BISON
IST-2001-38923

Biology-Inspired techniques for
Self Organization in dynamic Networks

Architecture of the Simulation Environment

Deliverable Number:	D11
Delivery Date:	June 2003
Classification:	Public
Contact Authors:	Alberto Montresor, Gianni Di Caro, Poul E. Heegaard
Document Version:	Final (January 29, 2004)
Contract Start Date:	1 January 2003
Duration:	36 months
Project Coordinator:	Università di Bologna (Italy)
Partners:	Telenor Communication AS (Norway), Technische Universität Dresden (Germany), IDSIA (Switzerland), Santa Fe Institute (USA)

Project funded by the European Commission under the Information Society Technologies Programme of the 5th Framework (1998-2002)

Abstract

This document describes the architectural design of the simulation environment for the BISON project. Given the different simulation requirements (in terms of scalability, network models, level of detail) needed by ad-hoc networks and overlay networks, we propose two different architectures. The simulation architecture for ad-hoc networks builds up on an existing packet-level simulator, namely the publicly available GloMoSim. The use of its commercial and enhanced version QualNet is also envisaged. On the other hand, the architecture of the overlay network simulator is based on a completely custom design specifically optimized for the simulation of very large networks and the analysis of graph-theoretical structures.

Contents

1 Introduction

The objective of workpackages WP2 and WP3 is to develop algorithms and models inspired by biological systems that can be used to solve several optimization and control problems that arise in dynamic communication networks, such as the routing of messages in ad-hoc networks and the distribution of workload in grid systems. All these problems show a high degree of *stochasticity* which is coupled with their intrinsic *combinatorial complexity*. Orthogonal to these two dimensions, we find *dynamicity*: the network topology may evolve rapidly, due to mobility and arrivals/departure of terminals; traffic patterns may change, due to variations in user behavior and location of information. The provided system solution must be able to adapt, following the dynamic evolution of the underlying environment.

Biology-inspired systems have shown their ability to cope with these challenges (complexity, stochasticity, dynamicity). Their solution strategy is typically based on the use of local information to co-operatively produce the answer to the global optimization problem. In general, under realistic, non-asymptotic time/resources, the solution found is sub-optimal, since these systems usually restrict the search to just few regions of the usually huge search space; precisely to those regions which are judged as the most promising ones according to some *heuristic* criteria.

The question is how to evaluate the goodness of the proposed solutions when the optimum is not known and multiple criteria have to be jointly optimized over time, as it is often the case for the class of network problems considered in BISON. More in general, