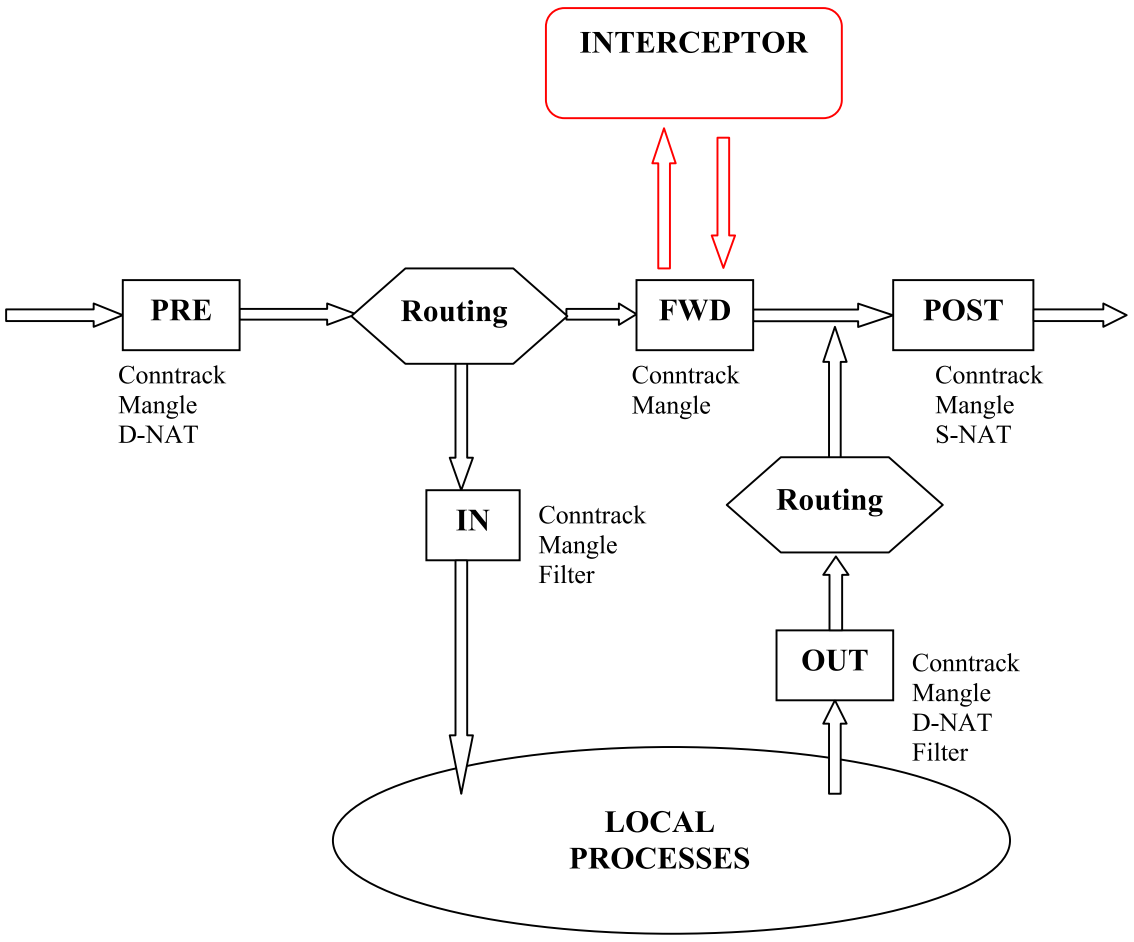


Figure 5.2: IPTABLES over the Netfilter Framework

In this case, the modules may decide to discard the packet, to allow it to pass, or to ask Netfilter to queue it into *userspace* (i.e. a special memory space reachable by users).

IPTABLES is a (loadable) module that implements a packet selection system built over the Netfilter framework. IPTABLES is composed of two main parts: the kernel-space modules and the userspace tool. The kernel-space modules are an integral part of the standard Linux kernel 2.4.0 and they are in charge of performing the appropriate function on the packets intercepted by Netfilter framework, depending on the previously defined rules in their rule-lists (called chains). The IPTABLES (at the userspace level) is a tool that adds, removes or edits rules in the chains. This packet selection system is used for packet filtering (the filter table), Network Address Translation (the NAT table) and general pre-route packet mangling (the mangle table). In the filter table, there are the rules to filter packets without altering them. The IPTABLES filter intercepts and inserts packets into Netfilter framework at the NF_IP_LOCAL_IN, NF_IP_FORWARD and NF_IP_LOCAL_OUT points (see the IN, FWD and OUT squares in the Figure 5.2). This means that for any single packet, there is only one right place to filter it. In other words, each packet sent to this computer may be filtered at NF_IP_LOCAL_IN hook, each packet passing through this computer towards another one may be filtered at NF_IP_ FORWARD hook and each packet sent from this computer toward another one may be filtered at NF_IP_LOCAL_OUT hook. If CONFIG_IP_NF_NAT_LOCAL is defined in the NAT table, the hooks NF_IP_PRE_ROUTING and NF_IP_POST_ROUTING (see PRE and POST squares in the Figure 5.2) are used for altering respectively the destination and source address (see D-NAT and S-NAT table in the same Figure). The network address translation supports also the connection tracking to restore the altering of the destination and source fields (Conntrack label in the Figure 5.2). Finally, the packet mangling table (the mangle table) is used for actual changing of packet information (for example, the Time-To-Live field of the IP header).

IPTABLES as packet selection system built over the Netfilter framework is very useful, especially the exploitation of its special target option *QUEUE* that allows one to queue the packet coming from the real network towards the userspace. The IPTABLES tool supplies a "queue handler" (controllable by means of libipq API) that supports the passing of packets between the kernel and the userspace. For this reason, the libipq API allows the interaction between the Netfilter framework and the user' space where user applications can receive, manipulate, and issue verdicts on IP packets. The following example shows how to use IPTABLES command to queue packets for userspace processing:

```
iptables -A FORWARD -j QUEUE -s 10.0.0.0/255.255.255.0
iptables -A FORWARD -j QUEUE -s 137.204.72.0/255.255.255.0
```

These rules specify to pass the forwarding IP packets with these source addresses to the "queue handler" (implemented by the *ip_queue* module), which will attempt to deliver the packets to a userspace application. If no userspace application is waiting, packets are dropped.

5.1.2 Implementation of an intermediate emulation or a routed emulation

The EI is able to route IP packets or to maintain the networked communications on the Interceptor consistent. The EI may be either an intermediate emulation server or a default gateway of the communication among the real computers of the experiment with Interceptor. As well as these considerations, the EI can realize either the Virtual Host Method or the Virtual Router Method (see the Section 3.4) to implement respectively the intermediate emulation and routed emulation.

A combination of the NAT and of the IP aliasing techniques is necessary to implement the *Virtual Host Method.* In this way, it is possible to address the computer running Interceptor with different real IPs (using the IP aliasing technique) and to change the destination and source addresses of an IP packet (adopting the NAT technique). Implementing the Virtual Host Method is necessary to map one-to-one real laboratory addresses with (reserved) aliases. This is due to the fact that the real devices should send packets to the network-reserved class of IP aliases to exploit the emulated/simulated scenario inside Interceptor.

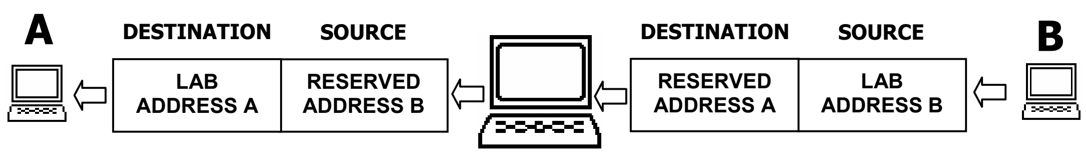


Figure 5.3: Incoming phase

Table 5.1: example of IPTABLES commands

```
# from A to B
iptables -v -A PREROUTING -t nat -d RESERVED A -j DNAT --to LAB A
iptables -v -A POSTROUTING -t nat -s LAB B -j SNAT --to RESERVED B
```

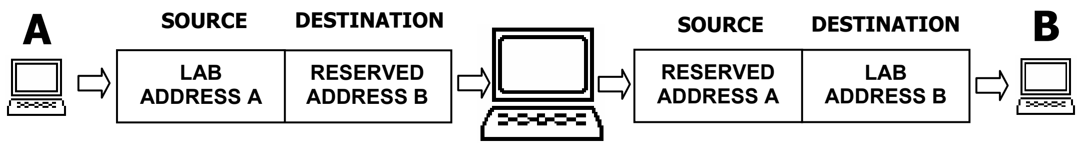


Figure 5.4: Coming back phase

In particular, a source sends its packets to a reserved alias address that identifies a specific destination. When these packets reach the Interceptor, the source address is mapped with another alias address and the destination is substituted with the real laboratory destination address. Therefore, during the incoming phase of a packet, the NAT will replace the IP number of the network reserved class in the field of destination address with that of the network class of the laboratory (D-NAT, prerouting) and that of the source with another alias (S-NAT, postrouting) (see the Table 5.1 and the Figure 5.3). If the destination replays to its source, it will send back the response packets to the alias address read in the arrived IP packets. This alias individuates as a source the computer on which Interceptor runs. Therefore, during the coming back phase, the connection tracking mechanism will be able to recover the packet to original contents, by restoring the previous address field assignments (see the Figure 5.4). In particular, Interceptor works as a userspace application of the Forward hook, after the prerouting and before the postrouting phases (see the Figure 5.2). In this way, it is able to manage IP packets without taking care of real routing, but only acting as an intermediate emulation server. This method, furthermore, allows the simulationist to use the lab computers (including the one on which Interceptor runs) to execute experiments even if other users are present.

In this case, eventually, the simulationist must understand how much the traffic of other users affects the experiment.

The *Virtual Router Method* is implemented by forcing each workstation to communicate only through the default gateway. This is possible if the configuration of the routing table of each workstation (belonging to the experiment) to reach the others (always belonging to the experiment) has a static route that specifies Interceptor as next hop for these communications. In this way, using the “route” command, a static route on a source “A” to send the IP packets to a destination “B” (adopting Interceptor as default gateway) must be added. This method highlights some limitations. First, only the system administrator has the privileges to run the “route add” command. Second, all communications between A and B (including that do not take part to the Interceptor experiment) pass through the emulative platform.

5.1.3 Implementation of the emulative interface

The libipq API exploiting the IPTABLES module over the Netfilter framework allows the EI in userspace to read, modify, discard, and finally route IP packets that pass through the Forward hook, (see the Figure 5.2). The Netfilter framework in Kernel mode allows the EI to capture to low-level data-stream that reaches gateway through network card. IPTABLES interfaces the EI (in system user mode) directly with Netfilter framework and it allows EI to pilot it through libipq API. In this way, the libipq API allows EI to get the IP packets arrived to the queue handler (in the userspace) or to deliver to it the IP packets toward the final destination. The libipq API allows the EI to control and to rewrite headers from the network protocol until application one. The EI waits to exchange IP packets with the (inter-arrival) queue handler and with the Event Manager, piloted by two fundamental events (see Figure 5.5):

- the *arrival event*, a packet is ready on the userspace (i.e. on the inter-arrival queue), waiting to be inserted in the EM where its departure time is computed (by means of one or more communication models) and

- the *departure event*, after a timeout expiration it notifies that packets must leave EM and must be re-inserted on the network towards their destination.

The former event is generated by the presence on the userspace of a packet coming from the FORWARD hook. This event is due to IPTABLES command with option QUEUE that specifies to Netfilter framework to make accessible the stream coming from network cards at the FORWARD hook. When an IP packet arrives at the userspace, it is inserted in a (inter-arrival) queue. A system-call of the libipq API captures the arrival event generated by the queue handler. In our implementation the EI calls this system-call (*ipq_read*) to receive incoming IP packets. In fact, the generation of this event activates the EI code that removes the packet from the userspace and that inserts it into the service queue (in the EM). Once the packet has been inserted, EI will wait for the (departure) time when will be forwarded again on real network.

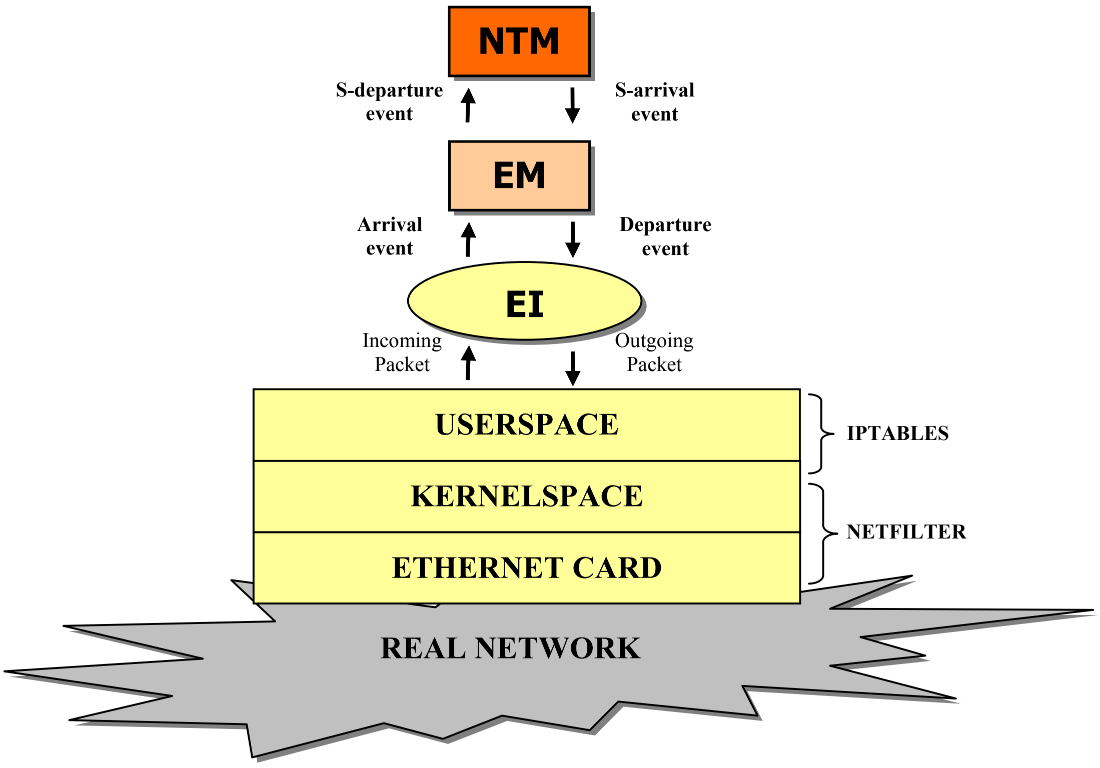


Figure 5.5: Interceptor over IPTABLES module of Netfilter framework

By means of an adequate routine it is possible to know the exact moment in which IP packet has arrived on Ethernet card. After the (departure) timeout expires the EM generates a departure event and passes again the IP packet to EI. The EI inserts the IP packet in the queue of the userspace, specifying to the queue handler to insert it through the FORWARD hook in the Netfilter framework by means of another system-call (*ipq_set_verdict*). This framework is in charge of routing correctly the packet according to its address fields and sending it through the Ethernet card outside the laboratory network. The value of departure timeout is the difference between the moment in which EI has enqueued actual packet and its departure. The timeout is calculated adopting a series of communication models (i.e. computing diverse service times) configured by the simulationist to reproduce a specific large scaled scenario. After an IP packet leaves the EI, this eliminates the structure referencing this packet.

5.2 Event Manager

The most important component of Interceptor is Event Manager (EM) that manages the service queue of the entire system. In essence, the EM is in charge of the synchronization of the components inside Interceptor, by implementing the event-driven scheme of the entire system. The event driven scheme advances (in real time) the evolution of the state of the synthetic environment. This scheme schedules the IP packets by managing the simulation time and the service queue. The main tasks of the EM are two:

- managing the service queue of the action related to the stored live IP packets, and
- calculating the new position of an action (i.e. the service time) inside the service queue by means of a Communication Model.

The action is a data structure that represents the future operations (related to a single IP packet) to do when the service time expires. In essence, an action explains, for example, which is the next hop for this expiring packet, or if the next hop is a simulated node or a real host. The queue is managed by means of four events: arrival, departure, S-arrival and S-departure. In particular, the arrival and the S-arrival events generate the insertion of an action (related to a packet) in the service queue ordered by time expirations (i.e. the moment at which service times expire), while the departure and the S-departure generate the extraction of one or more actions (related to diverse packets) from this queue. The arrival and departure events are necessary to manage the arrival to EM and the departure from the EM of the live traffic by means of the EI. The arrival event notifies the insertion in the service queue of a new action related to a packet coming from live traffic, while the departure the extraction of an action related to a packet that has already exploited the synthetic scenario and must be inserted again in the live traffic. Instead, the S-arrival and the S-departure are necessary to implement the communication with simulated nodes inside the network topological scenario. The EM generates an S-departure event to notify to NTM nodes that packets are arrived. A node inside the NTM generates an S-arrival event to notify to EM that an IP packet should be sent to its next hop, after the expiration of a new service time. The service time represents the time spent to reach the next hop (i.e. the time spent into the communication channel). The next hop may be a simulated node or a real (final) host. The new service time is computed adopting the communication model specified by the node, which generated the S-arrival event. Only when an arrival event is generated, the communication model is specified on the configuration file.

5.2.1 Service queue handler

The first task of the EM is to manage the service queue of the actions related to the stored live IP packets (see the Figure 5.6). This task is implemented by a mechanism that advances simulation time and guarantees that all the actions occur in a correct chronological order based on the Future Action List (FAL). Interceptor must service the simulationists with real time constraint. For this reason, a real time version of the Action Scheduling / Time-Advance Algorithm has been implemented. At each moment the service queue contains all previously scheduled future actions and the time at which they occur. The service queue is ordered chronologically by service time expiration. The clock of the computer running the EM sets the value of the simulated time. The action with time expiration nearest to the simulated time value is called the imminent action. The classical Action Scheduling / Time-Advance Algorithm, after generating the action at time “t”, advanced the simulated time to the next imminent action. This is not possible in Interceptor, because the real time constraint must be respected. In fact, EM extracts actions only when the system clock reaches their time expirations. Therefore, when the clock reaches the time of an action notification, the EM removes from the service queue the action and executes it generating an S-departure or a departure event. Before the next action notification, it is possible that new future actions may be inserted with a S-arrival or an arrival event. In this case, the action is scheduled and inserted into the right position on the service queue. The process of inserting and extracting actions is repeated until the synthetic emulation ends. The length and the contents of the service queue change during the simulation time. The correct management of the service queue will improve the system efficiency. The operations performed on the service queue are two: the extraction of the imminent action (or imminent actions) and the insertion of a new action. The efficiency of the search inside the service queue depends on its logical organization and on the searching mechanism. Our service queue is ordered according to the next time expiration, so the first position is that of the imminent action. In this way the searching cost of an extraction is zero. Instead, the insertion will require the scanning of the queue until the right position is reached. If the length of the service queue grows up quickly, the time for scanning will become one of the main bottlenecks of the entire system. Due to the high variability of the service time expiration, it is not possible to know on a priori basis in which position of the queue the next action will be inserted. If the length has a high value, the insertion may be (in unlikely cases) very expensive. To minimize this side effect, an ordered structure that points to several positions of the queue and that specifies the service time expiration of the pointed action has been built. In this way, when an arrival or S-arrival event occurs, the insertion mechanism checks the ordered structure, which represents the best entry point to begin the scanning of the queue. The cost of this insertion is the scanning of the ordered structure and of that part of the queue detected by the entry point. The management of the ordered structure is dynamic because it is bound to the service queue. In fact, when the action referenced as entry point is extracted (i.e. S-departure and departure event) the structure must delete this pointer. Vice versa if a new inserted action becomes a new entry point, a pointer must be added to the structure.

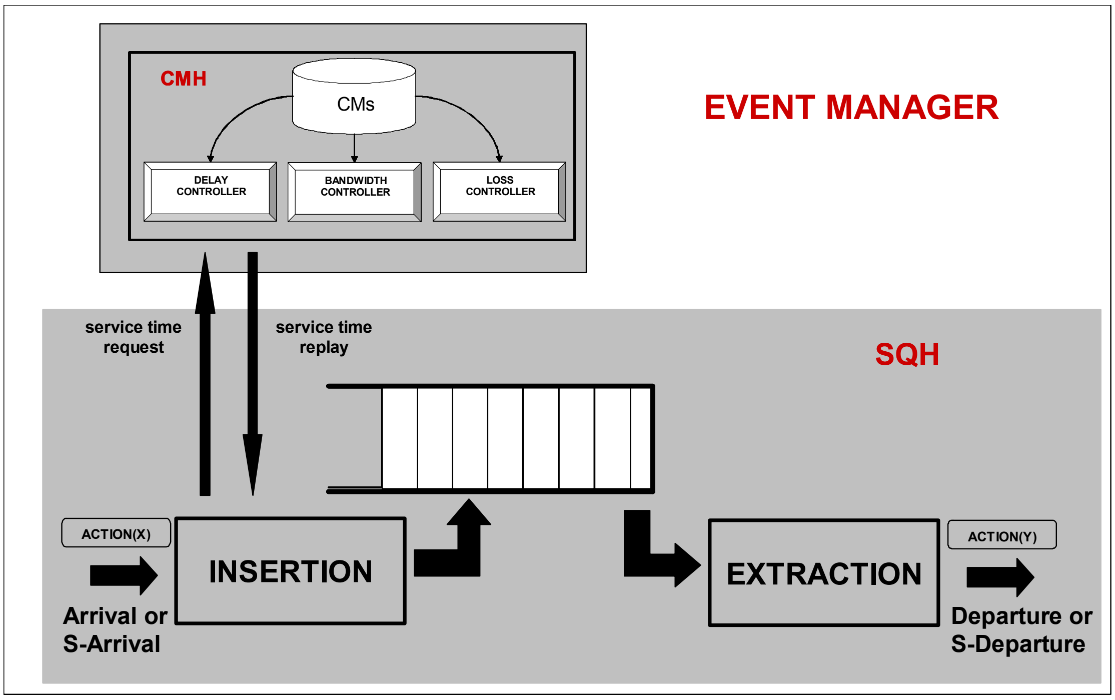


Figure 5.6: Event Manager: CMs, Controllers and Service Queue

The policies to decide which of the actions should be referred to as entry point might be different. In our implementation, the policy is to sample the queue with a fixed number of actions: this means that each entry point is referred to the next “n” actions (where “n” is configured by the simulationist). For example, the length of the queue is thousand and “n” is equal to hundred, (i.e. between two entry points there are hundred packets). In the worst case, the insertion mechanism scans 110 actions instead of a thousand, where ten is the size of the ordered structure and hundred is the sample size.

5.2.2 Communication model handler

The second task of the EM (strongly related to the first) is to calculate the new position (i.e. service time) of an insertion (of an action) into the service queue by means of a Communication Model (CM). A CM reproduces the delay caused by network communications between a real host and a simulated node or between two simulated nodes. In essence, when an Arrival or an S-arrival event is generated, an action related to a new incoming packet (in the first case) or related to an old packet (in the second case) is inserted (or inserted again) into the service queue. The node that has generated the Arrival (real host) or the S-arrival (simulated node) event must specify the parameters of CM and the next hop for this packet. In this way, the CM handler can compute the service time for this IP packet. The parameters of a CM depend on time granularity of the simulation environment and on emulation abstraction layer (i.e. Network Layer Emulation):

- Delay/ Latency
- Bandwidth limitation
- Packet loss

Then, the EM implements a Delay Controller, a Bandwidth Controller and a Loss Controller (see the Figure 5.6). Each single controller can be configured to work in several modes. Further, the three controllers can be combined together in different ways to model several CMs. In particular, each controller produces, as result, the time that the IP packet must spent according to its mode. Hence, the (service) time computed by CM includes the contribution of each configured controller.

5.2.3 Delay controller

The Delay Controller is in charge of reproducing the delay either due to the propagation time of a channel (i.e. single link) or due to the communication time of a path comprehensive of several links and routers. It can be configured to operate in five modes, each of which returns a time as contribution to the computation of the service time.

- *Fixed delay*: in this mode a constant delay is specified by the simulationist and all the packets passing through the channel between two nodes are delayed by this amount. This mode can be used to reproduce the fixed components of delay in Internet paths.

- *Defined delay pattern*: in this mode the simulationist specifies with a file the sequence of delays, which is read by the Delay Controller (in cycles). For example, the user might specify the delay sequence as 100 ms, 200 ms and 300 ms. The Delay Controller would then delay the first, fourth by 100 ms, the second, fifth by 200 ms, the third, sixth by 300 ms and so on. This mode is helpful in debugging network protocols because it provides a high degree of control over the delays.

- *Empirical delay distribution*: in this mode the simulationist specifies a file containing the delay values collected during different trails. Once a packet requests a delay to this controller the corresponding value in the file is chosen as the next delay value. This mode can be used to accurately model delay patterns observed in the Internet.

- *Analytical law*: in this mode the simulationist can use pre-specified analytical laws based on suitable parameter. For example, if the suitable parameter is the packet dimension, when a packet arrives to Interceptor it is possible to calculate the new delay value; checking the size of the packet and using that formula (see the Paragraph 6.2).

- *Stocastical model*: in this mode the simulationist can use pre-specified stocastical models. When a packet arrives to Interceptor it is possible to calculate the new delay value through an appropriate statistical distribution. Interceptor presents different wired/wireless models (see the Paragraph 6.1).

5.2.4 Bandwidth controller

The bandwidth of a channel is one of the most important parameters affecting network performance. A reasonable approach to introduce bandwidth limitation is a "leaky bucket". A leaky bucket limits outgoing traffic to the specified bandwidth. Surplus traffic is stored in a FIFO queue. If the queue is full, incoming packets are dropped. Leaky bucket behavior is determined by the bandwidth and the queue length. The packets in the queue are served at a user-defined rate of "M" bits per second. The value of "M" sets an upper bound on the bandwidth available to packets passing through the link. For example if M is set to 10 Kbps it is possible effectively to emulate a 10 Kbps line on a 10 Mbps link. Therefore the value of M determines the bandwidth of the channel. The bandwidth can be limited for each direction separately. The maximum queue length is another configurable parameter.

A better possible approach to reproduce limitation due to the bandwidth is to build a FIFO queue where a simulated background traffic injector introduces daily traffic. In fact, to mimic a network scenario, also the real daily traffic coming from other applications (or protocols) must be reproduced. For this reason, it must be possible to distinguish between live traffic coming from the applications (or protocols) under study and the background traffic that represents the traffic coming from all other applications used daily on the Internet. Hence, a simulated background traffic injector to reproduce the effects of the daily traffic has been implemented. The background traffic should be expressed as inter-arrival time and packet dimension that take up space in the FIFO queue. In this way, the live traffic will compete with the background one to get positions in the FIFO queue (Dam *et al.*, 1998). The background traffic generation may be used to calculate in a more realistic way the transmission time of IP traffic between two nodes (for example two routers). This approach allows one to simulate the taking up in the node (router) queues. The implementation of our coarse-grained approach (i.e. to maintain the computational performances, without affecting the simulative consistency) is possible by processing in the live and background packets.

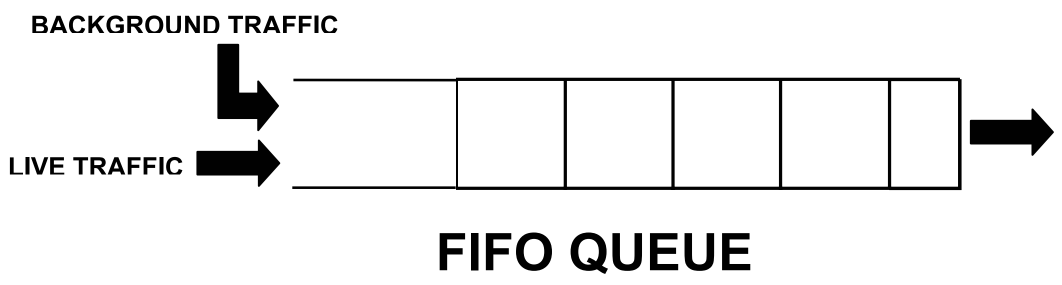


Figure 5.7: the combination of live and background traffic

Therefore, live packets are processed one by one, while all background ones are processed in a cumulative way. In particular, this means that the injector will not create real background packets, but it will calculate as bytes that will take up space in the queue between two live packets. Hence, the bandwidth controller handles the injector mechanism as shown in the Figure 5.7. By controlling parameters like the maximum queue length and bandwidth value, and with an appropriate background traffic injector, many interesting delay and loss patterns can be generated by this bandwidth controller. The controller returns a time (according to the maximum queue length and to the bandwidth limitation) as a contribution to the calculation of the service time.

5.2.5 Implementation of the Background Traffic Generation

Here below, how to model the background traffic model is shown:

- $P_0, \ldots, P_n$ the IP packets inside the FIFO queue,
- $T(P_j)$ the inter-arrival time (in the FIFO queue) of the "j-th" IP packet,
- $S(P_j)$ the size of the "j-th" IP packet,
- And B the bandwidth of the queue.

Where $\bar{P}_i$ is the "i-th" live packet and P_i is the "i-th" background packet. Our attempt is to describe the background traffic as the inter-arrival time of the background packets $T(P_j)$ and their sizes $S(P_j)$ by means of the Fractional Sum Difference model (FSD) (Cao *et al.*, 2002a; Cao *et al.*, 2002b). This model allows the simulationist to compute the inter-arrival time and the size of an IP packet by means of statistical distributions. In particular, these two characteristics depend on Active Connection Load (ACL). An ACL measures the number of transmitting connections that use the same link. The given distributions show the probability to find specific values of the inter-arrival time or size depending on ACL. To obtain these results (x = $T(P_j)$ or x = $S(P_j)$) their reverse functions are necessary. In the following formula “x” is the inter-arrival time of an IP packet (in the bandwidth queue) and Φ is the ACL measure:

$$F(x,\Phi) = 1 - e^{\left(-x/\beta\right)^{\alpha}}$$

Where α and β depend on Φ. Instead, in the next formula, “x” is the size of an IP packet and Φ is the ACL measure (see the Figure 5.8):

$$\text{F(x,}\Phi\text{)=} \begin{cases} 0 & \text{if } x < 40 \\ \Psi(40) & \text{if } x = 40 \\ \Psi(40) + \dfrac{(x-40)*(1-\Psi(1500)-\Psi(40))}{1460} & \text{if } 40 \leq x < 1500 \\ 1 & \text{if } x \geq 1500 \end{cases}$$

Where Ψ(40) and Ψ(1500) depend on Φ (i.e. assuming that 40 bytes is the minimum size of an IP packet, while 1500 is the maximum). The background traffic injector generates a series of packet sizes and their inter-arrivals before the beginning of an Interceptor run, and stores them inside a data structure. During the experiment, the bandwidth controller checks in this data structure how many background packets are injected between the incoming live packet and the last one in the FIFO queue. The Figures 5.9 and 5.10 show two possible states of the FIFO queue when an incoming real packet arrived. The inter-arrival times of each packet are positioned on the X axis, while their sizes are located on the Y axis. The arrow shows the next expiring packet. Then, to know how much time a live packet waits in the queue, it is necessary to compute how many bytes must be processed before this. Every time an incoming packet enters the FIFO queue, the possibility is:

- That there are other real packets before it,
- That there are no real packets before it,
- And that the queue is full.

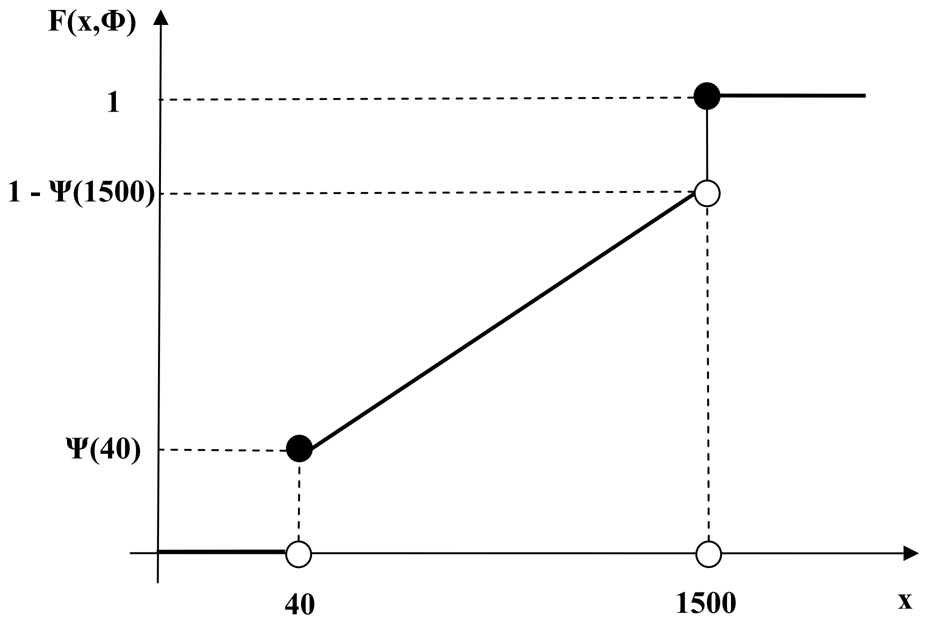


Figure 5.8: Statistical distributions of IP packet size

The Figure 5.9 describes the state of the queue in the first case. The following formula allows one to compute the quantity of data "X" (expressed in bytes) between the last real packet $\bar{P}_j$ (arrived at the time $T(\bar{P}_j)$) and the incoming real packet $\bar{P}_n$ (just arrived at the time $T(\bar{P}_n)$):

$$X = \sum_{i=j}^{n-1} x_i \quad \text{where}$$

$$x_i = \begin{cases} 0 & \text{if} \quad S(P_i) - ((T(P_{i+1}) - T(P_i)) * B \le 0 \\ S(P_i) - ((T(P_{i+1}) - T(P_i)) * B & \text{if} \quad S(P_i) - ((T(P_{i+1}) - T(P_i)) * B > 0 \end{cases}$$

Hence, the time that the packet $\bar{P}_n$ must wait in the bandwidth queue (defined $W(\bar{P}_n)$) is computed by the following formula:

$$W(\bar{P}_n) = (W(\bar{P}_j) - (T(\bar{P}_j) - T(\bar{P}_n))) + \frac{X}{B} + \frac{S(\bar{P}_n)}{B}$$

Therefore, this time value is the sum of the three time values (i.e. those divided by the plus): i) that packet $\bar{P}_j$ should wait before being processed, ii) that to process the packets generated by the background traffic and finally iii) the time to process the packet $\bar{P}_n$ itself.

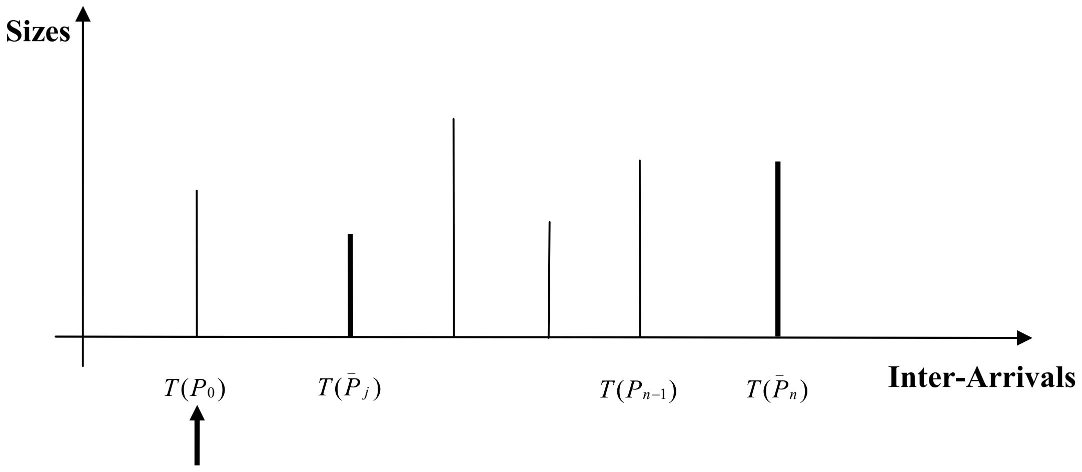


Figure 5.9: The state of the queue with other real packets

The Figure 5.10 describes the state of the queue in the second case. The following formula computes the quantity of data "X" (expressed in bytes) between the first background packet P_0 (arrived at the time $T(P_0)$) and the incoming real packet $\bar{P}_n$ (just arrived at the time $T(\bar{P}_n)$):

$$X = \sum_{i=0}^{n-1} x_i \quad \text{where}$$

$$x_i = \begin{cases} 0 & \text{if} \quad S(P_i) - ((T(P_{i+1}) - T(P_i)) * B \le 0 \\ S(P_i) - ((T(P_{i+1}) - T(P_i)) * B & \text{if} \quad S(P_i) - ((T(P_{i+1}) - T(P_i)) * B > 0 \end{cases}$$

Hence, the time that the packet $\bar{P}_n$ must wait in the queue is computed by the following formula:

$$W(\bar{P}_n) = \frac{X}{B} + \frac{S(\bar{P}_n)}{B}$$

Therefore, this time value is the sum of the two time values i) that to process the packets generated by the background traffic and ii) the time to process the packet $\bar{P}_n$ itself. In the last case, the queue is full and the real incoming packet is discarded. This is due to the fact that a queue has a maximum length.

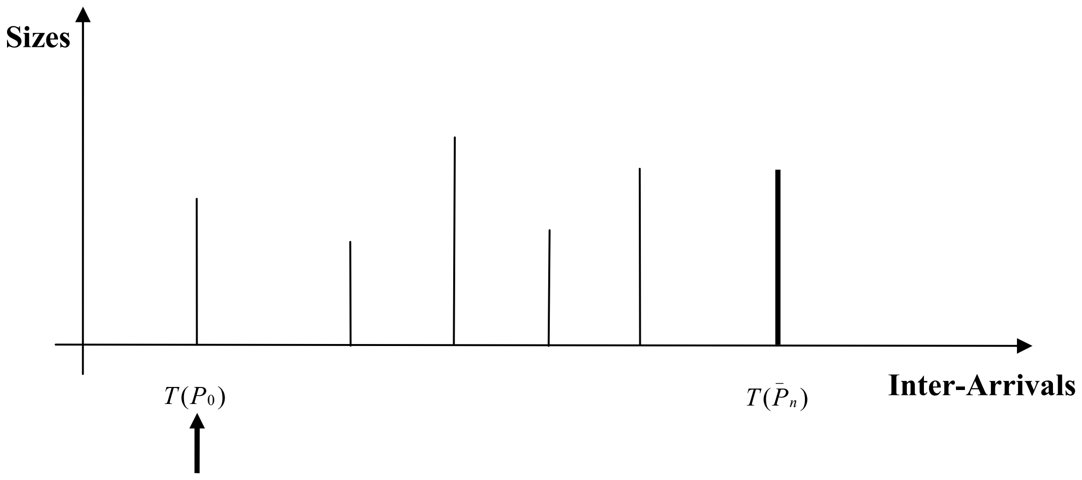


Figure 5.10: the state of the queue in the second case

$W(\bar{P}_n)$ is the contribution to the computation of the service time. This approach shows two main limitations. The first is that each pre-generated data structure containing the background traffic is dependent on the ACL. For this reason, if the background traffic injector must be used on links with diverse ACLs, there is the need for a data structure for each link. The second is the computation of the time that the incoming real packet must wait in queue. In fact, the computation of $W(\bar{P}_n)$ may be very expansive. Then, the best solution should be to have a statistical distribution that depending on ACL directly is able to show how much time the incoming real packet must spend in the queue. One of the main considerations, in reproducing how a router processes the incoming packets, concerns the interaction between the average of the traffic crossing it and its bandwidth constraints. Several works reproduce the router as a point of interconnection among several links where the bandwidth limits are the most important features. This produces two main possible effects deriving from the quantity of incoming traffic and from the amount of available bandwidth. In fact, if the average traffic crossing is greater than the bandwidth limit, the derived effect is that the queue is congested. Instead, if it is lower, the queue is always empty. We think that is not enough, because the time consumed by the router in scheduling the packets from a queue to another according to its policy may be non-null. For this reason, it is necessary to reproduce this behavior as in the next Paragraph (see the paragraph 5.3).

5.2.6 Packet Loss controller

The Packet Loss Controller is used to assign loss characteristics to the live traffic. Differently from the other two controllers, Loss Controller does not return a time for the computation of service time but it (directly) discards the packet. It can be configured to operate in two modes.

- *Fixed loss rates*: the unconditional loss probability determines the mean rate with which packets are dropped by the controller. The packets are randomly dropped by the controller.

- *User defined packet loss sequence*: in this mode the user specifies which packets must be dropped out of a sequence of consecutive packets. For example the user might specify that in a sequence of five consecutive packets the second and the third packets must be dropped. With this configuration the controller drops the second and third packets in every batch of five consecutive packets. This mode can be very useful in debugging and analysis of network protocols because it allows the user to selectively drop packets (for example ACK packets).

5.3 Network Topology Manager

The *Network Topology Manager* (NTM) implements a support for network topology modeling and for communication management by means of two network entities: nodes and links. Hence, the NTM represents any network topology as a set of nodes and links. A node represents router or general gateway while links connect two (simulated) nodes or a (simulated) node and a (real) host. Additional information about the network can be added to the topological structure by associating it with the nodes and links. In fact, The NTM is also in charge of indicating to the EM which is the communication model (and which are their parameters) to reproduce the passing of an IP packet from a node to the next one (or from a node to the final host).

The NTM supplies to the simulationist a configuration file where it is possible to specify the network topology scenario. Once the network scenario is started, the NTM is in charge of the management of the communication inside the specified topology. Therefore, when an action (related to an IP packet) must transit through the predefined topology from a node toward its next one, the NTM replies the indication to EM for computing the new service time expiration. Therefore, actions related to the live packets pass alternatively from EM to NTM and vice versa until they reach the last node before their final destination (see in the Figure 5.11 the vertical arrows).

The interaction between EM and NTM works through two events, named S-arrival and S-departure. With an S-departure event, the EM informs NTM that an action (related to a packet) is arrived at a node. With an S-arrival event, the NTM informs the EM that a new action (related to a packet) generated by a node has arrived. This action specifies the next node and a precise link to reach it. In this way, the NTM indicates the parameters of the communication model to the EM.

As Interceptor is a synthetic emulative platform, the source and the destination hosts are real devices. Therefore, if an action (related to an IP packet) is on the last node (see the node “C” in the Figure 5.11) before the destination host, a final S-arrival event will be generated to communicate to the EM to send this packet after the expiration of the next service time (i.e. the departure time) towards its external networked destination (generating a departure event). Afterwards, the first S-departure event introduces an action related to this packet into NTM to a first node (see the node "A" in the Figure 5.11). In Figure 5.11, it is possible to see an example of the interactions between NTM and EM. In particular, it is possible to see the insertion and the extraction of an action (at different times) from the service queue. In this Figure, the only service queue is represented at different times.

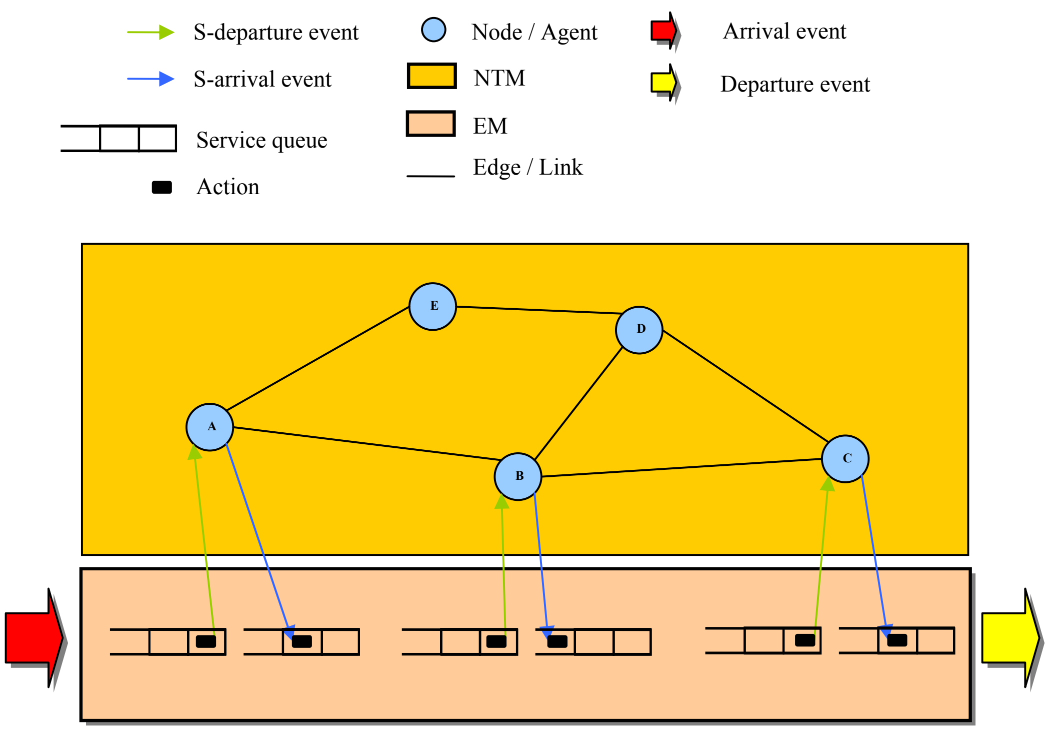


Figure 5.11: Integration of Event Manager with Network Topology Manager

5.3.1 Implementation of an agent-based NTM

The Network Topology Manager is implemented by a multi-agent system, called SPADES. The System for Parallel Agent Discrete Event Simulation (SPADES) is a middleware system for agent-based distributed simulation. SPADES is designed to build agent-based simulations where each computational entity receives sensations from a simulated world (sensing phase), spends some amount of computation (thinking phase) and then returns some actions (acting phase). SPADES provides a world where agents can live. The world model operates by scheduling simulation events realized one by one, and the agents simply receive sensations and send actions.

SPADES has four main components (see the Figure 5.12): a simulation engine, a communication server, agents and a world model. The simulation engine and the communication server are supplied as part of SPADES, while the agents and the world model must be implemented by a user according to the simulation of a particular scenario. The simulation engine adopts a discrete event scheduler to advance the simulation. This component is in charge of queuing all pending events and coordinating all network communications. A communication server is in charge of interconnecting all (TCP/IP) agent communications to the simulation engine. This means that the agents communicate with their communication server via pipes and this server communicates with the simulation engine via sockets (see in the Figure 5.12 where lines are pipes and arrows are sockets).

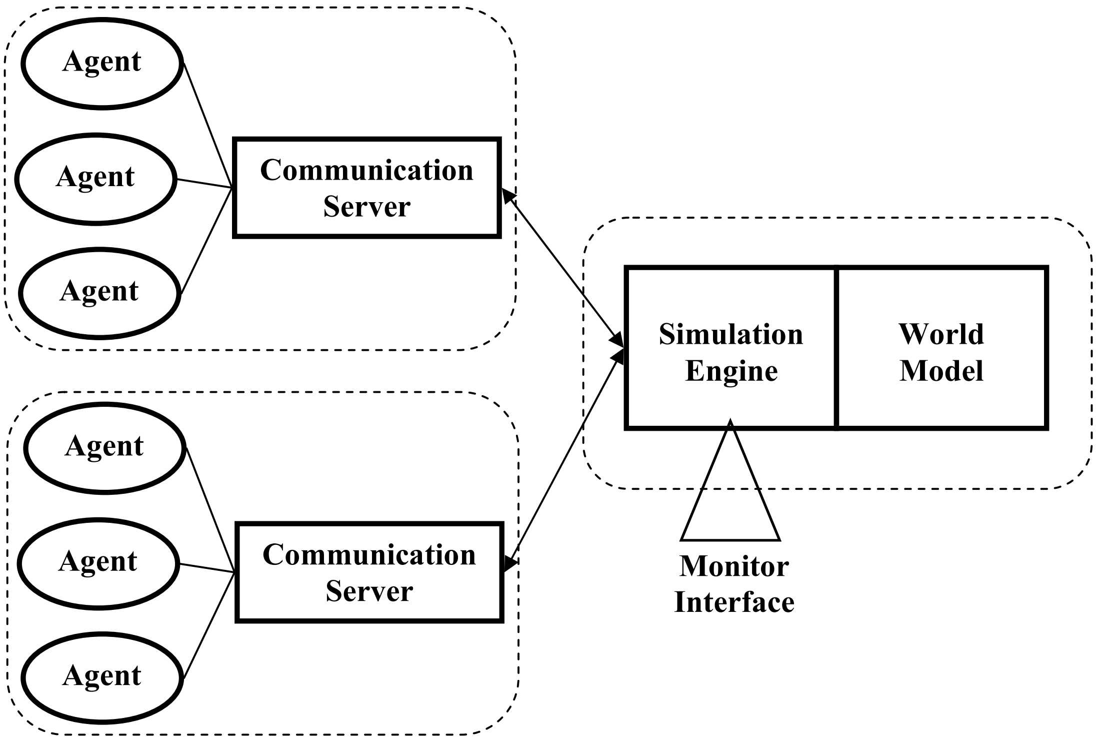


Figure 5.12: The architecture of the SPADES

In particular, if no distributed agents are required it is possible to run an integrated communication server, which is executed in the same process as the simulation engine. The world model is built by a simulationist to reproduce a particular scenario. The world model links the simulation engine as a library to constitute a single process. In particular, the world model inherits the C++ classes of SPADES in order to interact with the simulation engine. Finally, the agents interface with the communication server by means of read and write system-calls over pipes, thus the agents may be programmed as the simulationist wants. Further, it is possible to connect directly the simulation from an external program and to get periodic updates about the evolution of the simulated world (like the reproduction of large scaled networks) by means of a monitor interface. This monitor interface allows one also to send commands back to the simulation engine or to world model to perform different procedures.

In order to create our NTM by means of SPADES, agents and world models are implemented according to the simulation of a network topology (see the Figure 5.13). Along with this idea, the management mechanism of each node is a programmable agent, while the link represents the communications among the agents.
All agents work inside the world. The world is in charge of synchronizing the agents and exchanging the messages between them and the EM. In particular, the EM and the NTM exchange messages by means of the monitor interface. Our world model, (called NTMWorld) after the starting phase where it takes up its configuration, connects all agents and sends them their settings (by means of a *CreateNormSenseEvent*). At this point, the NTMWorld waits for the setup of all agents and only after that it stops waiting for a connection on the monitor interface. When a connection is established with the EM, different messages are exchanged on this channel. In particular, each message contains information for different agents. When the message reaches the NTMWorld is parsed (by means of the *parseMonitorMessage* method) and divided into messages that are sent to the agents. To obtain this result the NTMWorld adopts the *CreateMsgSenseEvent* that creates the senses (that contains the messages) for the specified agents. When the agents receive the senses, they call the *doAction* method to interpret the messages, to check their policy and to write a response message.

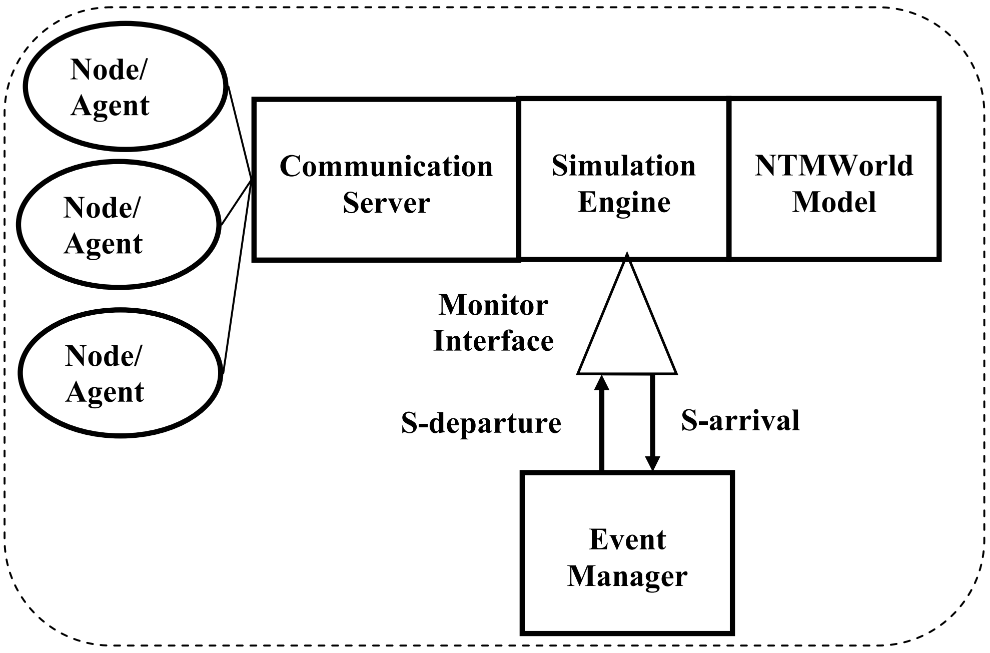


Figure 5.13: The implementation of the integration between the EM and the SPADES as NTM

At the end, the agent sends the response to the NTMWorld that uses the *parseAct* method to call the *MessageToMonitorEvent*. This last SPADES event sends the message to EM through the Monitor interface generating a S-arrival event.

The communication between the NTM and EM works on TCP. An application protocol has been built to exchange information (i.e. actions) between the NTM and EM. When a S-departure event is generated one or more packets have expired their service times and are ready to be delivered to the next nodes of the topology. Then, the EM will build a new message for the NTM that has the following format:

W <SentData> <<idAgent> <idIPPacket> <destination>>……………………………………
……………………………………………………………………………………………………
………………………………………………………… <<idAgent> <idIPPacket> <destination>>

where “W” informs the Simulation Engine that the message is addressed to the world model, <*SentData*> is the number of bytes that the Simulation Engine must receive, <idAgent> is the identifier of the agent that must receive a message, <idIPPacket> is the identifier of the IP packet that expired its service time and <destination> is the final destination that the IP packet must reach passing through the network topology. The Simulation Engine receives the message through the monitor interface, deletes the “W”, waits for <*SentData*> bytes and passes it to NTMWorld. The NTMWorld parses the message deleting the <*SentData*> and dividing it into messages as

<idAgent> <idIPPacket> <destination>

that will be sent to their specific agents (simply reading the <idAgent> field). When the agent reads the message, he creates a response message (like the following) and sends it to the NTMWorld:

<nextAgent> <idIPPacket> <CM>

Where <nextAgent> is the identifier of the next agent that after the expiration of the next service time will receive this action, <idIPPacket> is again the identifier of the IP packet and CM is the specifications of the communication model that EM must use to compute the next service time. This next time expiration will generate a future S-departure (or a future departure event). Once the NTMWorld receives this message, it passes directly through the monitor interface towards EM generating an S-arrival event.

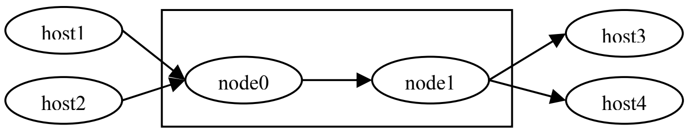
5.3.2 Implementation of an agent as a network node

The management mechanism for each node is represented by an agent and each link is represented by an interaction between two agents. The EM asks to the nodes (i.e. agents) to solve a routing problem for an IP packet. Each agent knows the other ones in its neighbourhood and which is the kind of channel to directly reach one of them. This means that each node has its routing table and also a communication table (where it is specified which is the CM that describes the link to reach directly the next agent). This information (i.e. routing and communication settings) is read from the agents at the beginning of an experiment by means of a *CreateNormSenseEvent* (see the Figure 5.14).

There are two possible kinds of links: emulated and simulated. In the first, the link is between a simulated (agent) node and a real host. This node can be at the beginning of the emulated link or at the end. In the second, both nodes are simulated agents interconnected by a simulated link.

Once it gets a request, each agent supplies a replay where it specifies the next agent, the identifier of the IP packet and an CM. The CM specifies the kind of link to reach the next agent and the parameters, the delay due to the node (as router) computation and if the link is down or up. In the first case, it is possible to specify the characteristics of the link between the two nodes. In the second, a further delay that reproduces the router computation can be specified. This delay is derived from the router management, like time to read routing table, time to get a packet from a link, time spent in the entry router queue and time to insert packets into the exit queue for a specific link. In the third, a link can go down or up at a “t” time from the beginning of the Interceptor experiment. In this case, when a router sends a packet over a link that is down, it discards this packet. In particular, each packet that is over a link when it is going down is discarded.

Further, it is possible to specify more routes towards the same final destination (inside a routing table). This means that it is possible to manage the load-balancing policy by indicating preferences among routes. When an agent chooses the link to the next agent (toward the final destination) he must take into consideration the preference (and also if the link is up or down). As a final consideration, there is also the possibility to manipulate both nodes and links at a run-time simply acting on the agents (i.e. in interactive mode). A second tool connects the agents through the world by means of the monitor interface. This tool permits the simulationist to change by means of a command shell the settings of all agents by adopting the same syntax of the configuration file. In this way, it is possible at a run-time to update: i) the routing table of any node, ii) the CM between two nodes (i.e. the communication table), and iii) the network topology (without the possibility to create new nodes). This operation is executed in mutual exclusion and can involve single nodes or a group of these. Similar to routing and communication settings at the starting phase, this operation updates the data structures of the agents at a run-time.



```
# simulated link
#nSrc nDst  lQueue  delay  lDown    lUp
node0 node1 r10000 f40000 d50000 u80000

# emulated link (real source)
#Src  Dst    nDst      delay
host1 host3 nnode0 f30000
host1 host4 nnode0 f40000
host2 host3 nnode0 f50000
host2 host4 nnode0 f60000

# emulated link (real destination)
#Src Dst   lQueue delay
node1 host3 r150 europa_gprs_buona v4 gauss
node1 host4 r1000 f30000

# route
#nSrc Dst     nHop
node0 host3 nnode1
node0 host4 nnode1
node1 host3 nnode255
node1 host3 nnode255
```

Figure 5.14: The configuration file of the NTM for the above scenario

5.4 Simulating Communication Time without Additional Delay

The emulative interface of Interceptor introduces live traffic into a synthetic environment that processes this traffic by means of different software components. A synthetic emulative platform must support the live traffic under real time constraints without adding additional delay due to the time spent to schedule their software components. Unfortunately, an excessive amount of incoming real packets (i.e. burst traffic) may cause the platform to miss time deadlines (i.e. the system becomes livelocked (Fall, 1999)). In essence, IP packets may leave late because their actions are not extracted from the service queue at the right moment. It is possible to avoid the livelock by exploiting a prefetching technique. In particular, we define the lookahead as the duration of the time period between the imminent action (related to a specific IP packet) and the last event occurred.

The value of the lookahead can affect the simulation performance and the real time constraint of the emulation. In fact, in this period, it is possible to run the software components of the simulative environment and the emulative interface avoiding the livelock. Unfortunately, if real hosts generate burst traffic, the lookahead period is reduced by involving the performance of Interceptor. This burst traffic may generate a high number of events within a small time frame during which Interceptor might not keep up with the real time. The interference between these processes and the real time constrains depends on the duration of the lookahead. In fact, the greater the lookahead, the lesser the probability that the interference causes additional delay. Therefore, every time an action is not extracted by the queue according to its timeout expiration, the additional delay grows. In particular, this means that every time an S-departure event or a final Departure event is not generated according to their timeout expirations, the IP packets leave late the simulated links. Then, the calculation of the time spent (for each IP packet) before the departure moment may be expressed by the following formula:

$$\text{Time}_{\text{departure}} = \text{Time}_{\text{arrival}} + \text{Time}_{\text{waiting}}$$

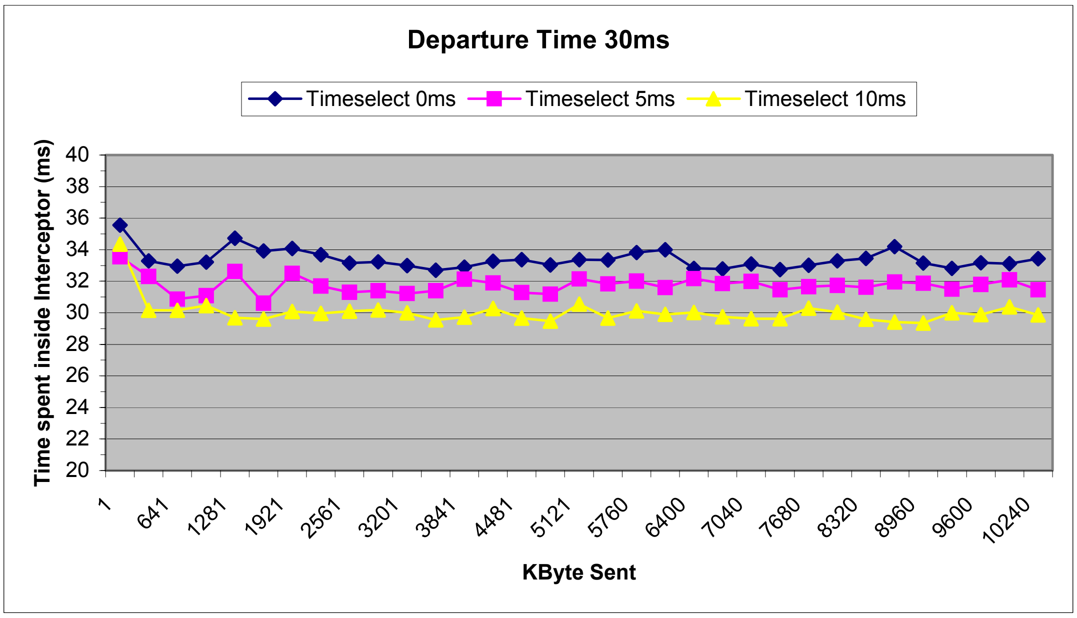
The Netfilter framework writes the characteristics of each live incoming packet on a structure, included the arrival time on the Ethernet card (i.e. $\text{Time}_{\text{arrival}}$). $\text{Time}_{\text{departure}}$ is the moment in which the packet leaves Interceptor. $\text{Time}_{\text{waiting}}$ is the sum of the times assigned by the synthetic scenario to the emulated or simulated links by means of several CMs (i.e. from CM(1) to CM(k-th)):

$$\text{Time}_{\text{waiting}} = \text{Time}_{\text{CM(1)}} + \text{TimeLocked}_{\text{CM(1)}} + \ldots + \text{Time}_{\text{CM(k-th)}} + \text{TimeLocked}_{\text{CM(k-th)}}$$

$\text{TimeLocked}_{\text{CM(x)}}$ is the extra time spent if during the passing through the “x” link a livelock occurs. In an ideal experiment all the TimeLocked components are null. In particular, the greater the number of the links between two real hosts, the greater the probability to accumulate additional delay. Being the service queue the repository of all actions of our synthetic scenario, it is very likely that several of them could expire very close one to the others. This situation is very dangerous because if the next actions have very short time expirations, they are forced to wait for “n” cycles of program execution (i.e. waiting for a higher time). In order to limit additional delay growth and in order to remove it, a mechanism that supports a prefetching technique has been developed. This mechanism is able to extract from the service queue all the actions that could violate their timeout expirations. It is possible to compute the next expiring actions forcing the next lookaheads to zero. In particular, every time an action reaches the expiration moment, the mechanism checks up the next ones that fall inside a period given by the current instant plus a time constant called time-select. The time-select is currently set lesser or equal to the time granularity of the machine. All actions in the time-select interval are extracted immediately and their events are generated. In a complex scenario, an action related to an IP packet might pass through different emulated or simulated links. This implies that the action is inserted and extracted several times. The mechanism does not affect the $Time_{departure}$ making it lower than the value calculated by the previous CM. This is possible because, during each insertion phase, the next time expiration is computed by adding to the previous one the new contribution due to the new CM. This means that the moment during which the previous event was generated is not taken into consideration, while the previous time expiration is taken into account. In this way, the least possible inconsistency is generated; that is for a time lesser than the granularity precision this action has been inserted into the next link in advance. In this way, it is possible to amortize the delay of different inserting and extracting phases, without affecting the $Time_{departure}$ by releasing the packet too early or too late. In a few words, the time-select amortizes the delays due to the additional delay. In this way, this mechanism tries to perform the extraction of actions without affecting the real execution of the running applications (see an example on the Figure 5.15).

5.5 The Schedule of Interceptor

Now, we explain how Interceptor works by means of the scheme of its states (see the Figure 5.16). After the starting phase, where all the settings are configured, Interceptor enters an endless cycle during which it passes through several states depending on different events. The first state of the cycle is the QUEUE. In this state, the content of the service queue is checked up.



KByte	Time-select 0		Time-select 5		Time-select 10	
1	35,55055	±2,50330	33,54004	±2,05924	34,35844	±2,53646
321	33,28769	±0,33491	32,29506	±0,41853	30,16315	±0,14194
641	32,95216	±0,22955	30,86735	±0,18609	30,15103	±0,16437
961	33,20946	±0,19279	31,08551	±0,21429	30,44385	±0,21015
1281	34,72764	±0,49501	32,61661	±0,24204	29,69390	±0,11455
1601	33,90771	±0,28037	30,61524	±0,13148	29,59947	±0,09269
1921	34,08215	±0,24056	32,49469	±0,29359	30,08475	±0,13992
2241	33,67372	±0,12942	31,69343	±0,16387	29,96314	±0,12108
2561	33,15287	±0,11001	31,30406	±0,14944	30,10470	±0,10630
2881	33,22671	±0,17873	31,39224	±0,13000	30,18140	±0,11717
3201	32,99217	±0,10012	31,22876	±0,12132	29,99103	±0,10732
3521	32,70544	±0,08689	31,39384	±0,11286	29,55307	±0,08368
3841	32,89831	±0,11848	32,11900	±0,13903	29,71712	±0,07990
4161	33,26725	±0,13019	31,88860	±0,12273	30,27629	±0,10030
4481	33,36183	±0,17762	31,28810	±0,09998	29,67142	±0,07496
4801	33,02575	±0,07641	31,17460	±0,09513	29,47609	±0,07063
5121	33,36983	±0,10912	32,14417	±0,12268	30,55266	±0,11153
5440	33,34212	±0,07551	31,82486	±0,11441	29,66567	±0,06807
5760	33,81028	±0,10822	32,00908	±0,10957	30,12530	±0,08249
6080	33,98785	±0,08418	31,59159	±0,09165	29,91044	±0,07281
6400	32,81368	±0,06508	32,17445	±0,10829	30,01412	±0,07761
6720	32,76928	±0,06409	31,85753	±0,09512	29,75066	±0,07315
7040	33,09119	±0,11542	31,99053	±0,09971	29,62027	±0,06397
7360	32,72658	±0,05999	31,46644	±0,08684	29,62752	±0,05778
7680	33,02072	±0,07727	31,64469	±0,08623	30,28782	±0,07957
8000	33,29145	±0,08255	31,73705	±0,08762	30,03947	±0,06953
8320	33,45153	±0,07955	31,60649	±0,07891	29,59380	±0,05280
8640	34,20061	±0,08862	31,94323	±0,08769	29,41757	±0,05267
8960	33,14709	±0,05868	31,86399	±0,09259	29,35969	±0,04606
9280	32,81908	±0,06904	31,50544	±0,07914	30,00423	±0,06884
9600	33,16915	±0,10033	31,79180	±0,08266	29,89044	±0,06270
9920	33,10744	±0,08667	32,09470	±0,08561	30,38015	±0,07799
10240	33,42284	±0,06771	31,46365	±0,06932	29,87044	±0,06081

Figure 5.15: top, the effect of the mechanism supporting lookahead real time in a scenario where the time spent is 30 ms and the time-select varies from 0 to 10 ms; bottom, confidence intervals at 95% of probability

The first time, the queue is empty, so, Interceptor goes to SELECT, where it suspends itself waiting for the generation of an event that could come from either the EI or the NTM. If the EI generates an event, Interceptor goes to ARRIVAL. Instead, if NTM generates an event, it goes to S-ARRIVAL. In ARRIVAL state, a real IP packet is extracted from the inter-arrival queue that is located inside the EI and inserts into the EM, where a new action containing the information about this packet is inserted into the service queue.

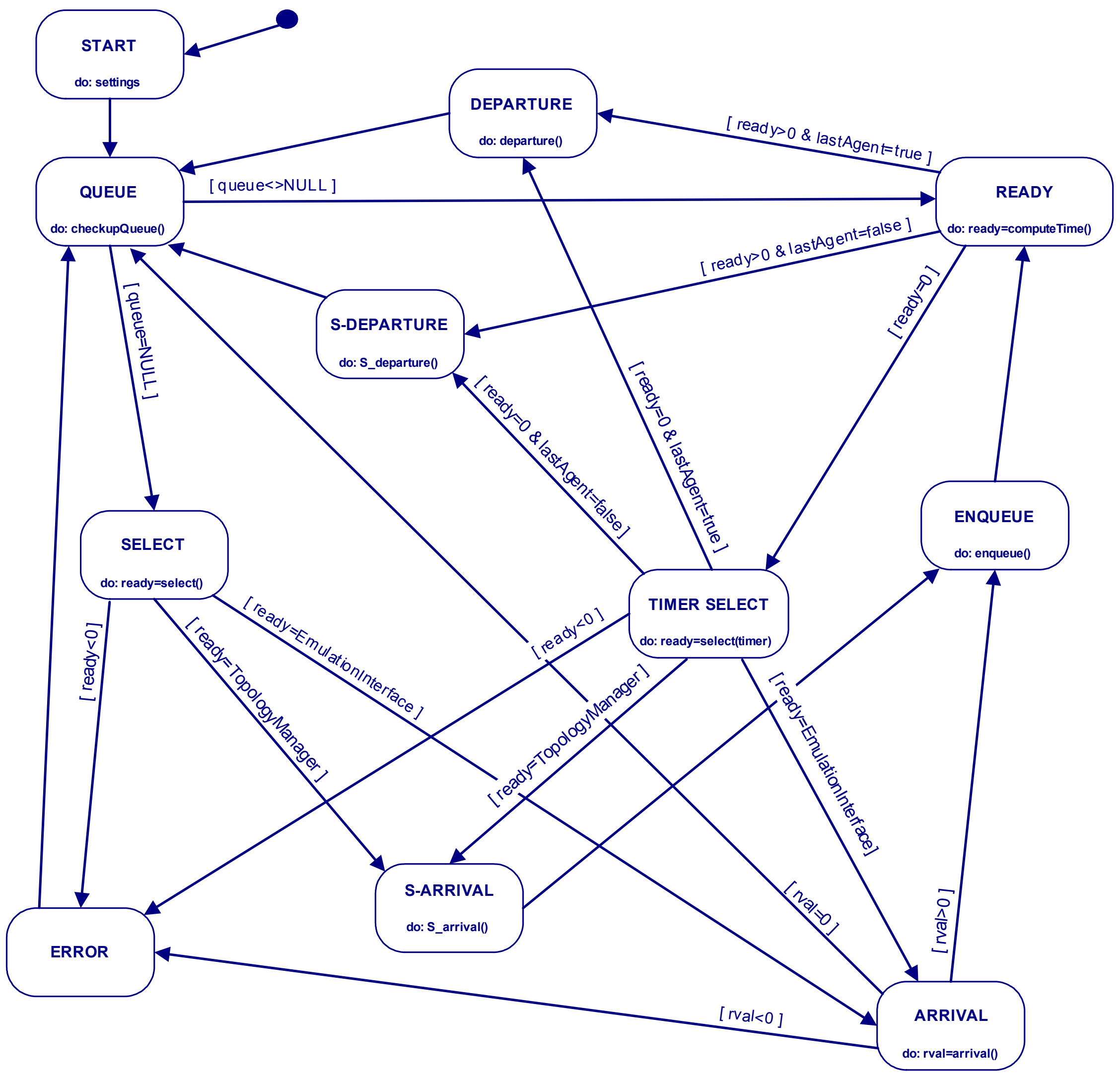


Figure 5.16: The scheme of the states of Interceptor

This insertion brings Interceptor into the ENQUEUE state. The only next state is READY. Here, the “computedTime” function establishes the amount of actions ready to be extracted (i.e. this function implements the prefetching mechanism). From this point it is possible to reach three states.

- If the number of the actions that must be extracted is zero, Interceptor goes to the TIMER SELECT where it waits for the expiration of the service time of the imminent action or for an event generated by EI or by NTM. Otherwise if this number is greater than zero, the field nextAgent of each action is checked up.
- If nextAgent is equal to LASTAGENT, the action sends the IP packet to the final destination generating a departure event. In this case, Interceptor goes to the DEPARTURE, where the real packet is inserted into the live network towards its final destination.
- If nextAgent is different from LASTAGENT, the action sends a message to NTM generating a S-departure event. In this case, Interceptor goes to the S-DEPARTURE. In particular, in this last state the action related to the IP packet is sent to the specific node of the NTM.

From the S-DEPARTURE and the DEPARTURE events Interceptor can go only to the QUEUE state. The TIMER SELECT is the most critical state because it allows the Interceptor to go to the five main states. At this point Interceptor calculates the lookahead time and interrupts itself until this time expires or an event is generated. If during this period an event is generated by EI or by NTM, Interceptor goes, respectively, to the ARRIVAL or the S-ARRIVAL. Otherwise if the time expires, it goes to S-DEPARTURE or DEPARTURE. If Interceptor arrives to the S-ARRIVAL it means that a new action from the NTM has reached the EM and it specifies the CM and the parameters to compute the next service time. Once the timeout has been calculated, the new action (related to a specific IP packet) is inserted according to the service time in the service queue (ENQUEUE). Finally, there is also an ERROR state, where all spurious data structures are deleted and Interceptor is forced to go to the QUEUE state.

5.6 Implementation of the Workload Traffic Generator

The WTG fills the inter-arrival queue by sending IP packets (in emulation mode) according to the communication patterns typical of an application (or protocol). In this way, it exercises the synthetic networks and the final hosts as if multiple users were simultaneously running the same application. In particular, our WTG, called M^{2}G (MultiMedia workload traffic Generator), reproduces the downloading requests, coming from Web and Multimedia application and in particular generated by mobile devices (such as Personal Digital Assistant and Laptop). M^{2}G includes the two following software components: a workload generator and a traffic generator. The workload generator reproduces the behavior of multiple users generating typical communication patterns. Therefore, it implements the set of distributional models that indicates how requests must be done. The traffic generator produces the requests for the multimedia service according to the workload generator by implementing the User Equivalents. Hence, it implements the communication protocol of the application generating the correspondent real traffic. This code is an exchangeable part of the traffic generator, because different application (or protocol) implementations can be built for different multimedia services. In particular, the M^{2}G derives from the SURGE and the GISMO workload traffic generators.

5.6.1 M^2 workload generator

The workload is generated by means of five routines (developed in C-ANSI) executed in the following order (see the Figure 5.17):

1. Zipf
2. Sizes
3. Match
4. Lru
5. Surfoff

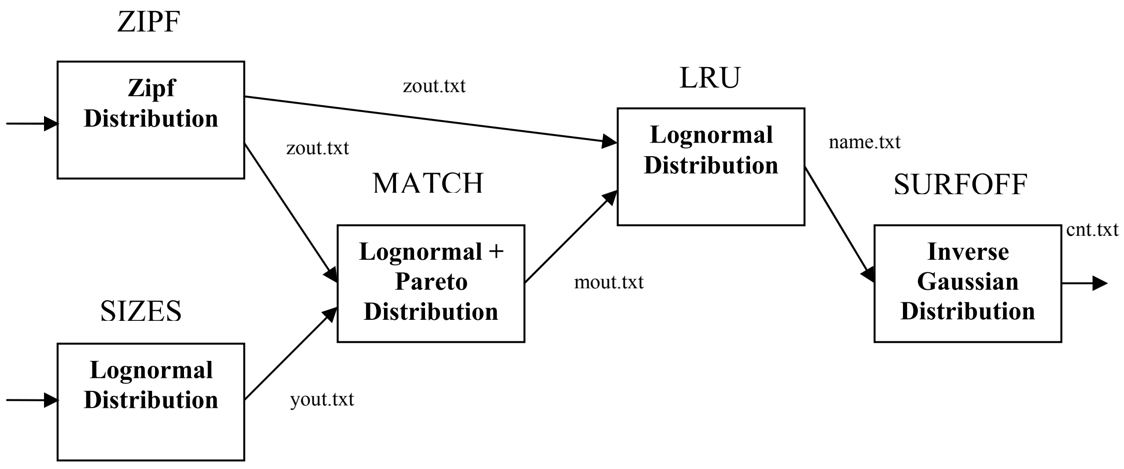


Figure 5.17: The architecture of M²G workload generator

Each routine gets (as input) parameters or files and creates (as output) files. The output files become either the input of the next routines or the repository of information that will describe the traffic patterns. If a simulationist wants to change the parameters of any routines, it will be necessary to execute the workload generator again to get the correct input files. The workload generator implements the following set of distributional models.

The "Zipf" routine implements the "Popularity" distributional model that describes the total number of requests made to each file on the server, during the experiment. The "Zipf" routine of SURGE requires two input parameters: i) the total number of requests for the most popular files and ii) the total number of files used in the experiment. Differently from SURGE, our "Zipf" routine requires just a single input parameter that is the total number of files used in the experiment. In this way, the GISMO approach that does not set an upper bound to the maximum number of requests for the most popular files is adopted. The "Zipf" routine creates an output file, called "zout.txt" which is a two column list: first column contains the indexes of the files stored on the servers, while the second column contains the total numbers of requests (for each of them) in descending order.

The "Sizes" routine implements the "File Sizes", that is a distributional model that describes the collection of files stored on the server according to empirical measurements. The "Sizes" routine requires just a single input parameter that is the total number of files used in the simulation. The “Sizes” routine of SURGE is broken down into three separate models: one for HTML pages with embedded images, one for embedded images and one for pages without images. Differently from SURGE, our “Sizes” routine uses just a model for the multimedia files according to GISMO. Further, the workload generator provides with a module that simulates a multimedia user downloading through a PDA. In this case, this distribution is adapted to the size of the file downloaded (from 10 KB to 10 MB) on these devices that have a limited quantity of memory (≈ 32 MB). The “Sizes” routine creates an output file, called “yout.txt” which is a column list (i.e. the sizes of the files that are set up on the server).

The “Match” routine implements the “Request Sizes” distributional model that generates a one-to-one match between the file sizes and the total requests. The “Match” routine requires two input files and a single input parameter: i) the “zout.txt” (output of the “Zipf” routine), ii) the “yout.txt” (output of the “Sizes” routine) files and iii) which determines which of three matching algorithms to use. In particular, “1” the value can be the optimal matching over the entire function, “2” the value can be the optimal matching focusing on the tail only and “3” the value can be the optimal matching focusing on the head only. The “Match” routine solves the problem of matching the request size distribution with the collection of file sizes stored on the server. The “Match” routine creates an output file, called “mout.txt”, which contains the one-to-one mapping of total requests (column one) to file sizes (column two). This is the original routine of SURGE.

The “LRU” routine implements the “Temporal Locality” distributional model that generates the actual sequence of multimedia files that will be requested by traffic generator. The “LRU” routine requires two input files: i) the “zout.txt” (output of the “Zipf” routine) and ii) the “mout.txt” (output of the “Match” routine). The main difference between our implementation and that of SURGE is here (see the Figure 5.18). In fact, also the “LRU” routine of SURGE requires two input files, respectively, the “objout.txt” (output of the “Object” routine) and the “mout.txt” (output of the “Match” routine). The “Object” routine (not present in our workload generator) is the program that generates the descriptions of the objects, which will be requested by SURGE and it is the implementation of “Embedded References” distributional model. Each object consists of either a loner file or the combination of a base file and one or more embedded files.

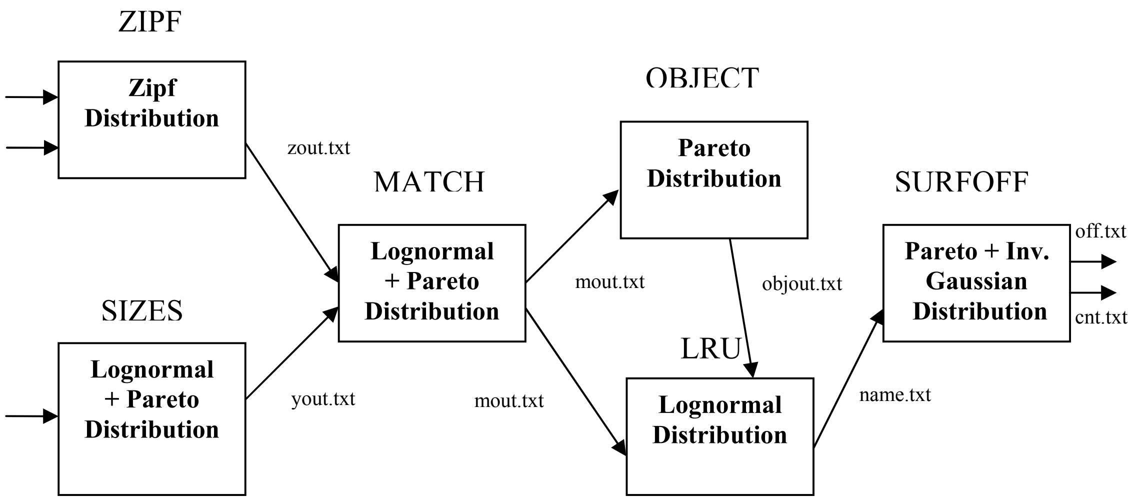


Figure 5.18: The architecture of SURGE workload generator

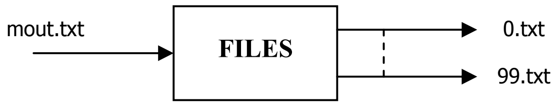


Figure 5.19: The routine that creates the files for the multimedia server

The "Object" routine requires "mout.txt", as an input file. The "Object" routine creates an output file, called "objout.txt" that contains the identifiers of files, which are assigned to each object. The identifiers are actually indexes into the "mout.txt" file. In our design the multimedia files have not embedded files, therefore the "Object" routine is not necessary.

For this reason, differently from SURGE implementation, our "LRU" routine requires the index of all files (i.e. those in "zout.txt"). The aim of this routine is to generate the actual sequence of files that will be requested by traffic generator. The "LRU" routine creates an output file, called "name.txt" that contains the sequence of files that are indexed in the file "zout.txt".

The "Surfoff" routine implements the "inactive OFF Times" and the "Session Length" distributional model. In the former case, this routine generates a set of OFF times that are interposed among the requests during an experiment. In the latter case, this routine generates the session length in terms of the numbers of multimedia files, which are to be requested during a session period. The "Surfoff" routine requires the "name.txt" (output of the "LRU" routine), as input files. The "Surfoff" routine of SURGE generates the

"OFF times" limited the maximum to 30 minutes. Differently from SURGE, in our implementation the value of the "inactive OFF Times" is equal to zero. This is because the user (in our design) makes requests to the multimedia server by means of a list. The time to download the multimedia files contained in the list represents the session period and the number of multimedia files represents the session length. The "Surfoff" routine creates two output files: i) "off.txt" that contains the set of off times in milliseconds (i.e. set of zeros in our implementation) and ii) "cnt.txt" that contains the set of file numbers, which are to be requested during a session period.

Finally, a special routine is necessary to set up the multimedia files on the servers. This special routine generates a set of simple test files with a specific identificator and specific sizes according to the "Match" routine (see the Figure 5.19). The special routine requires the "mout.txt" (output of the "Match" routine), as an input file. This routine creates the number of output files specified in "Zipf" routine, with the sizes specified in "Sizes" routine and the identifier, which corresponds to their index in "mout.txt".

5.6.2 M² traffic generator

The traffic generator creates the requests for the multimedia service together with the workload generator. The traffic generator has both a multiprocess and multithreaded implementation and consists of two main parts: MasterManager (MM) and UserManager (UM). The MM reads, as an input, the files produced by the workload generator, forks to create children that manage the UMs, and sets up pipes, which are used to synchronize the UMs. The UM reads the pipes (created by MM), runs the UEs and maintains constant the number of UEs. Further, the UE contains an exchangeable code that implements the application protocol of the multimedia service under study. To activate the traffic generator, the following command is run:

MM {#UserManager} {#UserEquivalent/UserManager} {runtime}

where, the first parameter is the number of generated UM, the second is the number of generated UEs (for each UM) and finally the third is the default time period of the experiment (i.e. how many seconds the traffic generator works). When the runtime expires, the traffic generator will stop the experiment. The MM reads the input files produced by the workload generator:

- "name.txt" because it contains the sequence of the multimedia files that will be requested, and
- "cnt.txt" because it contains the session length.

Differently, SURGE (see the Figure 5.20) reads also the "off.txt" file that contains the Inactive OFF Times. The MM forks a number of times, as specified in the "#UserManager", to create children that run and manage the UMs. These children run the UMs by means of a system-call "system". Each UM (being a single process) works in parallel with the MM and the other UMs. MM sets up pipes, which are used to synchronize the UMs. Through these FIFO pipes the MM communicates the data read from the input files to UM. In our implementation, there are two pipes:

- "seqfifo" transfers the values read from "name.txt",
- "cntfifo" transfers the values read from "cnt.txt",

Differently, SURGE (see the Figure 5.21) also uses two more pipes: the "offfifo" that transfers the values read from "off.txt" and the "outfifo" that transfers the values of the indexes that will be used to collect the logging data. Finally, MM waits for the end of the experiment by listening to the end of all UMs. After this event, it destroys the pipes. Another difference is that SURGE collects the logging data in a centralized way at the end of the simulation, storing them in the Surge.log file. In this way, it is not so simple to understand the behavior of each single UE; for this reason in our implementation each UE writes its own log file.

To activate the UM, the MM runs the command below (by means of a "system" system-call):

UM {#UserManager} {#UserEquivalent/UserManager} {runtime} {ID UserManager} where, the first parameters are the same of MM and the last one is the identifier that MM assigns to each UM. The UMs need other parameters that are stored on the UM.conf file. These parameters are the couple IP address / port number of the multimedia servers and the mode of UEs (i.e. the UEs can run over a Laptop or over a PDA). The UM creates the requests for the multimedia server by opening a connection with MM and by running UEs. UM connects the pipes (created by MM).

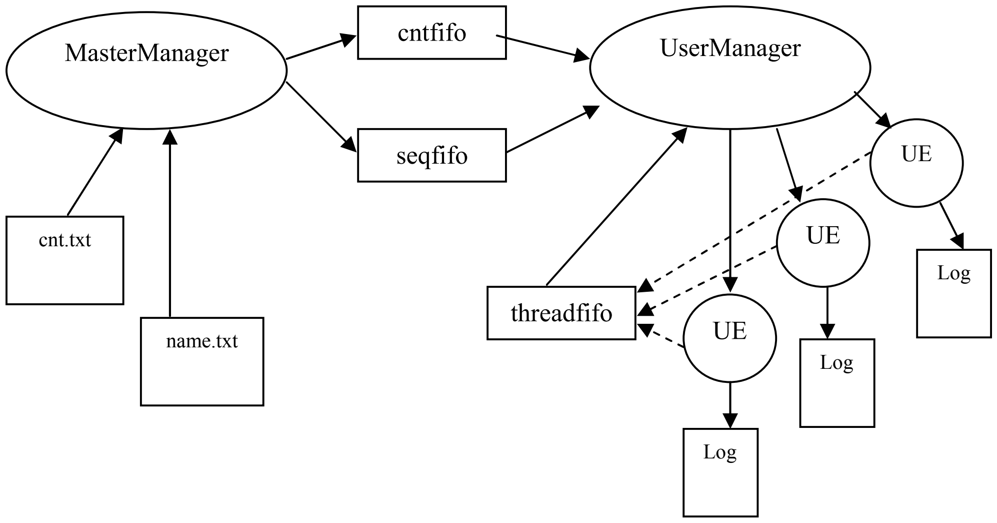


Figure 5.20: The architecture of M²G traffic generator: MasterManager, UserManager and UserEquivalent

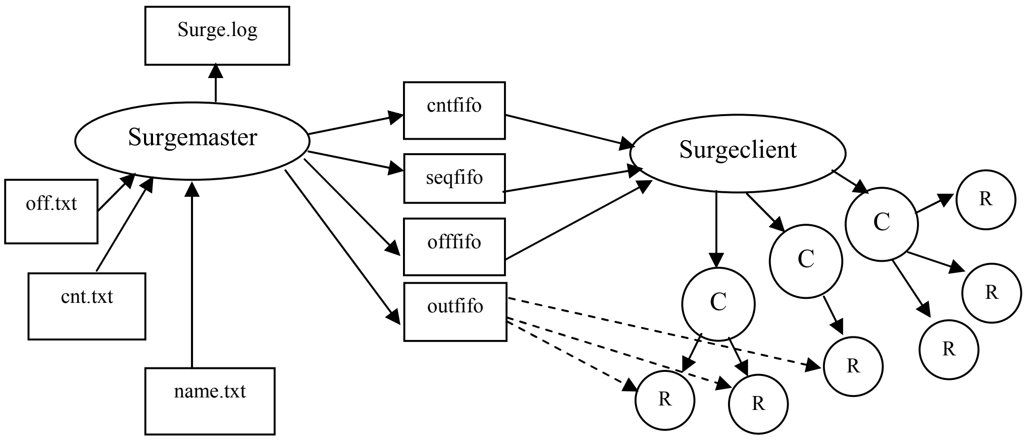


Figure 5.21: the architecture of SURGE traffic generator: Surgemaster, Surgeclient and Clients

From the “cntfifo” pipe it extracts the number “n” of multimedia files. After, from the “seqfifo” pipe it extracts a number “n” of identifiers of those files. After the connection, UM runs a “#UserEquivalent/UserManager” number of UEs by means of the system call “pthread_create()”. Each UE is implemented as a single thread that shares memory with the others and with the UM. Each UE receives the list of identifiers of the multimedia files (i.e. the previously extracted data) and requests these to the multimedia server. When an UE obtains the last one, it writes its performance on its “ID.log” file and it dies sending a message on the “threadfifo” pipe (see the Figure 5.21). In the “ID.log” file, UE writes on each row:

- The identificator of UM (it own),
- The identificator of UE (it own),
- The identificator of the multimedia resource
- The size of the downloaded file (in byte), and finally
- The download time (in seconds and microseconds).

Further UM is in charge of maintaining the number of Ues constant. When UM receives the “ID UserManager” on the “threadfifo” pipe, it knows that the UE, named ID, is dieing and it is ready to run another one. The UE also implements the application protocol of the multimedia service. The UM kills all UEs, when the runtime expires.

Chapter 6

Interceptor: Case Studies

This section demonstrates that Interceptor can be useful to evaluate different (multimedia) network applications and that can support several different scenarios. Furthermore, it shows different experimental assessments developed to validate the effectiveness of our proposed synthetic emulative platform. Along with these considerations, Interceptor becomes useful to evaluate the performance of different case studies in which mechanisms, applications and network architectures are involved:

- the Client Centered Load Distribution (C^2LD) mechanism to accelerate the downloading of remote files based on the use of replicated Web Servers,
- the wireless Internet architecture to deliver multimedia contents to mobile devices, and
- the system architecture to support the "*participatory simulation*" of complex systems from mobile device.

The first case describes how the C^2LD downloading accelerator works. The effectiveness of this mechanism is shown by means of either experimental campaign or synthetic emulative campaign. In particular, these campaigns allow the comparison between the C^2LD performance (Ghini et al., 2001) and the downloading of standard HTTP protocol mechanisms (Fielding et al., 1999). The second case reports on our experience in developing and evaluating a wireless Internet architecture designed to deliver advanced musical services to mobile consumers over wireless links. This case shows the design/development of this architecture and its performance on a synthetic emulative scenario. Finally, the last case illustrates how it is possible to support the execution of multi-agent participatory simulation activities inside Interceptor. The users of these collaborative systems demand tools that are able to visually present the results of cooperative simulation activities on the screen of mobile devices.

6.1 The Evaluation of the C^2LD mechanisms

This section describes how the C^2LD works and the effectiveness of this mechanism by means of the two separate evaluation exercises: an experimental and a synthetic emulative scenario. The former consists of evaluating the mechanism inside the actual Internet. In this evaluation, a single browser program, incorporating the mechanism and accessing a geographically replicated Web service was involved. Unfortunately, the experimental evaluation was based on a single client. In this way, these results were not enough to demonstrate the effectiveness of the C^2LD mechanism because the typical scenario (in which it should work) shows a replicated Web service accessed, concurrently, by a large number (hundreds) of clients. Thus, as it was unpractical to evaluate the mechanism by reproducing this scenario in the real Internet, the latter evaluation exercise is conducted by setting a synthetic complex (IP-based) networked scenario for Interceptor. Hence, a synthetic emulative scenario has been configured within which the implementation of the C^2LD mechanism as part of the client software has been evaluated. In this scenario, four replica servers supply the Web service for hundreds of clients.

6.1.1 The C^2LD mechanism

In the Internet context, a successful deployment of the geographically-distributed replica servers approach will depend on the ability to achieve the following two main goals:

- Dynamically binding the client to the most convenient replica server, and

- Maintaining data consistency among the replica servers.

Firstly, the dynamic binding of clients to replica servers can turn out to be difficult to implement, owing to the location-based naming scheme used in the Web. This scheme provides a one-to-one mapping (i.e., the Uniform Resource Locator - URL) between a name of a resource and a single physical copy of that resource; hence, dynamic binding of a client to distinct replica servers requires that the client-side software be adequately extended in order to be able to select an available replica, at run-time. Secondly, maintaining replica consistency on a large geographical scale can be hard to achieve, without affecting the overall service performance. In addition, the Internet environment is subject to (real or virtual) partitions that can prevent communications between functioning nodes; hence, within this environment, both clients and replica servers may hold mutually inconsistent views about which replica server is available and which is unavailable. In view of these observations, a mechanism was developed, in order to provide the clients of a replicated Web service with responsiveness, exploiting the parallelism inherent to the replicated servers architecture that implements that service. Specifically, the main goal of this mechanism is to minimize what we name the User Response Time (URT), i.e. the time elapsed between the generation of a browser request for the retrieval of a Web page, and the rendering of that page at the browser site (issues of replica consistency fall outside the scope of this mechanism). To this end, rather than binding a client to the most convenient replica server, as proposed in (Ingham *et al.*, 2000), the C^{2}LD mechanism intercepts each client browser request for a Web page, and fragments that request into a number of sub-requests for separate parts of that document. Each sub-request is issued to a different available replica server, concurrently. The replies received from the replica servers are reassembled at the client end to reconstruct the requested page, and then delivered to the client browser.

The mechanism is designed to adapt dynamically to state changes in both the network (e.g. route congestion, link failures), and the replica servers (e.g. replica overload, unavailability). To this end, the mechanism periodically monitors the available replica servers and selects, at run-time, those replicas to which the sub-requests can be sent, i.e. those replicas that can provide the requested page fragments within a time interval that allows the mechanism to minimize the URT. As the mechanism effectively implements the distribution of client requests among the available replicas, it was called Client-Centered Load Distribution (C^2LD). C^2LD can be implemented as an extension of the client-side software (i.e., in the browser) or as a module of a Proxy server to which the clients send their requests. The design of C^2LD is based on an analytical model (for more details see (Ghini *et al.*, 2002)) that is essential in order to determine the size of the page fragment that can be requested to each replica, and the extent of the monitoring period C^2LD to be used in order to execute a Web page request. We have assessed the effectiveness of the C^2LD mechanism by validating it through both an experimental evaluation over the Internet, and simulation.

6.1.2 How the C^2LD works

In general, in order to access a Web service, a user invokes a browser by providing it with the URL of that service. However, as explained earlier, the C^2LD requires that the user of Web service sets his/her own User Specified Deadline value (USD i.e. a value that indicates the extent of time a user is willing to wait for a requested Web page to be rendered at his/her workstation before invoking the browser that he/she will use to access that service. Thus, the mechanism provides its users with a configuration procedure that allows them to set their own USD timeout values before invoking their browser programs (a default USD value is used if a user does not make use of that configuration procedure). Once the USD has been set, the C^2LD mechanism operates as described below.

Access to a Web service from a browser entails that a HTTP GET method will be invoked by that browser. The C^2LD mechanism intercepts that HTTP GET invocation, starts the USD timeout and, using the URL in the GET invocation, interrogates the DNS. The DNS maintains the IP addresses of the N_{REP} replica servers that implement a Web service. When C^2LD submits a request to the DNS to resolve a URL, the DNS returns the IP addresses of all the replica servers associated to that URL. As the replica servers' addresses are available to the C^2LD mechanism, this mechanism interacts with each replica as illustrated by the skeleton code in Figure 5.1, and summarized below.

C^2LD invokes a HTTP HEAD method on each replica i. The reply from replica i to the first HTTP HEAD invocation is used by C^2LD to: i) get the size of the requested page, ii) estimate the current data rate that replica i can provide, and iii) assess the size of the first fragment that can be fetched from that replica. The C^2LD maintains a global variable (download_done, in Fig. 6.1) that indicates whether or not all the fragments of a requested page have been delivered. Until a page is not fully downloaded, C^2LD uses the Eq. (1) and (2) in the Figure 6.1 to compute the size of the fragment that is to be requested to the replica i. Where we denote with $DR_{i,r}$ the data rate that a given replica server *i* is able to provide during a given time interval $k_{i,r}$, with $PS_{i,r}$ the size of the document fragment requested to a certain server *i* with the request *r*, and with $S_{i,r}{}^{*}$ the *URT* expected from the execution of the request *r* directed to the replica *i*. In order to adjust adaptively to possible fluctuations of the communication delays that may occur over the Internet, the fragment size is computed each time a fragment is to be requested, based on the value of the *URT* experienced in fetching the previous fragment. Thus, in essence, as the GET request *r-1* directed to a given replica *i* terminates, a new GET request *r* can be issued to the replica *i* with the fragment size value $PS_{i,r}$ computed on the basis of the $URT_{i,r-1}$ response time. Once the requested fragment size has been calculated, C^2LD issues a HTTP GET request to the replica *i*, in order to retrieve the required fragment of that size. (Specifically, a fragment of Z bytes size is requested by invoking a HTTP GET method with the following option set: "*Range: bytes=Y-X*", where X and Y denote the bytes corresponding to the beginning and the end of the requested fragment of size Z, respectively). Note that some of the replica servers may not respond timely to the HTTP (HEAD and GET) invocations described above (e.g., they may be unavailable owing to network congestion). Thus, C^2LD associates a timeout to each HTTP request it issues to each server *i*. If that timeout expires before C^2LD receives a reply from a replica server *i*, it assumes that the server *i* is currently unavailable, and places it in a *stand_by* list. Replica servers in that list are periodically probed to assess whether they have become active again. Requests for replicas in the *stand_by* list are redirected to active replicas.

```
/* C^2LD */
…
within USD do                    /* set USD timeout */
...
HEAD(…)                          /* send HEAD request to replica i */
URT(i) := …                      /* assess URT replica i can provide */
PS(i,1) := …                     /* compute 1st fragment size for replica i */
within S do                      /* set timeout of length S */
    if not download_done then    /* check if page download completed */
        GET (…)                  /* get fragment from replica i */
        $DR_{i,r} = \frac{PS_{i,r-1}}{URT_{i,r-1}}$ ;/* compute expected data rate, based on URT of previous request */ Eq. (1)
        $PS_{i,r} = DR_{i,r} \cdot S_{i,r}$ ; /* compute size of next fragment to be requested */ Eq. (2)
    else
        return;
od
    …                            /* handle S timeout exception */
od
```

Figure 6.1: Implementation of the C^{2}LD Service

6.1.3 Experimental scenario

A large number of experiments (4000, approximately) were carried out during a two-month period. These experiments consisted essentially of a client program (i.e. a browser) downloading Web documents of different size from up to 4 geographically distributed replica servers. These documents were downloaded by the same client program using both the C^{2}LD mechanism, and the standard HTTP GET downloading mechanism, for comparison purposes. In this sub-section, we describe in detail the scenario within which these experiments have been carried out, introduce the metrics we have used to assess the implementation, and discuss the performance results we have obtained. The performance of the C^{2}LD implementation has been evaluated using the Internet to connect a client workstation (equipped with the C^{2}LD software) with four replica servers (see Figure 6.2). The client workstation was a SPARCstation 5 running the SunOS 5.5.1 operating system, located in Bologna (IT). The four different replica servers were running the Apache Web server over the Linux platform. For the purposes of our evaluation, these servers were located in four distinct geographical areas. Specifically, a replica server was located close to the client workstation in Cesena (IT). This Laboratory is four network hops far from the client workstation in Bologna. The connection between the client workstation and this

Laboratory has a limited bandwidth of 2 Mbps, and is characterized by a rather high packet loss rate (between 2% and 9%). A second replica server was located in Trieste (IT). This server was reachable through 9 network hops via a connection whose bandwidth ranged between 8 and 155 Mbps. A third replica server was located in Newcastle (UK). This server was reachable through 15 network hops from the client workstation. Finally, a fourth replica server was located in S. Diego (US). This server was reachable through a 19 network hops transatlantic connection. Within this scenario, a replicated Web service can be configured so as to use one of the following 11 combinations of the four available replica servers. Namely, a service can be replicated across the two servers in Cesena and Newcastle (C+N, in the following), only, or those in Cesena and Trieste (C+T), or in Trieste and Newcastle (T+N), or in Cesena and S. Diego (C+S), or in Trieste and S. Diego (T+S), or Newcastle and S. Diego (N+S). A more redundant service can be implemented across one of the following four combinations of three servers, instead: Cesena, Newcastle and Trieste (C+N+T); Cesena, Newcastle, and S. Diego (C+N+S); Cesena, Trieste, and S. Diego (C+T+S); Trieste, Newcastle, and S. Diego (T+N+S).

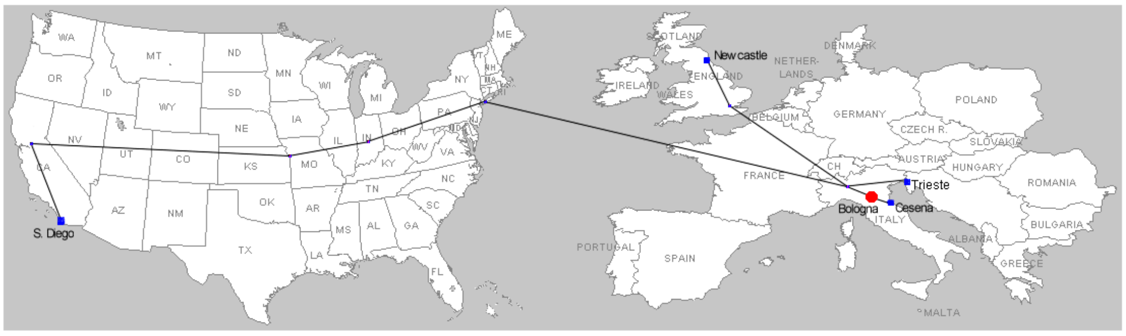


Figure 6.2: Routes between the client and the Web replica servers

Table 6.1: The measured round trip time and lost packet percentage

Ping from Bologna to:	Cesena	Trieste	Newcastle	S. Diego
RTT 90% of arrived pkt (ms)	107	95	160	450
RTT min (ms)	10	38	59	190
Lost Packet	2%-9%	0%	0%	0%-3%

Finally, a redundant Web service can be implemented across the four replica servers in Cesena, Newcastle, Trieste, and S. Diego (C+N+T+S). The experiments have been carried out using all the 11 combinations of replica servers. It is worth noting that the four machines running the server replicas were moderately loaded during the experiments. In contrast, the routes to the European Web replica servers were heavily loaded during daytime; network traffic over these routes typically decreased during the night. The average network traffic conditions on both the European and transatlantic routes, experienced during the evaluation of the C^2LD mechanism, are reported in the following Table 6.1. Specifically, in this Table, the first row indicates the 90% of the Round Trip Time (RTT) obtained with the ping routine from the client workstation in Bologna to the four replica servers; the second row reports the minimum RTT experienced over the routes to those servers; the third row reports the packet loss rate we measured over those routes, as obtained with the ping routine (ICMP).

The following two metrics have been used to evaluate the effectiveness of the C^2LD mechanism: i) the percentage of document retrieval requests successfully satisfied by the mechanism, and ii) the URT (as defined earlier) the mechanism provides. Both these metrics capture the principal user requirements; namely, that a requested document is effectively retrieved, and that it is done timely. Based on these metrics, the evaluation of the mechanism has been carried out using different values of the following four parameters: i) the number of server replicas that implement a given Web service, ii) the data rate that each different server replica can provide, iii) the download monitoring period adopted by the C^2LD mechanism, and, finally, iv) the size of the document to be retrieved. The performance of the C^2LD mechanism has been compared and contrasted with that provided by the standard HTTP document retrieval mechanism, as this is implemented by the HTTP GET function. To this end, each replica server maintained a set of downloadable files of different size, ranging from 3 Kbytes to 1 Mbytes. These servers were requested to retrieve the same Web document, at the same time of the day (i.e. under the same network traffic conditions, approximately) using, alternatively, both the standard HTTP GET request, and the C^2LD. The download monitoring period S used by the mechanism ranged from 50 milliseconds to 10 seconds. A number of experiments were carried out for each given file size and download monitoring period S. Each experiment consisted of 15 consecutives download requests. 4 out of 15 of these requests were executed by invoking the HTTP GET function; the other 11 requests were executed by the C^2LD mechanism. The experiments used files whose size was 3, 10, 30, 50, 100, 200, 500 and 1000 Kbytes. For each file size, the experiments with the C^2LD mechanism were repeated using a different download monitoring period, ranging from 50 milliseconds to 10 seconds. As mentioned earlier, the experiments were carried out on the 11 possible configurations of a replicated service; however, the values reported in the Tables and Figures in this sub-section are the average of the results obtained in all the experiments. As a first result, Table 6.2 summarizes the page loss percentage obtained with the C^2LD, and that obtained with the standard HTTP GET downloading mechanism (performed with S. Diego, Newcastle, Trieste and Cesena, respectively). It can be seen from this Table that the C^2LD mechanism provides a highly available service, since it always guarantees the downloading of the requested document. Instead, the standard HTTP mechanism is not always able to provide its client with that document, as shown by the page loss percentage experienced by the S. Diego and Cesena servers, in particular. The two higher curves, instead, represent the URT values provided by the two fastest Web replica servers, when interrogated with the standard HTTP downloading mechanism. Specifically, the URT values provided by these two replica servers were obtained as follows. The four different replica servers were exercised with the standard HTTP mechanism; then, out of these four servers, the URT results obtained by the two fastest ones were selected to plot the graph in Fig. 6.3.

Table 6.2: Fault Percentage

	USD (seconds)	C^2LD	S. Diego (HTTP)	Newcastle (HTTP)	Trieste (HTTP)	Cesena (HTTP)
50 Kbytes	120	0 %	0.3 %	0 %	0 %	1.65 %
100 Kbytes	120	0 %	0.25 %	0 %	0 %	5 %
200 Kbytes	180	0 %	0.2 %	0 %	0 %	4 %
500 Kbytes	300	0 %	0.3 %	0 %	0 %	1 %
1 Mbyte	600	0 %	0.45 %	0 %	0 %	2.5 %

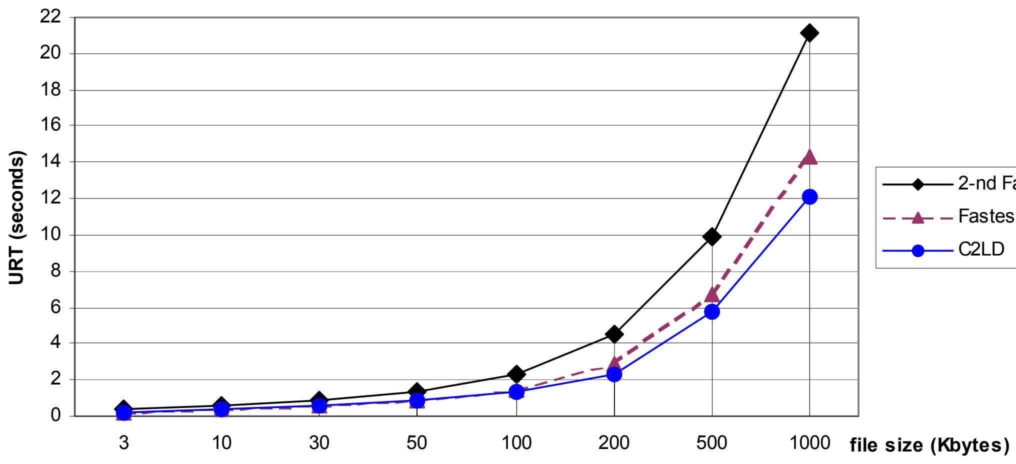


Figure 6.3: C^{2}LD vs. HTTP

As shown in that Figure, the performance of the C^{2}LD mechanism and the HTTP mechanism are equivalent when the document size less than 50 Kbytes. Instead, when the document size is larger than 50 Kbytes, the C^{2}LD mechanism outperforms the standard HTTP downloading mechanism. As already mentioned, the C^{2}LD URT values in Fig. 6.3 represent an average URT value, as it results from all the experiments we have carried out. For the sake of simplicity in this figure we do not report the value of the variance of the URT, as measured in the experiments. However, this value for the C^{2}LD mechanism varied from 0.099 s to 5 s, approximately, depending on the size of the requested page. In addition, it may be interesting to assess the URT improvement that can be obtained by the C^{2}LD as the number of replica servers, concurrently used, varies. To this end, we have carried out experiments in which the number of active replicas was varying from 1 to 4. Fig. 6.4 illustrates the results of the experiments, in these four cases. Specifically, in these cases, the URT is measured as a function of the file size. As shown in this Figure, the larger the number of replicas, the better the URT values; in addition, as the file size increases, the advantage of using the C^{2}LD mechanism becomes more notable. Table 6.3 shows the average percentage improvement of the URT values that has been experienced in the experiments depicted in Fig. 6.4, as the number of replica servers grows.

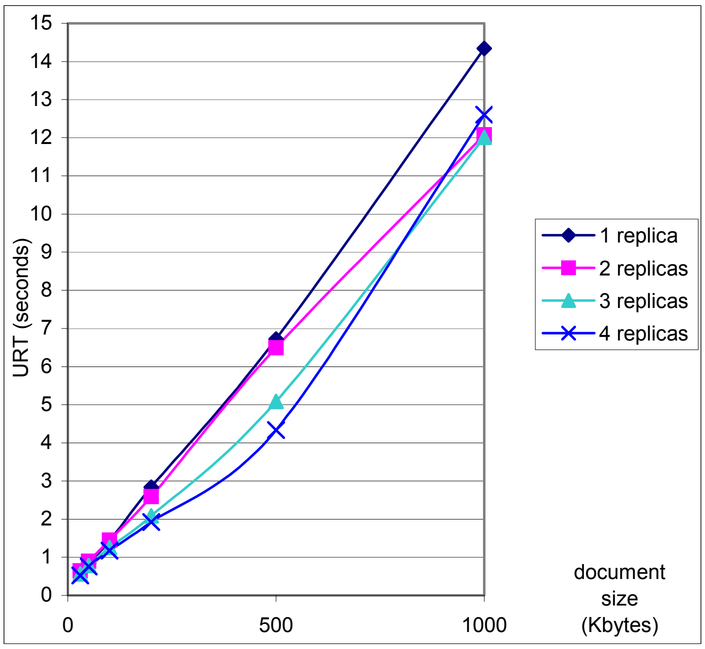


Figure 6.4: URT improvement provided by C^2LD as the number of replicas grows

Table 6.3: Average URT percentage improvement

Number of Replica Servers	URT percentage Improvement
2	4%
3	17.2%
4	21.5%

6.1.4 Synthetic emulative scenario

The results discussed in the previous paragraphs indicate that the C^2LD mechanism (when incorporated in a browser's software) can provide a notable speed-up in the communications between that browser and a replicated Web service, compared to the HTTP standard document fetching policy. Unfortunately, the experimental trials were conducted with a single client. These results were not enough to illustrate the effectiveness of this mechanism in a real computer network where several C^2LD clients request files, concurrently, to the same replica servers. As it was impractical to configure and manage the scenario described before by means of experimental trails, it is necessary to tackle two main problems to find a solution. The former is to confine in a lab-based environment such real clients and replica servers, while simulating a geographically distributed setting with appropriate network parameters. The latter is to improve laboratory scalability, by supporting an increasing number of UEs that run the C^2LD clients.

Interceptor tries to tackle these problems by involving real clients and servers inside a laboratory and by "synthetic emulating" a complex network scenario. Hence, the core components of Interceptor (i.e. EI, EM and NTM) synthetic emulate the geographically network distributed scenario, while the WTG manages the request of the C^2LD clients by scaling up the UEs. For this reason, we develop for Interceptor wired models that simulate the main communications delay among different geographical area (Europe, USA, Asia and Australia). Therefore, to validate the effectiveness of Interceptor, different experimental assessments developed to highlight its capability to reproduce a real network scenario are shown. Finally, to complete the evaluation of the C^2LD, its behavior is compared with that of the standard HTTP downloading mechanism inside the validated scenario (by contrasting, as metric, the downloading datarate).

The synthetic emulation trials integrated the synthetic environment with an external real lab-based scenario where, respectively, four different Web servers and sixty five client workstations were accommodated. The synthetic environment was designed to reproduce complex IP-based networks connecting four locations where a real Web service constituted by four Web replica servers of the Linux Debian official site, respectively located in Europe (in Newcastle), USA (in San Diego), Asia (in Tokio) and Australia (in Sydney), and a set of sixty five client workstations distributed in the same areas as the servers (as shown in Figure 6.5) were accommodated. In addition, on each single client workstation a number of up to 10 different browsers were simultaneously running, according to the UE methodology introduced in (Barford *et al.*, 1998). In our emulative scenario (see the Figure 6.6) the host, that accommodates Interceptor, is Pentium III 900 with 1 GB RAM connected by means of Ethernet cards (100 Mb/s) running Linux with Kernel family 2.4.0. The four replica servers are Pentium II 233 with 256 MB RAM connected by means of Ethernet card (100 Mb/s) running Windows 2000 server. The clients are Pentium III 900 with 256 MB RAM connected by means of

Ethernet card (100 Mb/s) running Windows 2000 professional. Interceptor has been configured in order to simulate both the delay of the IP packets traveling between two end systems in the same geographical area (e.g. Europe), and the delay of the IP packet traveling between two end systems located in two separate geographical areas (e.g., Europe and Asia). The Table 6.4 reports on the approximately average value of the RTT measured with a ping using the synthetic emulative scenario. For the sake of simplicity, a symmetric approach to implement (inside Interceptor) the communication channels is used and the intra area delay has the same for each area. For this reason, some squares of the Table 6.4 are not reported and others have the same value.

To validate the effectiveness of this scenario, the first trails are simulation assessments conducted by considering a single browser (located in Italy) that downloads files of different sizes from two separate Web servers located, respectively, in England and in the USA. Along with the simulation trials, two real world experiments directly carried out over the Internet are also conducted (i.e. necessary for the comparison). The final downloading datarate was adopted as fundamental metrics for all the experiments. The following Figures 6.7, 6.8, 6.9 and 6.10 account for the simulative/experimental trials respectively conducted with the England-based and the USA-based servers. In particular, each figure contrasts the results related to the simulations with those obtained through the real Internet. For each figure, the x-axis represents the size of the downloaded files, while the y-axis represents the datarate (expressed in KB/sec) provided by the server while downloading those files. As we can see from all figures, the curves of the downloading datarate values, obtained in both the lab-scaled simulated setting and in the real Internet, present quite similar slopes. In particular, Figures 6.1 and 6.2 represent the average value of the obtained results, while Figures 6.3 and 6.4 account for the statistical variation of those values. For the sake of completeness, table 6.5 reports, in the fourth and fifth columns, the values of the half-width *h* of, respectively, the 99% and 95% two-side confidence intervals, calculated for the trials.

The second trails has been used to compare the performance of two different methodologies for Web downloading, namely, the standard HTTP protocol and the C^2LD mechanism, by contrasting their respective final downloading datarate inside the already validated scenario.

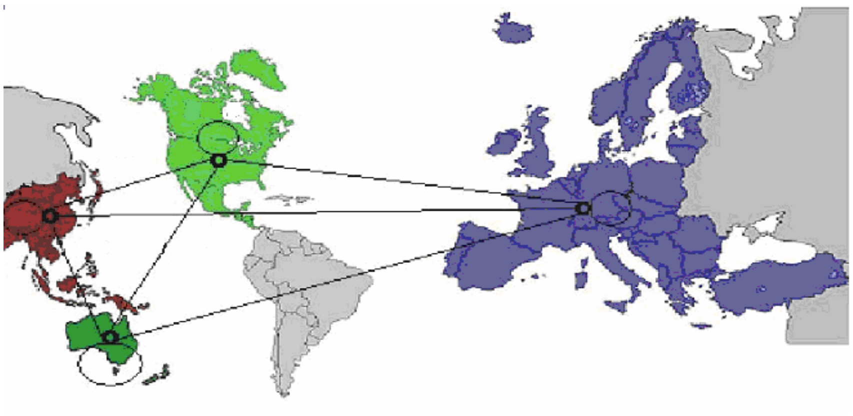


Figure 6.5 Geographical scenario

Table 6.4: Average value of the RTT measured with a ping

RTT ms	Europe	USA	Asia	Australia
Europe	60	190	310	350
USA	-	60	250	240
Asia	-	-	60	220
Australia	-	-	-	60

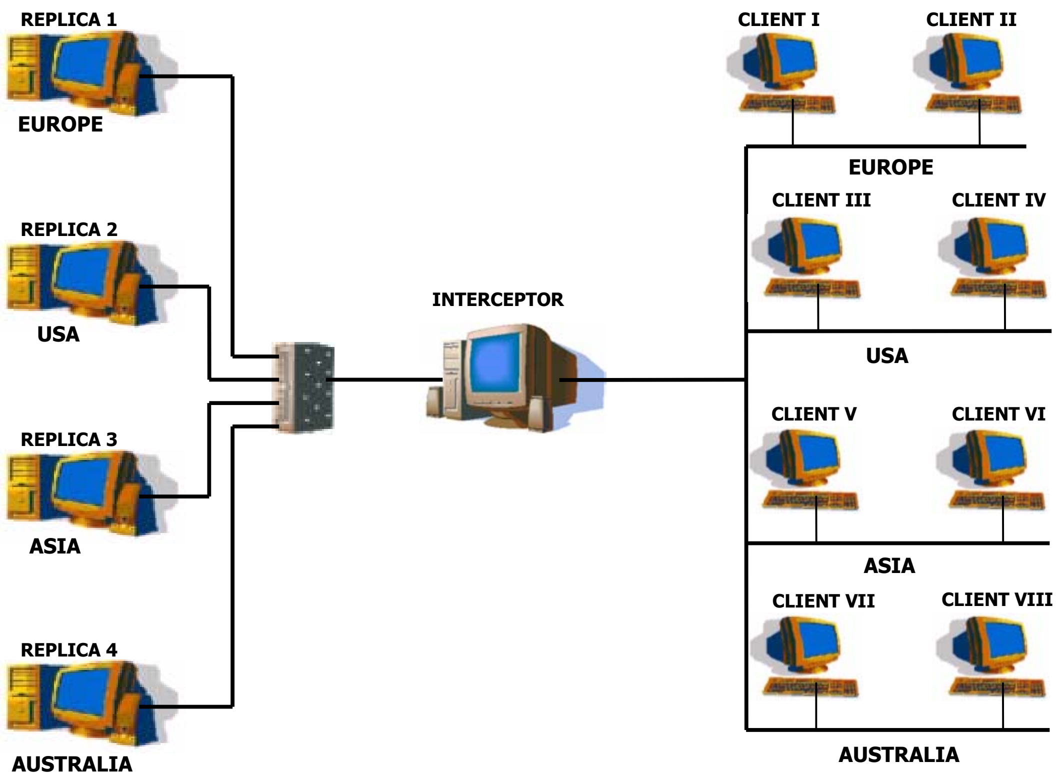


Figure 6.6 emulative scenario

Table 6.5 Confidence intervals for the simulative/experimental trials

Sim/Exp	Size	Mean	*h* (99% confidence level)	*h* (95% confidence level)
Simulation UK	3k	7.318	5.842	4.063
Simulation UK	25k	28.228	16.419	11.418
Simulation UK	250k	0.880	0.646	0.449
Simulation US	3k	5.424	5.554	3.819
Simulation US	25k	15.462	8.945	6.220
Simulation US	250k	0.718	0.474	0.330
Experimental UK	3k	6.172	3.208	2.231
Experimental UK	25k	28.91	16.286	11.325
Experimental UK	250k	0.737	0.385	0.268
Experimental US	3k	3.526	1.443	1.004
Experimental US	25k	18.146	3.835	2.667
Experimental US	250k	0.543	0.138	0.096

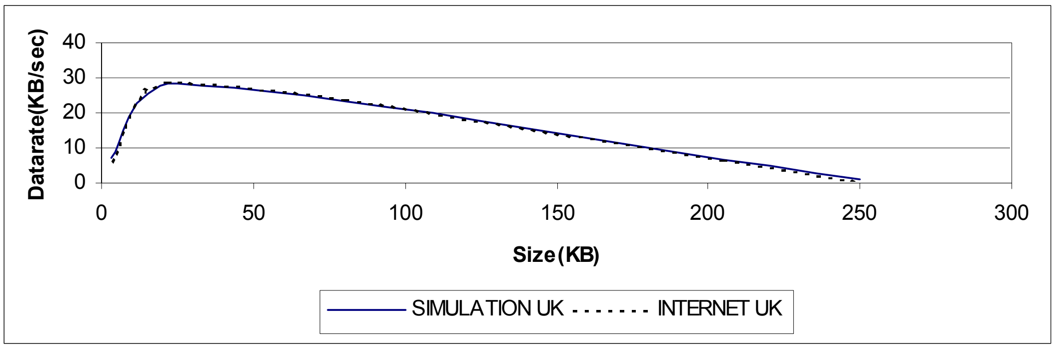


Figure 6.7: Average value of downloading data rate: simulative vs. experimental (client in Italy - server in the UK)

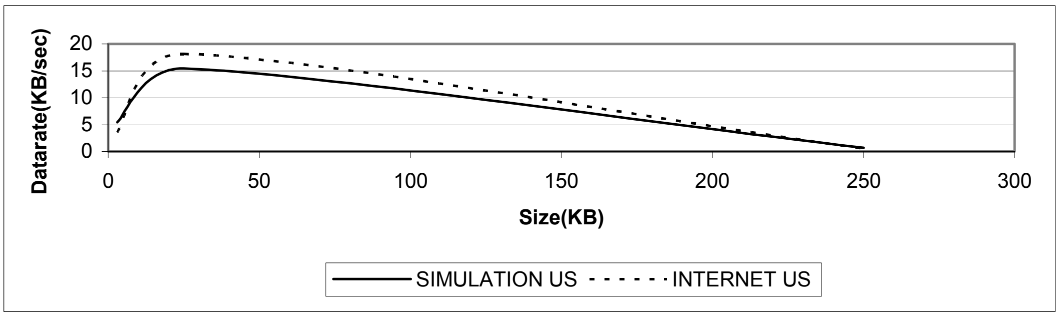


Figure 6.8: Average value of downloading data rate: simulative vs. experimental (client in Italy - server in the USA)

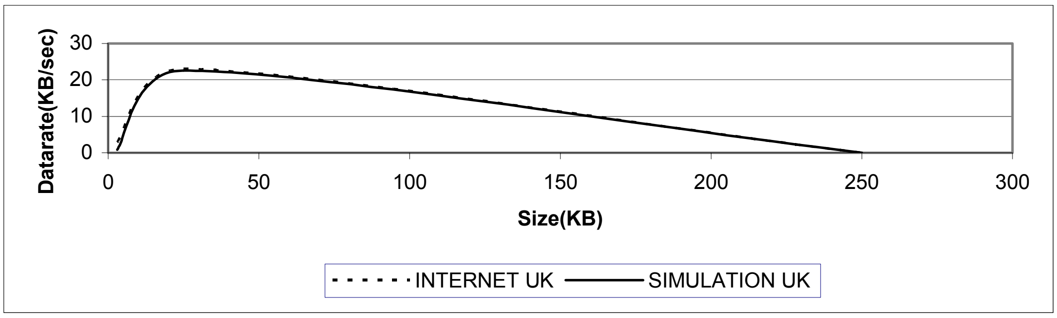


Figure 6.9: Statistical variation of the downloading data rate: simulative vs. experimental (client in Italy - server in the UK)

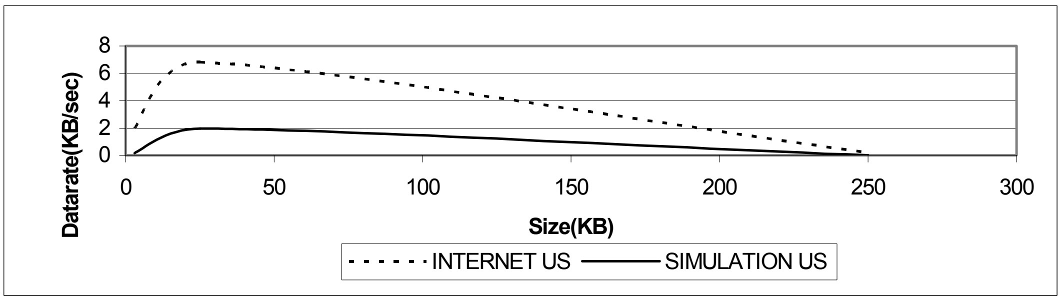


Figure 6.10: Statistical variation of the downloading data rate: simulative vs. experimental (client in Italy - server in the USA)

A set of experiments is carried out to assess the capability of each mentioned method to support an increasing number of clients, when Web resources of different sizes are downloaded in a validated scenario. The scenario was that validated before (Figure 6.11) with a replica server for each geographical area and different number of UEs dispersed in the USA area (close to the Replica 2). In particular, each UE executes its client to request files ranged from 850 to 920 times. The size of files stored in each replica server has range from 0 to 330 KB. These are in agreement with the distribution law of Pareto with scale k = 1 and shape $\alpha = 1.5$. From these analyses (see Figure 6.11), it is evident that all the curves related to the use of the HTTP have different slopes w.r.t. the correspondent curves obtained with the C^2LD. In general, the C^2LD (in comparison with HTTP) generates a considerable delay for small size files, while it generates low delay for big size files. In fact, there is a file size threshold value (ranging from 55 KB to 110 KB over different curves) beyond which C^2LD outperforms HTTP. In particular, C^2LD outperforms HTTP approximately at 110 KB in the 200 UEs case. If we increase UEs, subjecting the servers to a greater workload, C^2LD outperforms HTTP at lower dimension (approximately 65 KB in the 400 UEs case and approximately 55 KB in the 650 UEs case). Namely, the larger the file size, the greater is the advantage of using the C^2LD mechanism. This result is in accord with the previous experimental trails of the Paragraph 6.2.3. In addition, our experiments show that this trend is improved by the increase of the numbers of clients. In particular, the three HTTP curves have a peak around 17-23 KB and while increasing file size, they decrease their slopes. Instead, the three C^2LD curves increase their slopes. This analysis allows us to understand two fundamental aspects:

- HTTP performs better when downloading small files, instead C^2LD downloads big files, and
- C^2LD has a better performance increasing the UEs.

Then, we conclude these analyses claiming that the HTTP is suitable for the traffic due to users surfing on the web sites, while the C^2LD for the traffic due to downloading of bulk data transfer (to es: avi, images, mp3, mpg, operating system iso, patch, rpm, tool and update). Furthermore, the Figure 6.11 shows that increasing the downloading file size and increasing the workload (in order of UEs) the C^2LD outperforms the HTTP.

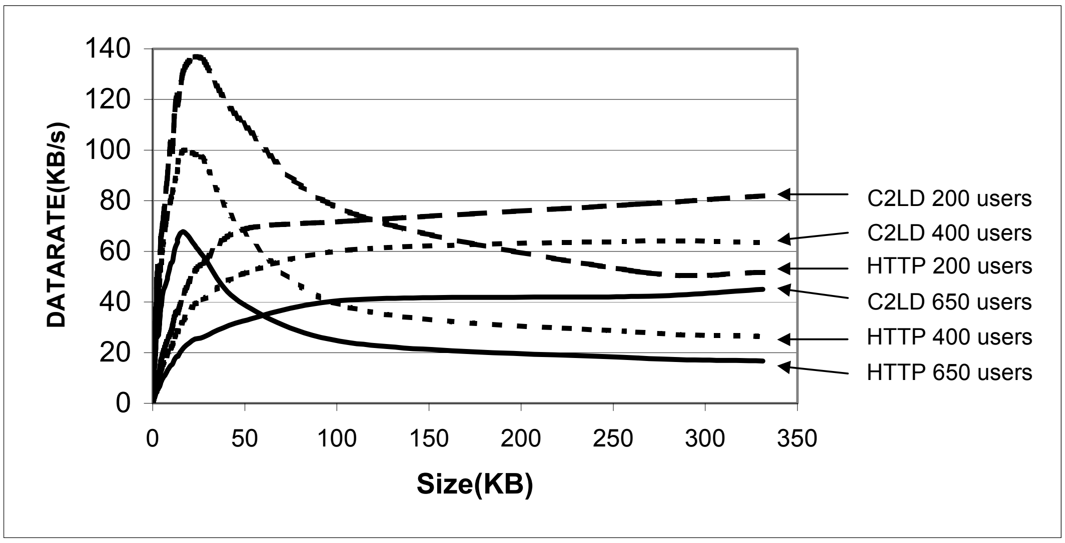


Figure 6.11: Comparison of downloading data rates between C^2LD and HTTP

These trails highlight that the C^2LD is a good mechanism for downloading multimedia contents. Therefore implementing it inside an architecture that delivers multimedia contents is a good idea (see the Paragraph 6.2). Finally, these considerations prove the effectiveness of the mechanism in a real computer network where several C^2LD clients request files, concurrently, to the same replica servers. Furthermore, these analyses illustrate the benefit of testing a mechanism like the C^2LD inside a synthetic emulative scenario built by means of Interceptor.

6.2 Delivering Multimedia Contents over the Wireless Internet: from Song Distribution to Interactive Karaoke on Wireless Devices.

Another case study shows the performance of a wireless Internet architecture that delivers multimedia contents from song distribution to interactive karaoke on mobile devices (Roccetti *et al.*, 2003a). The pace of developments in wireless technology is enabling a wide range of exciting applications including location-aware systems, wearable computers, wireless sensor networks and novel use of cellular telephony systems. Many of those applications may play the role of key drivers of future wireless technology provided that they may guarantee affordable access, ubiquitous reach and an effective service delivery model. In this challenging scenario, a growing demand is emerging for delivering modern multimedia entertainment services to wireless handheld devices, on a large scale. In this paragraph, we report on our experience in developing and evaluating a wireless Internet application designed to deliver advanced musical services to mobile consumers over wireless links. This contribution is composed by two main parts. The former shows the design, the development and an experimental campaign on the field of this application. The latter reports the development and the results of a synthetic emulative scenario especially configured to evaluate this application.

In essence, our application allows mobile users to listen to songs and to see karaoke clips on wireless devices by exploiting the Internet as a large musical storehouse. The Paragraph 6.2.1 shows the world scenario about delivering music. The Paragraph 6.2.2 discusses on some technical obstacles surmounted for integrating the wireless access technology with the wired Internet. The Paragraph 6.2.3 alongside the operational description of our wireless Internet application illustrates the design principles followed to design our system. Field trials have been conducted and confirm that an adequate structuring of the wireless Internet application may be effective for the delivery of modern musical entertainment services to mobile consumers. In particular, Paragraph 6.2.4 reports on a large set of performance results gathered from experimental trials conducted on the field with UMTS technology.

Unfortunately, the previous experimental campaign has been conducted over a single data link communication type, i.e. UMTS, by means of a single wireless device. To understand the behavior of our application in a more realistic scenario, several experiments where it is possible to choose among different wireless technologies and to scale up the number of users are necessary. Our multimedia service is a distributed application that integrates wireless and wired communications. For this reason building experiments in a real scenario is not so simple. In this context, a synthetic emulative scenario is setup to make an accurate study of this multimedia service. Therefore, wireless models have been designed and developed for simulating different data link communication (Paragraph 6.2.5). Further, UEs (Paragraph 6.2.6) able to request the multimedia service to our application are built.

6.2.1 Introduction

With the advent of the 21st century, market forces are accelerating the pace of wireless technology evolution. It is widely anticipated that future mobile users will enjoy a near-ubiquitous access to high-bandwidth wireless networks. Probably, mobile phone services represent the most prominent example of widespread access to wireless communication networks. Recently, notable efforts, such as i-Mode and WAP, have been carried out to exploit the phone-based wireless technology, along with the Internet, to provide web services to mobile users. As of today, unfortunately, those efforts have resulted in little more than email access and limited web browsing. Although this situation is slowly improving, it is easy to envisage that old pricing schemes, along with contents not suitably formatted for mobile phones, may limit the interest of mobile consumers in wireless services. However, in parallel with the recent improvements of high-bandwidth wireless communications, the wired Internet's progress has determined a significant innovation among music distribution paradigms. The maturing *Distributed File Sharing* technology, implemented by systems like Napster, has enabled the dissemination of musical contents in digital form, allowing consumers to link to stored music files from around the world. Hence, with users adopting Internet-enabled cellular phones and similar handheld devices we may expect that a growing demand may emerge for wireless services which enable mobile consumers to access music contents from the large musical storehouse represented by the Web.

In this context, the new wireless (like GPRS, UMTS, and Wi-Fi) technology promise to be the milestones in the process of building an adequate large-scale infrastructure for delivering musical services to mobile consumers. Important advantages of the wireless technology amount to the fact that packets originating from wireless devices can be directly transmitted to IP networks (and vice versa), while specific QoS guarantees may be provided over the wireless links. Additionally, they offers higher data rates (from 56Kb/s of the GPRS, going through few Mb/s of UMTS and up to a several Mb/s of the Wi-Fi) and an increased capacity. These data rates, plus compression techniques, will allow the access to HTML pages, to video/audio streaming, as well as to enhanced multimedia services, for laptops and smaller devices. In particular Wireless LAN and WPAN technologies, used to provide *local* access to the Internet, are likely to be coupled with the existing widespread telecommunication infrastructures (e.g., GPRS, UMTS), thus providing an opportunity to drastically improve the range, throughput and performance of music distribution services. Nevertheless, it is well known that, even with the adoption of the wireless technology, various technical communication problems may arise due to:

- The time-varying characteristics of the wireless links,
- Possible temporary link outages,
- Protocol interference between the radio link level protocols and the Internet transport protocols, and
- Typical high bit error rates of the radio links.

As a result, all the above mentioned limitations may keep some older applications designed for music distribution over the wired Internet from working efficiently and effectively when deployed over wireless mobile communication scenarios.

In this chapter, we describe a modern Wireless Internet application we have designed to support the widespread delivery of musical services to Internet-enabled mobile phones, and similar handheld devices, over wireless links. In particular, the prototype implementation of our wireless application allows mobile consumers equipped with wireless devices to exploit the Internet as a vast storehouse of music resources, where two different musical services are provided, namely:

1. A *mobile song-on-demand distribution service*, and
2. A *mobile karaoke distribution service*.

The *mobile song-on-demand distribution service* permits to mobile users to download and to listen to Mp3 files on wireless devices. Specifically, our wireless application provides its users with: i) a simple and rapid Internet-based mobile access to a music-on-demand *download* service, ii) a robust and widely available music-on-demand distribution system, based on the technique of replicated web servers, and finally, iii) the possibility of interactively customize the use of the system. From a user's perspective, it is worth noticing that different types of clients may exploit the developed service. In particular our wireless application may be exploited by the following categories of users.

- *Music Listeners*, equipped with a wireless device and connected to their wireless cell, may search for their favorite songs over the Internet, download them onto their wireless devices, and finally playout them at their earliest convenience.
- *Music Producers* may wish to exploit our system to distribute their own music songs to be listened to on wireless devices. (At the current state of the art of our system, this type of users needs a regular wireline Internet connection in order to upload to the system their musical resources).
- *Musical Service Providers* may exploit our system to organize, to build and to maintain structured repositories of musical resources over the Internet for use

from wireless consumers.

Karaoke is a MTV-type multimedia entertainment service which has gained a lot of popularity in Asia, US and Europe. With the advent of the wireless Internet and of Internet-enabled cellular phones, a growing demand is emerging for delivering such kind of multimedia entertainment service to wireless handheld devices, on a large scale. In this context, the *mobile karaoke distribution service* we have developed allows mobile users to download and to play out (on their wireless devices) multimedia karaoke clips, constructed out of synchronized multimedia resources, such as music, text and video. A karaoke clip is represented by means of a SMIL file containing the formal specification of the media scheduling and synchronization activities concerning all the audio, video and textual resources which compose that karaoke clip. The important experiences of P2P systems, such as, for example, Napster, Freenet and Gnutella, are at the basis of our musical delivery services, but our Internet application is essentially new, in the sense that it allows a reliable and distributed music sharing service over wireless links.

6.2.2 System issues

The aim of this Section is to discuss some technical issues which are at the basis of the system we have developed to support the distribution of mobile musical services to wireless devices. Some of the most prominent technical issues for the development of our system are those related to the problems of integrating the wireless access technology with the Internet. It is well known that choosing the wireless technology as a means to provide mobile access to the Internet poses a number of obstacles. In this context, the first problem is to decide if advanced TCP/IP based applications will be able to behave well over radio communications protocols. With regard to this fact, it is important to notice that the Internet TCP/IP protocol stack has not been especially designed for wireless communications. The standard Transmission Control Protocol (TCP) provides a sliding window based ARQ (Automatic Repeat reQuest) mechanism that incorporates an adaptive timeout strategy for guaranteeing end-to-end reliable data transmissions between communicating peer nodes over wired connections. Since the ARQ mechanism of TCP essentially uses a *stop and resend* control mechanism for ensuring connection reliability, under question, here, is whether this TCP retransmission mechanism may trigger a TCP retransmission at the same time when the radio link level control mechanism is already retransmitting the same data. Secondly, an even more significant problem of mobile wireless is that of temporary link outages. If a user, in fact, enter an area of no signal coverage, there is no way that the standard TCP protocol may be informed of this link-level outage. After having considered all the above challenges, a third and final problem is strictly related to the internal architecture of those advanced Internet-based applications that should be accessed through radio interfaces. Those applications, in fact, must exhibit a high rate of robustness and availability, since the mobile access to those applications should not be influenced by possible problems occurring at the Internet side. To overcome the aforementioned obstacles we adopted a number of important strategies, which are illustrated in the following.

First, in order to ensure both the availability and the responsiveness of our mobile musical service, we have structured our application according to the special technology of *replicated web servers*. Following this technology, a software redundancy has been introduced at the Internet side by replicating the multimedia resources across a certain number of web servers distributed over the Internet. A typical approach to guarantee service responsiveness consists of dynamically binding the service client to the available server replica with the least congested connection. An approach recently proposed to implement such one adaptive downloading strategy at the Internet side amounts to the use of a software mechanism, called the Client-Centered Load Distribution (C^2LD) mechanism (see the Paragraph 6.1). With this particular mechanism, each client's request of a given multimedia resource is fragmented into a number of sub-requests for separate parts of the resource. Each of these sub-requests is issued concurrently to a different available replica server, which possesses that resource. The mechanism periodically monitors the downloading performance of available replica servers and dynamically selects, at run-time, those replicas to which the client sub-requests can be sent, based on both the network congestion status and the replica servers workload.

Second, our wireless application has been structured following an ALL-IP approach, where a wireless session level has been additionally developed (on the top of the standard

TCP protocol) to guarantee connection stability in the possible cases of temporary link outages.

Third, as the download of musical resources over wireless links may experience long duration (and high costs) our wireless application has been designed to exploit in the cheapest service class as for example Wi-Fi technology, in general, and UMTS *background* class, in particular. We remember that is this kind of service class with lower costs, since it has been designed for supporting non-interactive, best-effort traffic.

6.2.3 A wireless Internet application for music distribution

The comprehensive visual representation of the architecture of the wireless Internet application that we have developed is reported in Figure 6.12. The client part of our software application, running on the wireless device, provides users with the possibility of searching, downloading and playing out the required musical resources. Musical resources may be of a different type, based on the musical service selected by the user. If a consumer wishes to enjoy the song-on-demand delivery service, then musical resources are represented by simple musical files (typically encoded with the Mp3 format). Those musical files are stored over different web servers, geographically distributed over the Internet, which act as music repositories. In general, different web servers can be managed and administrated by different music service providers, and may also offer different set of replicated songs. Simply stated, this replication scenario can be thought of as a loosely coupled replication system, where, potentially, different servers support different sets of musical resources. Needless to say, each single song may be replicated within a number of different web servers.

If the selected service amounts to the mobile karaoke, instead, the corresponding musical resource is represented by a more complex set of data constituting a *karaoke clip*. A karaoke clip, in practice, is represented by a textual SMIL-based file with pointers to all the multimedia resources that compose that given clip. All the multimedia resources (specified in the SMIL description file) are stored on different replicated web servers, geographically distributed over the Internet. Like in the case for the song-on-demand delivery service, those web servers perform as redundant repositories for the multimedia resources that compose the karaoke clip of interest. As seen from Figure 6.12, between the mobile clients and the Internet-based multimedia repositories an Intermediate software System (IS) has been interposed. The responsibilities of the IS are:

- Providing each user device with a wireless access point to the Internet-based music distribution service;
- Discovering and downloading all those multimedia resources (songs or karaoke clips) that are requested by a user.

In essence, the IS is constructed out of three main subsystems, namely:

- An *Application Gateway* subsystem,
- A *Discovery* subsystem, and
- A *Download Manager*.

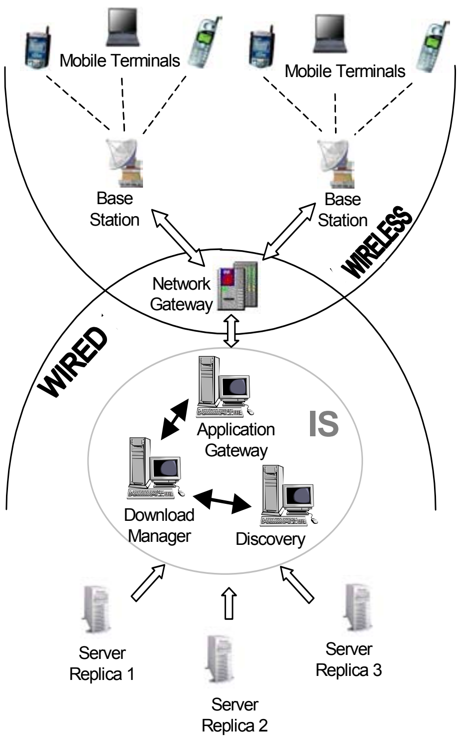


Figure 6.12: Wireless Application Architecture

Those subsystems co-operate to support the download of the required musical resources to the mobile consumer, as detailed below.

6.2.3.1 Search and Download of Musical Resources

The client part of our application represents the interface between consumers and services, and provides a set of *search-and-download* functions. Taking into account that a typical mobile device (such as a smartphone or a PDA) possesses a very limited memory capacity and disk size, we have taken the decision to move all the *search and discovery* functions to the IS side. This entails that the client part of our application needs the collaboration of all the IS software subsystems to determine which musical resources are available, and where they are located. This also implies that users have to perform the search and download activity by stepping through several different phases.

As an example, a complete search and download session for karaoke clips steps through three different phases, and is as follows. In a first phase, a user from his/her wireless device issues to the Application Gateway subsystem a request for a given karaoke clip. (Such request may refer either to a specific song author or to a given song title). The Gateway subsystem passes this request down to the Download Manager. The Download Manager asks to the Discovery subsystem for the complete list of all the available karaoke clips matching the request issued by the user. The Discovery subsystem performs the research of the clips required by the client. Such activity steps through two additional different phases, and proceeds as follows.

First, the Discovery tries to establish a relationship between the titles of the songs requested by the user and the SMIL description files that represent the required songs. (Note that different clips stored in different web servers may exist in the system that matcheshe request issued by the user).

Once this activity is completed, the Discovery passes to the user (via the Application Gateway) the list of all the clips (and correspondent SMIL files) that match the initial request. Upon receiving this list, the user chooses one of the proposed clips. This choice activates an automatic process to download the correspondent SMIL file. It is the Download Manager, now equipped with all the relevant information needed to locate it, that downloads the required SMIL file, and finally delivers it to the software application running on the wireless device. Upon receiving this SMIL file, the software application running on the wireless device examines that file and, following the specified schedule, calls again for the help of the Discovery subsystem and of the Download Manager in order to locate and download all the multimedia resources specified in the SMIL file.

This represents the beginning of the third phase of the discovery/download activity, at the end of which all the multimedia resources are delivered to the wireless device that can perform their play back according to the time schedule defined in the SMIL file. It goes without saying that during this final phase, it is under the responsibility of the Discovery subsystem to find the web locations of all the required multimedia resources, while the Download Manager carries out the download activity by engaging all the different replica servers that maintain a copy of the requested multimedia resources.

Needless to say, in order for the system to work correctly, a preliminary phase has to be carried out where each karaoke server announces the list of the clips it wishes to make available for sharing. Each karaoke server that wants to add its own repository to our IS system may do so by running a software application called the *Data Collector* which is in charge of communicating to the Discovery subsystem the list of the clips along with the associated multimedia resources.

When the song-on-demand service is used, a similar activity is carried out by the system, with the only difference being that the intermediate phase when a SMIL file is searched, downloaded and interpreted is not needed. The mobile user may activate the automatic download of a given song, simply after he has chosen from the list which has been submitted to him at the end of the first phase. Figure 6.13 illustrates a part of the evolution of the above-mentioned process (the interface is in Italian language) which has been developed in our prototype implementation by resorting to the Visual C++ programming language on a Microsoft Windows Pocket PC platform.

6.2.3.2 Design Principles

The next Paragraph is devoted to highlight the technical attributes that were significant for the development of all the software subsystems we have previously introduced.

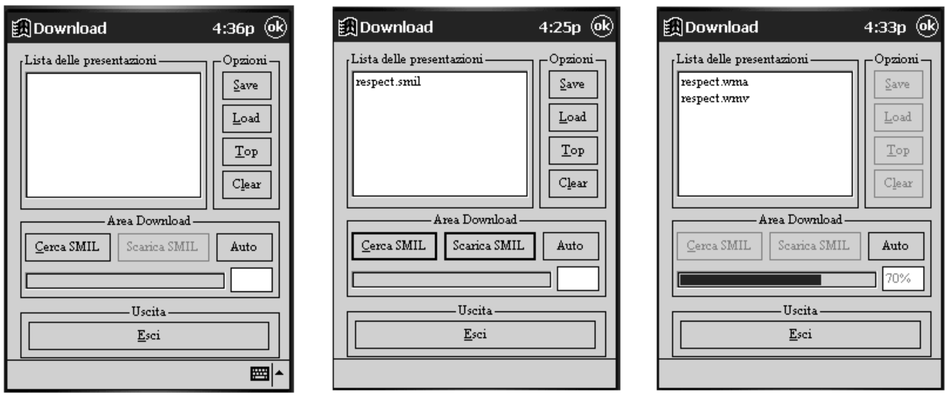


Figure 6.13: Search and Download over the Wireless Internet

We have already mentioned that our wireless Internet application exploits a standard TCP-IP stack for carrying out the communications between the Application Gateway subsystem and the client part of our application. According to the adopted approach, the client part of our application works as any other Internet-connected device, and the end-to-end connection is guaranteed by using the standard TCP protocol. However, to avoid all possible problems due to the time-varying characteristics of the wireless link, our wireless application incorporates a session layer developed on the top of the TCP stack. This additional protocol layer provides stability to the download session, which may suffer from possible link outages.

Figure 6.14 shows the protocol stack we have designed and developed to support all the Gateway-related communications. In particular (as shown in the leftmost side of the Figure) the Gateway subsystem communicates with the client part of the application over a Wireless link. As seen from the Figure, on the Wireless data link protocol stack an IP layer, based on the MobileIP (version 4) protocol is implemented. On the top of this MobileIP level, a standard TCP layer has been built. Finally, to avoid all the network problems due to the radio link layer, the application layer built on the top of TCP has been designed as constructed out of two different sub-layers:

- A Session Layer; this protocol layer is devoted to organize and to manage a download session which may possibly consist of different subsequent communication patterns, in view of possible link outages.

- An Application Layer; this protocol layer is in charge of supporting the different connections needed to search and to download songs.

It is worth noticing, here, that our designed session layer provides users with the possibility of resuming a session that was previously interrupted due to subsequent temporary link outages. The session management mechanism we have designed and implemented has a greater importance for the full success of the download activity of musical resources onto a Wireless device. It is easy to understand, in fact, that very large files (e. g., songs of about 3/5 MB) have to be delivered to the Wireless terminal, and this must be carried out in the presence of a wireless access, which typically exhibits a scarce connection stability and an unpredictable availability. Stated simply, our session layer works as follows: when the Wireless client application opens a connection to the Application Gateway, the Gateway assigns a unique identifier to this new session. If, eventually, the Gateway detects a network failure (i.e., the TCP connection comes out to be closed), the download status is saved at the Gateway side. In particular, a pointer to the last byte received of the multimedia resource is saved, along with the session identifier. At the same time, the identifier of the suspended session is saved at the client side of the application too. As soon as the mobile client application is able to open a new TCP connection to the Gateway, the client application tries to resume the interrupted session by exploiting the session identifier that was previously saved.

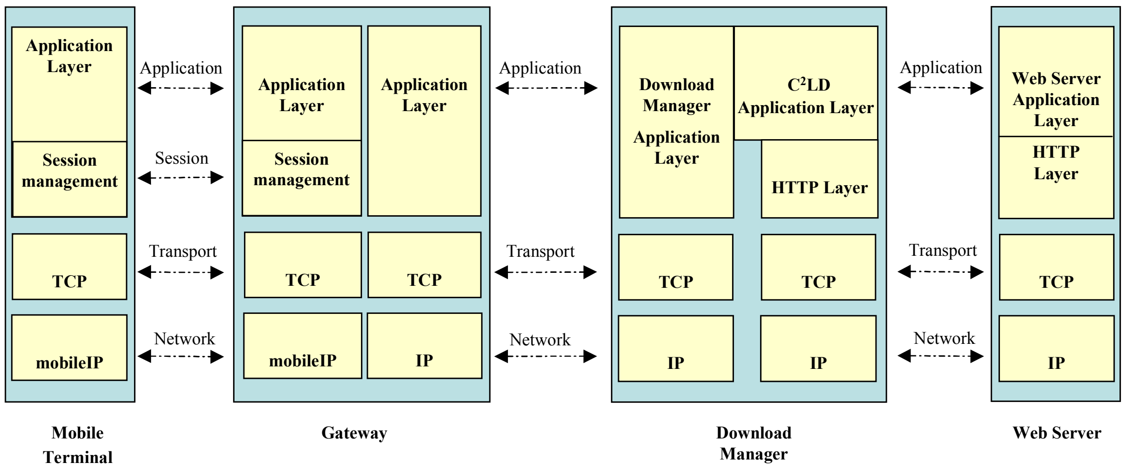


Figure 6.14: Wireless Internet Application: Protocol Stacks

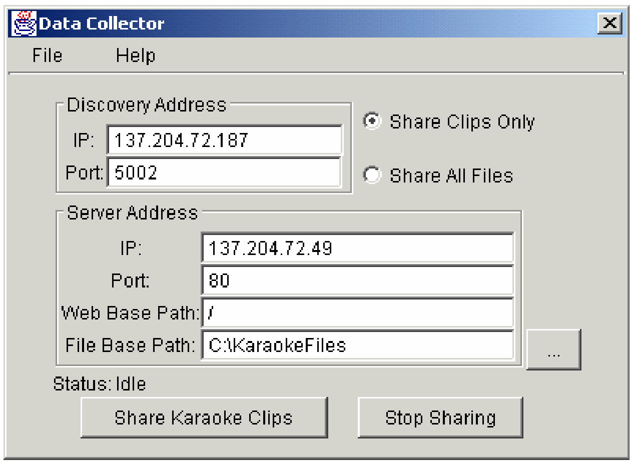
As a final note, it is important to remind that the session management mechanism we have developed is suitable for recovering sessions that are interrupted due to temporary link outages, but it is not adequate to recover from system failures occurring at the Wireless terminal or at the Application Gateway subsystem.

The Download Manager is the real agent responsible for the download process. It has been incorporated in our application built on the top of the HTTP protocol. With the aim of maximizing the service availability and responsiveness, it makes use of the web server replica technology along with the Client-Centered Load Distribution (C^2LD) mechanism. The C^2LD mechanism implements an effective and reliable download strategy that splits the client's requests into several sub-requests for fragments of the needed resource. Each of these sub-requests is issued concurrently to a different available replica server. The C^2LD mechanism is designed so as to adapt dynamically to state changes both in the network and web servers: in essence, it is able to monitor and select at run-time those replicas with the best downloading performances and response times. Figure 6.14 shows the Download Manager protocol stack. As seen from the Figure, the Download Manager has to communicate with each different web server replica, and hence forks into different children processes for each different requested resource. Each process uses a C^2LD application protocol to carry out different download activities from different web server replicas. It is worth noticing that the use of the C^2LD mechanism does not force music providers to organize musical repositories, which are all perfect replicas of the same list of musical resources. A musical resource, in fact, may be replicated within only some of the available server replica of our system.

The software component of our developed system where the relevant information about musical resources are stored and indexed is the Discovery subsystem. The main responsibility of the Discovery is to perform a sort of naming resolution for musical resources, which are requested by clients. In particular, it carries out the activity of:

- Accepting user's requests to establish a formal relationship between them and the correspondent musical resources stored in the system.
- Locating the exact Internet location where a given musical resources is stored throughout the entire system composed of replicated web servers.

To carry out this activity, the Discovery subsystem calculates for each of the multimedia resource embedded in the system a 32 bit-long identifier (called the checksum). This value is computed on the basis of both the file content and of other additional information (such as file name, file creation time, file length, song title and author). Two different indexes are maintained by the Discovery subsystem: the former is needed to resolve users' requests and the latter is used to locate the corresponding musical resources. We have devised a decentralized method to calculate the checksum. According to this scheme, each host server computes the checksum of its musical files and communicates the results to the Discovery system. To minimize both the computational and the traffic overheads, each server has to run a software application locally, termed the Data Collector, which provides the possibility to add or delete the musical resources to be referenced by the Discovery system. In essence, the Data Collector locally performs the checksum computation, and after having computed the checksum of all the files that a given music provider wishes to distribute, opens a TCP connection toward the Discovery subsystem. As a final task, the Data Collector application uploads the computed checksums to the Discovery subsystem. It is worth noticing that the Data Collector is implemented as a Java application to enhance the software portability, and also meets standard security constraints; as it can only read from the local file system, but it cannot execute local write operations. To minimize both the computational and the traffic overheads, each server has to run a software application locally, termed the Data Collector, which provides the possibility to add or delete the musical resources to be referenced by the Discovery system. In essence, the Data Collector locally performs the checksum computation, and after having computed the checksum of all the files that a given music provider wishes to distribute, opens a TCP connection toward the Discovery subsystem. As a final task, the Data Collector application uploads the computed checksums to the Discovery subsystem. It is worth noticing that the Data Collector is implemented as a Java application to enhance the software portability, and also meets standard security constraints; as it can only read from the local file system, but it cannot execute local write operations. Figure 6.15 shows a screenshot of the Data Collector application.



Discovery Address
IP: 137.204.72.187
Port: 5002

[✓ Share Clips Only] Share All Files

Server Address
IP: 137.204.72.49
Port: 80
Web Base Path: /
File Base Path: C:\KaraokeFiles

Status: Idle

Figure 6.15: Screenshot of the Data Collector Application

As seen from the Figure, two different kinds of information must be specified: the address of the Discovery subsystem (under the form of IP and port numbers) and the complete address of the server where the musical resources are stored (including the data path within the local file system).

6.2.3.3 Structuring Karaoke Clips

The SMIL mark-up language is an XML-derived technology designed to integrate continuous media into synchronized multimedia presentations. SMIL allows one to: 1) manage the timing behavior of the presentation, 2) manage the layout of the presentation on the device screen, and 3) associate hyperlinks with media objects. The design of a SMIL-based presentation is performed according to two different phases: first, the author creates spatial regions that will contain the associated multimedia objects, then those multimedia objects are specified along with the timing schedule of their presentation. A SMIL file contains two main elements: a header (between <head> and </head>) and a body (between <body> and </body>). A SMIL header may specify spatial areas by using the <region> tag. (In a SMIL header it is also possible to define meta tags that allow one to insert meta-information). In a SMIL body, it is possible to define which multimedia objects are to be loaded in specific regions. To this aim, tags such as <video> for video files, <audio> for audio files, and <text> for text strings are exploited. The SMIL body is also used to schedule the synchronization of different multimedia objects. Two basic synchronization methods are provided:

- parallel (<par>,</par>): all multimedia objects are executed concurrently in their own regions;
- sequential (<seq>,</seq>): with the sequential method, each multimedia objects is executed in its own region according to a predefined sequential time schedule.

By using the SMIL technology, it is easy to specify a karaoke clip that includes audio, video and the text that periodically flows following the song melody. An example of a karaoke clip, specified by using SMIL, is presented in Figures 6.16 and 6.17 (the title of the song is A little respect, by WHEATUS). In particular, Figure 6.16 reports a code fragment with the SMIL header. As shown in the Figure, two different regions are defined, respectively termed region1_1 and region1_2. Three meta tags are used to specify: i) title, ii) author of the song, and iii) title of the album that contains the song. Figure 6.17 shows a code fragment representing the body of the SMIL file in which the following three different multimedia objects are executed in parallel:

- An audio file (respect.wma),
- A video file (respect.wmv), loaded in region1_1,
- Asequence of textual information, flowing on the screen in region1_2 for a limited time duration, which is specified in seconds by using the dur attribute.

Summing up, the wireless karaoke service we have provided exploits the SMIL technology, thus allowing users to enjoy:

1. A search session where karaoke clips may be searched by simply indicating a part of the song title or a part of the author name;
2. A download session during which the SMIL file, and, subsequently, the associated multimedia resources are downloaded;

3. A playout session when the SMIL player plays back the multimedia objects according to the time schedule specified in the SMIL file.

```
<smil>
    <meta id="meta1" name="Titolo" content="A little respect" />
    <meta id="meta2" name="Autore" content="Wheatus" />
    <meta id="meta3" name="Key1" content="Wheatus" />
    <head>
        <layout>
            <root-layout/>
            <region id="region1_1" top="76%" left="2“
                        height="24%" width="100%"/>
            <region id="region1_2" top="1" left="15"
                        height="75%" width="100%"/>
        </layout>
    </head>
```

Figure 6.16: Header of a Karaoke SMIL File

```
<body>
    <par>
        <video src="respect.wmv"
                  region="region1_2" fit = "slice" repeatCount="6">
        </video>
        <audio src="respect.wma"></audio>
        <seq>
            <text begin="6s" dur = "7s" region="region1_1">
                I tried to discover a little something to make me
            </text>
            <text dur = "10s" region="region1_1">
                sweeter Oh baby refrain from breaking my heart
            </text>
            ...
            <text dur = "4s" region="region1_1">I hear you calling</text>
            <text dur = "17s" region="region1_1">
                Oh baby please give a little respect to me.
            </text>
        </seq>
    </par>
</body>
</smil>
```

Figure 6.17: Body of a Karaoke SMIL File

6.2.4 An experimental study

This Section introduces an experimental study we have conducted in order to assess the effectiveness of our proposed musical services. We carried out several experiments (about 4000) consisting of the download either of a set of Mp3-type songs or of a set of different karaoke clips, according to the selected service. During the experimentation of the song-on-demand distribution service, four different replicated web servers were exploited at the Internet side as song repositories, which were dispersed throughout the world; as per the mobile karaoke service, instead, we used three different replica servers that were located within a metropolitan scenario. The communication between the IS and the mobile client was simulated by means of an UMTS simulator which was able to produce the transmission delay time of each frame at the UMTS radio link layer. Detailed information concerning the experimental models we adopted for our experiments are discussed below.

6.2.4.1 UMTS Simulation Model

At the date of the experimentations, no real measurements of UMTS wireless data were available, hence in our experiments the communication between the IS system (at the Internet side) and the mobile client application was carried out through a simulated UMTS network (running the background traffic class). To this aim, an UMTS simulator provided by the "Fondazione Marconi" (a public Italian foundation for wireless computing) was exploited. The UMTS network simulator we used was able to return after simulations Wireless Network Transmission Time (WNTT) values, that is, the time spent to download musical resources, as computed at the UMTS radio link control (RLC) layer. The simulated transmission of an IP packet over an air interface is illustrated in Figure 6.18. The RLC layer received a PDCP frame, which comprised an IP data packet to which various headers were added for each different level of the protocol stack (indeed, the PDCP layer was not simulated, for the sake of simplicity). We conducted experiments with IP packets coming from the Internet of different possible sizes (namely, 160, 480 and 960 bytes). The resulting PDCP frames were then segmented into RLC data blocks, each of which was 36 bytes large. The result of this segmentation activity at the RLC level was that 5, 15 and 30 RLC data blocks were needed to transmit IP packets with the dimension of 160, 480 and 960 bytes, respectively. Summarizing, if the transmission slot was available, 10, 30 and 60 milliseconds were needed to transmit over the air interface IP packets with the dimension of 160, 480 and 960 bytes, respectively. It goes without saying that the obtained WNTT values depended on some operational parameters, such as the amount of traffic present in the simulated cell, the number of active clients and their speeds. WNTT measurements included also the time spent for possible retransmissions at the RLC level.

The main problem that derives from adopting an approach where simulated results for RLC frames (traveling over the wireless link) are combined to the real measurements obtained for the TCP segments (traveling over the wired Internet) is that segment errors and resulting retransmissions at the TCP level are not taken into account. To solve this problem, our experiments included the possible retransmission time delays incurred at the TCP level obtained by exploiting an external delay management mechanism. This external delay management mechanism was designed to take into account the typical TCP error recovery method based on received ACKs.

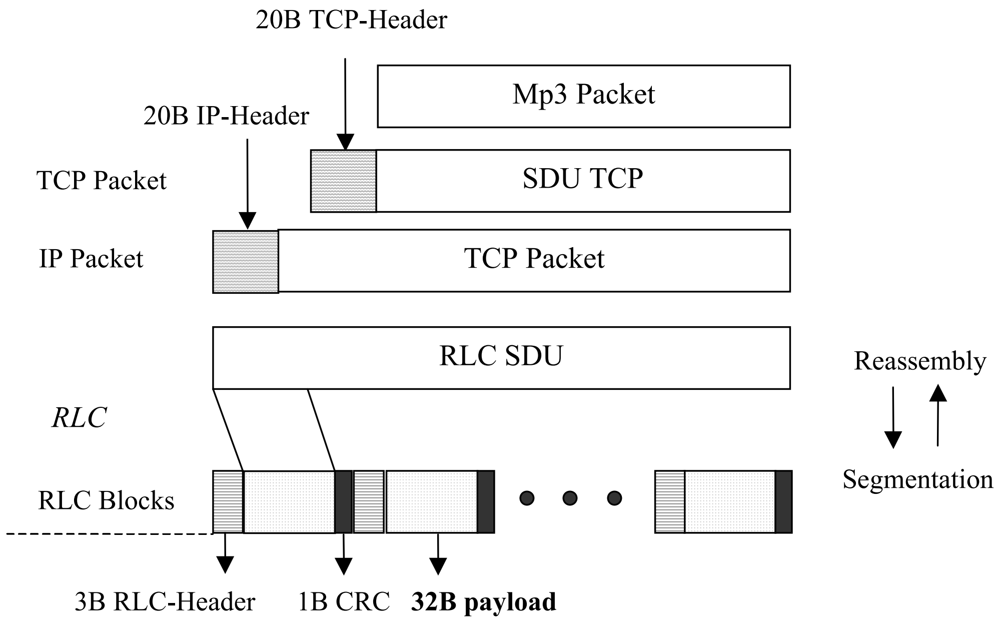


Figure 6.18: Segmentation of an IP packet into RLC data blocks

The delay mechanism compared the WNTT values obtained through the UMTS simulation against the timeout values computed by TCP. If the simulated WNTT value was larger than the correspondent TCP timeout value, then a retransmission must have occurred at the TCP level. In such one case, the WNTT value of that given TCP segment was augmented by an additional value chosen as equal to the next WNTT value extracted from the set of the UMTS-based simulated values. Consequently, the TCP timeout value was updated as follows. If a retransmission at the TCP level was detected according to the method mentioned above, then the subsequent TCP timeout value was calculated as double w.r.t. to the previously computed value. If no retransmission at the TCP level was detected, then the traditional adaptive formula for the calculation of the TCP timeout value, as proposed by Jacobson, was followed. Below, we present the two different measurement architectures (along with obtained results) we adopted at the Internet side to evaluate, respectively, the song-on-demand distribution service and the mobile karaoke service we have implemented.

6.2.4.2 Song-On-Demand: Measurement Architecture and Results

To evaluate the efficacy of the song-on-demand distribution service we have developed, we used four different web servers, geographically distributed over the Internet, providing the same set of 40 different songs. The four different replica servers were respectively located in Finland, Japan, USA and New Zealand (see Figure 6.19). Our designed Intermediate System was running over a Pentium 3 machine (667 MHz, 256 MB RAM) equipped with the Windows 2000 Server operating system, and was located in Italy (Bologna). Finally, the UMTS device, on which the client of our application was running, was emulated by means of a Pentium 2 computer (266 MHz, 128 MB RAM) equipped with the Windows CE operating system. In order to provide the reader with an approximate knowledge of the transmission times experienced over the considered Internet links, it is worth mentioning that the Round Trip Times (RTTs), obtained with the ping routine, between the client and the four different servers (i.e., Finland, Japan, USA and New Zealand) measured, respectively 70, 393, 145 and 491 ms.

As far as the downloading process is concerned, we took the two following basic assumptions:

- Mp3 file dimension: in our experiments we used 40 different Mp3-based songs, whose correspondent file dimension ranged from 3 to 5 MB. The file dimension of 3-5 MB corresponds to the average file dimension of the songs maintained in the Napster system.
- Number of downloads activities: our software application provides support to two different types of song download services: the former consists of downloading a single song, the latter amounts to the downloading of a complete set of songs (song compilation). To evaluate the performance of our system under both these circumstances, we conducted the following different experiments:

    - A set of independently replicated experiments consisting of the download of single songs.
    - A set of independently replicated experiments, with each one consisting of the download of a set of songs. The number of songs for each compilation ranged in the set of (3, 5, and 10). These three values were chosen based on the consideration that the average disk capacity of typical Mp3 players never exceeds 50 MB.

For the sake of brevity, in this chapter we only report on the results we obtained for the download of single songs. Later we report on a large set of results obtained during many experimental trials based on the above mentioned models. In particular, we begin by presenting the measurements we obtained for our wireless application at the Internet side. In essence, with the term WireLine Network Transmission Time (WLNTT) values, we refer to the time spent over the wired Internet links to download a requested song from the replicated web servers towards the IS at the Internet side. These measurements have been compared with those that may be obtained by downloading the same Mp3-based song with a standard HTTP GET method. The first row of Table 6.6 reports those results for Mp3 files whose size is 5 MB.

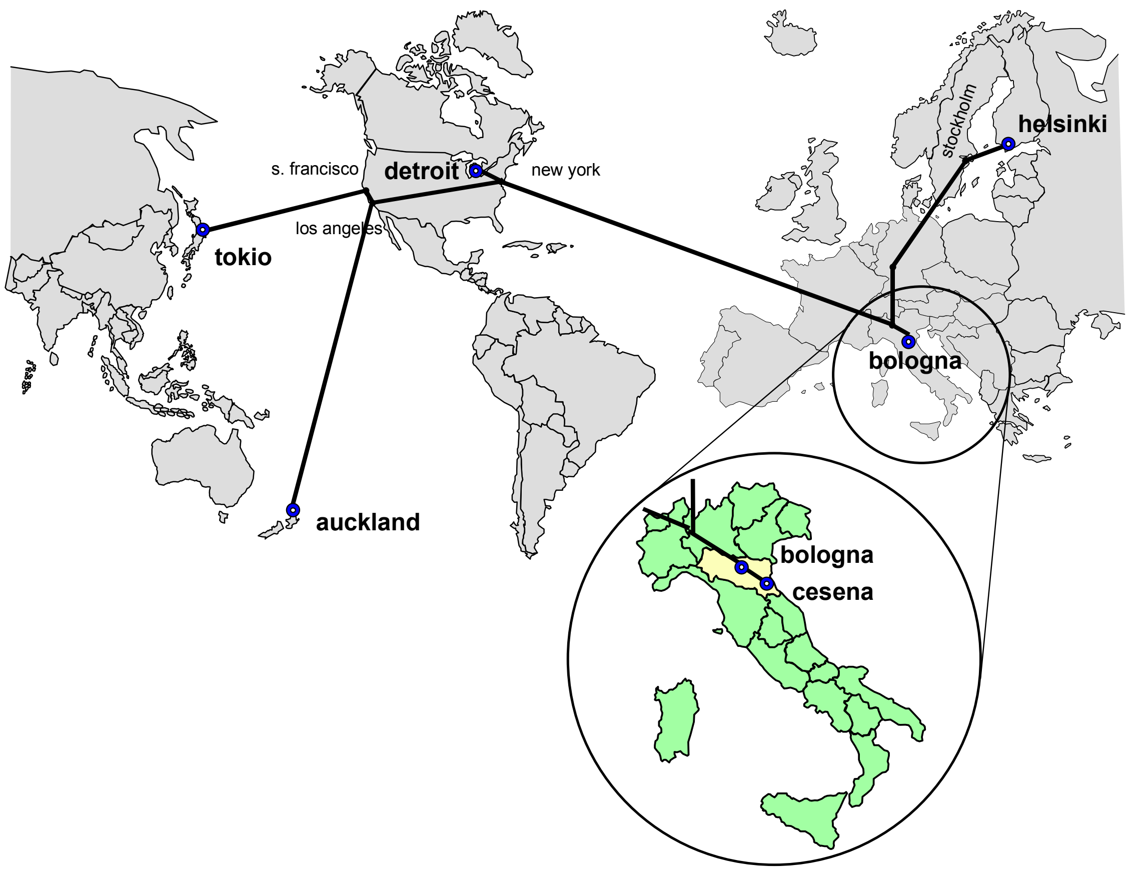


Figure 6.19: Web Server Replicas and Clients: (big picture) Song-on-Demand, (small picture) Mobile Karaoke

Table 6.6: Song-On-Demand: WLNTT Results

	C^2LD (4 Servers)	HTTP			
		Finland	USA	Japan	New Zealand
Download time (seconds)	32.547	47.889	122.191	248.740	624.195
C^2LD improvement (percentage)	–	32%	73.4%	86.9%	94.7%

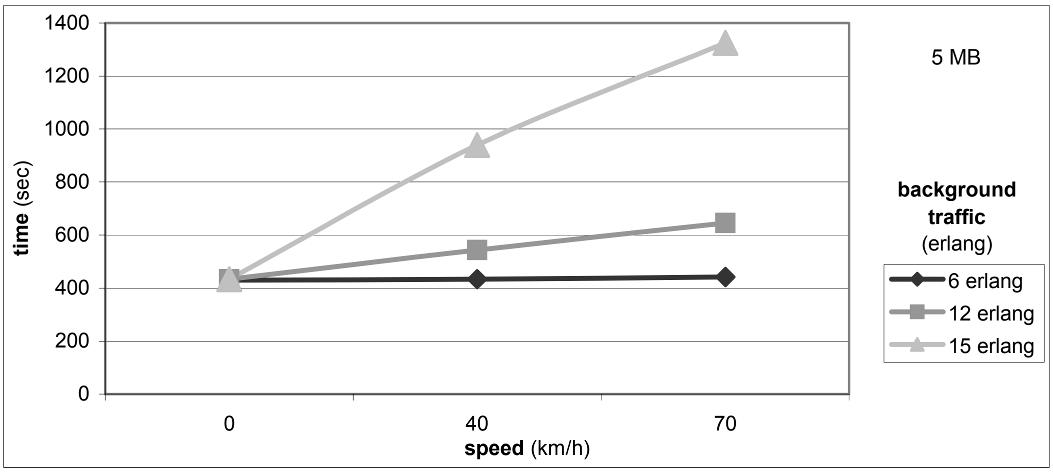


Figure 6.20: Song-On-Demand: WNTT Results for 5 MB-sized Songs

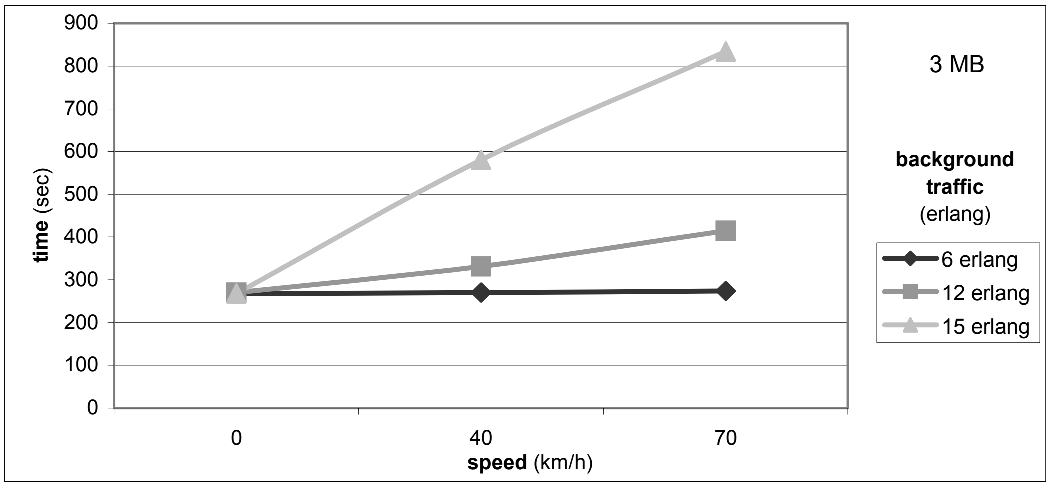


Figure 6.21: Song-On-Demand: WNTT Results for 3 MB-sized Songs

The second row shows the average WLNTT percentage improvement obtained by our system that exploits the already mentioned C2LD mechanism, with respect to the standard HTTP protocol. As shown in the Table, our system obtains an average percentage improvement over the fastest HTTP replica, which is equal to 32 %. As already mentioned, with the term Wireless Network Transmission Time (WNTT) values we refer to the time spent to download songs to the mobile devices through the wireless links. (It is worth reminding that these values have been obtained through UMTS simulations). Figures 6.9 and 6.10 show the WNTT values (respectively, for 5 MB-sized songs and 3 MB-sized songs) depending on the two following traffic parameters:

- The speed at which users move throughout the cell (expressed in Km/h),
- The additional traffic in the cell (expressed via Erlang values).

In addition, it is worth noticing that the WNTT values we have plotted have been obtained by averaging over all the experiments conducted with IP packets of different dimensions (i.e., 160, 480, 960 bytes). The main considerations that derive from an analysis of the results presented in Figures 6.9 and 6.10 are: i) the larger the traffic in the cell (and the users' speed), the larger the corresponding WNTT values, and ii) the best WNTT result may be obtained when the mobile device is completely still. (In such one case a data rate of about 12 KB/s may be obtained). Additionally, it is also worth noticing the impact that the download time values, obtained at the Internet side, have on the total time requested to download songs on UMTS terminals. The obtained average download delays at the Internet side (about 33 seconds) seem to be quite irrelevant if compared with the WNTT values which have been experienced on the wireless links (ranging from 250 to 1325 seconds, i.e. from about 4 to about 22 minutes). This optimal result at the Internet side is probably due to the use of the adopted web replication technology along with the use of our distribution mechanism (C^2LD). Note, in fact, that if we try to download songs from a single web server (such as the New Zealand web server) with the standard HTTP, this can lead to an increase of the WLNTT value by about 600 seconds (10 minutes).

6.2.4.3 Mobile Karaoke: Measurement Architecture and Results

To evaluate the efficacy of the mobile karaoke distribution service we have implemented, three different web replica servers were used. They all maintained the same set of karaoke clips, along with the associated multimedia resources. As shown in Figure 6.19, out of these three web servers, two were located on the same LAN at the Department of Computer Science of the University of Bologna (namely a 100 Mb/sec Ethernet). Instead, the third server and the IS system were deployed on a different 100 Mb/sec Ethernet LAN located in a remote site of the University of Bologna (the Computer Science Laboratory of Cesena). The two different LANs were approximately at a distance of 10 hops each from other, interconnected through a 34 Mb/sec link. The IS application was running over a Pentium 3 machine (800 MHz, 512 MB RAM) equipped with the Windows 2000 Professional operating system. Finally, the client side of our mobile application was emulated on a Pentium 3 machine (667 MHz, 512 MB RAM) equipped with the Microsoft Pocket PC operating system emulator. (It is worth mentioning that the round trip times, obtained with the ping routine, between the emulated client and the three different web servers measured about 10 milliseconds, in average). As far as the downloading process is concerned, we took the two following basic assumptions:

1. SMIL files: we used SMIL files with dimensions ranging from 3 to 4 KB. SMIL files of such dimensions are typically enough large to specify complete karaoke clips;
2. Multimedia resources: SMIL files were used, which typically pointed to two different multimedia objects: i) a WMA (Windows Media Audio) file, and ii) a WMV (Windows Media Video) file. As per audio files, we used a set of WMA files with dimension ranging approximately from 1.5 to 2.0 MB, corresponding to songs sampled at 64 Kb/sec (and lasting approximately from 3.5 to 4 minutes). As per video files, we used WMV video clips lasting approximately 30 seconds, with a quality needing a data rate of 190 Kb/sec, thus yielding file dimensions ranging from 750 to 850 KB. As previously explained, the execution of audio and video resources were synchronized (along with textual information) by using SMIL commands. In the case when an audio file had a duration larger than the video file, the execution of the video file was scheduled to be repeated till the conclusion of the music song.

As far as the obtained results are concerned, the first consideration is about the time spent over the wired links to download karaoke clips from the replicated web servers towards the IS (i.e., WLNTT results). Those WLNTT results amounted to quite small values.

Indeed, 0.2 seconds, in average, were needed to download the SMIL files, while 5/6 seconds, in average, were measured to download the corresponding multimedia objects.
Much larger values were measured for the WNTT results experienced over the wireless link. Like in the case of simple song distribution, those measurements were taken depending on the two following traffic parameters: i) the speed at which users moved throughout the cell, ii) the additional traffic in the cell. The WNTT values needed for delivering SMIL files to the mobile device were as much as 1 second, in average. Instead, as an example, in Figure 6.21, the WNTT values are reported that were measured to deliver, over the wireless link, the multimedia resources of two different karaoke clips, respectively: "Loosing my religion" by REM (hereinafter referenced as song 1) and "A little respect" by WHEATUS (hereinafter referenced as song 2). In particular, song 1 was composed by a WMA audio file of 2.15 MB and by a WMV video file of 765 KB. Song 2 was composed by a WMA audio file of 1.6 MB and by a WMV video file of 850 KB. Figure 6.22 presents three different graphs for each song, with each different graph plotted for a different Erlang value. In each different graph the WNTT values needed to deliver the audio and the video file are presented separately, through two different curves. Each curve varies depending on the user speed. (Yet again, it is worth noticing that the WNTT values we plotted were obtained by averaging over all the experiments conducted with IP packets of different dimensions). As in the case of simple song distribution, it is easy to notice that: i) as the traffic in the cell increases, the corresponding WNTT values increase, and ii) the lowest WNTT results may be obtained with the mobile device still. In addition, the karaoke clip for song 1 was available at the handheld device after an average time interval ranging from about 3.5 to 9.5 minutes ([220, 570] seconds), while the karaoke clip for song 2 was delivered to the UMTS device after an average time interval ranging from about 3 to 8 minutes ([180, 490] seconds). To conclude this Section, it is worth noticing that the WNTT results obtained for the mobile karaoke service appear to better than those obtained with the song distribution service only due to the fact that, in the case of mobile karaoke, multimedia resources of smaller size were exploited in the field trials.

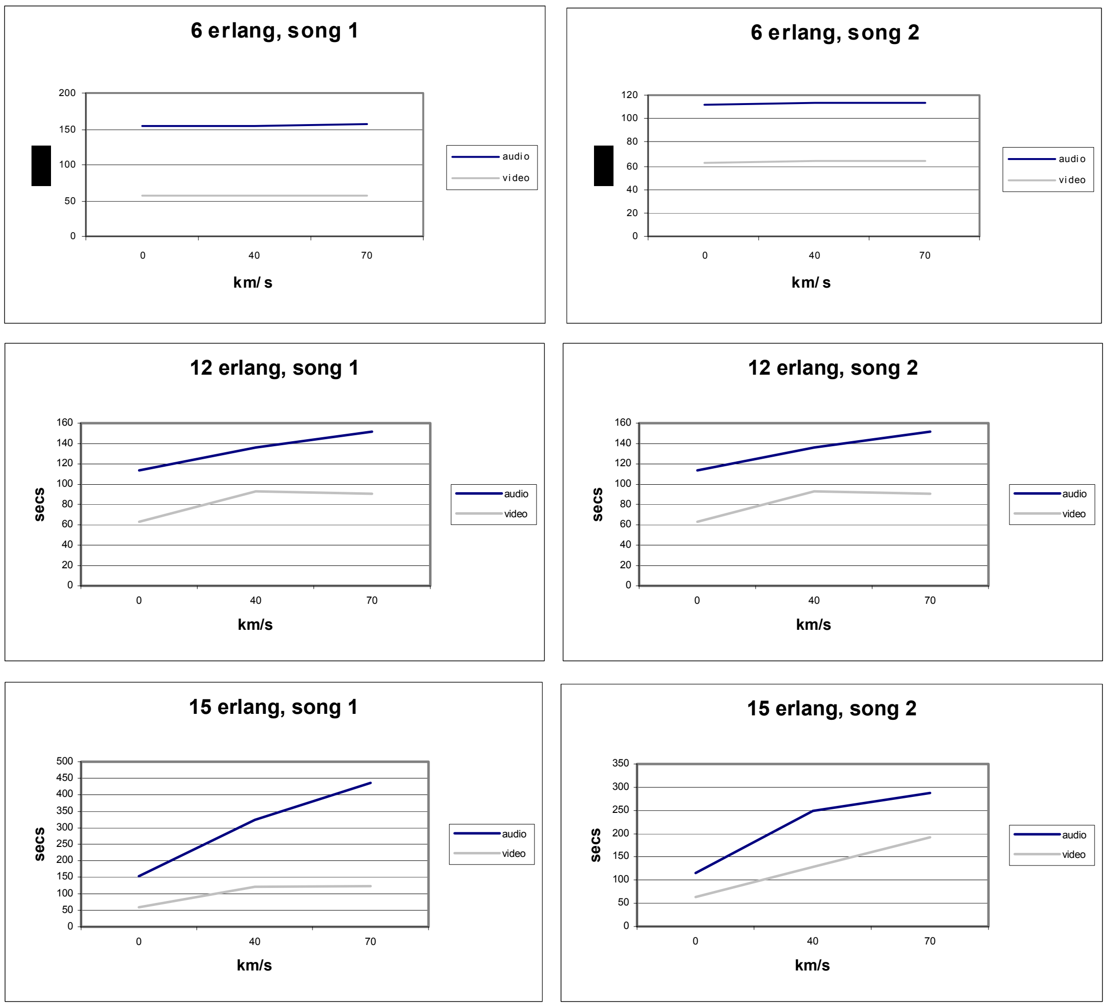


Figure 6.22: Mobile Karaoke: WNTT measurements for delivering song 1 (left graphs) and song 2 (right graphs)

6.2.5 Wireless simulation modelling

In the previous Paragraphs, our experience on implementing a wireless Internet application, designed to support the distribution, on a large scale, of musical resources (from simple songs to synchronized karaoke clips, as shown in Figure 6.23) has been reported. Our application allows mobile consumers to play songs or karaoke clips on handheld wireless devices by exploiting the Internet as a vast storehouse of music resources. Experimental results show that fast large-scale wireless musical services may be provided. Measurements confirm that both songs and karaoke clips may be downloaded from the Internet to wireless devices in few minutes. It is widely discussed if the role of wireless networks is limited to extending the Internet reach, or if there are basically new applications that may be enabled by the wireless access. We claim that our wireless application demonstrates that exciting musical services may be implemented profitably using the wireless technology available today.

Unfortunately, the experimental evaluation was based on a single client exploiting the UMTS data link communication. In this way, these results confirm the feasibility study of this wireless (i.e. UMTS) Internet application, but they were not enough to demonstrate the effectiveness of the application (and its distributed architecture) over also other different wireless infrastructures (for example, GPRS and Wi-Fi) while it is exploited by a large number (hundreds) of clients. It is very difficult to reproduce, in the real Internet, a complex experimental scenario where a distributed system architecture connects to replicated servers who are geographically dispersed in a wired network, which is reached by an increasing number of users that connect through wireless devices.

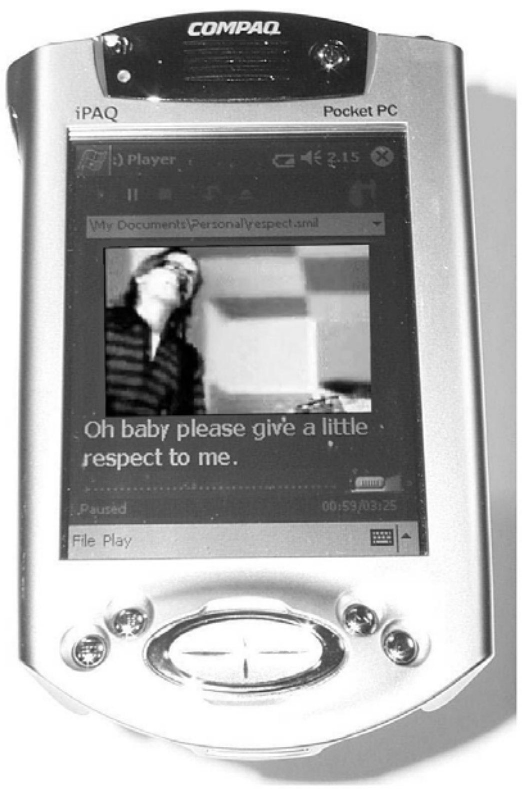

COMPAQ
iPAQ
Pocket PC
Player
\My Documents\Personal\respect.smil
Oh baby please give a little respect to me.
Paused
00:59/03:25
File Play

Figure 6.23: A Screenshot from the Mobile Karaoke Service

Starting from this point, we report on the design, the development and the results of a synthetic emulative scenario especially configured to reproduce the above-specified conditions. In this contest, wireless models to simulate different data link communication have been obtained from regression analyses of real data traces (Roccetti *et al.*, 2002b) (Salomoni, 2003) and have been embodied in the Event Manager as communication models. In particular, the real data come from three wireless infrastructures, namely GPRS, UMTS and Wi-Fi using a Personal Digital Assistant (Figures 6.24) as final device and from Wi-Fi, using a Laptop (Figure 6.25). After the modeling phase, it is necessary to verify that models represent the true system behavior closely enough for be used as a substitute for a real scenario. The *verification* process is concerned with building the model right. It is used in comparison to the conceptual model of the computer representation that implements that concept. Further, it is necessary to increase to an acceptable level the credibility of the models. The *validation* process is concerned with building the right model. It is used to determine that a model is an accurate representation of the real system.

6.2.5.1 Regression and Verification

This section shows how to build a model by means of the real data obtained during on-the-field trials and how to verify it by adopting regressive tests (that confirm it goodness). Specific attributes of each different technology were varied to build the correspondent CM. In essence, the model may be thought as a black-box (see Figure 6.26), where the input variables *(I={IN.1,…,IN.k})* represent these (real) attributes of the wireless infrastructure, while the output (O={OUT}) is computed on the basis of a regressive function *(f(I)=O)* that accounts for the time needed to transmit/download files (in form of IP packets) over a wireless link (i.e. Download Time). To carry out this analysis several regressive functions have been created (Cacciaguerra, 2002). Only those that turned out to verify well the regressive tests have been used as CMs for the synthetic environment(see Table 6.7). Each kind of wireless infrastructure has it own input attributes. For the sake of completeness, it is worth mentioning all the input attributes of each wireless infrastructure was: i) the size of the transmitted file (expressed in KB), ii) the speed at which users move throughout the cell (expressed in Km/h), iii) the additional traffic in the cell (expressed via Erlang values), iv) the coverage quality (good: device under coverage; bad: hand over situations) and v) sizes of the TCP segments (expressed in byte). In particular, the attributes may be qualitative and quantitative. If the attribute is qualitative, a number of models for each value of this characteristic has been created. For example, in GPRS case, we have created a model for good and another for bad coverage (see GPRS case in Table 6.7). Instead, if the attribute is quantitative, this characteristic varies with continuity in the model (see, for example the size of transmitted file in GPRS case).

In conclusion, we report (in Figures 6.24 and 6.25) on the modeling results obtained for the examined wireless architectures. From up to bottom, the simulative models for the Wi-Fi, GPRS and UMTS connections are presented where the time needed to transmit/download test files (Download Time expressed in seconds) is the output value, which depends on the input attribute values mentioned previously.

6.2.5.2 Experimentations

For a better understanding of these results, it is worth reporting the following considerations on the hardware components involved in the experiment and on the values of the attributes monitored during these trails. The downloaded files used for the experiments were stored on a Web server (a Pentium 3 machine, 1 GHz, 512 MB RAM with Windows 2000 Advanced Server). Each client runs on a PDA or a Laptop (an IPAQ 3850 compact with Microsoft Pocket PC or a DELL Inspiron 8000 Pentium III 900 GHz 512 MB RAM with Windows XP pro). To collect real traces over the Wi-Fi infrastructure, a PCMCIA expansion pack with IEEE 802.11b card and a Access Point connected directly to LAN of server Web was adopted. In this case, we assumed that users were still and their numbers were quite low. This is a typical housework case, where:

i) The size of the transmitted file (expressed in KB),

ii) The speed at which users move throughout the cell is near zero,

iii) The number of users in the coverage area is low, and

iv) The coverage quality is good.

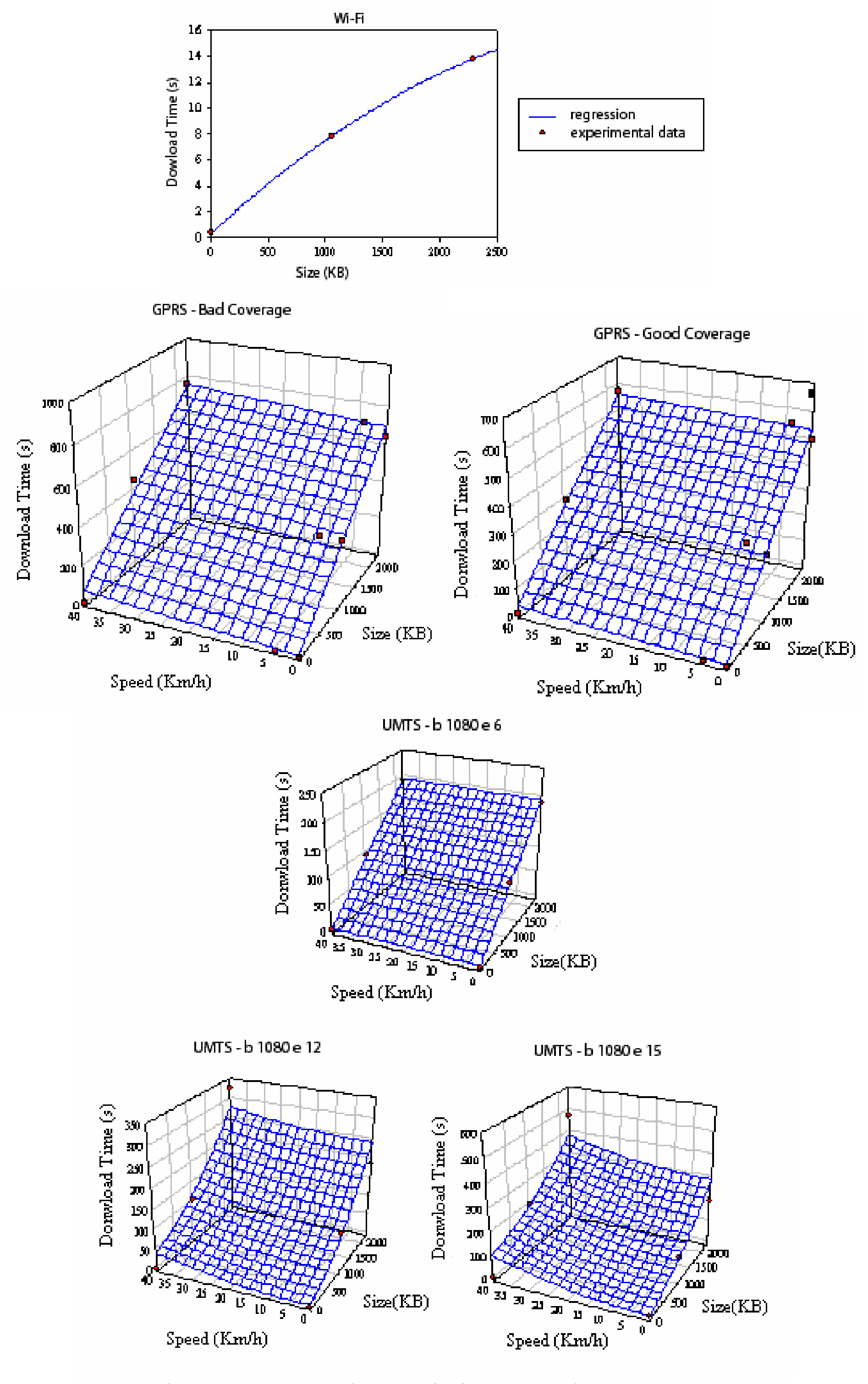
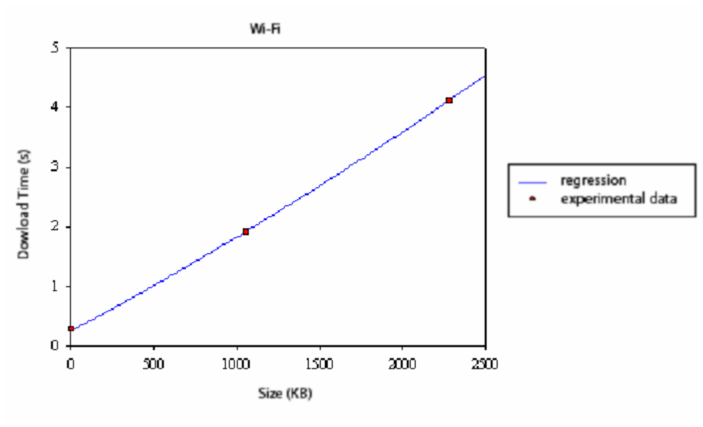


Figure 6.24: Regressions - Wi-Fi, GPRS and UMTS - PDA



Figures 6.25: Regression - Wi-Fi - Laptop

The obtained regressive functions are shown, respectively, on the top of Figure 6.24 for the PDA and on the Figure 6.25 for the Laptop. The (above) hypotheses justify that the regressive function has one (dimension) degrees of freedom on a plane (i.e. the size of the transmitted file).

In the GPRS case, the IPAQ was equipped with a GPRS expansion pack. In particular, the output of a traceroute command has shown the path (of the real IP packets) from PDA to Web server. In this way, it was possible to estimate communication traffic either via wireless or via wired. The attributes of GPRS architecture are:

i)The size of the transmitted file (expressed in KB),

ii)The speed at which users move throughout the cell (expressed in Km/h),

iii)The number of users in the coverage area represent an metropolitan case, and

iv)The coverage quality (good: device under coverage; bad: hand over situations).

The regressive functions obtained of two models are planes (Figure 6.24, middle).

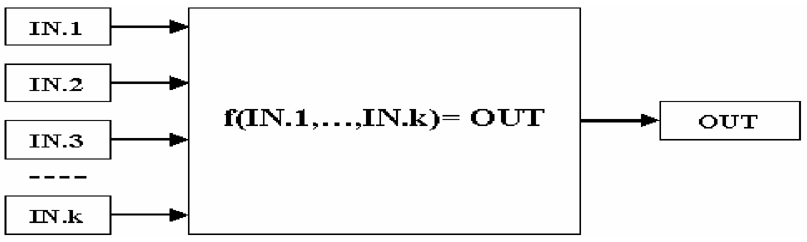


Figure 6.26: Black-Box Model for CM.

Table 6.7: Test for Regressive Functions (Wi-Fi, GPRS, UMTS)

Wi–Fi	**Function**	R	R^2	R^2_{adj}	**Fisher**	**Durbin Watson**	**Std Err of estimate**
PDA	$f = Y_0 + ax + bx^2$	1,000	1,000	---	---	---	---
Laptop	$f = Y_0 + ax + bx^2$	0,998	0,997	0,997	14330	1,99	0,0887
***X*: File Size [KB]**							

GPRS	**Function**	R	R^2	R^2_{adj}	**Fisher**	**Durbin Watson**	**Std Err of estimate**
Good coverage	$f = ax + by$	0,997	0,994	0,993	1212,111	1,922	19,0794
Bad coverage	$f = ax + by$	0,993	0,987	0,985	513,892	2,572	37,9972
***X*: File Size [KB]**				***y*: User's Speed [Km/h]**			

UMTS b1080	**Function**	R	R^2	R^2_{adj}	**Fisher**	**Durbin Watson**	**Std Err of estimate**
6 Erlang	$f = ax + by$	0,999	0,999	0,999	6260,119	2,002	2,3966
12 Erlang	$f = ax + by$	0,956	0,913	0,855	15,759	2,379	47,237
15 Erlang	$f = ax + cx^2 + dy^2$	0,911	0,829	0,716	7,290	2,337	94,5876
***X*: File Size [KB]**				***y*: User's Speed [Km/h]**			

Finally, there are no UMTS repeaters in Italy. Hence, to study this wireless architecture an UMTS network emulator on PDA is adopted. The emulator is designed by "Fondazione Marconi" and uses background class. The attributes of UMTS architecture are:

i) The size of the transmitted file (expressed in KB),
ii) The speed at which users move throughout the cell (expressed in Km/h),
iii) The additional traffic in the cell (expressed via Erlang values) and
v) The sizes of the TCP segments (expressed in byte).

The regressive functions obtained are planes or hyper-planes (Figure 6.24, bottom). To obtain more verifiable models, we adopted qualitative approach. For example the values of v) attribute are just two, so two different models have been created.

6.2.5.3 Validation of the Wireless Models

After the modeling and verification phases, it is necessary to pass through the validation. To control the viability of our wireless models, we have followed two alternative schemes (Cacciaguerra *et al.*, 2003). According to the first scheme, we have contrasted the real data collected with on-the-field trails, with the results generated by our wireless models. In particular, these data are carried out with the scenario explain in the Paragraph 6.2.6, see the Figure 6.30. Based on the second scheme, instead, we have evaluated the probabilities of rejecting a valid model (Type I error) and accepting an invalid model (Type II error). As to the first scheme, we have obtained an average difference of approx. 5% between real and simulated data. Examples of this validation activity are reported in Figures 6.27-6.29, for respectively the Wi-Fi, GPRS and UMTS architectures.

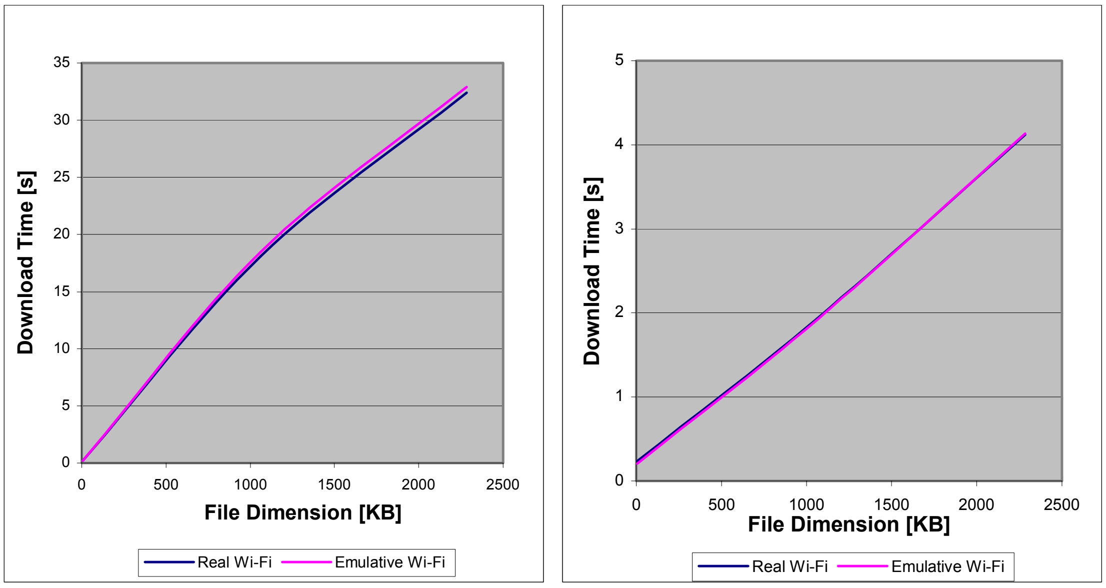
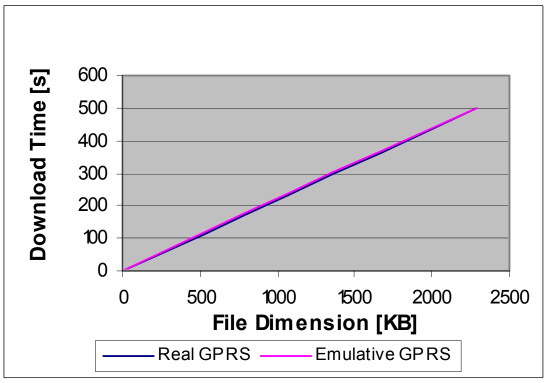
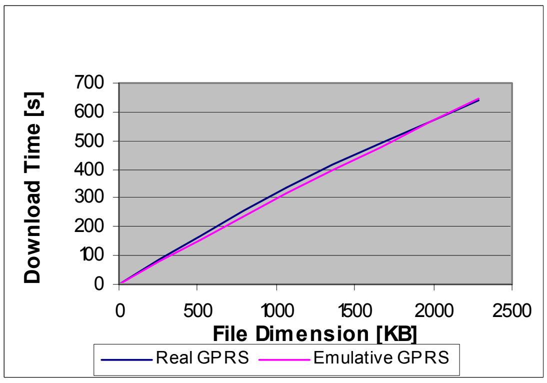


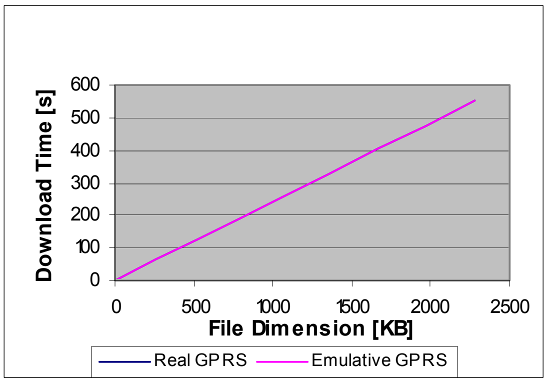
Figure 6.27: Wi-Fi - Comparison between real and emulated data –left) PDA– right) Laptop



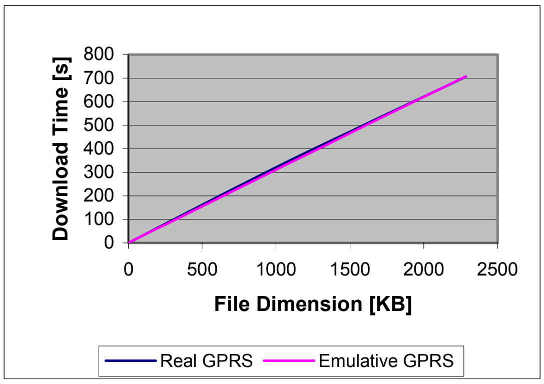
Good coverage - 0 Km/h



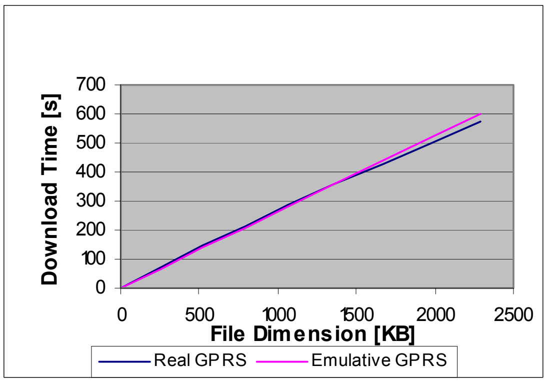
Bad coverage - 0 Km/h



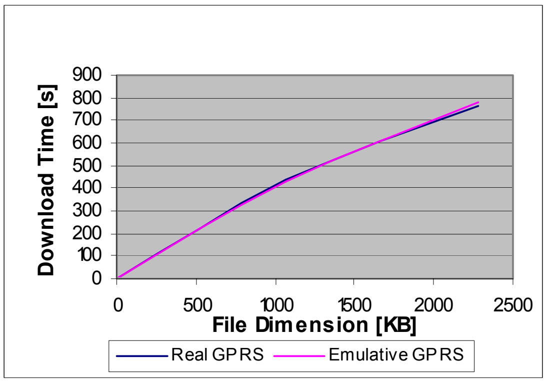
Good coverage - 4 Km/h



Bad coverage - 4 Km/h

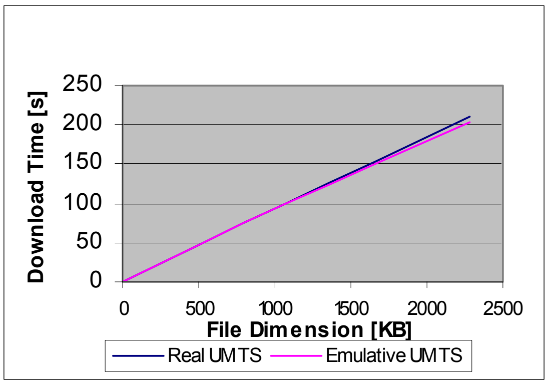


Good coverage – 40 Km/h

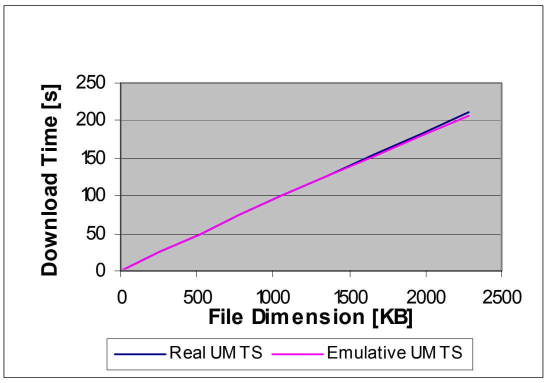


Bad coverage - 40 Km/h

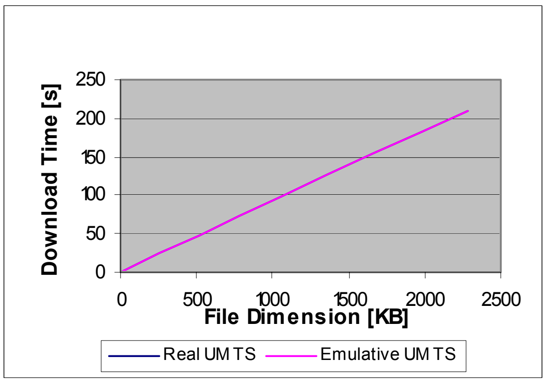
Figure 6.28: GPRS - Comparison between real and emulated data



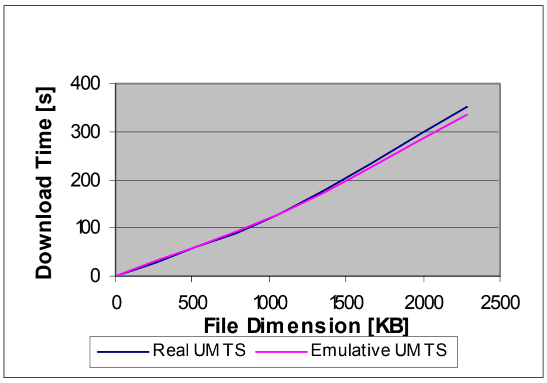
1080 B - 0Km/h - 6 erlang



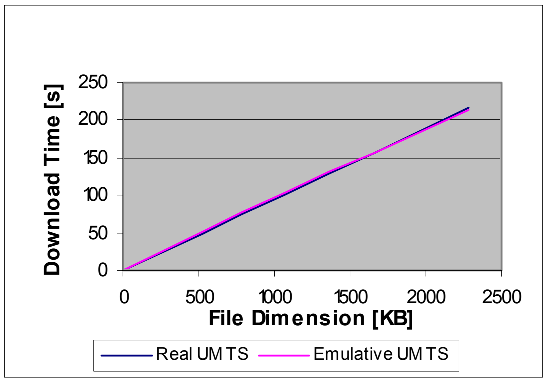
1080 B - 40Km/h - 6 erlang



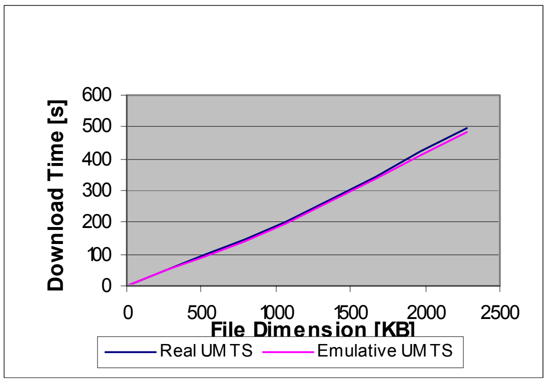
1080 B - 0Km/h - 12 erlang



1080 B - 40Km/h - 12 erlang



1080 B - 0Km/h - 15 erlang



1080 B - 40Km/h - 15 erlang

Figure 6.29: UMTS - Comparison between real and emulated data

According to the second scheme, and based on the standard theory for the calibration of the simulative models, we have formally conducted a statistical test of the null hypothesis, as described above in Equation 6.1. After an experimental campaign to test these models, we compared the simulative results with the real ones. Then, we adopted two validity's tests, named Type I error (or α) and Type II error (or β). Both them test the null hypothesis (see the Table 6.8).

$$\begin{aligned} &H_0 : E(T) = \mu_0 \\ &\text{versus} \\ &H_1 : E(T) \neq \mu_0 \end{aligned} \qquad \text{Eq. 6.1}$$

The validation test consist of comparing the average delay of the system response μ_0 to the model response *E(T).* In particular, when testing hypothesis, rejection of the null hypothesis H_0 is a strong conclusion, because

$$P(H_0 \text{ rejected} | H_0 \text{ is true}) = \alpha \quad \text{Eq. 6.2}$$

and the level of significance α is small ($\alpha = 0.01$). The equation above says that the probability of making the error of rejecting H_0 when H_0 is in fact true is low ($\alpha = 0.01$). This means the probability of declaring the model invalid when it is valid. Failure to reject H_0 must be considered as a weak conclusion unless the power of the test has been estimated and found to be close to 1. In other words, this test detects no inconsistency between the sample data and the specified mean μ_0. The test t student is used to study Type I error. Instead the β, where

$$P(H_0 \text{ failing to rejected} | H_1 \text{ is true}) = \beta \quad \text{Eq. 6.3}$$

is the probability of accepting the model as valid when it is not. The power of the test is the probability of detecting a departure from $H_0 : \mu = \mu_0$ when such departure exists. This power may also be expressed as 1 - β. In the validation context, the power of the test is the probability of detecting an invalid model.

Table 6.8: Type of Error in Model Validation

Statistical Terminology	**Modeling Terminology**	**Associated Risk**
Type I: rejecting H_0 when H_0 is true	Rejecting a valid model	α
Type II: failure to reject H_0 when H_1 is true	Accepting an invalid model	β

To consider failure to reject H_0 is a strong conclusion. The modeler would want β to be small. β depends on the sample size n and on the difference between $\overline{T}$ and μ_0. For a fixed sample size *n*, increasing α will decrease *β*. Once α is set, and the critical difference to be detected is selected, the only way to decrease β is to increase the sample size. A Type II error is the more serious of two types of errors, and thus it is important to design the simulation experiments to control the risk of accepting an invalid model.

According to this validation approach we carried out the test hypothesis H_0 with a probability of rejecting H_0, when H_0 is true, equal to $\alpha = 0.01$ and a sample of $n = 30$ experiments. Hence, we have computed the test statistics:

$$t_0 = \frac{E(T) - \mu_0}{S/\sqrt{n}} \quad \text{Eq. 6.4}$$

where μ_0 is the specified value in the null hypothesis H_0, E(T) is the average delay of the model response and S is the standard deviation of T. In Table 6.9, we have reported the t_0 values when different wireless technologies are used to download files of three different sample sizes (5KB, 1MB, 2MB). Obviously, the values of t_0 are functions of E(T) and μ_0 as reported in Table 6.9. Comparing our t_0 results with a t-student value equal to $t_{\alpha/2,n-1} = t_{0.01} = 2.46$, we can conclude that we have not rejected a valid model as the following inequality is satisfied:

$$|t_0| < t_{0.01} \quad \text{Eq. 6.5}$$

Further, we have conducted a test hypothesis H_1 to compute the probability of accepting the model as valid when it is not valid, i.e. the β (see Equation 6.3). This test statistics has been conducted with the value $\hat{\delta}$ of the difference between E(T) and μ_0 computed over the sample values. In Table 6.10, we have reported the β values when different wireless technologies are used to download files of three different sample sizes (5KB, 1MB, 2MB).

Table 6.9. Type I error (α = 0.01)

Test n=30		**5KB**			**1MB**			**2MB**		
		μ_0	E(t)	t_0	μ_0	E(t)	t_0	μ_0	E(t)	t_0
Wi–Fi	**PDA**	0.20	0.22	0.02	18.1	18.5	0.26	32.4	32.9	0.20
	Laptop	0.24	0.21	1.40	1.93	1.92	0.07	4.12	4.14	0.15
GPRS		1.28	0.40	0.93	256.1	253.3	0.75	550.2	552.0	0.66
UMTS		0.60	0.20	0.52	101.0	101.6	0.17	211.8	205.2	1.28

Table 6.10. Type II error (β)

Test n=30		**5KB**		**1MB**		**2MB**	
		$\hat{\delta}$	β	$\hat{\delta}$	β	$\hat{\delta}$	β
Wi–Fi	**PDA**	0.47	0.2	0.47	0.2	0.37	0.25
	Laptop	4.42	0.1	0.22	0.25	0.48	0.2
GPRS		1.70	0.1	1.36	0.1	1.21	0.1
UMTS		0.95	0.1	0.32	0.1	2.34	0.1

Obviously the values of β are function of $\hat{\delta}$. It is possible to observe that in most situations the probability of accepting a wrong model is satisfying (10%). In a few situations this probability increases up to 25%. This occurs typically when the Wi-Fi technology is exploited. This result does not take us unaware as with this technology we have large variations in throughput due to different loads.

6.2.6 Building an Emulative Scenario with Wireless Models

The wireless models of Interceptor have been used to test the Intermediate software System (IS) and its music delivering service. The IS is hosted in an Internet server and it is in charge of managing all the communications between the mobile devices and the Internet infrastructures. In particular, the IS integrates:

- The *wireless* communication with the mobile devices and
- The *wireline* communication with the replicated Web servers.

In essence, IS is in charge of implementing a *reliable* and *responsive* discover-and-download service based on the user's requests over the integration of the wireless and wired network. This means making service request to IS by means of a PDA that sends IP packets to it via wireless. Then, when a service request has reached IS, the IS downloads the data from the replicas via wired and reply to the mobile device via wireless (see in the Figure 6.30). Hence, reproducing this network scenario in a laboratory is possible by connecting Web server replicas, IS and mobile devices to Interceptor. In this case, Interceptor is in charge of reproducing, at the same time, wireless and wired communications by adopting the appropriate wireless and wired communication models. In particular, the computer, that hosts Interceptor, interconnects network workstations from different LANs (10.0.0.0/24, 10.0.1.0/24 and 10.0.2.0/24; see the Figure 6.30). This computer is Pentium III 900 with 1GB RAM running Linux (2.4.0 Kernel family) and has three Ethernet card (100 Mb/s). Each Ethernet card is plugged to a single network:

- eth0 → the LAN (10.0.0.0/24) that contains the computer running IS,
- eth1 → the LAN (10.0.1.0/24) that contains the replica servers, and
- eth2 → the LAN (10.0.2.0/24) connected to the rest of the world.

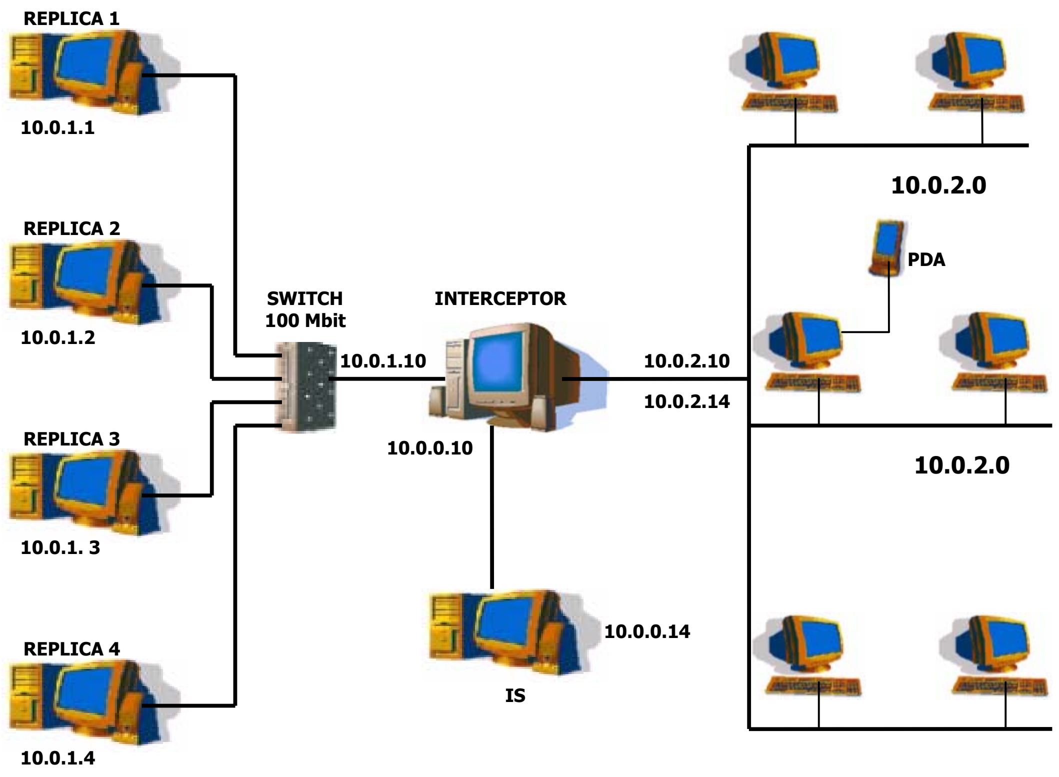


Figure 6.30: Emulative Scenario

The computer, that accommodates IS, is Pentium III 900 with 512 MB RAM connected by means of Ethernet card (100 Mb/s) running Windows 2000 server. The four replica servers are Pentium II 233 with 256 MB RAM connected by means of Ethernet card (100 Mb/s) running Windows 2000 server. Finally, there is also a PDA IPAQ 3870 linked by means of an USB 1.0 cable (11 Mb/s) to a workstation running PocketPC 2002/WinCE. This workstation stays inside the 10.0.2.0/24 LAN and forwards the IP packet coming from the USB device to the Ethernet card.

In the previous (real) experiments (see the Paragraph 6.2.4), IS was tested with a mobile device. The aim of this (synthetic emulative) campaign is to test the performance of IS by scaling up the number of users (on mobile devices). In this context, the wireless (communication) models are useful to simulate the communication delay between a mobile terminal and its base station. Scaling up the number of users (i.e. mobile devices) is possible by means of two different approaches. The former adopts, as mobile device, the PDA linked via cable to the LAN (see the Figure 6.30), while other workstations run some clients that request service to IS. Also the workstations were delayed with the wireless (communication) models and request file sizes in PDA fashion. The latter adopts a Workload Traffic Generator that reproduces the requests coming from Web and Multimedia application when a user exploits mobile devices (such as PDA and Laptop).

6.2.6.1 First Approach: Real PDA and PDA approximation

In this former approach, the performance of the IS are monitored by the real PDA (linked via cable to the LAN), while different workstations make service request to generate workload as coming from several traffic sources. This was only a first approach useful to study the behavior of our application running on a PDA when the IS is stressed. This (synthetic emulative) campaign starts by generating the traffic with 5 clients and continues by doubling them until the limits of IS are not reached. Each client is able to interface with IS using its communication protocol and runs a simple program that requests cyclically the same series of files. The communication of each client passes through the synthetic environment and is delayed with the wireless (communication) models. The models and their (respective) attributes adopted in this campaign for the wireless communications are:

- Wi–Fi
- GPRS
    - Good coverage, 4 Km/h
    - Good coverage, 40 Km/h
- UMTS
    - Frame Dimension 1080, 12 Erlang, 0 Km/h
    - Frame Dimension 1080, 12 Erlang, 40 Km/h

Two different graphs report the average values of the download time increasing the size of the requested file, for each trail: the former highlights the behavior of the critical curve while the latter that of the single PDA. This is due to the fact that the two curves are very different in scale and it is not possible to plot them in the same graph without losing same important details. In the Wi-Fi plot (Figure 6.31, left), the curve with 40 clients increases the download time, quickly. The Figure 6.31 on the right is more interesting because (showing the curve of the single PDA) highlights that increasing the number of clients the performance of IS decreases and, in particular, the download time increases very quickly. Also in both the GPRS cases (Figures 6.32 and 6.33, left) the curve with 40 clients is critical for IS. Differently from the previous case, until 20 clients are reached, the system is stable and the download time seems independent of this number. In particular, the single PDA curve has lower performance than the 5, 10 and 20 curves (Figures 6.32 and 6.33, right). This is due to the implementation of the client. In fact, each client requests the same files to the same replica servers; in this way the default caching technique of each web server decreases the average of download time. For example, when the second request reaches the web server, it finds the wanted files in cache. This fact is present in each case. The UMTS cases are similar to GPRS. 40 clients are critical for IS (Figures 6.34 and 6.35, left) and until 20 clients are not overcome, the system is stable and the download time seems independent of this number (Figures 6.34 and 6.35, right). Differently (from the GPRS), there is less difference between the performances of the single PDA than that of clients until 20.

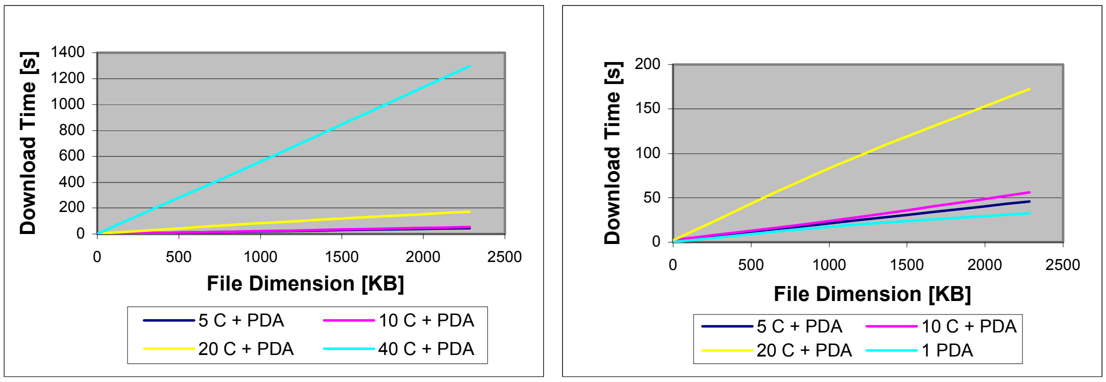


Figure 6.31: Wi-Fi - Comparison among different numbers of clients: left) critical curve and right) single PDA curve

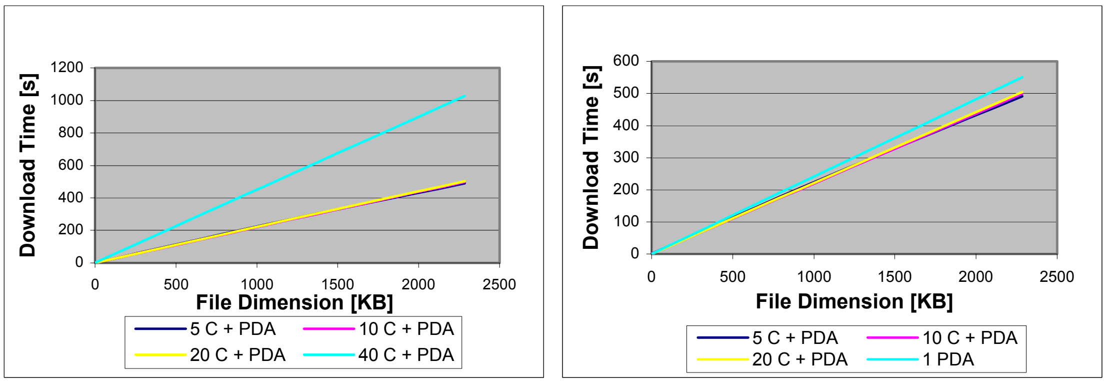


Figure 6.32: GPRS - good coverage – 4 KM/h Comparison among different numbers of clients: left) critical curve and right) single PDA curve

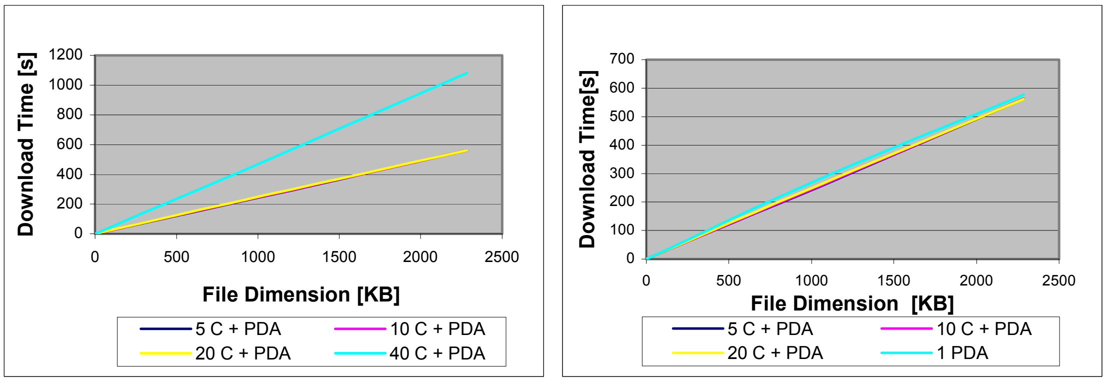


Figure 6.33: GPRS - bad coverage - 40 KM/h Comparison among different numbers of clients: left) critical curve and right) single PDA curve

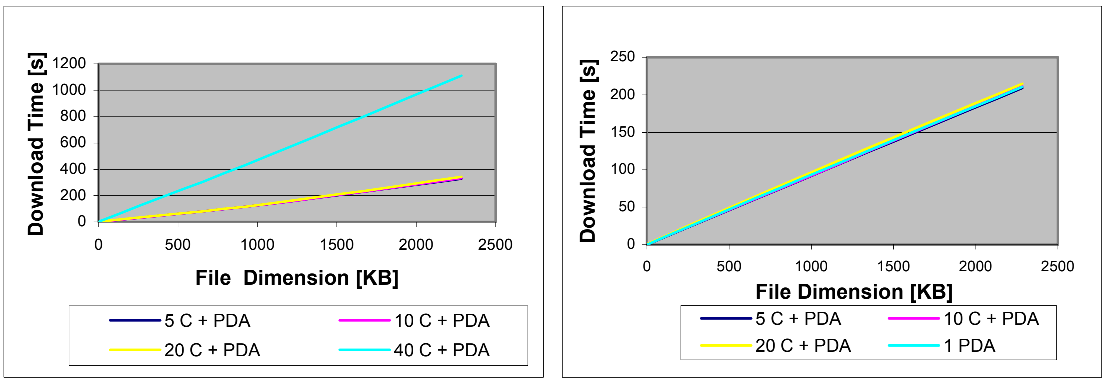


Figure 6.34: UMTS – 1080 B -12 erlang – 0Km/h Comparison among different numbers of clients: left) critical curve and right) single PDA curve

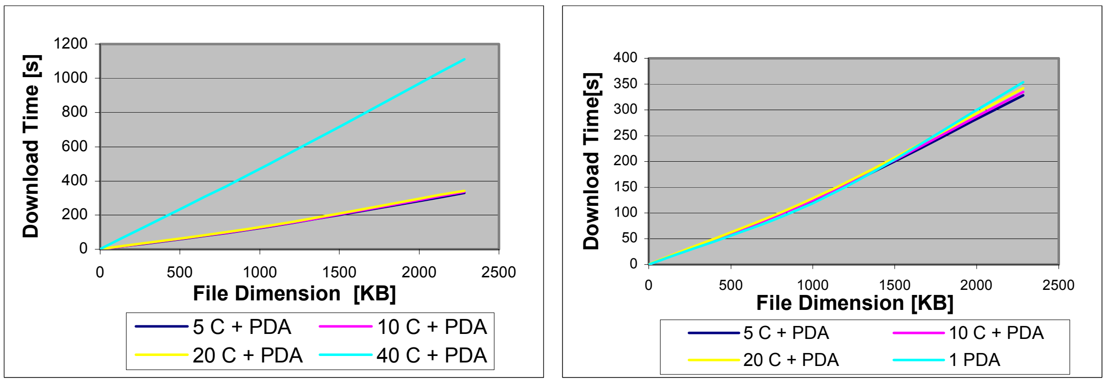


Figure 6.35: UMTS – 1080 B -12 erlang – 40Km/h Comparison among different numbers of clients left) critical curve and right) single PDA curve.

There are two final considerations related to all these curves. The former relates to the bandwidth of the wireless infrastructure and the latter to the number of clients. The bandwidth is different in the three infrastructures and causes different behaviors. In the UMTS and the GPRS the bandwidths are lower while in the Wi-Fi is higher. Until both UMTS and GPRS requests do not overcome 20 clients, the interarrival request rate is not critical for IS, because the bottle-neck of this communications is the wireless link (low bandwidth). In this case, the IS wired downloader is faster than the IS wireless uploader. Differently, in the Wi-Fi case, the IS wired downloader is slower than the IS wireless uploader. For this reason, the interarrival request rate stresses more the IS. The results of these two cases are that the users adopting a Wi-Fi infrastructure download the same files speeder that those using UMTS/GPRS one. Further, the performance of IS is sensitive not only to kind of wireless infrastructures, but also to the increasing number of clients. When the requests come from 40 clients (independently of the wireless infrastructure), IS is not able to manage the service and, consequently, the download time increases quickly. There are two possible ways to increase the performances of IS. In the former the power of the hardware components should be increased or the components of IS should be distributed. In the latter, a caching technique on IS should be adopted. This caching technique should store the most requested files on the Gateway component. In this way, the IS prevents to download, again, the requested file from the replica. This may be more effective in the Wi-Fi case, where the bottleneck is the wired communication. A side effect of this campaign is a first comparison between GPRS/UMTS and Wi-Fi infrastructures. In fact, while GPRS/UMTS supply an efficient service of mobility but a low bandwidth, the Wi-Fi does not supply an efficient service of mobility, but a high bandwidth. GPRS/UMTS have a large diffusion because it can adopt the GSM infrastructure. This infrastructure has good coverage and mobility in a lot of country, but probably, when the multimedia applications will request a higher bandwidth, the Wi-Fi will win this challenge.

6.2.6.2 Second Approach: M^{2}G and PDA UEs

We developed a Workload Traffic Generator (called M^{2}G) that is in charge of reproducing the behavior of a user running Web and Multimedia application on its mobile devices and of generating its corresponding communication traffic. In this second approach, the hardware configuration of the synthetic emulative scenario is the same of the previous one except for the absence of PDA. In fact, differently from the previous case, the workstations run the M^{2}G instead of simple clients that request cyclically the same series of files. As already mentioned before, the Wi-Fi is considered better for the multimedia applications (i.e. its download times are lower than those of GPRS/UMTS), so this synthetic emulative scenario adopts, as wireless infrastructure, only the Wi-Fi. Each M^{2}G (running on a workstation) simulates a multimedia user that downloads files through a PDA (called “PDA” user equivalent). The “PDA” user equivalent is useful if not enough real PDAs may be involved in the synthetic emulative scenario. Building a “PDA” user equivalent over a workstation is possible by reproducing its following features: the storage capacity, the communication capability and the hardware power. Each PDA has a limited short quantity of memory (≈ 32 MB; during the 2002/2003) to store data. To reproduce this feature on a larger storage workstation, an appropriate file size distribution is adopted. In this case, the distribution of the size of the downloaded files is limited by the storage capacity of a PDA (i.e. from 10 KB to 10 MB). The communication capability of a PDA is related to two main aspects: the communication delay due to network infrastructure and the implementation of the socket options. Being the PDA a mobile device, it adopts a wireless card. In our scenario, the communication delay (due to pass through a wireless infrastructure) is reproduced by the apposite Wi-Fi

CMs. The second aspect is related to the study of the socket options for PDA that are different from the default ones used on a standard PC. The getsockopt system call showed the socket options of the PDA (PocketPC 2002/WinCE). For this reason, at the beginning, each PDA equivalent set its socket options (by means of the system call setsockopt). In particular, the getsockopt system call highlighted that the size of the receive buffer and of the send buffer were 4 KB (lesser than a standard PC). Finally, the hardware power of a PC is higher than that of a PDA: speeder CPUs, speeder buses and larger caches. To obtain a final version of the PDA equivalents we should study a way to limit the hardware power of the PC (this work is on progress). In this Paragraph, we use the approximation of the PDA equivalents working with the implementation of the first two features.

The behavior of each PDA user equivalent passes through different phases. It connects the IS, it opens a session, it requests a list of files (song, clip and text), it downloads these from IS and finally it closes the session. The plots generated by PDA user equivalents are reported in Figures 6.36-6.38. These plots show the comparison between the curves that interpolate the resulting data of this second approach and that obtained in the first approach (Figure 6.31). Finally, the Figure 6.39 reports all curves produced with different PDA user equivalents. This Figure highlights the behavior of the IS by increasing the number of the PDA user equivalents. In the Figure 6.36, the curve interpolating the data, requested from 1 PDA equivalent, shows a linear slope. In particular, the PDA equivalent is able to download a file of 1 MB less than 20 seconds. This Figure highlights the comparison between the behavior of a single client running on a real PDA and that of a PDA user equivalent. The two plots are very close from 0 KB to 1 MB. The main difference is that one has a linear slope and the other has a quadratic one. This Figure can claim that the PDA equivalent represents an upper bound of the behavior of the real PDA. In the Figure 6.37, the curve interpolating the data, requested from 5 PDA equivalents, has a linear slope. It is very interesting because both curves present a linear slope and they are very close. In particular, the performance of the trails with PDA equivalents are just better than the others until 1MB (less than 20 seconds) while after this threshold these makes just worse. In the Figure 6.38, also the curve interpolating the data, requested from 10 PDA equivalents, has a linear slope. This Figure shows a little difference between the two curves: there is an increment of 4 seconds for 1MB and of 8 seconds for 2MB in the data monitored in the first approach. The behavior of the 10 clients is more stressing than that of the same number of PDA user equivalents generated by the M^{2}G. The reason is that the 10 clients request cyclically without any pause while the M^{2}G alternates a phase in which each PDA user equivalent make requests, to a phase in which each PDA user equivalent dies and another one will be born. In this last phase, the PDA equivalent does not load the IS and the replica servers with its request. In this way, the performance of the other PDA user equivalents is better. In the Figure 6.39, it is possible to see that also the function interpolating the data requested from 15 PDA user equivalents has a linear slope (there is not a corresponding curve in the first approach). Finally, in this last Figure all the curves of the PDA equivalents are reported. In this way, it is possible to highlight some final remarks on increasing the number of PDA equivalents. All the curves have a linear slope. The download time of the files delivered by IS increases with the number of PDA user equivalents. Also this second approach shows that scaling up the number of the PDA user equivalents (for example 20 as in the first approach) the communications with the IS become critical. In this case, the IS was not able to deliver its service satisfying the requests coming from all PDA user equivalents. In particular, the log files show that a lot of PDA user equivalents were not able to open or to conclude its session in a correct way. The problems were two kinds: related to connect IS and to receive the file. In the former case, the PDA user equivalent tries to reconnect until a timeout expires. After the expiration, the session is close and another PDA user equivalent is born. Most probably also this next PDA equivalent will have the same problem of the previous one. In the latter case, the PDA equivalent that does not receive any communication on the connected socket, after a timeout expiration, contacts the IS asking for an acknowledgement. If the IS does not respond, the PDA equivalent closes the session, otherwise, if it does, the service goes on until the expiration of another timeout or the conclusion of this session is reached. In both cases, the result is that the number of active PDA user equivalents is lesser than that specified at the beginning of the simulation. For this reason, the active UEs can exploit to the utmost the IS, showing better performances.

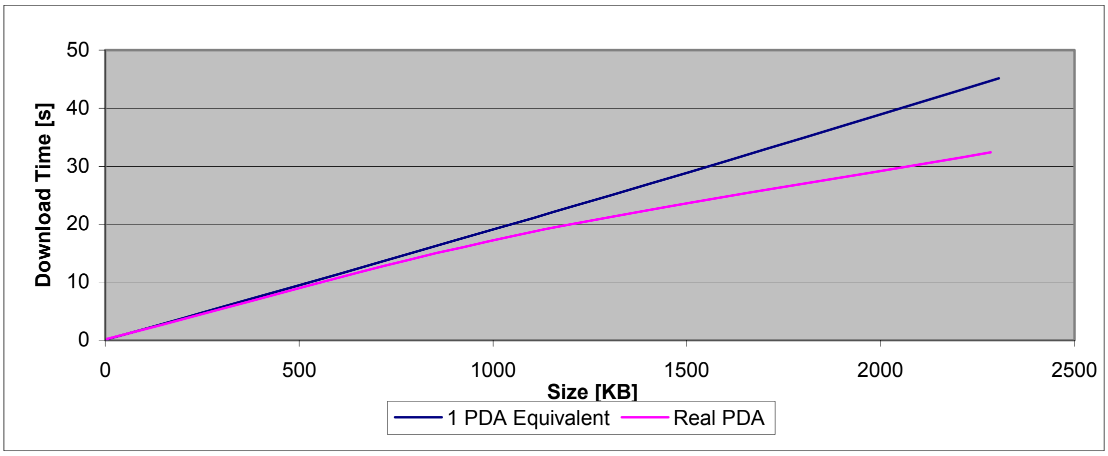


Figure 6.36 Comparison between PDA equivalent and real PDA

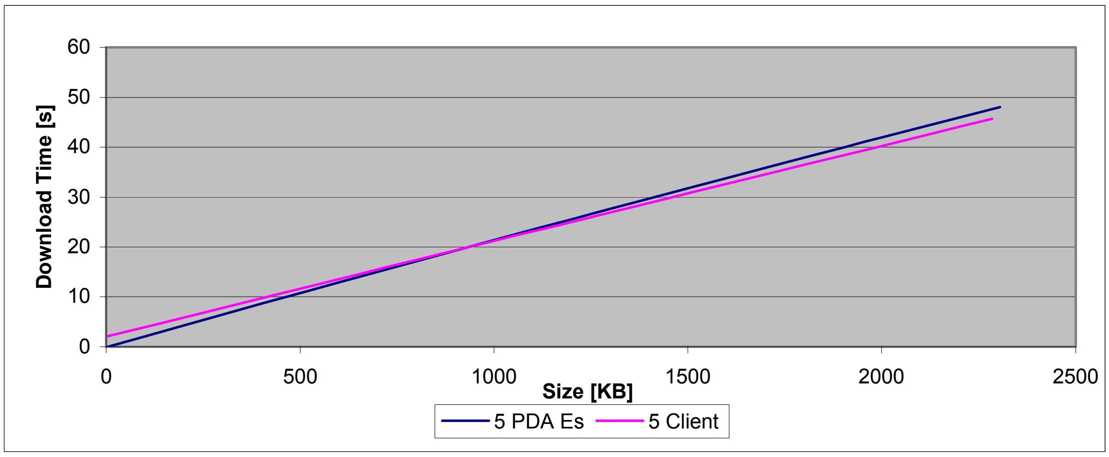


Figure 6.37 Comparison between 5 PDA equivalent and 5 client (first approach)

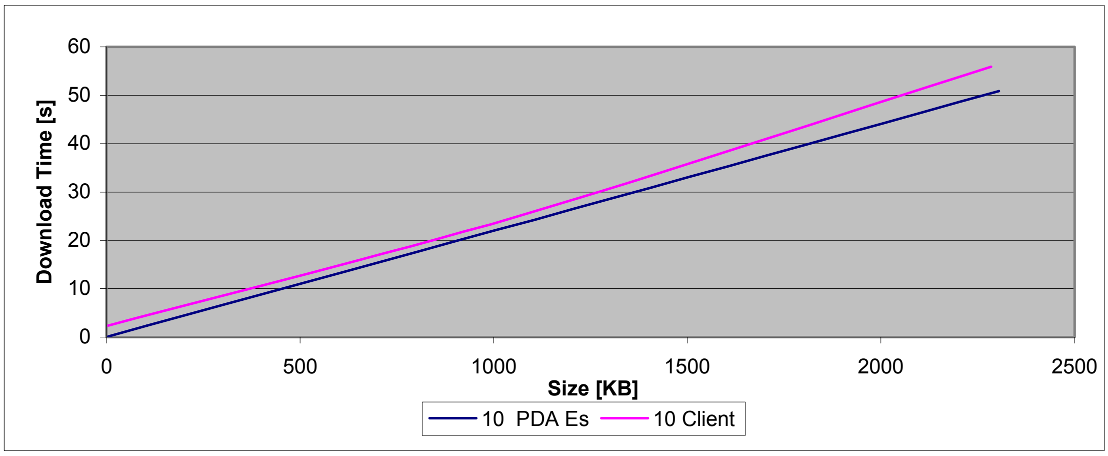


Figure 6.38 Comparison between 10 PDA equivalent and 10 client (first approach)

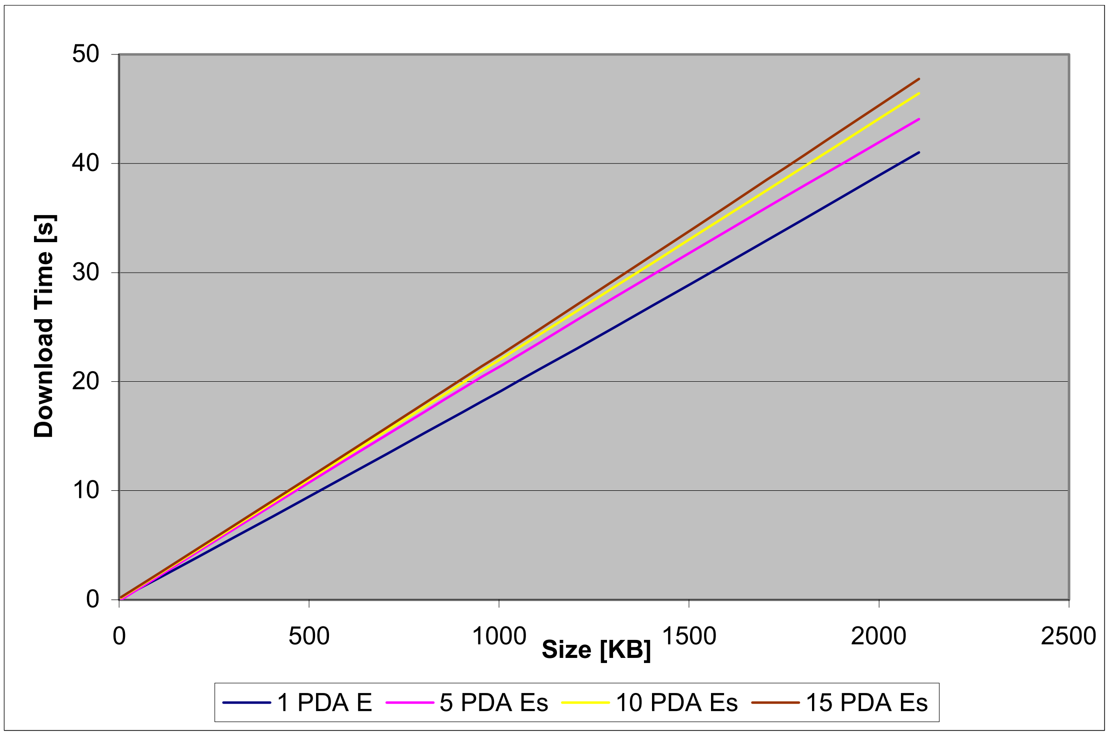


Figure 6.39 Comparison among 1, 5, 10 and 15 PDA equivalent

6.2.6.3 Second Approach: M^{2}G with Laptop UE

The plots generated by the monitored data resulting from a user equivalent running on a Laptop are reported here. Today, modern Laptops have the same hardware characteristics of Desktops when the level of battery is good. So, it is not necessary to reduce the performances of a Desktop (to simulate a Laptop) if the hypotheses claim that the level of battery is good. Therefore, in this case, it is important only to use the appropriate file size distribution and the appropriate wireless models. To compare the results of these trials with the previous ones we use the same file size distribution. The reason is that this file size distribution may also reproduce the sizes of file coming from services like distribution of songs and distributed karaoke. To adopt the appropriate wireless (communication) models (for Laptop) it is necessary to specify during the configuration phase of Interceptor. In the next Figures (6.40-6.41), functions interpolating the resulting data are reported. In particular, these Figures show the download time by increasing the size of the downloading file when a fixed number of UEs run on a Laptop. The curves with 1 and 5 UEs have a linear slope while that with 10 UEs (see the Figure 6.40) is quadratic. The first two curves show, more or less, the same downloading time for files of the same sizes. Instead, the curve with 10 UEs is not so close to the previous ones, showing a high variability of the data (for example a file of 400 KB is downloaded by an UE in 8 seconds and by another in 43 seconds). The UEs running on a Laptop are able to request and download files speeder than the PDA equivalents. Therefore, the behavior of these UEs results more stressful for the IS, so its performance decreases more quickly when the number of UEs increases (see Figure 6.40). In particular, the curve with 15 UEs shows better performance than 10 UEs (see Figure 6.41).

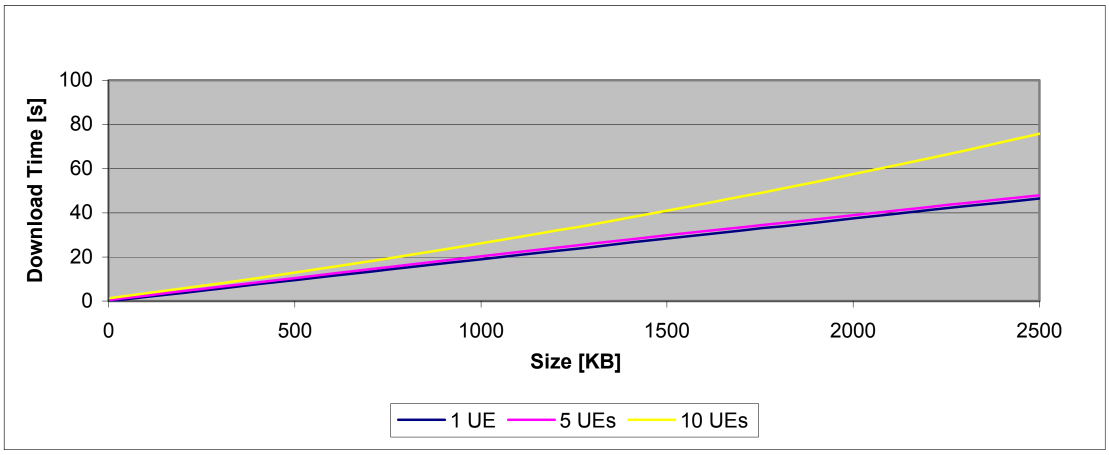


Figure 6.40 Comparison among 1, 5 and 10 Laptop UEs

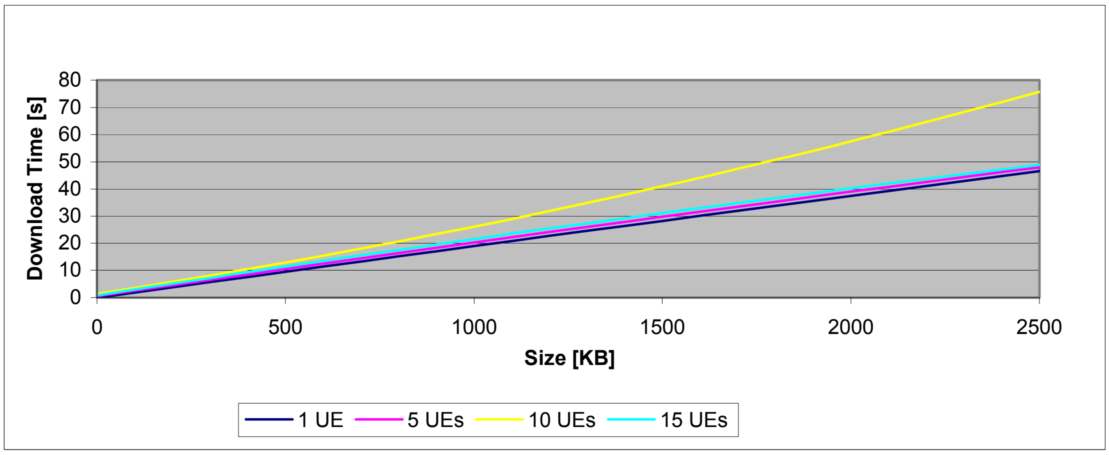


Figure 6.41 Comparison among 1, 5, 10 and 15 Laptop UEs

The log files show that a lot of UEs were not able to open or to conclude their session (look at the Figure 6.41: the number of the active UEs was between 5 and 10; in particular, closer to 5). Concluding, the downloading time increases with both the number of the UEs and the size of requested file. For example, with a file of 1.5 MB there is no trouble in the case with 5 UEs, but with 10 UEs some of them were enable to download. Finally, (where is possible) the curves of M^{2}G with Laptop and with PDA UE are compared. The behavior of both cases is similar:

- the downloading time increases with the size of the requested files showing a linear slope,
- the downloading time increases with the number of the UEs, and finally
- after a certain threshold the downloading has critical problems because the IS is not able to provide correctly the delivery of its multimedia service.

The main difference is the value of the critical threshold that is lower in the Laptop case (10 VS. 20). Therefore, it is possible to support 10 users adopting a Laptop as final device, while it is possible to support 20 users adopting a PDA (with lesser performance for each user!). The results of the comparison involve two main considerations: the wired network is the bottleneck, and the higher is the speed on the wireless network, lower the critical threshold. This synthetic emulative platform reproduces either the delay due to the wired networks (from the IS to the replica servers and vice versa) or that due to wireless networks (from IS to final devices and vice versa). In particular, the model of the wireless device for the Laptop is about four times faster than that for the PDA (due to the better performance of the Laptop: better CPUs, buses, etc.). In this scenario, the wired network is the bottleneck of the entire service. In fact, the wired downloading time (i.e. the time for downloading a file from replica servers to IS) is the same in both cases because they adopt the same communication model. This explains because the downloading times calculated on the final Laptops and PDAs depend only on the wireless communication models. Furthermore, the higher the speed on the wireless network, the lower the critical threshold. In fact, if the transmitting times are lower, the number of interarrival requests is higher. Then, increasing the number of UEs means increasing the number of these interarrival requests. Further, IS must run a session manager for each UE (to supply the multimedia service) and this action has a cost. If these UEs do not receive data, saturate the capability of IS and go in timeout. After this moment, IS refuses the new connections. Then, decreasing the wireless downloading time (i.e. the time for downloading a file from the IS to the final device) means increasing the interarrival requests. This explains the fact that Laptop case reaches with a lower number of UEs more quickly the critical threshold.

6.3 A System Architecture to Support the Execution of Agent-Based Participatory Simulation Activities

In this last Paragraph, we show how it is possible to integrate agent-based participatory simulation activities within Interceptor. The use of participatory simulation based on multi-agent models to mimic and control complex systems is becoming increasingly common. The users of these collaborative systems demand tools that are able to interact with the remote environment from any-device and to present visually the results of these cooperative simulation activities on their screens. Furthermore, they want to connect the remote environment from any-where using their fixed or mobile device and to play these cooperative simulation activities any-times. It is possible to exploit the integration of the wireless communication connectivity and the Internet, and instructing their agents to play also if the user is not online. As Network Topology Manager of Interceptor, an agent-based model simulative platform is integrated. Then, it is possible to use it either to manage the topology of complex computer networks or to model environment for participatory simulation activities. In this Paragraph, we want to demonstrate that Interceptor could be useful to emulate the communication delay of the interactions of the participants with the remote environment. Differently from the Paragraph 6.2, where the multimedia delivery service passes transparently through Interceptor, in this case, the remote environment is integrated in Interceptor. This means that the service end-points are inside Interceptor. By means of these considerations, we want to demonstrate that Interceptor can be useful to build participatory simulative scenarios inside a Laboratory.

Along with this idea, we show a software prototype of architecture implementing an agent-based participatory simulation, built on the Netlogo platform (Cacciaguerra *et al.*, 2004). Hence, we explain how is possible to build agent-based participatory simulation activities within Interceptor substituting the Netlogo platform for SPADES middleware.

6.3.1 Introduction

The wireless revolution has started with Internet phones and continued with many kinds of wireless hand devices that allow users to access the Internet. Thanks to the technical developments in high-speed chips, mobile networks and software protocols, the wireless technology is enabling a wide range of exciting possibilities, including, for example, wireless sensors networks, wearable computers, ubiquitous computing and innovative use of Web phones. In this context, it is easy to envisage that future mobile users will enjoy a near ubiquitous access to the vast storehouse of technical and intellectual resources offered by high bandwidth (wireless) networks. For example, in social organizations, or in collaborative human environments, workers are starting to exploit wireless technologies to connect to colleagues and carry out different kinds of cooperative tasks, including brainstorming, task planning, resource sharing, instant messaging, and waving the Internet together. In essence, due to the use of these new technologies, secure virtual spaces (or environments) may be created where workers, after been identified, may cooperate to accomplish common tasks. In this challenging scenario, a great popularity has been gained by a modern form of "semi-automatic" collaborative scheme, named participatory simulation (Kanter, 2003). With the term participatory simulation Wilensky and Stroup (Wilensky *et al.*, 2000) refer to such role-playing cooperative activities aimed at exploring how complex dynamic systems evolve over time. As an example of participatory simulation consider that of a virtual stock exchange, where each player could play the role of a virtual buyer or of a seller who engages in the activities of the resulting share exchange dynamics. Obviously, we have just mentioned only a simple example, but a wide set of possible content areas for participatory simulation include different scientific and technical fields, ranging from the spread of a disease, to the flow of energy in an electric network, to the diffusion of innovation, to the distribution of goods in an inventory system (Lomi *et al.*, 1999). From a scientific standpoint, participatory simulation typically employs some form of Agent Based Modeling Simulation (ABMS) technology. Simply put, an ABMS platform is a programmable modeling environment for simulating complex systems where programmers can give instructions to several independent agents working in parallel. In essence, in an ABMS the global state of the system emerges as a result of the interaction of hundreds, or thousands, of elementary agents engaged in a variety of local processes such as exchange, cooperation and competition. These agents (which may be either completely controlled by humans or automatically programmed) can play their moves based on an intrinsic capability of local investigation and local action. Perhaps the most common way to present the simulation results produced by the complex interactions of an ABMS-based virtual world is visual display. A prominent example of using visual representation to display the results of ABMS-based simulation is reported in Figure 6.42, where we show the typical graphical interface of these platforms by means of three different models. The problem here is that the actual ABMS-based software platforms only provide for limited graphical functionalities. For example, well known ABMS platforms such as Netlogo, Swarm, Jas, and Repast only render their agent based models through a 2D raster graphics visualization methodology (Netlogo, 1999; Swarm, 1999; JAS, 2002). In this context, the main contribution of our work is the design of a distributed architecture that allows remotely to control your agents inside the ABMS-based virtual environments and to represent visually, by means of a 3D visualization engine, simulation results of these virtual environments on mobile devices, such as laptops, PDAs and smart phones. From a graphical standpoint, it is worth mentioning that, as typical 3D rendering problems are here exacerbated by the need to display virtual worlds on wireless (possibly handheld) devices, we resorted to special rendering techniques based on triangular meshes that guarantee an optimal trade off between fast visual reproduction and device compatibility. Alongside a detailed description of the architecture of our remote control and visualization engine, we report a set of experimental results which confirm that an appropriate integration of the remote control and the visualizer with the software architecture of the ABMS system enables a fast 3D representation on wireless devices.

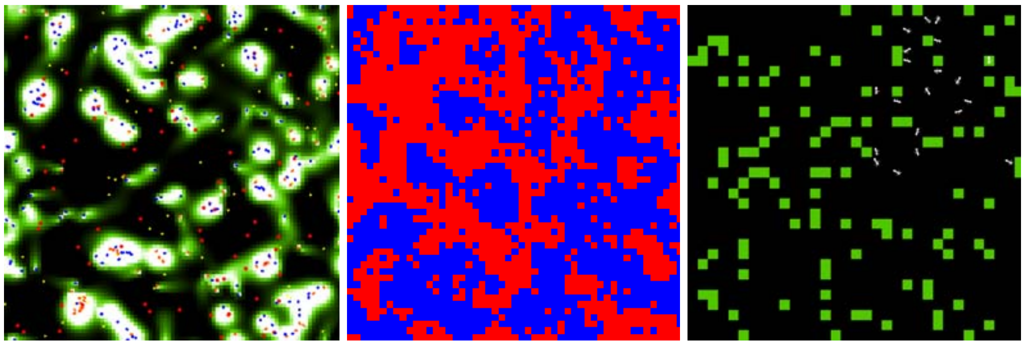


Figure 6.42 The 2D typical graphical interface of ABMS

This study is organized as follows. In Paragraph 6.3.2, we illustrate the main features of the software architecture (i.e. a prototype based on Netlogo platform) of the agent-based participatory simulation we designed. In Paragraph 6.3.3, we present a set of empirical results we obtained with a prototype implementation of our software architecture. Again, Paragraph 6.3.4 presents an example of modeling a complex system. Furthermore, in Paragraph 6.3.5, we show how is possible to build participatory simulative scenarios within Interceptor emulating the participation of users geographically dispersed in different country. Finally, the Paragraph 6.3.6 concludes our work with some hints for future developments of our work.

6.3.2 System architecture

We have designed a (client-server) software architecture able to support the execution of ABMS-based virtual worlds and their 3D rendering on wireless devices (see Figures 6.43-6.44). The three main software components of our architecture are the following: i) the ABMS platform, ii) the 3D visualizer and, iii) the wireless network communication subsystem. As shown in Figure 6.43, a complete "execute and visualize" session of our system works as follows. Initially, a user from his/her device issues an order to his/her set of controlled agents in a virtual world. This order is intercepted by a dedicated user process, termed Agent Manager (AM). After collecting orders from a given user, the AM sends them to its software counterpart on the ABMS platform, called the Request Manager (RM). Apart from the RM, the ABMS platform (hosted on a wired machine) is comprised of a State Updater (SU) and a Snapshot Creator (SC). In essence, the SU computes, on a periodical basis, a new state of the virtual world, based on the interactions with the system users. After that a new state has been computed, the SC constructs a text-based image of the newly calculated state. Upon reception of these data, the visualizer displays them on the screen of the wireless device. It is worth pointing out that all the above-mentioned communications are carried out by the wireless network communication subsystem based on a TCP/IP stack.

The AMBS platform, on the top of which our system is built, is partially based on the Netlogo software environment (Netlogo, 1999). In particular, the RM and the SU are technologies provided by Netlogo; instead, we developed a Snapshot Creator (SC) that is able to capture the states of the system generated by Netlogo, and to transform them into input parameters for our visualizer. As to the functions provided by Netlogo, it is important to notice that all the most complex models produced by Netlogo may be built based on three different types of agents, namely: observers, turtles and patches. In substance, each model has only one observer that represents the most general framework where all other agents live and cooperate. Turtles, in turn, are the most active types of agents which can be defined in a Netlogo model: they can perform several types of actions, on a local basis, ranging from spatial movements, to visual interpretation of other agents actions, and generic data exchange.

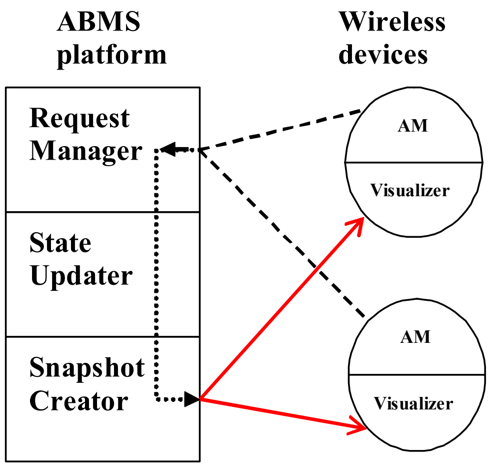


Figure 6.43 An “execute and visualize” session

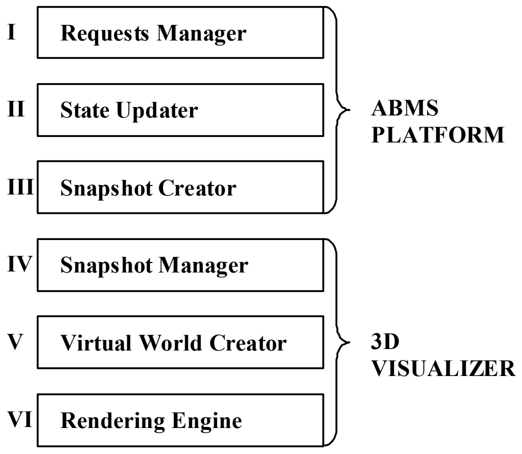


Figure 6.44 System architecture

Finally, patches are agents, which are typically used, within the Netlogo framework, to represent static pieces of information, such as background colors and spatial landscapes, for example. Patches may also represent "resources" that turtles produce and/or consume. Hence, within Netlogo, agents are able to perceive their environment and respond to changes in a timely fashion, further they are able to interact with each other to perform cooperative activities. Our contribution to the Netlogo platform, here, has been that of developing a software modul, which captures, on a periodical basis, all the data which refer to a given model and inputs them into a FIFO queue. Subsequently, following the order given by the queue, the data are fetched and sent to the visualizer for 3D rendering, as described in the following Subsection.

The main obstacles that need to be tackled for the development of the visual engine for an ABMS system concern the choice of the most appropriate structuring of the hardware/software architecture. This architecture must be able to render visually on a wireless device the states of the 3D virtual worlds generated by the simulative platform. To this aim, we have developed a visual engine, based on open source graphical libraries, whose architecture is depicted in Figure 6.44 The main software components of our 3D visualizer are the Snapshot Manager (SM), the Virtual World Creator (VWC) and the Rendering Engine (RE). Upon receiving data into a FIFO queue from the ABMS through a wireless connection, the SM manages the subsequent activity of data decompression, and checks for their integrity. In turn, the VWC extracts data from the FIFO queue and creates the 3D virtual world based on a scene graph model which exploits triangular meshes as basic 3D objects (Clarke Wilson, 1994; Bethel *et al.*, 2002; Clark, 1976). This is accomplished by following an augmented reality strategy, where real images, captured with a camera, can be attached to the virtual world generated by Netlogo. The final activity is performed by the RE which renders in 3D on the wireless screen the virtual world generated by the VWC. It is important to mention that our RE is able to support the introduction of graphical optimizations (such as the use of textures) without affecting the rendered data. In addition, it has the capability to sustain dynamically the display of a given frame rate (on the wireless device) while scaling down with respect to the graphical quality of the rendered 3D objects. Further, this software module allows for frame skipping, when a too large transmission delay is experienced at the client side. Finally, it is worth mentioning that our RE displays the virtual world generated by the VWC based on the OpenRM/OpenGL graphical libraries (Bethel,1999 ;Segal *et al.*, 2002), the main advantage of this graphical library being its portability across different operating systems. (Thus, our visualizer may run on different hardware equipments, such as laptops, tablet PCs and PDAs.)

6.3.3 Empirical results

To test the efficacy of this architecture, we developed an experimental study based on the use of the following prototype implementation of our system. We run the Netlogo-based ABMS platform (RM+SU+SC) on a server equipped with a Windows 2000 Pentium III, working at 900 MHz and with 512 MB RAM. The client, instead, was running on either a laptop machine (a Dell Inspiron 8200 equipped with a NVIDIA GeForce4 Go 440 with 64Mb of RAM) or on a PDA (iPAQ 3970). The laptop mounted a Windows 2000 OS, while the PDA mounted the Familiar Linux OS equipped with the GPE x-Windows graphical interface (Familiar, 2003; GPE, 2003). Both clients were connected to the server through a TCP/IP wireless connection on a Wi-Fi 802.11b network.

Now we wish to illustrate the results we obtained both on a visual and on a numerical standpoint. To this aim, in Figure 6.45 we present the 3D results we have obtained by rendering, on the wireless device, the virtual model termed “Climb-the-Hill” produced by the Netlogo engine. Several considerations are in order here. First, we wish to point out that our 3D models are rendered dynamically, following the evolution of the simulative models produced by Netlogo. In the example above, turtles are rendered, in a timely fashion that climbs the hill, looking for the highest patch in their neighborhoods (local maximum).

Second, it is easy to understand that the observer may manipulate dynamically the virtual world, for example by changing the perspective under which the world is considered, or by zooming on a detail, or by highlighting (using wireframe techniques) the lattice over which the turtles play their moves. This final characteristic is of particular interest when the observer wants to verify if the visual representation rendered by our 3D engine matches the underlying numerical data. Based on the consideration that our 3D engine guarantees that the correctness of the numerical data is maintained in the visual representation, it is easy to understand that local visual manipulations of the virtual world are made possible, without affecting the integrity of the performed operations. As a final consideration, with respect to the discussion above, we wish to encourage the reader to note that a great visual difference exists between the graphical 3D representation, which our system is able to produce and the standard 2D graphical representation generated by the Netlogo platform shown in Figure 6.46 as an alternative display strategy.

Besides the visual results, it is also important to provide quantitative measurements, which capture the performance of our designed system. To this aim we carried out two different sets of experiments. The former set (30 experiments) refers to the ability of the ABMS platform to update the virtual state of the virtual world, as a function of the number of agents. In Figure 6.47, we plotted the average number of states, on a logarithmic scale, that our ABMS platform is able to generate per each second, depending on the number of agents involved in the simulation. In particular, the lower curve accounts for the performance of the ABMS system including the data export activities performed by the Snapshot Creator (SC) we have developed. It is easy to assume from an analysis of Figure 6.47 that the larger the number of agents, the lower the number of the states of the virtual world that can be updated per each second. For example, Figure 6.47 shows that with a thousand of cooperating agents the ABMS platform is able to provide state updates at the very low frequency of 1.5 states per second.

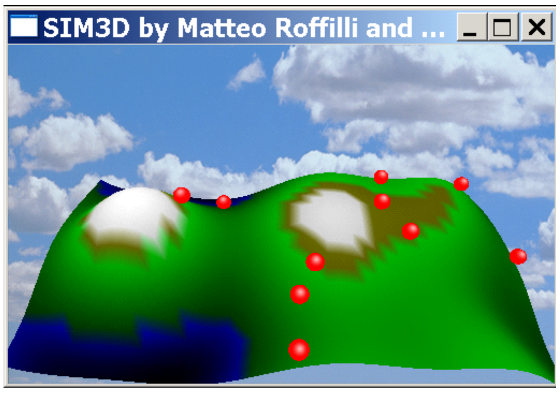
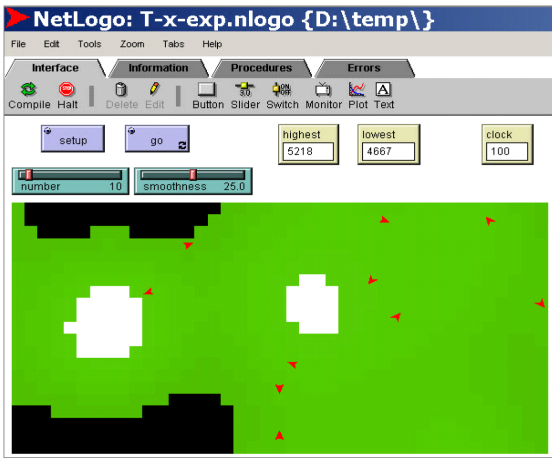


Figure 6.45 A 3D visual representation produced by our visualizer


NetLogo: T-x-exp.nlogo {D:\temp\}
File Edit Tools Zoom Tabs Help
Interface Information Procedures Errors
Compile Halt Delete Edit Button Slider Switch Monitor Plot Text
setup
go

highest 5218
lowest 4667
clock 100
number 10
smoothness 25.0

Figure 6.46 A Netlogo-generated visual representation of the same world of Figure 6.45

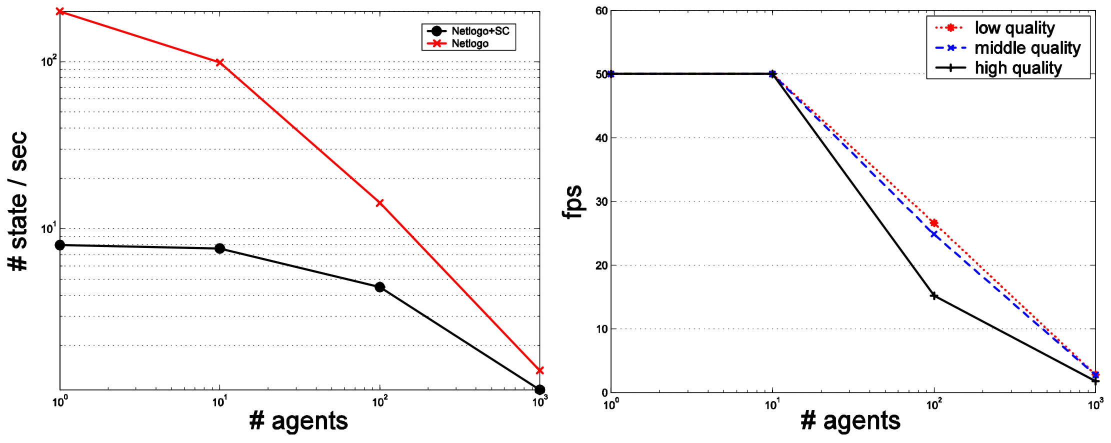


Figure 6.47 Performances: left) of the ABMS (server-side) and right) of the visualizer (wireless device)

The latter set of experiments aims at highlighting the number of frames per second that may be displayed by our visualizer on the wireless client device, yet again depending on the number of agents involved and on the quality of the 3D rendered graphical objects. In particular, in Figure 6.47 right, the number of the frames per second are plotted which were obtained, on average, over thirty different experiments carried out on wireless devices, as a function of the number of agents involved in the simulation. Precisely, three different curves are plotted, denoted as low, middle and high, respectively. The curve denoted as low represents a situation where the quality of the rendered image was obtained with 8 triangular-meshed faces per each graphical object. The quality of the middle curve was obtained by exploiting 32 faces per each rendered object. The quality of the high curve was, instead, obtained with 128 faces. As shown in the Figure, the larger the number of agents, the lower the number of frames which may be created and visualized per each second. Further, the higher the graphical quality of the rendered 3D object, the lower the number of frames that can be displayed per each second. From a comparative analysis of Figures 6.47 left and right, it is easy to understand that our visualizer has been tuned to support a rate of rendered frames, which is attuned with the frequency according to which the ABMS platform provides updates of the virtual world state. In other words, from a communication viewpoint it is sufficient for our wireless communication subsystem to be able to transport, over the air, per each second the same number of state updates produced by the ABMS platform. This would guarantee a perfect synchronization between the producer (the ABMS platform) and the consumer (the visualizer). Following this consideration, our experiments with an iPAQ 3970 PDA, equipped with the pocketGL graphical library, show that an average rate of 10-15 frames per second may be safely tolerated over a Wi-Fi connection with a hundred of agents.

6.3.4 Modeling a complex system: organizational decision chemistry on a lattice

The software architecture, we have designed, supports the modeling of different complex systems simulating their components by means of different types of agents. In this way, it is possible to reproduce the behavior of each complex system by writing appropriate code for each kind of agent of this software architecture. Therefore, to prove the usefulness of this architecture, the usefulness of modeling complex systems with multi-agent platforms must be highlighted. Along with this consideration, a concrete example exploring the original insight of the "Garbage Can Model" of organizational choice (GCM) proposed by M. Cohen, J. March, and J. Olsen (1972) is shown. From this viewpoint, we realize a simple multi-agent model that explores organizational decision processes with NetLogo (Lomi *et al.*, 2003a-b). This Paragraph shows how it is possible to implement the "Garbage Can Model", which are its features and a series of experimental trails obtained by varying the initial settings and the value of the characteristics of the agents.

6.3.4.1 Introduction

Social organization can be studied at many different levels of abstraction and analysis. For example, in the analysis of organizational decision making processes a common strategy is to reduce a complex social activity to an individual constrained optimization problem. Understandably, this analytical strategy emphasizes the information properties of alternative organizational arrangements (Burton *et al.*, 1995) and the capacities of decision makers (individually and in teams of variable size) to find, assemble, exchange and process the various pieces of information that formal organizations produce and make available to its participants (Miller, 2001). Typically, organizational decision processes are assumed to follow a "logic of consequence" according to which the outcomes of decisions are evaluated in terms of personal preferences (March, 1994). According to consequence framework, a number of coherence assumptions are needed for inferring from the observable consequences of decisions, an implicit decision rule such that the organization as a whole can be said to behave as if it were following that rule (McFadden, 1976).

An alternative way of thinking about how decisions happen in organizations, concentrates on the aggregate flows of problems, solutions, opportunities and decision makers through organizational networks (Padgett, 1980). This perspective focuses on how aggregate regularities are produced and reproduced through the interaction of elementary components ("agents") defined at lower levels. Such a view starts with a very different position on what the term "organization" means when referred to socially constructed entities. Organizations are seen as regulated and partly self-maintaining flows emerging from the interaction of elementary agents. Organization "structure" is patterns in these flows that may take the form, for example, of routines (i.e. systematic connections between actual or potential "problems" and "solutions"). Within the broadly defined field of "Organization Science," a prominent example of this second strategy to make sense of organizational decision processes is the Garbage Can Model of Organizational Choice (GCM), originally proposed by M.D. Cohen, J.G. March and J.P. Olsen (1972). The motivating claim behind the GCM is that: "[A]lthough organizations can often be viewed conveniently as vehicles for solving well-defined problems […] they also provide sets of procedures through which participants arrive at an interpretation of what they are doing and what they have done while in the process of doing it. From this point of view, an organization is a collection of choices looking for problems, issues and feelings looking for decision situations in which they might be aired, solutions looking for issues to which they might be the answer, and decision makers looking for work" (Cohen *et al.*, 1972). Our effort is based on the intuition that (within the original GCM) the representation of "organizational decisions" as outcomes of quasi-random collisions among "participants," "problems," and "solutions" invites development of a more explicit bio-chemical metaphor for organizational decision process (Fontana *et al.*, 1994). In this view, new constituents of an organizational system are generated from within by the interaction of elementary "agents" following different laws of composition (Padgett *et al.*, 2003). The present work is not the only or the first attempt to reconstruct the GCM. Other examples are available that have emphasized and developed different aspects of the original model (Masuch *et al.*, 1989; Warglien *et al.*, 1995). Since its appearance, the implications of the GCM have also been explored in a number of empirical contexts (Carley, 1986; Cohen *et al.*, 1974; Levitt *et al.*, 1989; March *et al.*, 1976; March *et al.*, 1986) and theoretical elaborations (March, 1978). Differently from the other examples, at the heart of our representation is a co-evolutionary view of organizations whereby "participants," "problems" and "solutions" have to interact to produce decisions, but at the same time are transformed through and by the interaction process that they activate.

6.3.4.2 Model issues

In this model, organizations are viewed as crossroads of time-dependent flows of four distinct classes of entities: “problems”, “solutions”, “participants” and “opportunities”. Collisions among the different entities generate events called “decisions”. In our NetLogo-based reconstruction of the GCM, the type of decision is determined by the relative levels of energy accumulated by “participants” and “opportunities” up to the moment of collision. We make no attempt to reproduce all the features of the original model. Some features of our representation are not present in the original model and are included to cast new light on specific aspects of decision processes in organizations. Our model is a highly preliminary attempt to represent organizational decision processes when agents live in a structured socio-physical space, and are capable of reproducing, i.e. of creating identical copies of them. In its current state of development the model serves mainly didactic purposes and as an illustration of the value of the NetLogo programming environment for representing complex organizational systems. The general learning point that the model can be used to illustrate is that dynamic complexity at the organizational level does not depend on complexity at level of individual agents. We claim that there is a conceptual link between aspects of social organizations and the study of emergence in natural and artificial systems. Furthermore, we realize a module to represent and explore a variety of theoretically grounded mechanisms that may be behind the emergence of organizational routines. For this purpose we define routines as relatively stable self-reproducing communities of “problems,” “solutions,” and “participants”; the basic constituent elements of organizations. In our model, these communities of entities emerge from decentralized activities of opportunity seeking and problem solving defined at the level of individual agents. In this way, we try to extend the basic GCM representation by bringing to light, and developing more clearly, its more constructive features that are implicit in the tendency of the basic constituent elements of organizations to (i) reproduce by creating copies of themselves; (ii) give rise to new entities through interaction, and (iii) impose order in their environment through the routinization of problem solving activities. The main goal of our modeling exercise is to suggest one possible way in which individual task-oriented activities performed by agents endowed only with local communication and sensory capacities may give rise to regular patterns of organization at more aggregate levels.

6.3.4.3 Implementation of the model

Here below, we present the logics behind the implementation of our model. We start by describing the basic lattice representation of our agent-oriented organizational structure, and the characteristics of the entities that populate our simplified organizational world. Then, we describe the collision and transformation rules that regulate the interaction among entities in the lattice. Finally we discuss the main characteristics of the entities in terms of their communication capacities, and sensory and information processing mechanisms. This section introduces the notion of organizations as ecologies of entities. To build an effective connection between this image of organizations and computational models we need to define: i) the basic structure of the organizational environment, ii) the rules of interaction and transformation among the various entities populating this environment, and iii) The sensory and information processing mechanisms of the entities. We discuss these issues in turn (here below).

Consider *the stylized structure of the organizational environment* (called world) inhabited by three types of entities called “Problems”, “Solutions” and “Participants”. A fourth type of object called “Choice Opportunities” - or simply “Opportunities” - emerges from the collision among individual “Problems” and “Solutions”. Suppose that each object is randomly allocated to sites in a two dimensional lattice with periodic boundary conditions (see the Figure 6.48). Assume that entities are endowed with an initial level of energy. During their life on the lattice, entities dissipate energy by moving around. Entities move following a random walk through the lattice. An entity expires when its energy level is zero. Entities can also increase and exchange their energy levels through collisions. When a maximum (exogenously set) level of energy is reached, participants and opportunities reproduce by splitting, i.e., by forming identical copies of themselves.

In keeping with the original formulation of the model, both participants (“decision makers”) and choice opportunities are characterized in terms of energy levels. Energy exchange among the different types of entities is regulated by *the rules of interaction and* *transformation* (called collision rules) described below. Collisions between "Participants" and "Opportunities" give rise to three possible outcomes, each inducing specific types of "decisions."

- If (Energy-of-Participant > Energy-of-Opportunity), then the participant destroys the opportunity. As a consequence the opportunity disappears and the participant increases its own energy level (new-energy-of-participant = old-energy-of-participant + energy-of-opportunity). The outcome of this collision is called decision by "Resolution."

- If (Energy-of-Participant < Energy-of-Opportunity), then the opportunity destroys the participant. As a consequence the participant disappears and the opportunity increases its own energy level (new-energy-of-opportunity = energy-of-participant + old-energy-of-opportunity). The outcome of this collision is called decision by "Flight."

- If - within a given range - (Energy-of-Participant = Energy-of-Opportunity), then the opportunity separates into its component elements: a problem and a solution. In this case the "problem" and the "solution" retain the energy level of the opportunity. The energy level of the participant remains unchanged. The outcome of this collision is called decision by "Oversight."

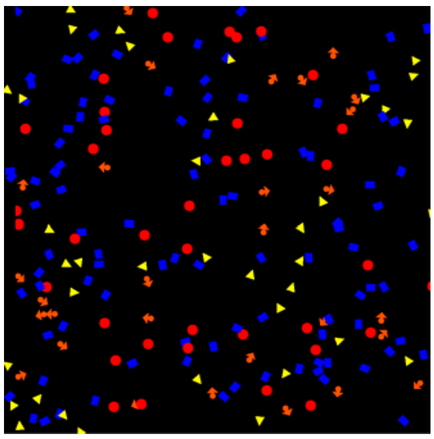


Figure 6.48. Initial distribution of entities on the lattice

In our model, mobility and task-oriented behavior of the entities are controlled by *the sensory and information processing mechanisms*. In particular, they are implemented by three parameters defining (i) local communication capacity (or “coverage area”), (ii) search capacity (or “vision”) and (iii) sensitivity to information (or “attention”). We assume that entities communicate by leaving information trails that other entities might be able to detect and follow by means of theirs sensors. This simple biologically-inspired communication mechanism has a strictly local character (i.e. it occurs only in a limited neighborhood of sites defined around the entities). The patches (or “sites”) visited by the various entities accumulate and dissipate information, thus the shape and strength of information trails change continuously as the entities navigate their simplified organizational world. As Figure 6.49 shows, the different intensity of information trails is represented in the model by a color scheme according to which black patches contain no information and white patches signal the presence of the strongest information trails. The various shade of green represent intermediate situations, with the information content of the patches increasing from dark green to light green patches. Entities are endowed with a generic sensory (information-detection) mechanism and a visual mechanism that allows them to follow the gradient of information intensity stored in the patches within their range of vision.

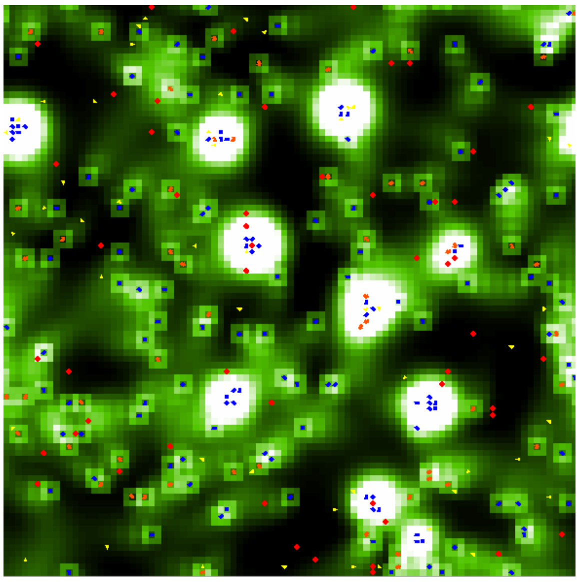


Figure 6.49: Communication trails

Table 6.11: Rules of movement

```
to move
    if else (information > detect-threshold)                        ;1
    [turn-toward-information]                                       ;2
    [ rt random rotate-angle - random rotate-angle + rotate-bias]   ;3
    fd 1                                                            ;4
End
```

The sensory mechanism is based on a threshold function: entities can only detect information above a minimum (exogenously set) threshold. This implies that (above this threshold) entities are attracted to patches with higher levels of information. This process of local optimization is used to guide the entities' search behavior. When entities do not sense the presence of information in their vicinity their movement follows a random walk through the lattice. The visual mechanism is defined in terms of the angle of vision (in degrees) within which an entity can actually detect information. Thus, the movement of entities on the lattice is driven by the distribution of information across patches that are within their angle of vision as illustrated by the pseudo-code reproduced in Table 11.

In words, if an entity reaches a patch (or a "site") holding sufficient information (i.e., an amount that is greater than the threshold), the entity uses its search capacity defined above to choose the best direction (see instruction in rows 1 in Table 11). The best direction is that toward the greater quantity of information (see instruction in row 2). If the entity is positioned on a patch that does not contain sufficient information (i.e., it is below the information detection threshold), the entity chooses a random direction driven by the Rotate-Angle and Rotate-Bias parameters (see instruction in rows 3). The Rotate-Angle is the maximum amount, in degrees, that an entity can turn left or right in its random movements. When Rotate-Angle is set to zero, an entity will hold its direction constant until it finds an information gradient to follow. The Rotate-Bias is the bias of an entity's average rotation. When Rotate-Bias = 0, on average the entity will move straight ahead. When Rotate-Bias > 0, the turtle will tend (on average) to rotate to its right. When Rotate-Bias < 0, the turtle will tend (on average) to move to its left. Finally, when a direction is chosen, the agent moves one step in that direction (instruction in row 4). The relatively simple, biologically-inspired communication and sensory mechanisms that we have defined are sufficient to induce complex clustering behavior. As Figure 6.50 illustrates, the system evolves over time from a disordered initial state toward a state clearly characterized by a higher degree of spatial organization and temporal stability. In the next section we conduct a series of computational experiments to learn more about the emergence of these patterns of self-organization.

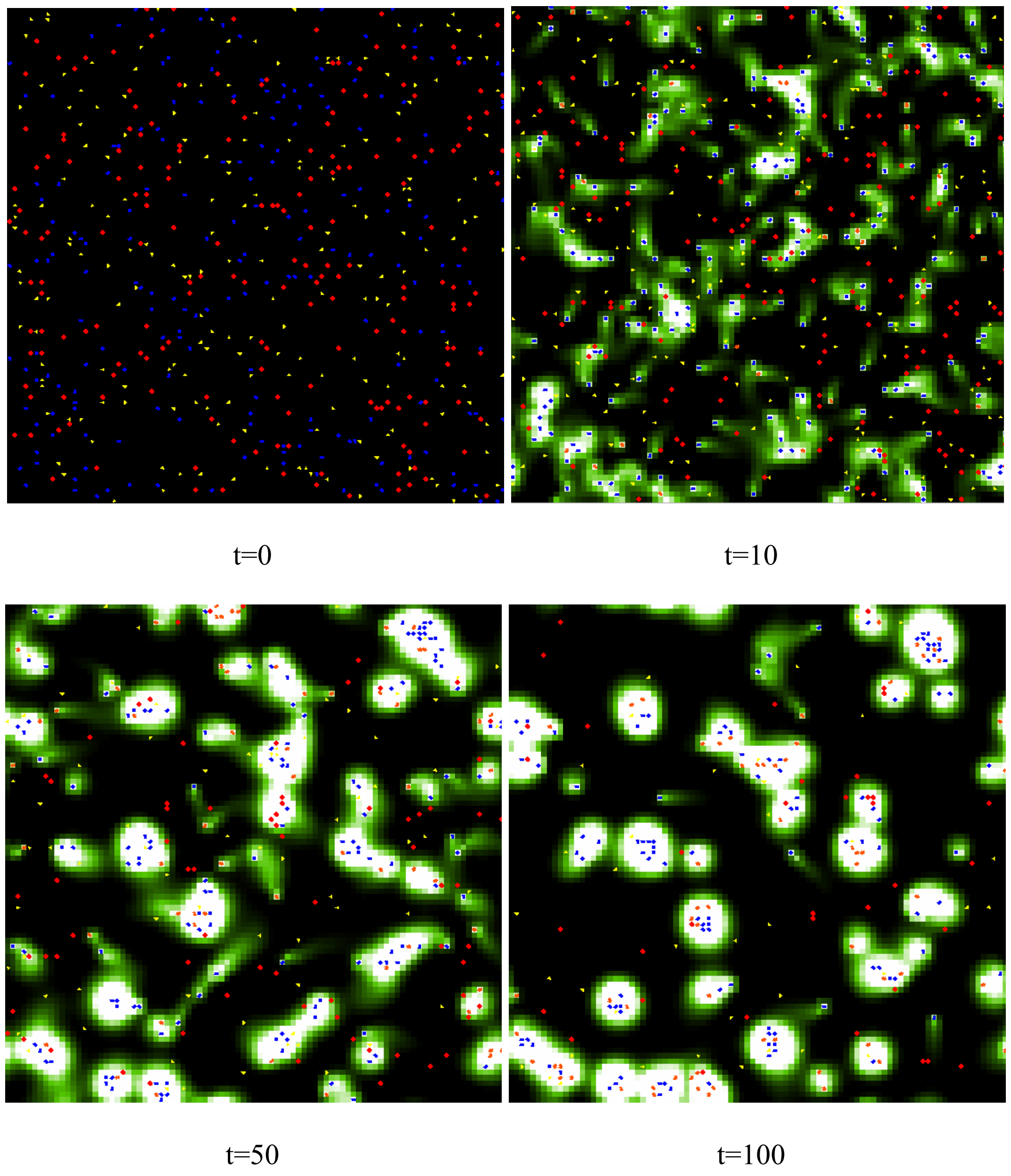
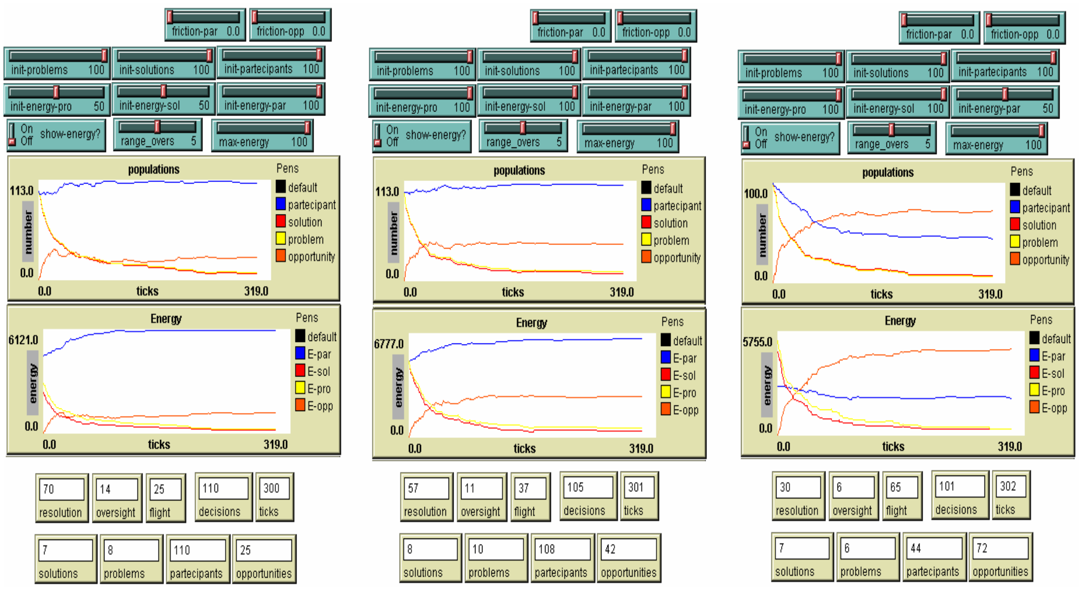


Figure 6.50. Space-temporal evolution of the system (examples)

6.3.4.3 Experimental campaign

Three different kinds of trails have been conducted to explore the behavior of our model. The first series of these trails show the typical behaviors of the model starting from baseline runs. The second series explore the routinization either with visual or with numerical data. Finally, the third series explain the emergence of different system behavior driven by the angle of vision whether with visual or with numerical data.

Figures 6.51 a), b) and c) report *the results of baseline runs*. Above each (time series) plot the relevant sliders show the setting up of the parameters of the model. Further, Figures 6.51 also report various monitors that provide final numerical information. For example, the INIT-ENERGY sliders in Figure 6.51 a) imply that most of the participants are endowed with an initial energy level that is greater than that of solutions and problems. Because the energy of an opportunity is computed as the average of the energy of its parent entities (problems and solutions) in this configuration the energy of participants will tend to be higher than the energy of opportunities. Thus when a participant and an opportunity collide, the "decision style" prevalent in the system is likely to be "by resolution." As the various monitors illustrate, after 300 time periods ('ticks'), the number of decisions "by resolution" (70) is more than twice the number of "decisions by flight" (25), and five times the number of "decisions by oversight" (14). In the second baseline run illustrated in Figure 6.51 b), the energy level is likely be the same across all types of entities. In this case, when participants and opportunities collide, we have little prior information to guide our prediction about the decision style that will be prevalent in the system. After 301 periods (or "ticks"), the number of decisions "by resolution" (57) is again somewhat greater than the amount of flights (37), and is five times greater than the number of decisions "by oversight" (14). Number of participants, problems and solutions in case 4 (b) are comparable to the corresponding numbers in of Figure 6.51 a), but the number of opportunities in the system has almost doubled (42 vs. 25). Decisions "by resolution" are less frequent in this system (70 vs. 57) and the number of decision "by flight" is higher (37 vs. 25). Finally in the situation illustrated in Figure 6.51 c) it is likely that the most of solutions and problems will be endowed with a level of energy that is higher than that of participants.



a)
friction-par 0.0
friction-opp 0.0
init-problems 100
init-solutions 100
init-partecipants 100
init-energy-pro 50
init-energy-sol 50
init-energy-par 100
show-energy? Off
range_overs 5
max-energy 100
70 resolution
14 oversight
25 flight
110 decisions
300 ticks
7 solutions
8 problems
110 partecipants
25 opportunities

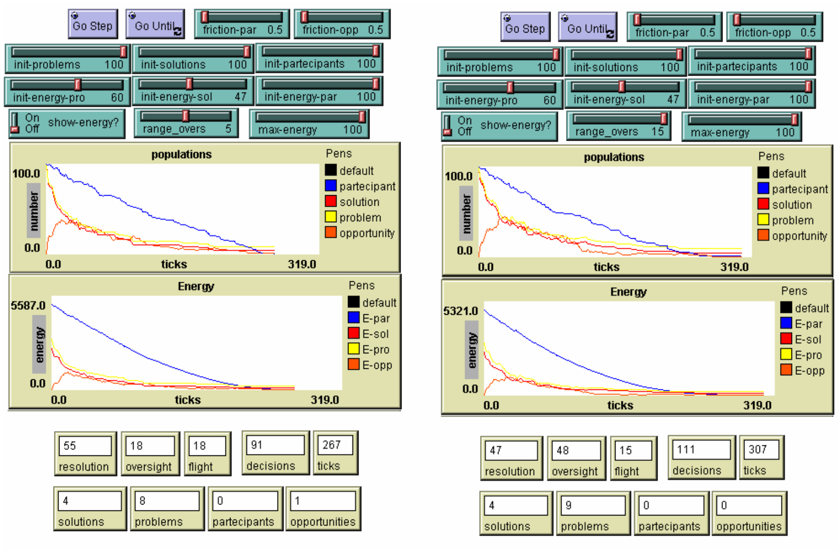
b)
friction-par 0.0
friction-opp 0.0
init-problems 100
init-solutions 100
init-partecipants 100
init-energy-pro 100
init-energy-sol 100
init-energy-par 100
show-energy? Off
range_overs 5
max-energy 100
57 resolution
11 oversight
37 flight
105 decisions
301 ticks
8 solutions
10 problems
108 partecipants
42 opportunities

c)
friction-par 0.0
friction-opp 0.0
init-problems 100
init-solutions 100
init-partecipants 100
init-energy-pro 100
init-energy-sol 100
init-energy-par 50
show-energy? Off
range_overs 5
max-energy 100
30 resolution
6 oversight
65 flight
101 decisions
302 ticks
7 solutions
6 problems
44 partecipants
72 opportunities

Figure 6.51: a), b) and c). Baseline runs of the model

In this case, when a participant collides an opportunity the resulting decision will be likely to be "by flight." After 302 simulation steps, the number of decision "by flight" (65) is more than twice the amount of decisions "by resolution" (30), and eleven times greater than the number of decisions "by oversight" (6). The number of participants surviving in this system has dropped dramatically (44 vs. 108). These simple experiments confirm a considerable degree of sensitivity of the system to changes in initial conditions in predictable directions. So far we have experimented exclusively with conservative model configurations. The model is very sensitive to friction that we can manipulate experimentally by setting the corresponding sliders to desired levels (that can be interpreted as "search costs" for participants and "timing" or "duration" for opportunities). Figure 6.52 a) illustrates what happens when dissipation of energy through movement is allowed. After 267 time periods we have no participants left, only 4 solutions, 8 problems and 1 opportunity. The system halts because it is no longer possible to take decisions. As expected, the value of RANGE_OVERS controls the number of decisions "by oversight" that will be taken (Figure 6.52 b).

Routines are defined as relatively stable self-reproducing communities of entities (participants, problems and solutions) connected by information flows. These connections form localized areas of high information intensity (visible as adjacent white patches of variable size and shape) with clear boundaries. *The study of the routinization* highlights the forces that regulate the dynamics of routine formation and dissolution in our simplified organizational world. In other words, we want to learn how order (in the form of recurrent patterns of actions and stable relationships among organizational entities) might emerge in garbage can decision processes. The first step in this direction necessarily involves the definition of an accounting mechanism capable of identifying the distinct clusters and of keeping track of what goes on within the different clusters of adjacent patches.



a)
friction-par 0.5
friction-opp 0.5
init-problems 100
init-solutions 100
init-partecipants 100
init-energy-pro 60
init-energy-sol 47
init-energy-par 100
show-energy? On Off
range_overs 5
max-energy 100
55 resolution
18 oversight
18 flight
91 decisions
267 ticks
4 solutions
8 problems
0 partecipants
1 opportunities

b)
friction-par 0.5
friction-opp 0.5
init-problems 100
init-solutions 100
init-partecipants 100
init-energy-pro 60
init-energy-sol 47
init-energy-par 100
show-energy? On Off
range_overs 15
max-energy 100
47 resolution
48 oversight
15 flight
111 decisions
307 ticks
4 solutions
9 problems
0 partecipants
0 opportunities

Figures 6.52: a) Baseline runs with friction, and b) baseline runs with different values of the RANGE_OVERS slider

For example, Figure 6.53 shows a space-time frame of the model and how entities are grouped into distinct clusters. Table 6.12 provides basic demographic information about the composition of the different communities of entities (note that the lattice has periodic boundary conditions, i.e., its surface is a torus). In the actual implementation of these ideas a "routine" (R) is defined in terms of the following tuple:

R=(#C-pat #C-tur #C-par #C-opp #C-sol #C-pro #C-nrg #C-inf)

Where, #C-pat is the number of patches in a cluster; #C-tur is the number of entities in a cluster (= #C-par + #C-opp + #C-sol + #C-pro); #C-nrg is the total energy present in the cluster, and #C-inf is the total quantity of information present in the cluster. Figure 6.53 illustrates the routines that are active in the specific time period to which the space-time frame refers. Table 6.12 summarizes the basic properties of the different clusters. The first column in the table reports the coordinates of the patches that identify the distinct clusters produced by the model. The second columns provide information that could be used to reconstruct the evolutionary trajectory of the size distribution of clusters over time.

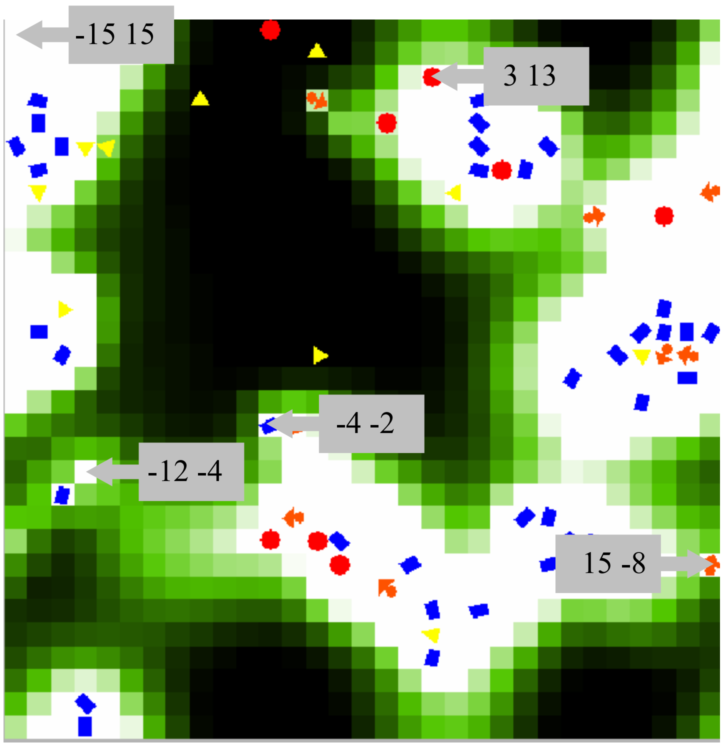


Figure 6.53: Communities of entities

Table 6.12: Quantitative characterization of the various communities of entities

X Y	#C-pat	#C-tur	#C-par	#C-opp	#C-sol	#C-pro	#C-nrg	#C-inf
-15 15	149	48	34	8	1	5	2883	408.8
3 13	35	11	7	0	3	1	1042.5	71.0
-4 -2	95	24	15	4	4	1	949.5	171.5
-12 -4	2	1	1	0	0	0	83.0	2.2
15 -8	1	1	0	1	0	0	35.5	1.1

The communities of entities that we have interpreted as the analogues of organizational routines are produced endogenously by our model. The graphs reported in the bottom of Figure 6.54 show, both the number of clusters (blue line), as well as the proportion of patches located within clusters (grey line) tends to increase over time. In this model, routines are very sensitive to the variation in the parameters that define the sensory mechanisms of the entities. Along with this consideration it is possible to claim that there is *the emergence of structures* driven by individual search capacity (called vision).

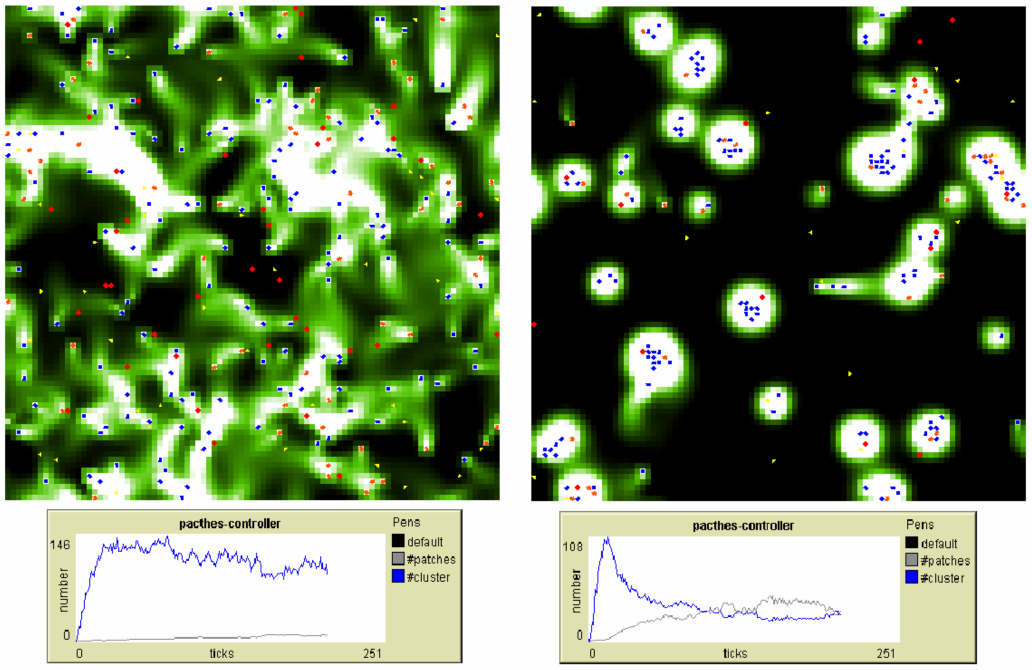


Figure 6.54 Effect of Vision-Angle on routine formation:
(left = 10 degree, right 45 = degree)

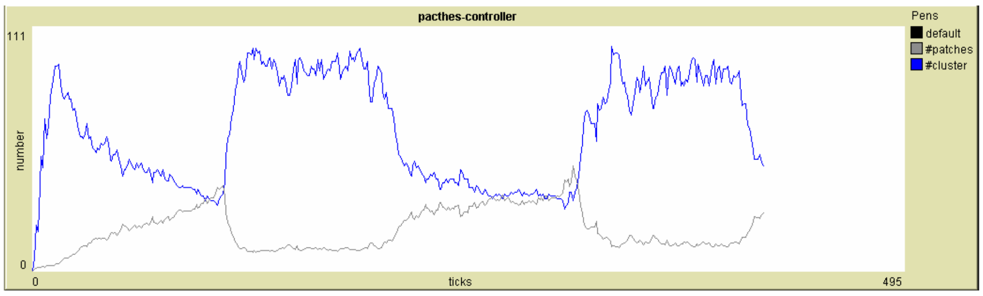


Figure 6.55 Implications of periodic changes in the Vision-Angle parameter
(45deg-10deg -45deg-10deg)

This is shown clearly in the two panels in Figure 6.54. The panel to the left in Figure 6.54 is produced by a model in which all agents have Vision-Angle = 10. In the panel to the right all agents have Vision-Angle = 45. The aggregate patterns produced by this change in individual information search capacity are appreciably different. As clearly illustrated in the time-series plots of Figure 6.54, the space-time frame to the left suggests the presence of a high number of clusters with few patches and a small number of bigger clusters. The frame to the right of Figure 6.54 implies the presence of a smaller number of clusters each containing an approximately equal number of patches. In Figure 6.55, we report a time series plot that illustrates the implications of periodic changes in the Vision-Angle parameter every 100 time periods. Figure 6.56 below reports the three pairs of space-time frame that represent the global state of the system immediately before and immediately after these changes. For example, the first frame in the first pair of frames at the top of Figure 6.56 represents the configuration of the system at time t=95 (when Vision-Angle = 45deg). The second frame in the first pair of frames represents the global configuration of the system at time t=105, immediately after changing the Vision-Angle to from 45deg to 10deg. As we discussed, the size distribution of clusters and their shape is highly sensitive to changes in the parameters that define the sensory mechanisms of the entities and, consequently, their search strategies.

6.3.4.4 Concluding remarks and future works

We have discussed a highly preliminary attempt to represent organizational decision processes when agents live in a structured socio-physical space, and are capable of reproducing, (i.e. of creating identical copies of themselves). According to the original “Garbage can model of organizational choice” (Cohen *et al.*, 1972) on which our model is based, three different types of agents populate the organizational space: “problems,” “solutions” and “participants.” A fourth species, “opportunities”, is produced endogenously as a combination of “problems” and “solutions.” Our model does not attempt to reproduce all the features of the original GCM.

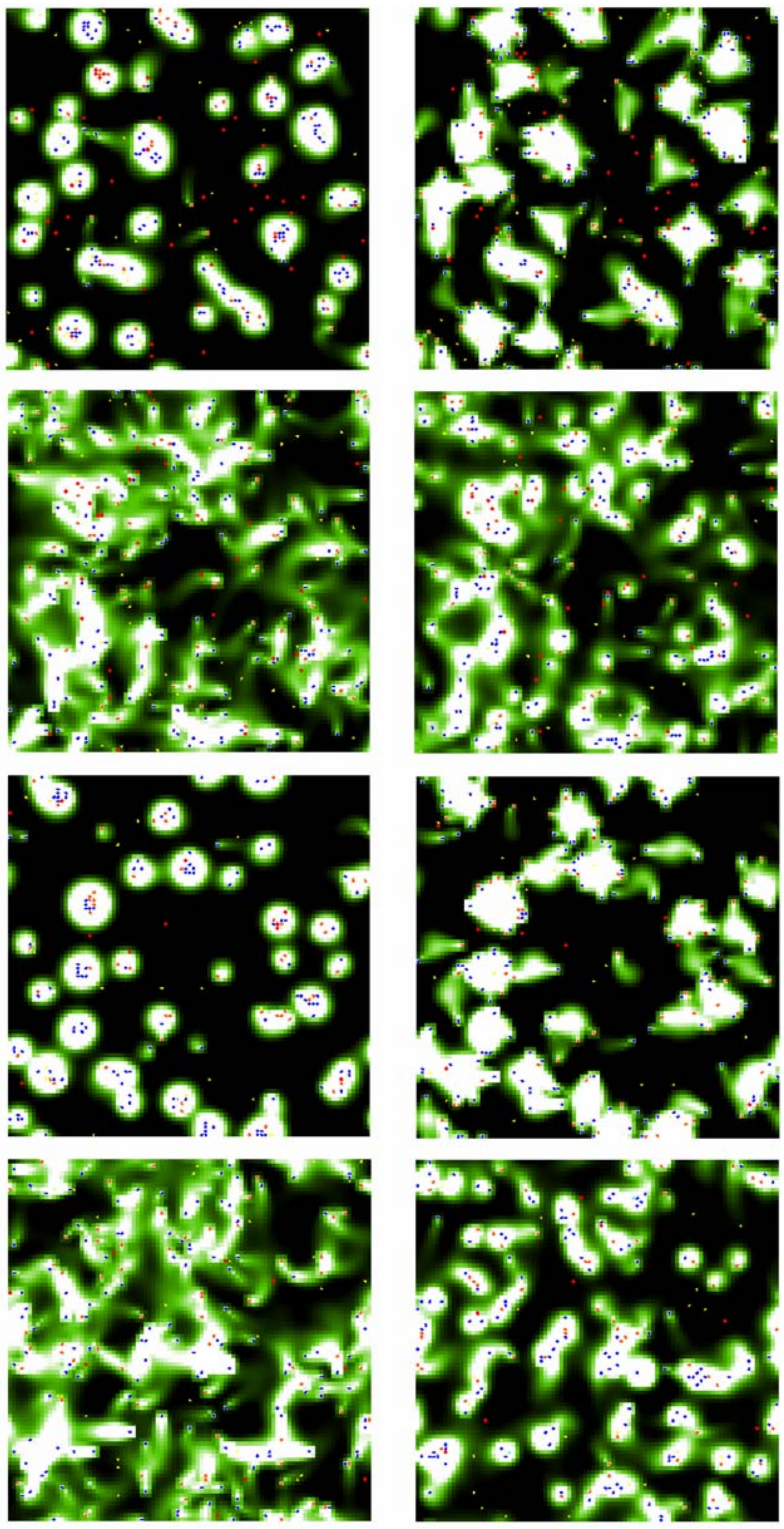


Figure 6.56 Local structures and global system configurations induced by periodic changes in the Vision-Angle parameter

For example, access structures (i.e., predefined connections between problems and choices) are not represented directly in the current model, which is based on very different topological assumptions about the physical structure of the space in which the various entities live. Further, some features of our model are not present in the original GCM. For example, evolutionary concerns related to reproduction rates of the various entities are central to our reconstruction, but are absent in the original model in which the arrival of “problems” “solutions,” “opportunities” and “participants” into the system is regulated by time-dependent, but otherwise exogenously given stochastic flows. The model that we present is based on a minimalist representation of an organizational world and no attempts are made to introduce elements of realism, or to suggest possible empirical contextualization. While we make no claim that actual social agents should conform even remotely to these assumptions (Carley *et al.*, 1994), we note that in our representation complexity at the system level does not seem to depend on complexity at level of the agents’ architecture. In its current form the simulator serves mostly didactic purposes. For example, it could be used as a computer-based learning environment to introduce students of social and economic institutions to a number of central issues in the study of organizational decision processes, and to the related representation problems. We see at least two directions for future research.

The first involves treating the model as a quasi-experimental observation-generation mechanism. Information on the states that the various entities occupy at any given time and on the duration of the entities in these states can be produced and treated as empirical data. Under certain assumptions statistical models could be estimated to explore patterns of state and time-dependence of the different entities on the lattice as a function of different combinations and levels of experimental parameters. The main idea behind such an analysis would be to arrive at a characterization of “organizational performance” in terms of patterns of duration dependence of problems, participants, solutions and opportunities. For example, stochastic models of state transition with multiple destinations could be estimated to explore conditions under which “problems” might tend to transform themselves into opportunities. In this perspective, the analysis of the conditions affecting the survival of “problems,” “participants,” and “solutions” on the lattice could reveal interesting aspects of the underlying model. A similar strategy has recently been followed by March, Schultz and Zhou (March *et al.*, 2000) in their detailed empirical study of rules in organizations. A second direction for further research involves additional programming work to inject elements of stability into Garbage Can decision making processes. Since its appearance, a historical criticism of the GCM has been based on its apparent inability to represent elements of stability, inertia and routinization and therefore to capture adequately what many consider being the most obvious feature of organizations (Cyert *et al.*, 1963; Hannan *et al.*, 1989; Nelson *et al.*, 1982; Padgett, 1980). In an attempt to address this problem, we presented a model in which "routines", relatively stable, self-reproducing and well bounded communities of "problems," "solutions," "opportunities" and "participants", emerge from simple local rules of information search and transmission that control the behavior of simplified mobile agents. The main learning point that the model can be used to illustrate is that complexity at the organizational level does not depend on architectural complexity at level of individual agents. This broad conclusion suggests the possibility of developing a conceptual link between aspects of social and economic organization, and the study of emergence in other natural and artificial systems; a link that remains poorly understood within current sociological and economic theories of organizations.

Finally, we retain that the model presented in this work is nothing more than a first step in a search for more constructive representations of organizational decision processes (i.e. for representations that do not presuppose the existence of entities that are meant to be explained (Fontana *et al.*, 1996)). Even as a first step the models falls short of its own promises in many respects. Perhaps the most obvious shortcoming involves the fact that the boundaries around primitive entities in the model are not allowed to change or evolve. In our model, entities are capable of reproducing, but their structure is kept constant throughout the simulation. In other words, the entities in the model might change quantitatively (e.g., their energy level may change), but they are capable of limited qualitative change as a consequence of their encounters with other entities (this is particularly true for participants, perhaps less so for "problems" and "solutions"). As a consequence and as a reflection in part of the original formulation of the Garbage Can Model, the state space of the model depends almost exclusively on a set of exogenous conditions with extremely limited scope for innovation, learning and the development of novelty at scales not specified at the level of individual entities (Crutchfield, 1994). In the context of our continuing modeling efforts, we believe that the main challenge ahead will be to remove these limitations, without increasing the complexity in the architecture of individual entities that populate the organizational landscape.

6.3.5 Participatory simulative scenarios within Interceptor

Here, we want to explain how it is possible to build participatory simulative scenarios within Interceptor emulating the participation of users geographically dispersed in different country. Along with this proposal, two are the main considerations. The former shows how is possible to build a participatory simulative scenarios within Interceptor, while the latter illustrates how is possible to run a participatory simulation inside a Laboratory emulating the participation of users geographically dispersed in different countries.

To build an Agent Based participatory simulation within Interceptor, the integration of an Agent Based Modeling platform is necessary. Along with this consideration, it must be possible to give instructions to several independent agents working in parallel inside Interceptor. These agents may be either completely controlled by humans or automatically programmed and must exchange information (more precisely, sensations and actions) among them in a shared virtual environment. Inside this shared virtual environment each of them can play their moves based on an intrinsic capability of local investigation and local action. We remember that Interceptor adopts an Agent Based Modeling platform (i.e. SPADES, the System for Parallel Agent Discrete agEnt Simulation that is a middleware system for agent-based distributed simulation) as Network Topology Manager. Therefore, the integration of an Agent Based Modeling platform within Interceptor is already done. In this way, building the reproduction of diverse complex system (inside a participatory simulation) is possible simply implementing different virtual environments (i.e. different worlds) and types of agents inside an Agent Based Modeling platform. Therefore, instead of adopting the "network topological" environment (i.e. NTMWorld; see the Figure 5.13) and the "router" agents in the highest component of the stack of the Interceptor system architecture (see the

Paragraph 5.3), we implement an environment and agents in accord with the simulation of other (i.e. not related to computer networks) complex systems. The ABMS platform integrated inside the prototype for our agent-based participatory simulation activities is partially based on Netlogo software. Nothing prevents us from substituting Netlogo with SPADES (look at the "execute and visualize session" Figure 6.43). Differently, from Netlogo that provide us, the Hubnet client to control remotely our agents, SPADES give us the possibility to interface directly our agents. Further, regarding the ABMS platform of Figure 6.44, the Request Manager and the State Updater components can be provided by SPADES. We can also write a version of the Snapshot Creator to capture the evolution of the states of the complex system generated by SPADES. Again, all the most complex models produced by Netlogo may be built based on three different types of agents (i.e. observers, turtles and patches) that can find corresponding ones in SPADES. Each model has only one observer that represents the most general virtual environment where all other agents live; its corresponding one is named World. Turtles that are the most active types of agents and patches that represent static pieces of resources are for SPADES simply active and passive agents. Finally, the software modules for the management of the 3D visualization can be directly connected to SPADES. In this way, we can build participatory simulative scenarios with Interceptor.

In particular, to play a participatory simulation is necessary to pilot the remote agents in a remote environment by means of an Agent Manager. These Agents live, interact and cooperate in the same remote centralized world. In our design, this world is inside the synthetic simulative environment of Interceptor (on the top of the stack of its system architecture). So, the communications of the Agent Managers (geographically distributed among multiple machines) must pass through the Emulative Interface of Interceptor to reach their software counterpart in the virtual environment. Then, it is possible to delay these communications with the remote environment by intercepting them. Once, a packet with new moves (for an agent) reaches the Emulative Interface is delayed according to a predefined emulative communication model. In fact, it is possible (if the configuration file is set up opportunely) to emulate the network communication delay between the real hosts, where users play, and the computer, where Interceptor runs. This is the typical operation that permits to the incoming packets to be delayed in accord with a specific CM before being deliver to one of the “entry” nodes of the synthetic topology. The Event Manager performances this operation by reading the configuration file (i.e. only after an arrival event). This means to configure all agents present inside the simulation environment as a possible entry simulated node to the proposed network topology. In this way, we can build participatory simulative environments inside a Laboratory emulating the participation of users geographically dispersed in different country. These participatory simulative environments allow to play model (i.e. not only network model) like the “Garbage can model of organizational choice” (Cohen *et al.*, 1972) inside Interceptor.

6.3.6 Conclusions of agent-based participatory simulation system

We have designed and developed a software architecture able to support the execution of agent-based participatory simulation activities and to render them in a 3D virtual world over wireless devices. We have conducted several empirical trials (both visual and numerical) that confirm that the structuring of the software architecture we have devised is able to guarantee the visual delivery of ABMS-based virtual worlds on wireless devices in a timely fashion. We have presented also a complete example of modeling a complex system. Hence, we report the design issues and an experimental campaign of the revision of the “Garbage can model of organizational choice”. Finally, we have explained how is possible to build participatory simulative scenarios within Interceptor simulating the participation of users geographically dispersed in different country. In particular, these users can participate to the simulations and can play model like he “Garbage can model of organizational choice”. Furthermore, we wish to point out here that the visual aspect of the 3D virtual worlds we are able to reproduce with our system is very realistic as our current implementation permits us to manage textures and other augmented reality-based objects in a sophisticated fashion. As an example of the use of texture-based 3D virtual worlds we report in Figure 6.57 a landscape where the photos of the four authors of this paper are mounted on the represented surfaces. Further, in Figures 6.58 two visual examples of use of 3D worlds are depicted which were obtained based on real geospatial data and augmented with virtual characters. We wish to conclude by mentioning that possible future developments of our system may foster new applications relevant to the field of participatory simulation such as mobile business, digital cinema, edutainment, multiplayer games and pervasive computing on wireless devices (Camponovo *et al.*, 2003; Ferretti *et al.*, 2003a; Ferretti *et al.*, 2003b; Espinoza *et al.*, 2001; Bates *et al.*, 1998; Iacucci *et al.*, 2000; Chen *et al.*, 1993; Colella *et al.*, 1998; Resnick *et al.*, 1998).

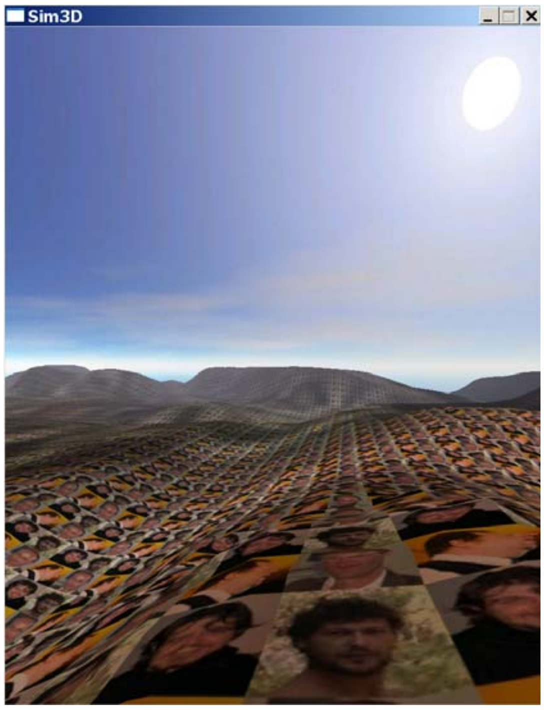


Figure 6.57: A virtual world with textures based on the photos of the authors of this work

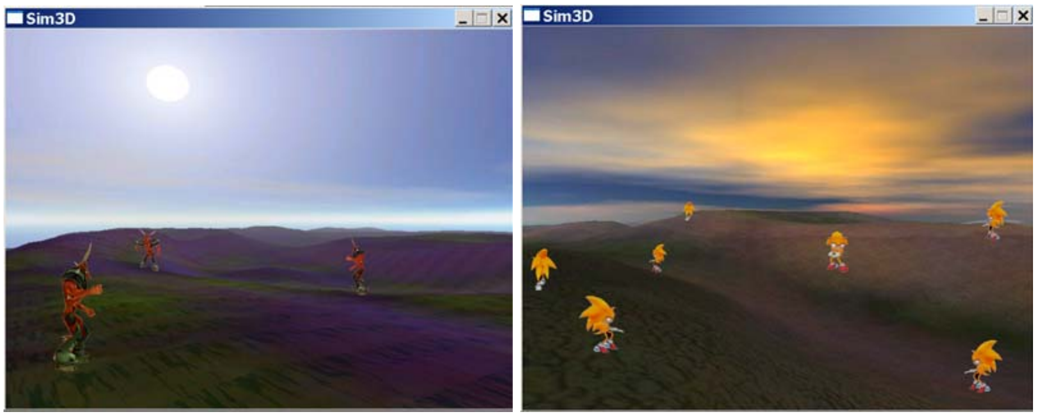


Figure 6.58: A 3D world based on real geo-spatial data where virtual characters are inserted

Chapter 7

Related Work

The literature on network emulation and simulation shows efforts by various groups. A lot of tools have been built, with different peculiarities. In this chapter, the comparison among the features of our Interceptor and the other tools are highlighted. In particular, Interceptor is compared with a well-known network emulator, named Dummynet (Rizzo, 1997) and with a well-known network simulator, named VINT/NS (Bajaj *et al.*, 1999). The reason is that Dummynet represents a stand point for all network emulators because it is one of the first emulators and because it has been integrated as the main atomic component inside the most recent and complex works in this field. Instead VINT/NS represents a standard *de facto* in the network simulation due to its capability of modelling complex network systems maintaining high performance execution. In particular, being VINT/NS a network simulator that incorporates an emulation interface, it represents the best choice for a comparison.

7.1 Comparisons with Network Emulators: Dummynet

The main difference with the network emulator, in general, and Dummynet, in particular, is that Interceptor introduces both a network topology manager and a traffic generator inside its architecture. In fact, some network emulators are used together with traffic generators (Nahum *et al.*, 2001: where they use Dummynet and SURGE) and some others have theirs own specific network topology manager (White *et al.*, 2001b; Vahdat *et al.*, 2002: where they use Dummynet as the main atomic component), but nobody has both (see the Paragraph 3.8). Interceptor integrates a network topology manager general purpose while, outside, a workload traffic generator, running on different real hosts, exploits it. Also the work behind Interceptor has been inspired by *Dummynet* (Rizzo, 1997). Two are the most important features that typically characterize network emulator tools, namely: the mechanisms adopted to integrate the models within the emulative environment with the emulative interface (see Table 7.1), and the mechanisms for avoiding interference from I/O processing, when the tool works in emulation mode (see Table 7.2).

As far as the former problem is concerned, Dummynet works by exploiting a finite queue at the interface between the TCP and IP layers in order to delay packets (left top side of Figure 7.1). An important limit of this approach is that Dummynet is able to manage just a single communication link for each router running an emulated TCP/IP stack (left bottom side of Figure 7.1). If several communication links have to be emulated simultaneously, several TCP/IP stacks must be emulated running over different machines (right side of Figure 7.1). A consequence of this fact is that Dummynet lacks a configurable tool to build simulation scenarios based on an arbitrary network topology. To fill this gap, more recent network emulators, like Netbed and ModelNet (based on Dummynet) have built complex network topologies, by physically involving high performance switches and powerful computers.

Table 7.1: Emulation Comparison (N: N >1, T: True, F: False)

EMULATOR / SIMULATOR	EMULATION LEVEL	NUMBER OF LINK COMMUNICATIONS	PACKET FORMAT	STREAM SIMULATED	SIMULATION SCENARIOUS
NS-2	IP	1 – N	NS packet	Static routing Dynamic routing (unicast & multicast)	T
DUMMYNET	IP	1	IP packet	TCP	F
INTERCEPTOR	IP	1 – N	IP packet	All over IP	T

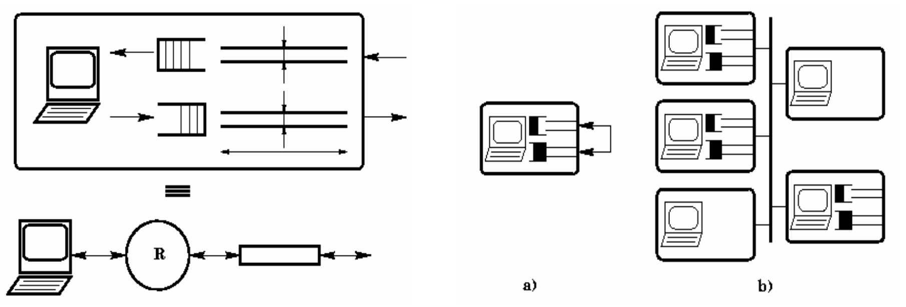


Figure 7.1: Dummynet (reported from (Rizzo, 1997))

Table 7.2: Simulation Comparison

EMULATOR / SIMULATOR	APPROACH	SIMULATION EVENT APPROACH	EVENT TYPE
NS-2	Fine	Discrete event	Next pending
DUMMYNET	-	Periodic time event	Periodic time
INTERCEPTOR	Coarse	Discrete event	Packet arrival and departure

In these network emulators, more instances of Dummynet work on different computers linked by the switches in order to implement the emulative scenario that reproduces the characteristics of a specified target topology on a link-by-link basis. Therefore, to change a topology model it is necessary to reconfigure a part of the computers working in the previous run. Further, to scale up the number of nodes of the network topology it is necessary to increase the number of computers involved in the experiment. Hence, conducting comparative analyses of different executions results more difficult.

Interceptor instead exploits the libipq libraries (Russell, 2002) in order to attach to packets, belonging to different communication channels, their correspondent delay values. In simple words, a single router running Interceptor is able to manage the emulation of several incoming communication channels. Furthermore, it can introduce these packets, coming from different communication channels, into its synthetic environment implementing a complex network topology. In order to avoid interference from the I/O processing when in emulation mode, Dummynet manages incoming packets during emulation by using a periodic time-driven approach, with a *T* given period. The problem here may be due to the fact that an excessive amount of incoming network traffic may cause the system to miss time deadlines. The direct consequence of this is that the emulator may become *livelocked* (Fall, 1999), i.e., notable performance degradation may be experienced in the emulation process. On the contrary, Interceptor uses a discrete event-driven approach to manage the advance of the synthetic emulation. Further, as already said, Interceptor adopts a prefetching mechanism to prevent the violation of real time constrain (see the Paragraph 5.4). Finally, we illustrate some technical details (see Table 7.3). Dummynet and Interceptor are both standalone systems and are written in “C”. Both Dummynet and Interceptor work only on a specific operative system, respectively, FreeBSD and Linux. In particular, Dummynet needs to recompile the (FreeBSD) kernel to work while Interceptor is currently supported as from the family 2.4.0 versions of the (Linux) kernel.

7.2 Comparisons with Network Simulators: VINT/NS

Most of network simulators have been designed and implemented as synthetic programs working on a single machine or on a cluster, without any live traffic introduction. Among the network simulators, NS-2 results one of the most suitable to be compared with Interceptor. The NS-2 has been designed and implemented as a network synthetic simulator working on a single machine, without any live traffic introduction (see Paragraph 2.5.1).

Table 7.3: Technical Report (T: True, F: False)

EMULATOR / SIMULATOR	**ARCHITECTURE**	**DEVELOPING LANGUAGE**	**OPERATING SISTEM**	**COMPILING KERNEL**	**WHY COMPILING KERNEL**
NS-2	Integrated in NS-2 system	C++ / OTcl	FreeBSD, Solaris & Linux	F	-
DUMMYNET	Standalone system	C	FreeBSD (version 3.4.0 or upper)	T	Including Dummynet
INTERCEPTOR	Standalone system	C/C++	Linux (kernel 2.4.0 or upper)	F	-

Only later an emulative interface and a real time scheduler have been integrated in NS-2 to introduce live traffic inside its simulation environment (Fall, 1999). For this reason, one of the main differences between NS-2 and Interceptor is the approach to the problem. Differently from NS-2, Interceptor is specifically developed as a system that intercepts live network packets by means of its emulative interface and introduces them in its synthetic simulative environment. In fact, NS-2 exploits a detailed fine-grained simulation model, while Interceptor adopts the coarse-grained approach to model network settings sacrificing simulation details, while it maintains the possibility of integrating external live traffic. Both NS-2 and Interceptor use a discrete event-driven approach to manage emulation/simulation. However, NS-2 has a single process software architecture that builds a monolithic structure. Instead, Interceptor interfaces and synchronizes different processes building a distributable architecture. In particular, all nodes (agents) belonging to the synthetic environment (of Interceptor) are single processes that can be distributed on a cluster of computers. For this reason, it is possible to integrate into the same synthetic environment different kinds of nodes that adopt different behaviours (for example different routing policy) and that can be configured at a run-time. Further, each single agent is fully programmable and at compile time it can be inserted in the environment. Finally, some different technical choices are shown on Table 7.3. Interceptor is a distributable standalone system, while NS-2 emulator (i.e. the emulative interface and the real time scheduler) is integrated into NS-2 system. Both Interceptor and NS-2 are written in “C/C++”. NS-2 runs on several operative systems: FreeBSD, Solaris and Linux, while Interceptor works only on Linux. Both Interceptor and NS-2 are currently supported by the operative system (where they run). The next paragraphs highlight more details about prominent differences in the software architecture, in the building of an emulative scenario, in the setting of a simulative scenario and, finally, several plots illustrate the comparison of the performances between the two systems.

7.2.1 Differences in Designing and Implementing the Software Architectures

When designing and implementing Interceptor, the computation for the management of the synthetic environment should not violate the real time constrain to integrate and process external live traffic by means of emulative interface. To prevent this violation, an appropriate mechanism must be adopted to integrate the models (i.e. that reproduce complex IP-based networks among the end-systems) within the synthetic environment with the emulative interface (see Table 7.1). NS-2 has the problem of integrating the emulative interface into the simulation. It has its own network packet format to manage the integration between the simulation model and the emulation interface (Figure 7.2). The emulation interface transforms IP datagrams into a particular NS-based format. In essence, NS converts network-native packets into simulator-compatible packets providing live network access (Fall, 1999). The problem with this approach is related to the fact that this transformation causes additional computational overhead. Instead, Interceptor stores each packet inside the synthetic environment without modifying it. This environment checks the packet and uses a model to calculate the simulation (service) time. No additional overhead is caused, because our architecture uses the IP format datagrams. Further, an appropriate real-time scheduler that avoids the interference from I/O processing, when the tool executes in emulation mode (see Table 7.2) must be built. The real-time scheduler of NS-2 runs by selecting the next imminent event, by executing it and returning to execute the next event.

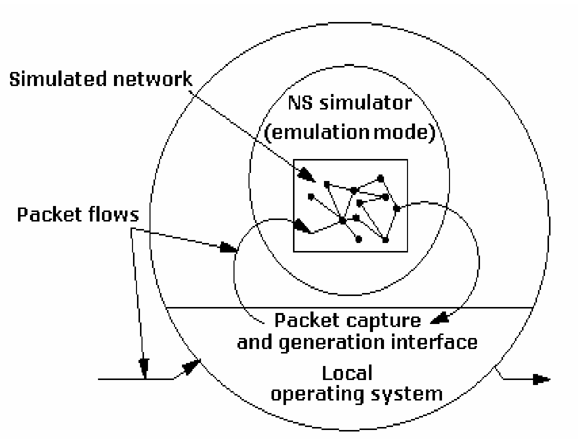


Figure 7.2: NS-2 emulation interface (reported from (Breslau *et al.*, 2000))

NS-2 is single-threaded and only one event is executed at any given time. If more events must be executed at the same time, their execution is performed on the first-scheduled first-served policy. Further, NS-2 does not support the partial execution of events or pre-emption. In particular, the real-time scheduler (adopted when NS-2 is on emulation mode) is implemented as a subclass of the list scheduler (i.e. one of the NS scheduler: list, heap and calendar). The list scheduler is implemented by using a simple linked-list structure. The list is handled in time-order (from imminent to latest), so event insertion and deletion require scanning the list to find the appropriate entry. Instead, the imminent event is always the head of the list. This implementation preserves event execution in a FIFO manner for simultaneous events. Therefore, the problem of avoiding interference from the I/O processing when in emulation mode is due to the fact that an excessive quantity of incoming network traffic may arrive. This fact may cause the system to miss time deadlines and to become *livelocked* (Fall, 1999). The real-time scheduler of Interceptor is similar to that of NS-2 but exploits a prefetching mechanism that: i) pays attention to the additional delay due to computation time and ii) handles the service queue (i.e. simple linked-list structure) avoiding that very close (in time) events were delayed waiting for their turn in the next simulation/emulation cycles. In the former case, this mechanism is able to calculate the simulation time of each packet in charge of the time spent transmitting and processing it. In particular, the mechanism controls the time to physically transmit a real packet from an end-system to another in the real lab without passing through Interceptor and the time to process this packet inside Interceptor. The processing time includes the phase where the packet is introduced into the architecture and that where the packet is injected outside in the real world. Therefore, the real execution time is equal to the simulation time subtracting the time calculated by this mechanism. Further, in the latter case, this mechanism is in charge of checking up all the expiring events in the service queue during a given period of time (up to 10 msec.) and eventually prefetches these if the system is becoming overloaded. This means that, every times, a service time expires all the events included in that period are prefetched (where 10 msec. is the minimum time granularity).

Another way to preserve the real time constrain is to build mechanisms of the simulative/emulative architecture that avoid extra computation if not necessary. For example, Interceptor provides with a mechanism (i.e. background traffic injector) that is able to inject virtual packets inside the synthetic environment. These virtual packets are not real and are only useful to create background traffic. This virtual traffic is not injected outside in the real world. With regard to this last consideration, it is not necessary that this kind of packets, different from the live packets, really exist inside the simulative environment. This kind of packets exists only in the phase of computation of the simulative delay and has no real size. These packets for the architecture are only system messages and no additional computation needs to manage them (i.e. it is not necessary to create and to delete background packets). In NS-2 this possibility is not supported; the only way to reproduce background traffic is to adopt other real computers generating real traffic (this possibility is supported also by Interceptor). However, in this way these background packets must have an NS-based format and they are processed as the others. This approach may cause additional computation as it manages a higher number of packets. This fact can disadvantage the real packets.

7.2.2 Building an Emulation Scenario with VINT/NS: Limitations

To build an emulation scenario with VINT/NS it is necessary that the computer, on which the emulator runs, acts as a default gateway for the other computers in this experiment. This implies the configuration of static rules in the routing table of each computer (taking part in the experiment). The official documentation about creating an emulative scenario with VINT/NS reports that if A wants to communicate to B via C, static rules in the routing table must be added:

- to add a static route on A to send the IP packets with B destination to C,
  route add B_IP C_IP, vice versa
- to add a static route on B to send the IP packets with A destination to C,
  route add A_IP C_IP.

The "route add" command defines which is the next hop for the sent packets. Using this command includes some limitations.

- Only the system administrator has the privilege to run the “route add” command. In this way the simulationist should have the system administrator’s privileges on each machine he wants to use. This is not always possible.
- All communications between A and B pass through C. In a lab with different users, everyone who wants to send packets from A to B (or vice versa) delivers them through C. For this reason, it is not possible to send packet directly from A to B, or vice versa.
- In a particular case, if there are some routers between source and destination (see the Figure 7.3), the simulationist must be able to configure each of them to route the packets coming from A toward B to C. In this case, the simulationist should have the network administrator’s privileges.

In particular, the last point that makes feasible the integration of computers belonging to different LANs is only possible if the transmission latency between the two hosts (i.e. A and B) is small with respect to the network emulated latency that the scenario tries to reproduce. This means that geographical networks are emulated by using LANs sufficiently close. Instead, Interceptor adopts an emulation consistent approach by implementing the virtual host method with IP Aliasing and IPTABLES techniques. As already said (see the Paragraph 5.1.2 and the Figure 7.4), Interceptor must not be the default gateway. In this case Interceptor adopts a virtual address to send a packet from A to B via C. Differently from VINT/NS, in this approach the only requirement is to have the system administrator’s privileges of the computer where Interceptor runs.

7.2.3 Setting a Simulative Scenario: Interactivity

The Interactivity is the possibility to interfere with the synthetic environment modifying it at a run-time. Interceptor supports this feature while NS-2 does not. To support this feature it is necessary to build a reactive synthetic environment that allows the simulationist to interact with each entity of its environment. These interactions must allow the simulationist to configure the entities at run-time.

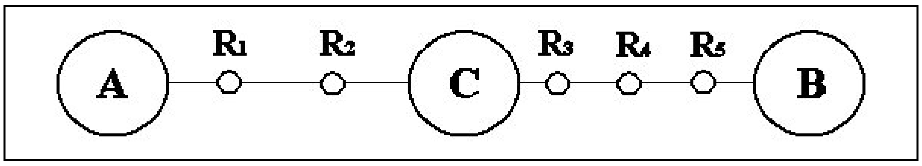


Figure 7.3: A,B,C belong to different LANs interconnected by some routers

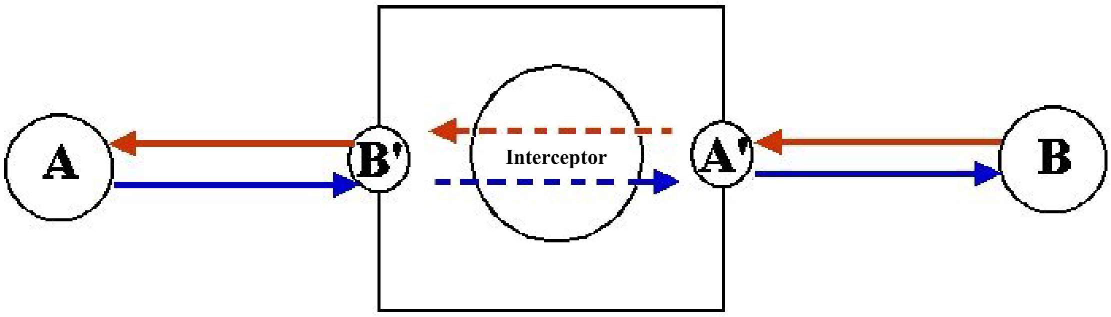


Figure 7.4: Route from A to B via C and vice versa with two aliases

In particular, to interact in a consistent way with the simulative environment, it is necessary to respect both the real-time constraints of the emulation and the run-time constraints of every single packet passing over each simulated node and link. For this reason, if the simulationist interacts with an entity (like a node or a link), it is extremely important that all the packets currently passing through this entity undergo new settings. This means that if the simulation model claims that the current packet should pass through the entity after the time at which the interaction is done, this packet undergoes the new settings, vice versa if the packet passes before, it does not. Interceptor works according to this run-time approach. The multi-agent system that implements the NTM allows the simulationist to interface each node/agent of the modelled network topology during the evolution of the simulation. This approach allows the simulationist to set at run-time several functionalities of each node and of each link (see Paragraph 5.3.2). For this reason, Interceptor does not compute *a priori* the time spent by the next packet and does not know *a priori* the simulated path between the source and the destination. This implies that Interceptor is forced to compute the next hop and the communication delay to reach it every time a packet reaches each node of the topology. In fact, it may be possible that in the meantime the simulative scenario has been changed.

NS-2 does not support the interactivity because it does not allow simulationist to interface with nodes and links during the simulation. The simulationist can only setup the configuration file before the beginning of the simulation. When a packet arrives at the simulative environment, NS-2 already knew the (static) network topologies and the characteristics of each link and node. For this reason, it knows the amount of time that this packet should wait inside the environment to simulate the passing through the path between the source and the destination. This means that it has already known (been the simulator a deterministic program) the path to reach the final destination (for the each incoming type of packet) and the time spent (for each hop before the destination). The only exception to this behaviour is when dynamic topologies are configured. Also these network dynamics are established on *a priori* basis: the startTime and finishTime variables set the configuration time during which a specific topological change occurs. This operation forces NS-2 to control that a packet does not pass through a wrong scenario. The network dynamics are only related to the characteristics of links. Hence, it is possible to model dynamic network topologies, where links may go down and may forces nodes to change routing policy.

7.2.4 Comparison of the performances

A series of experiments has been made in order to compare Interceptor and NS under the same conditions. The hardware scenario (see Figure 7.5) is composed of three computer running UDP senders (i.e. Pentium III, 677 MHz, 256 MB RAM, ethernet card 100 Mbit/sec., Linux Red-Hat 9.0), a single one running UDP receivers (i.e. Pentium III, 900 MHz, 512 MB RAM, ethernet card 100 Mbit/sec., Linux Red-Hat 9.0) and a final one where Interceptor or NS-2 are accommodated (i.e. as the previous). In order carry out comparable experiments on Interceptor and NS (see Paragraph 7.2.1), the computer accommodating the emulator/simulator must be the default gateway for all the network communications. The software scenario (see Figure 7.7) is obtained by increasing the number of UDP senders that generate traffic for an increasing number of UDP receivers. Each UDP sender sends an UDP datagram (i.e. 100 KB) and freezes for a time (i.e. 12 msec.), until the total number of sent datagrams (i.e. 1000 times) is not reached.

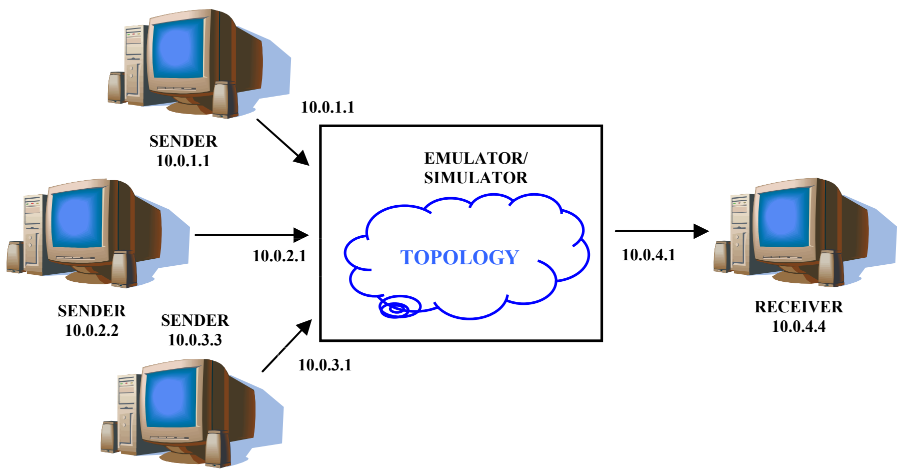


Figure 7.5: The hardware/software scenario

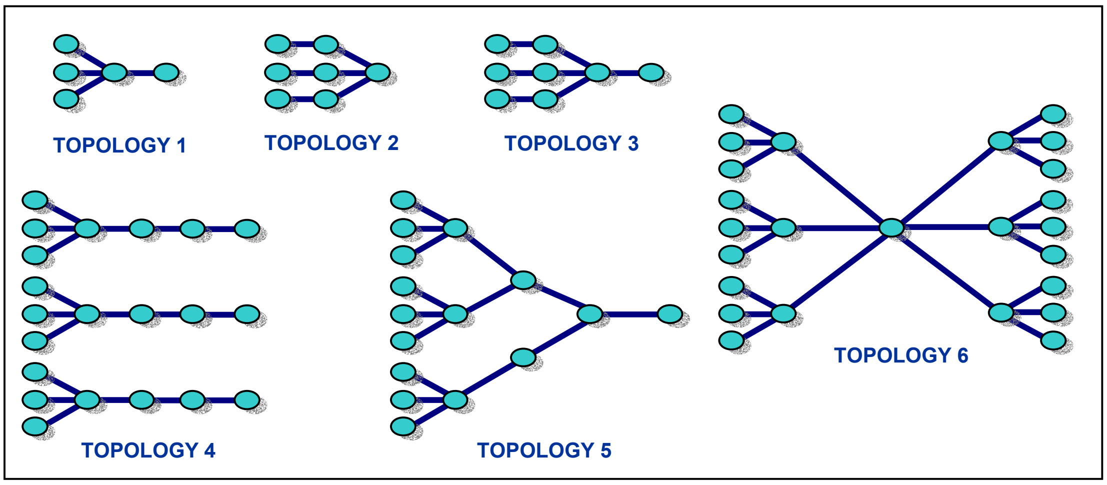


Figure 7.6: Different topologies

Each UDP receiver checks up the number of received datagrams and their integrity. The UDP datagrams (sent) reach the emulator/simulator (in form of IP packets) where they pass through diverse network topologies. These network topologies are differentiated by the number of the involved nodes and the length of the path between source and destination (see Figure 7.6). In these experiments, the capability of the emulative platform to support a higher detailed simulative environment by maintaining the real time constraints is tested. Hence, the experiments adopt network topologies with increasing complexity by monitoring the performance of the emulative interface. For this reason, it is interesting to evaluate the capability to schedule all received packets without discarding anyone. If the emulative platform is too busy to schedule packets instead of intercepting the incoming ones, these are discarded. The main aim of these experiments was to find a limit to the representation of Interceptor. This limit is quantified when the number of packets sent with additional delay reaches the 1% of the received ones. Further, it is important to evaluate the time difference between the time that a packet should have spent inside the simulative environment and that computed by the simulation for each packet. If this difference is positive, it means that packets are sent from the emulator/simulator to the receiver with additional delay. For each network topology of Figure 7.6 the number of senders (i.e. 3, 6 and 9) and the latency of each link (i.e. either 10 or 100 msec.) are varied by conducting more than 500 trails. The results obtained in these trails are shown on plots (see Figures 7.7-7.9) as the average of the monitored metrics according to the chosen topology. The metrics are: the percentage of the loss packets, the percentage of the packets that are send toward the final destination with additional delay (greater than 10 msec.) and the average values of the additional delays.

As shown in the plots with 3 or 6 senders, Interceptor is able to maintain the percentage of the packet loss (lower than 0.1%) and the packets sent with additional delay low (lower than 0.12%). We have established that the limit to the representation of Interceptor is reached when the percentage of packets sent with additional delay is greater than 1%; this occurs in the experiment with 9 senders and, respectively, with topology 6.

Therefore, from the comparison between the two emulators/simulators, we report that NS-2 has a lower percentage of packets sent with additional delay but the average values of the additional delays are greater than Interceptor. In particular, NS-2 shows no constant behavior in the percentage of the loss packets and its bursts are much greater than the corresponding values of Interceptor when they occur. In these cases, NS-2 generates single trails that are not reliable where it lost up to approximately 13,4% of the transmitted packets (i.e. it lost 400 packets over 3000). Instead, in the worst case of Interceptor a single trail lost 2,86% of transmitted packets (i.e. it lost 86 packets over 3000). To conduct this series of trails, we adapt Interceptor to work in an ideal environment for NS-2. These trails are conducted imposing the computer where NS-2 runs as default gateway. Interceptor does not require this constrain. Further, the core of the emulation/simulation will work exclusively on one computer.

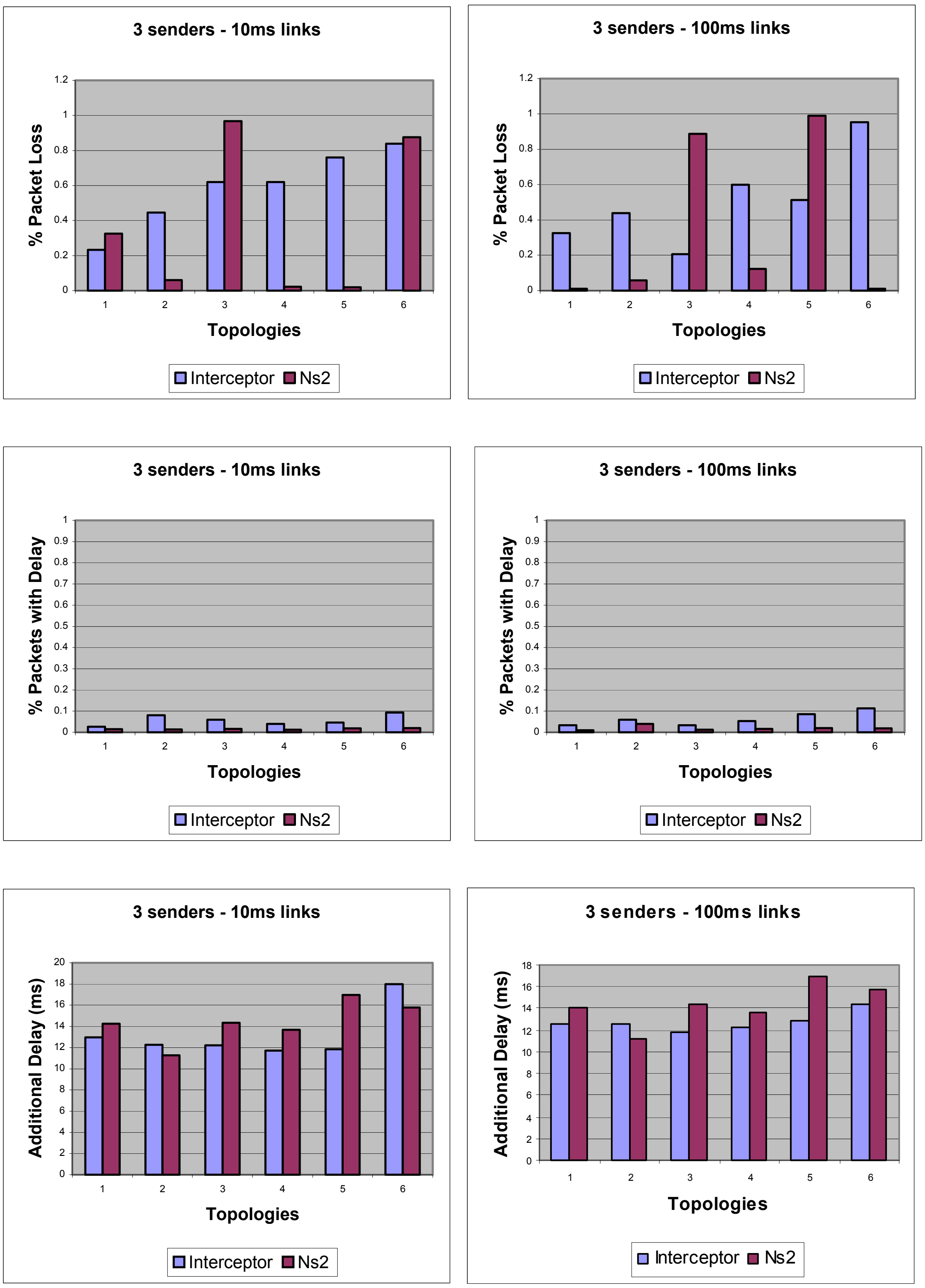


Figure 7.7: The percentage of the loss packets, the percentage of the packets sent with additional delay and the average values of the additional delays for a scenario comprehensive of three senders and, respectively, either 10 ms links or 100 ms links.

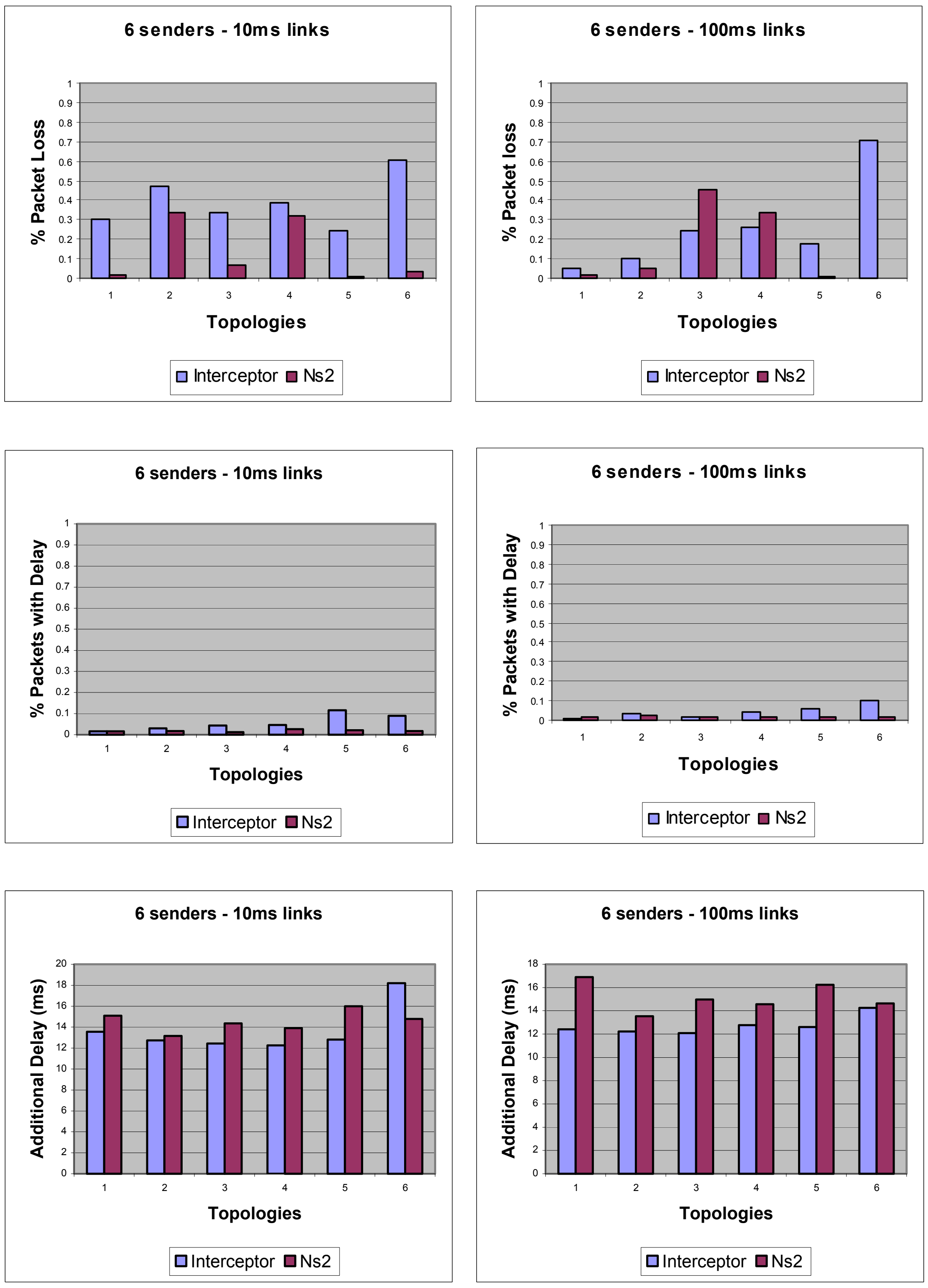


Figure 7.8: The percentage of the loss packets, the percentage of the packets sent with additional delay and the average values of the additional delays for a scenario comprehensive of six senders and, respectively, either 10 ms links or 100 ms links.

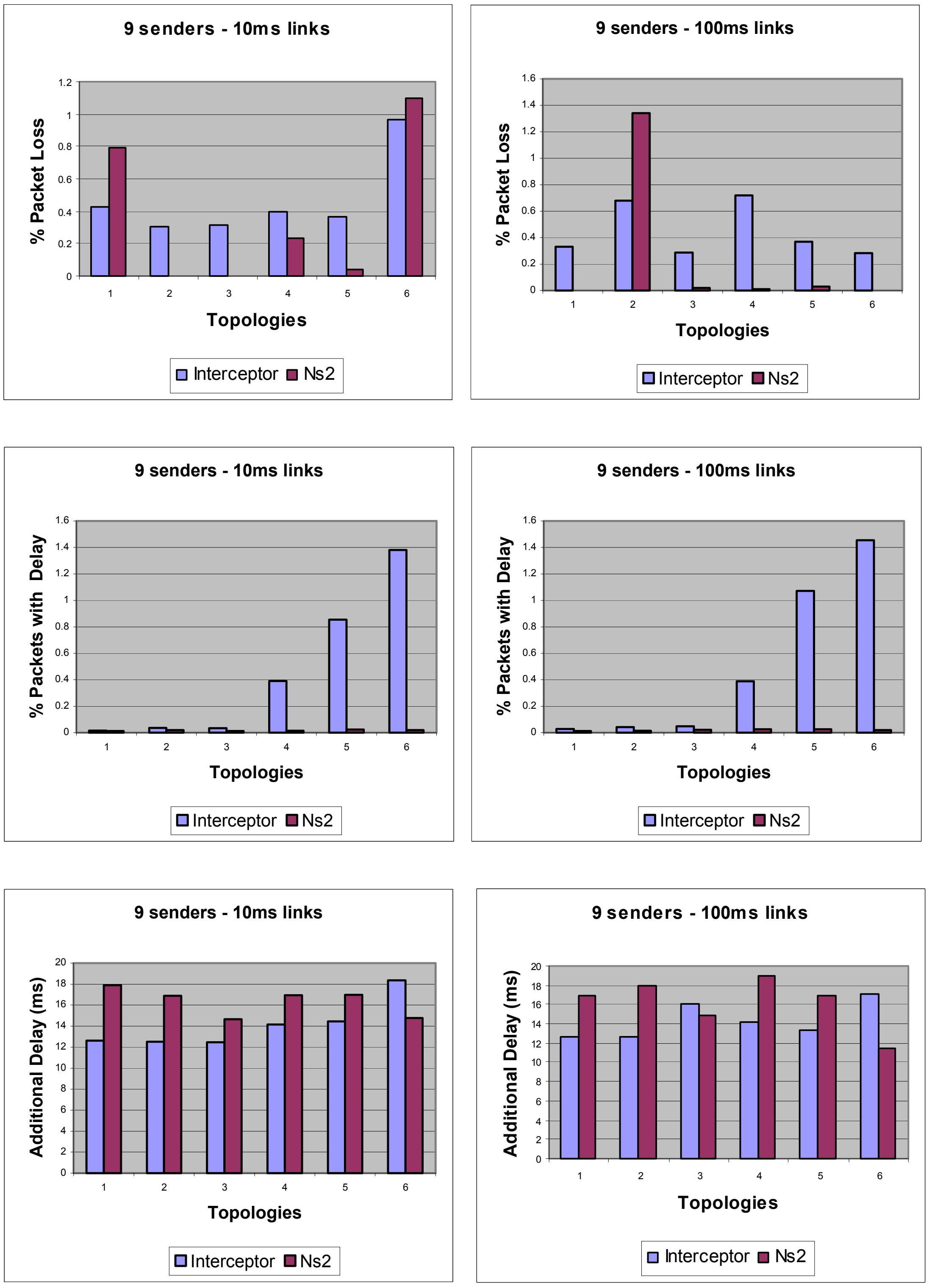


Figure 7.9: The percentage of the loss packets, the percentage of the packets sent with additional delay and the average values of the additional delays for a scenario comprehensive of nine senders and, respectively, either 10 ms links or 100 ms links.

Interceptor runs different processes one for each node, one for the each software layer. This is the main reason for decreasing the performance with 9 senders (i.e. increasing the interarrival of the packets and the number of nodes). By distributing these processes among few computers (for example only two) or by adopting a multi-processor hardware computer the system architecture of Interceptor can scale up its performances. All these processes are necessary to allow the simulationist to interact with the simulative environment at run-time (feature that NS-2 does not support). Concluding, the performances of Interceptor are very close to those of NS-2. For this reason, it is possible to interact in an active way at run-time with the simulative environment, eventually running an emulative scenario on a computer that does not work as a default gateway, with performances similar to larger diffuse NS-2. This means that the improved management of the scheduler (the prefetching mechanism) and the implementation (in accord with the coarse grained approach) of components that avoid extra computation allow a "real-time" and "run-time" constrained architecture (like Interceptor) to maintain the performances of the NS-2.

Chapter 8

Conclusions And Future Works

We designed and implemented a synthetic emulative platform that reproduces complex IP-based networks, called Interceptor. In particular, Interceptor is an emulative platform, which incorporates a synthetic coarse-grained environment, specifically developed to inject traffic coming from real network devices (running real applications) into simulated complex IP-based networks. A clear advantage of the coarse-grained approach our simulator is based on, derives from the fact that a large amount of the incoming live traffic may be processed, through the emulative interface, without remarkable performance degradation taking place during the simulation process. Interceptor provides many other advantages. It allows to set from simple to sophisticated network topology by reproducing a wide range of network communication models from single link to more general end-to-end communications. It supplies the possibility to change and update the synthetic environment without affecting the network interfaces at configuration time, while it provides the possibility to interact with simulated nodes and links at run-time. Furthermore, external workload traffic generators may exercise both end systems and synthetic emulative networks in accord with UE technology. Along with this idea, we developed a workload traffic generator that reproduces a high number of users running simultaneously the same application. A clear advantage of UE technology, on which our workload traffic generator is based, is the fact that it allows to move up several virtual users who simultaneously run the same application on the same computer in emulation mode.

Using Interceptor, we have obtained synthetic emulative results for well-known network applications, whose comparison with real experimental trials has confirmed the viability of our design choices. For this reason, Interceptor results useful to study the performance of network applications over complex IP-based networks. In essence, Interceptor works by delaying packets at the IP level, while the upper layers are directly executed in the synthetic network emulative environment. Three case studies show the effectiveness of Interceptor in evaluating network applications on synthetic complex IP-based networks within a lab-based scenario.

The first case illustrates the effectiveness of the C^2LD mechanism (i.e. a Web download accelerator based on the use of replicated Web Servers) has been assessed through the following two separate evaluation exercises: in an experimental and in an synthetic emulative scenario. Unfortunately, the experimental evaluation was based on a single client. These results were not enough to show the effectiveness of the C^2LD mechanism in the general case in which a replicated Web service is accessed by a large number of clients (say, hundreds), concurrently. In this case, the contribution of Interceptor was to provide the design and the development of a synthetic emulative scenario within which more clients running the C^2LD mechanism accessed, concurrently, the same replicated Web service. The plots obtained by the data monitored inside this scenario were useful to evaluate mechanisms in comparison with the standard downloading mechanism of HTTP.

The second of case studies shows the performance evaluation of an Wireless Internet architecture implementing a Song-on-Demand distribution service (from song distribution to interactive karaoke) for mobile devices. In this case, we reported on our experience in developing and evaluating this Wireless Internet application designed to deliver advanced musical services to mobile consumers over wireless links. The contribution of Interceptor was to provide the design and the development of the synthetic emulative scenarios built to evaluate this Song-on-Demand distribution service. We needed to use our synthetic emulative platform for two main considerations: the previous experimental campaign for evaluating this service has been conducted over a single wireless infrastructure (i.e. UMTS) and by means of just a single mobile device. Therefore, to understand the behavior of this service in a real scenario we needed a platform that provides several synthetic emulations where we can choose among different wireless infrastructures by scaling up the number of mobile devices. We implemented the appropriate scenarios and we ran a campaign by adopting Interceptor. In particular, these trails highlight that the main trouble of these architecture is due to scaling up of the number of mobile devices.

The last case explains the integration of participatory simulation based on multi-agent models to mimic complex systems within Interceptor. The users of these collaborative systems demand tools that are able to interact with the remote environment from mobile devices and to present visually the results of these cooperative simulation activities on their screens. In this context, we have explained how it was possible to build participatory simulative scenarios within Interceptor by emulating the participation of users geographically dispersed in different countries. Differently from the previous cases, where the service passes transparently over Interceptor, in this case the remote environment is the synthetic environment (i.e. the service end-point is inside the synthetic environment of Interceptor).

The results shown in these three case studies have confirmed that Interceptor may be successfully adopted to evaluate network architectures, applications and mechanisms since it guarantees a good compromise between the performance of the emulation and the accuracy of the synthetic IP networks.

The comparison with the related works have demonstrated that Interceptor permits a simulationist to interact in an active way at run-time within the synthetic environment with performances similar to larger diffuse NS-2. This means that the improved management of the scheduler due to the prefetching mechanism and the implementation of components (based on a coarse grained approach) that avoid extra computation allow the “real-time” and “run-time” constrained Interceptor to maintain the performances of the NS-2.

Future works with Interceptor will be devoted to the study of three main aspects: architectural improvements, architectural extensions and new case studies. The main improvement in the system architecture concerns the prefetching mechanism that prevents the *livelocked* problem. When a time expiration occurs (i.e. every time a packet arrives to the next node of the topology) this simple but effective mechanism checks how many other actions (related to the real IP packets) will expire in a very short time. In our implementation, all future actions (related to IP packets) are queued in the service queue (of the entire system) ordered by next time expiration. When the (service) timeout of the next imminent action is expiring, the prefetching mechanism checks up all the other actions whose timeout will expire in the next "time-select" period and schedule all of them. The value of the "time-select" parameter (based on microsecond) is established at configuration time. An interesting improvement of this mechanism relates to set dynamically (i.e. during the run of a synthetic emulative experiment) the values of "time-select" parameter. The dynamic choice of the "time-select" value may be obtained by checking up the number of actions inside the service queue (if it was increasing, decreasing or constant), the distribution of their interarrival times and if the last scheduled action was just in time or not. To prove the effectiveness of this new dynamic prefetching mechanism useful statistics about its performance should be reported.

With reference to architectural extensions, several new software modules may be useful to improve the reproduction of possible network effects. Each synthetic complex network is composed by two main kinds of entities: nodes and links. If we want to reproduce all possible network effects, it is necessary to increase the number of features of each single entity. Therefore, we can improve the features of the nodes by implementing software modules for different routing policies and different load balancing techniques. To further explore the generality of our approach, we can also implement extensions to our architecture in order to support the movement of the nodes. In this way, it becomes possible to reproduce wireless ad-hoc scenarios. Relating to the features of the links, new communication models for each controller (i.e. Latency/Delay, Bandwidth and Loss controllers) may be studied and implemented.

With regard to the new case studies, we promote the evaluation of the performance of our distributed architecture that supports networked multiplayer games (Ferretti *et al.*, 2003). By moving server functionality to the boundaries of the network it is possible to provide robustness, congestion control, scalability and cheating prevention. This architecture replicates the game state on the system, filters messages and uses application access points so as to optimize the communication with the clients. Our approach allows the distribution of the characters' information over the network and at the same time the replication of the virtual world's state only where necessary. Actually, we are planning a campaign of trails (Ferretti *et al.*, 2004; Palazzi *et al.*, 2004a; Palazzi *et al.*, 2004b). The typical scenario of this system architecture should be the Internet. Hence, we should evaluate the performance of this system architecture in a synthetic complex IP-based network scenario, using Interceptor. To further increase the effectiveness of our approach, we should implement a virtual player in line with UE technology.

References

(Aiello *et al.*, 2000) W. Aiello, F. Chung, and L. Lu, *"A Random Graph Model for Massive Graphs"*, In Proceedings of 32nd Annual Symposium in Theory of Computing, 2000.

(Allman, 1997) M. Allman, A. Caldwell, and S. Ostermann, *"ONE: The Ohio Network Emulator"*, In Technical Report TR-19972, Ohio University, August 1997.

(Avallone *et al.*, 2004a) Stefano Avallone, Antonio Pescapè and Giorgio Ventre, *"Analysis and experimentation of Internet Traffic Generator"*, In Proceedings of International Conference on Next Generation Teletraffic and Wired/Wireless Advanced Networking (New2an'04), St. Petersburg , Russia, February 2004.

(Avallone *et al.*, 2004b) Stefano Avallone, Donato Emma, Antonio Pescapè and Giorgio Ventre, *"A Distributed Multiplatform Architecture for Traffic Generation"*, In Proceedings of International Symposium on Performance Evaluation of Computer and Telecommunication Systems (SPECTS'04), San Jose, CA, USA, July 2004.

(Bagrodia *et al.*, 1994) R. Bagrodia, and W. Liao. *"Maisie: A Language for Design of Efficient Discrete-Event Simulation"*. In IEEE Transactions on Software Engineering, April 1994.

(Bagrodia *et al.*, 1997) R. Bagrodia and R. Meyerr, *"PARSEC: A Parallel Simulation Environment for Complex System"*, In Technical Report, UCLA, 1997.

(Bajaj *et al.*, 1999) S. Bajaj, L. Breslau, D. Estrin, K. Fall, S. Floyd, P. Haldar, M. Handley, A. Helmy, J. Heidemann, P. Huang, S. Kumar, S. McCanne, R. Rejaie, P. Sharma, K. Varadhan, Y. Xu, H. Yu, and D. Zappala, "*Improving Simulation for Network Research*" In Technical Report 99-702, University of Southern California, March 1999.

(Banks *et al.*, 2001) J. Banks, J. S. Carson II, B. L. Nelson, D. M. Nicol, "*Discrete-Event System Simulation*", Third Edition Book, Pretience Hall, Upple Saddle River NJ, 2001.

(Barford *et al.*, 1998) P. Barford, M.E. Crovella, *"Generating Representative Web Workloads for Network and Server Performance Evaluation*", In Proceedings of ACM Performance'98/SIGMETRICS, 151-160, Madison, WI, 1998.

(Barabasi *et al.*, 1999) A.L. Barabasi and R. Albert, *"Emergence of Scaling in Random Networks"*, Science, pages 509–512, October 1999.

(Bates *et al.*, 1998) J. Bates, M. D. Spiteri, D. Halls, J. Bacon, "*Integrating Real-World and Computer-Supported Collaboration in the Presence of Mobility"*, In Proceedings of the IEEE 7th International Workshops in Enabling Technologies: Infrastructure for Collaborative Enterprises, Stanford, CA, June 1998.

(Bethel, 1999) W. Bethel, RM Scene Graph, White Paper, December 1999, http://www.r3vis.com/RMSceneGraph.

(Bethel *et al.*, 2002) W. Bethel, R. Frank, J. D. Brederson, *"Combining a Multithreaded Scene Graph System with a Tiled Display Environment"*, In Proceedings of the 2002 IS&T/SPIE Conference on Electronic Imaging and Technology, San Jose, CA, January 2002.

(Bradford *et al.*, 2000) R. Bradford, R. Simmonds, B. Unger, "*A Parallel Discrete Event IP Network Emulator*", In Proceedings of 8th International Symposium on Modeling, Analysis and Simulation of Computer and Telecommunication Systems, San Francisco, CA, August 29 - September 01, 2000.

(Brakmo *et al.*, 1994) L. Brakmo, S. O'Malley, L. Peterson, *"TCP Vegas: New techniques for congestion detection and avoidance"*, In Proceedings of ACM SIGCOMM'94, pp. 24–35, 1994.

(Brakmo *et al.*, 1996) L. S. Brakmo, L. L. Peterson, *"Experiences with Network Simulation"*, In Proceedings of ACM SIGMETRICS '96 Philadelphia, PA, USA, May 1996.

(Breslau *et al.*, 2000) L. Breslau, D. Estrin, K. Fall, S. Floyd, J. Heidemann, A. Helmy, P. Huang, S. McCanne, K, Varadhan, Y. Xu, H. Yu, *"Advances in Network Simulatio*n", IEEE Computer, 59-67, May 2000.

(Burton *et al.*, 1995) R. Burton and B. Obel, *"Strategic organizational diagnosis and design: Developing theory for application"*, Kluwer Academic Publishers, Boston, MA, 1995.

(Cacciaguerra, 2002) S. Cacciaguerra, *"Wireless Simulation Modeling within Interceptor"*, In Proceedings of the Italian Society for Computer Simulation Conference (ISCS'02), Brindisi, Italy, December 2002.

(Cacciaguerra *et al.*, 2003) S. Cacciaguerra, M. Roccetti, V. Ghini and S. Ferretti, *"On Using An Emulative Middleware To Model Wireless Networks: Simulation Results And Validation"*, In Proceedings of the 1st European Simulation and Modelling Conference (ESMC'03), Naples, Italy, October 2003.

(Cacciaguerra *et al.*, 2004) S. Cacciaguerra, M. Roccetti, M. Roffilli and A. Lomi, *"A Wireless Software Architecture for Fast 3D Rendering of Agent-Based Multimedia Simulations on Portable Devices"*, In Proceedings of the 1st IEEE Consumer Communications and Networking Conference (CCNC'04), IEEE Communications Society, Las Vegas, Nevada (USA), January 2004.

(Calvert *et al.*, 1997) K. Calvert, E. Zegura, and M. Doar, *"Modeling Internet topology"*, In IEEE Communication Magazine, June 1997.

(Calzarossa *et al.*, 1993) M. Calzarossa and G. Serazzi, *"Workload Characterization: a Survey"*, In Proceedings of the IEEE, 8(81):1136-1150, 1993.

(Calzarossa *et al.*, 2000) M. Calzarossa, L. Massari, and D. *Tessera "Workload Characterization Issues and Methodologies"*, In Performance Evaluation - Origins and Directions, volume 1769 of Lecture Notes in Computer Science, pages 459-484, Springer-Verlag, 2000.

(Camponovo *et al.*, 2003) G. Camponovo, Y. Pigneur*, "Analyzing The Actor Game In M-Business"*, In Annals of Telecommunications, , Vol. 58, January - February 2003.

(Cao *et al.*, 2002a) J. Cao, W. S. Cleveland, D. Lin and D. X. Sun *"Internet Traffic Tend Toward Poisson and Independent as the Load Increases"* In NonLinear Estimation and Classification Book, eds. C.Holmes, D. Denison, M. Hansen, B. Yu, and B. Mallick, Springer, New York, 2002.

(Cao *et al.*, 2002b) J. Cao, W. S. Cleveland, D. Lin and D. X. Sun *"Internet Traffic: Statistical Multiplexing Gains"*, In Proceeding of Workshop on Internet and WWW Measurement, Mapping and Modeling (DIMACS'02), 2002.

(Cardellini *et al.*, 2001) V. Cardellini, E. Casalicchio, M. Colajanni, *"A Performance Study of Distribuited Architectures for the Quality of Web Services"*, In Proceeding of 34th Hawaii International Conference on System Sciences, 2001.

(Carley, 1986) C. Carley, *"Efficiency in a garbage can: Implications for crisis management"*, in J.G. March, J.G., and R. Weissinger-Baylon (Eds.), Ambiguity and command: Organizational perspectives on military decision making, Marshfield, MA, Pitman, 1986.

(Carley *et al.*, 1994) K. Carley and A. Newell, *"The nature of the social agent"*, Journal of Mathematical Sociology, 19(4): 221-262, 1994.

(Carniani *et al.*, 2001) E. Carniani, R. Davoli, *"The NetWire emulator: a tool for teaching and understanding networks"*, In Proceedings of the conference on Integrating Technology into Computer Science Education (ITiCSE'01), ACM Press, pp. 153–156, 2001.

(Carson, 1997) M. Carson, *"Application and protocol testing through network emulation"*, 1997. *http://www.antd.nist.gov/itg/nistnet.*

(Carson, 2000) M. Carson, *"NISTNet Network Emulator"*, 2000, *http://www.antd.nist.gov/itg/nistnet/*

(Chang, 1999) X. Chang, *"Network Simulations with OPNET"*, In Proceedings of the Winter Simulation Conference, 1999.

(Chen *et al.*, 1993) D. Chen, W. Stroup, *"General Systems Theory: Toward a Conceptual Framework for Science and Technology Education for All"*, In the Journal for Science Education and Technology, 1993.

(Clark, 1976) J. Clark, *"Hierarchical Geometric Models for Visible Surface Algorithms"*, In Communications of the ACM, October 1976.

(Clarke Wilson, 1994) S. Clarke Wilson, *"The Design of Virtual Environments – Value Added Entertainment"*, In Computer Graphics, Vol. 28, N. 2, 1994.

(Cohen *et al.*, 1972) M.D. Cohen, J.G. March, and J.P. Olsen, *"A garbage can model of organizational choice"*, In Administrative Science Quarterly, 17:1-25, 1972.

(Cohen *et al.*, 1974) M.D. Cohen and J.G. March, *"Leadership and ambiguity"*, Book McGraw Hill, New York, NY, 1974.

(Colella *et al.*, 1998) V. Colella, R. Borovoy, M. Resnick, *"Participatory Simulations: Using Computational to Learn about Dynamic Systems"*, In Proceedings of the Computer Human Interface Conference, Los Angeles, CA, April 1998.

(Crutchfield, 1994) J.P. Crutchfield, "Is anything ever new? Considering emergence", In A. Cowan, D. Pines and D. Meltzer (Eds.) Complexity: Metaphors, models and reality, Perseus Books, 515-538.Cambridge, MA, 1994.

(Cyert *et al.*, 1963) R.M. Cyert and J.G. March, *"A behavioral theory of the firm"*, Book, Prentice-Hall, New Jersey,1963.

(Dam *et al.*, 1998) K. K. Dam, L. M. Ni, *"Design and Implementation of a Network Emulator"*, In Technical Report MSU-CPS-ACS-98-16, Michigan State University, May 1998.

(Doar, 1996) M. Doar, *"A Better Model for Generating Test Networks"*, In Proceeding of IEEE GLOBECOM, November 1996.

(Emma *et al.*, 2004) D. Emma, A. Pescapè and G. Ventre, *"Analysis and experimentation of an open distributed platform for synthetic traffic generation"*, In Proceeding of 10th International Workshop on Future Trends in Distributed Computing Systems (FTDCS'04) Suzhou, China, May 2004.

(Espinoza *et al.*, 2001) F. Espinoza, P. Persson, A. Sandin, H. Nyström, E. Cacciatore and M. Bylund, *"GeoNotes: Social and Navigational Aspects of Location-Based Information Systems"*, In Proceedings of the Ubiquitous Computing International Conference, Atlanta, GA, September- October 2001.

(Familiar, 2003) Familiar Linux distribution official site, 2003, http://familiar.handhelds.org

(Fall, 1999). K. Fall, *"Network Emulator in the Vins/NS Simulator*", In Proceedings of the 4th IEEE Symposium on Computers and Communications, pp. 244-250, Red Sea, Egypt, July 1999.

(Faloutsos *et al.*, 1999) M. Faloutsos, P. Faloutsos, and C. Faloutsos, *"On Power-Law Relationships of the Internet Topology"*, In ACM Computer Communication Review, Cambridge, MA, September 1999.

(Ferretti *et al.*, 2003a) S. Ferretti and S. Cacciaguerra "*A Design for Networked Multiplayer Games: an Architectural Proposal*", In Proceedings of the Euromedia Conference, Plymouth (UK), April 2003.

(Ferretti *et al.*, 2003b) S. Ferretti, M. Roccetti, *"On Designing an Event Delivery Service for Multiplayer Networked Games: An Approach based on Obsolescence"*, In Proceedings of the 7th International Conference on Internet, Multimedia Systems and Applications, Hawaii, Honolulu, August 2003.

(Ferretti *et al.*, 2004) S. Ferretti, M. Roccetti & S. Cacciaguerra, *"On Distributing Interactive Storytelling: Issues of Event Synchronization and a Solution"* In Proceedings of the 2nd International Conference on Technologies for Interactive Digital Storytelling and Entertainment (TIDSE 2004), (S. Gobel, U. Spierling, A. Hoffmann, I. Iurgel, O. Schneider, J. Dechau, A. Feix Eds.), Darmstadt (De), Lecture Notes in Computer Science n. 3105, Springer Verlag, 219-231, June 2004.

(Ferris *et al.*, 1946) C. L. Ferris, F. E. Grubbs, and C. L. Weaver, "*Operating Characteristics for the Common Statistical Test of Significance*", In Annals of Mathematical Statistics, 1946.

(Fielding *et al.*, 1999) R. Fielding,J. Gettys, J. Mogul, H. Frystyk, L. Masinter, P. Leach, T. Berners-Lee, *"Hypertext Transfer Protocol - HTTP/1.1"*, RFC 2616, June 1999.

(Floyd *et al.*, 2001) S. Floyd, V. Paxson, *"Difficulties in Simulating the Internet"*, in IEEE/ACM Transactions on Networking, February 2001.

(Fontana *et al.*, 1994) W. Fontana and L. Buss, *"The Arrival of the Fittest: Toward a Theory of Biological Organization"*, In Bulletin of Mathematical, Biology, 56: 1-64, 1994.

(Fontana et al., 1996) W. Fontana and L. Buss, *"The barrier of objects; From dynamical systems to bounded organizations"*, in J. Casti and A Karlqvist (eds) Boundaries and Barriers: 56-116, Addison Wesley, 1996.

(Ghini *et al.*, 2001) V. Ghini, F. Panzieri, M. Roccetti, *"Client-centered Load Distribution: A Mechanism for Constructing Responsive Web Services*", In Proceedings of 34th IEEE Hawaii International Conference On System Sciences (HICSS-34), Maui, Hawaii, January 2001.

(Ghini *et al.*, 2002) V. Ghini and F. Panzieri, *"QoS-Adaptive Middleware Services"*, Ph.D Thesis, Bologna, Italy, February 2002

(GloMoSim, 2001) GloMoSim official site, February 2001, http://pcl.cs.ucla.edu/projects/glomosim/

(GPE, 2003) GPE, The GPE Palmtop Environment official site, 2003, http://gpe.handhelds.org

(Hannan *et al.*, 1989) M. Hannan and J. Freeman, *"Organizational ecology"*, Harvard University Press, Cambridge, MA, 1989.

(Herrscher *et al.*, 2002a) D. Herrscher, A. Leonhardi, and K. Rothermel, *"Modeling Computer Networks for Emulation"*, In Proceedings of the 2002 International Conference on Parallel and Distributed Processing Techniques and Applications *(PDPTA'02),* Las Vegas, NE, June 2002.

(Herrscher *et al.*, 2002b) D. Herrscher, K. Rothermel, *"A Dynamic Network scenario Emulation Tool"*, In Proceedings of the 11th International Conference on Computer Communications and Networks (ICCCN 2002), Miami, FL, October 2002.

(Iacucci *et al.*, 2000) G. Iacucci, K. Kuutti, M. Ranta, *"On the Move with a Magic Thing: Role Playing in Concept Design of Mobile Services and Devices"*, In Proceedings of the ACM Symposium on Designing Interactive Systems, Brooklin, NY, 2000.

(Ingham *et al.*, 1994) D. Ingham and G. Parrington, *"Delayline: A Wide-Area Network Emulation Tool"*, In Computing Systems, Vol. 7, No. 3, pp. 313-332, 1994.

(Ingham *et al.*, 2000) D. Ingham, S.K. Shrivastava, F. Panzieri, *"Constructing Dependable Web Services"*, In IEEE Internet Computing, Vol. 4, N. 1, pp. 25-33, January-February 2000.

(JAS, 2002) JAS official site, 2002, http://sourceforge.net/projects/jaslibrary/

(Jin *et al.*, 2000) C. Jin, Q. Chen, and S. Jamin, *"Inet: Internet Topology Generator"*, In Technical Report Research Report CSE-TR-433-00, University of Michigan at Ann Arbor, 2000.

(Jin *et al.*, 2001) S. Jin, A Bestavros *"GISMO: Generator of Internet Streaming Media Objects and workloads"*, In ACM SIGMETRICS Performance Evaluation Review, November 2001.

(Kanter, 2003) T. G. Kanter, *"Attaching Context-Aware Services to Moving Locations"*, In IEEE Internet Computing, Vol. 7, N. 2, March - April 2003.

(Knuth, 1994) D. Knuth. *"The Stanford GraphBase: A Platform for Combinatorial Computing"* in Addison-Wesley book, 1994.

(Lepreau *et al.*, 1999) J. Lepreau, C. Alfeld, D. Andersen, K. van Maren, *"A Large-Scale Network Testbed*", In SIGCOMM'99 New Research Session*,* Cambridge, 1999.

(Levitt *et al.*, 1989) B. Levitt and C. Nass, *"The lid on the Garbage Can: Institutional Constraints on Decision Making in the Textbook Publishing Industry"*, Administrative Science Quarterly, 34(2), 190-207, 1989.

(Lin *et al.*, 2002) T. Lin, S. F. Midkiff, J. S. Park, *"A Dynamic Topology Switch for the Emulation of the Wireless Mobile Ad Hoc Networks"*, In Proceedings of IEEE Conference on Local Computer Networks (LCN'02) Tampa, Florida, USA, November 2002.

(Lomi *et al.*, 1999) A. Lomi, E. R. Larsen*, "Learning with Simulation: Understanding the Strategic Implications of Deregulation and Competition in the Electricity Industry"*, in Proceedings of the Workshop on Agent Simulation: Applications, Models, and Tools, Chicago, IL, October 1999.

(Lomi *et al.*, 2003a) A. Lomi, S. Cacciaguerra *"Organizational Decision Chemistry on a Lattice"*, In Proceedings of the Seventh Annual Swarm Users/Researchers Conference (SwarmFest'03), Notre Dame, Indiana (USA), April 2003.

(Lomi *et al.*, 2003b) A. Lomi, S. Cacciaguerra, *"The Emergence of Routines in an Organizational Decision Chemistry"*, In Proceedings of the First European Social Simulation Association Conference (ESSA 2003), Groningen, Netherland, September 2003.

(March *et al.*, 1976) J.G. March and J.P. Olsen, *"Ambiguity and choice"*, Bergen, Universitetsforlaget, 1976.

(March, 1978) J.G. March, *"Bounded rationality, ambiguity and the engineering of choice"*, Bell Journal of Economics, 9: 587-607, 1978.

(March *et al.*, 1986) J.G. March and R. Weissinger Baylon*, "Ambiguity and command: Organizational perspectives on military decision making"*, Marshfield, MA, Pitman, 1986.

(March, 1994) J.G. March, *"A primer on decision making"*, The Free Press, New York, NY, 1994.

(March *et al.*, 2000) J.G. March, M. Schultz and X. Zhou, *"The dynamics of rules"*, Stanford University Press, Stanford, CA, 2000.

(Masuch *et al.*, 1989) M. Masuch and P. LaPotin, *"Beyond garbage cans: An AI model of organizational choice"*, Administrative Science Quarterly, 34: 38-67, 1989.

(McFadden, 1976) D. McFadden, *"The revealed preferences of a government bureaucracy: Theory"*, In Rand Journal of Economics. (6) No. 2 (Autumn): 401-416, 1976.

(Medina *et al.*, 2000) A. Medina, I. Matta, and J. Byers, *"On the Origin of Powerlaws in Internet Topologies"*, In ACM Computer Communication Review, 30(2):18–28, April 2000.

(Medina *et al.*, 2001) A. Medina, A. Lakhina, I. Matta, and John Byers, *"BRITE: An Approach to Universal Topology Generation",* In Proceedings of the International Workshop on Modeling, Analysis and Simulation of Computer and Telecommunications Systems (MASCOTS'01), August 2001.

(Milliken, 1993) W. Milliken, Product Overview on BBN Long-Link Emulator, 1993.

(Miller, 2001) J. Miller, *"Evolving information processing organizations"*, in A. Lomi, and E. Larsen (Eds.) Dynamics of Organizations: 307-328. Boston, MA/Palo Alto, CA. MIT Press/AAAI Press, 2001.

(Nahum *et al.*, 2001) E. M. Nahum, M. C. Rosu, S. Seshan and J. Almeida *"The Effects of Wide-Area Conditions on WWW Server Performance"*, In Proceedings of ACM SIGMETRICS/Performance, Cambridge, MA, 2001.

(Naylor *et al.*, 1967) T. H. Naylor and J. M. Finger *"Verification of Computer Simulation Models"*, In Proceedings of the Management Science, Vol. 2, pp. B92-B101.

(Netlogo, 1999)Netlogo official site, 1999, http://www.ccl.sesp.northwestern.edu/netlogo

(Nelson *et al.*, 1982) R.R. Nelson and S.G. Winter, *"An evolutionary theory of economic change"*, The Belknap Press of Harvard University Press, Cambridge, MA, 1982.

(NIST NET, 1999) N. Advanced Network Technologies Division. NIST Net home page, 1999, http://www.antd.nist.gov/itg/nistnet/

(NS) "The Network Simulator 2 official site", http://www.isi.edu/nsnam/ns/

(NS, 1999) B. N. research group. NS home page, 1999, http://www-mash.cs.berkeley.edu/ns/ns.html

(NS manual, 2003) Manual, Notes and Documentation of The Network Simulator 2, 2003, http://www.isi.edu/nsnam/ns/ns-documentation.html

(Padgett, 1980) J.F. Padgett, *"Managing garbage can hierarchies"*, Administrative Science Quarterly, 24(4): 583-604, 1980.

(Padgett *et al.*, 2003) J.F. Padgett, D. Lee, N. Collier, *"Economic production as chemistry",* In Proceedings of Forthcoming in Industrial and Corporate Change, 2003.

(Paxson *et al.*, 1999) V. Paxson and S. Floyd, *"Why We Don't Know How To Simulate The Internet"*, In Proceedings of the Winter Simulation Conference, Atlanta, GA, January 1997.

(Palazzi *et al.*, 2004a) C. E. Palazzi, S. Ferretti, S. Cacciaguerra & M. Roccetti, *"On Maintaining Interactivity in Event Delivery Synchronization for Mirrored Game Architectures"*, In Proceedings of the 1st IEEE International Workshop on Networking Issues in Multimedia Entertainment (NIME'04) - GLOBECOM 2004 - Satellite Workshop, (G. Pau, M. Roccetti, S. Lee Eds.), Dallas (USA), IEEE Communications Society, November 2004.

(Palazzi *et al.*, 2004b) C. E. Palazzi, S. Ferretti, S. Cacciaguerra & M. Roccetti, *"A RIO-like Technique for Interactivity Loss Avoidance in Fast-Paced Multiplayer Online Games: a Preliminary Study"*, In Proceedings of the 2nd Annual International Workshop in Computer Game Design and Technology (GDTW'04), Liverpool (UK), November 2004.

(Resnick *et al.*, 1998) M. Resnick, U. Wilensky, *"Diving into Complexity: Developing Probabilistic Decentralized Thinking through Role-Playing Activities"*, In Journal of the Learning Sciences, Vol. 7, N. 2, 1998.

(Rizzo, 1997) L. Rizzo, *"Dummynet: a Simple Approach to the Evaluation of Network Protocols*", In ACM Computer Communication Review, January 1997.

(Roccetti *et al.*, 2002a) M. Roccetti, V. Ghini, P. Salomoni, A. Gambetti, D. Melandri, M. Piaggesi, D. Salsi, *"The Structuring of a Wireless Internet Application for a Music-On-Demand Service on UMTS Devices*", In Proceedings of the ACM Symposium on Applied Computing (SAC'02), ACM Press, Madrid (Spain), March 2002.

(Roccetti *et al.*,2002b) M. Roccetti, V. Ghini & S. Cacciaguerra, "*Interceptor: a Tool for Enabling Large-Scale Simulation of Software Distributed Applications in a Networked Laboratory*", In Proceedings of the 6th World Multiconference on Systemics, Cybernetics and Informatics (ISAS/SCI 2002), IIIS Press, pp. 242-246, Orlando, FL, USA, July 2002.
(Roccetti *et al.*,2002c) M. Roccetti, P. Salomoni, V. Ghini, S. Ferretti, S. Cacciaguerra and A. Sorcinelli, *"MoKa: a Wireless Internet Application for Delivering Mobile Karaoke on UMTS Devices"*, In Proceedings of the IASTED International Conference on

Communications, Internet and Information Technology (CIIT'02), St. Thomas, Virgin Islands (USA), November 2002.

(Roccetti *et al.*, 2003a) M. Roccetti, P. Salomoni, V. Ghini, S. Ferretti & S. Cacciaguerra *"Delivering Music over the Wireless Internet: from Song Distribution to Interactive Karaoke on UMTS Devices",* in Handbook of Wireless Internet, (B. Furht, M. Ilyas eds.), CRC Press, pp. 537-565, Boca Raton, USA, March 2003.

(Russell, 1999) R. Russell. The Linux IP firewall chains page, 1999, http://www.rustcorp.com/linux/ipchains/

(Russell, 2002) R. Russell, *"NAT HOWTO*", *"Netfilter Extensions HOWTO*", *"Netfilter Hacking HOWTO*", *"Packet Filtering HOWTO*, February 2002, http://netfilter.samba.org/documentation/index.html#HOWTO

(Salomoni, 2003) P. Salomoni, "*A Multimedia Mobile City Guide for Outdoor Learning*", In Proceedings of the ICSEE/Western MultiConference on Computer Simulation (ICSEE/WMC'03), Orlando, FL, January 2003.

(Satyanarayanan *et al.*, 1997) M. Satyanarayanan and B. Noble, *"The Role of Trace Modulation in Building Mobile Computing Systems"*, In Proceedings of the 6th Workshop on Hot Topics in Operating Systems, Cape Cod, MA, May 1997.

(Segal *et al.*, 2002) M. Segal, K. Akeley, The OpenGL Graphics System: A Specification (Version 1.4), http://www.opengl.org

(Sibal *et al.*, 1994) S. Sibal, A. DeSimone, *"Controlling alternate routing in general mesh packet flow networks"*, In Proceedings of the ACM SIGCOMM'94, pp.136–147, 1994.

(Sidhu *et al.*, 1993) D. Sidhu, T. Fu, S. Abdallah, R. Nair, R. Roltun, *"Open shortest path first (OSPF) routing protocol simulation"*, In Proceedings of the ACM SIGCOMM'93, pp. 53–62, 1993.

(SSF, 2004) SSF version 2.0, Scalable Simulation Framework, January 2004, http://www.ssfnet.org/

(Stevens, 1998) W. R. Stevens, "*Unix Network Programming, Networking APIs: Sockets and XTI, Volume 1"*, Prentice-Hall, 1998.

(Swarm, 1999) Swarm official site, 1999, http://www.swarm.org

(Vahdat *et al.*, 2002) A. Vahdat, K. Yocum, K. Walsh, P. Mahadevan, D. Kostic, J. Chase, and D. Becker, *"Scalability and Accuracy in a Large-Scale Network Emulator"*, In Proceedings of the 5th Symposium on Operating System Designand Implementation, Boston, MA, december 2002.

(Veloso *et al.*, 2002) E. Veloso, V. Almeida, W. Meira, A. Bestavros, S. Jin, *"A Hierarchical Characterization of a Live Streaming Media Workload"*, In Proceedings of the ACM SIGCOMM Internet Measurement Workshop, 2002.

(Warglien *et al.*, 1995) M. Warglien and M. Masuch, "The logic of organizational disorder", Berlin, De Gruyter, 1995.

(Watts, 1999) D. J.Watts, *"Small Worlds"*, a Princepton Studies in Complexity Book, 1999.

(Waxman, 1988) B. M. Waxman, *"Routing of Multipoint Connections",* in IEEE journal Selection Areas Communication, vol. 6 no. 9, pp 1617-1622, 1988.

(White *et al.*, 2002a) B. White, J. Lepreau, S. Guruprasad, *"Lowering the Barrier to Wireless and Mobile Experimentation"* in Proceedings of the First Workshop on Hot Topics in Networks (HotNets-I), October 2002.

(White *et al.*, 2002b) B. White, J. Lepreau, L. Stoller, R. Ricci, S. Guruprasad, M. Newbold, M. Hibler, C. Barb, A. Joglekar, *"An Integrated Experimental Environment for Distributed Systems and Networks"*, in Proceedings of the Fifth Symposium on Operating Systems Design and Implementation *(OSDI '02)*, pp. 255-270, December 2002.

(Wilensky *et al.*, 2000) U. Wilensky, W. Stroup, *"Networked Gridlock: Students Enacting Complex Dynamic Phenomena with the HubNet Architecture"*, In Proceedings of the Fourth Annual International Conference of the Learning Sciences, Ann Arbor, MI, June 2000.

(Xu *et al.*, 2001) K. Xu, M. Takai, J. Martin and R. Bagrodia, *"Looking Ahead of Real Time in Hybrid Component Networks"*, In Proceedings of Workshop on Parallel and Distributed Simulation (PADS'01), Lake Arrowhead, CA, May 2001.

(Zegura *et al.*, 1996) E. Zegura, K. Calvert, S. Bhattacharjee, *"How to Model an Internetwork"*, In Proceedings of IEEE INFOCOM, April 1996.

(Zegura *et al.*, 1997) E. W. Zegura, K. L. Calvert, M. J. Donahoo, *"A Quantitative Comparison of Graph-Based Models for Internet Topology"*, In IEEE/ACM Transaction on Networking, vol. 5 no. 6, December 1997.

(Zeng *et al.*, 1998) X. Zeng, R. Bagrodia, and M. Gerla, *"GloMoSim: a Library for Parallel Simulation of Large-scale Wireless Networks"*, In Proceedings of the 12th Workshop on Parallel and Distributed Simulations, pp. 154-161, Banff, Canada, May 1998.

(Zheng *et al.*, 2002a) P. Zheng and L. M. Ni, *"EMPOWER: A Scalable Framework for Network Emulation"*, In Proceedings of the 2002 International Conference on Parallel Processing (ICPP'02), In Press, Vancouver, Canada, 2002.

(Zheng *et al.*, 2002b) P. Zheng and L. M. Ni, "*Experiences in Building a Scalable Distributed Network Emulation System"*, In Technical Report MSU-CSE-02-18, Department of Computer Science and Engineering, Michigan State University, 2002.

(Zheng *et al.*, 2002c) P. Zheng and L. M. Ni, *"EMWIN: Emulating a Mobile Wireless Network using a Wired Network"*, In Proceedings of the Workshop on Wireless Mobile Multimedia (WOWMOM'02), Atlanta, GE, September 2002.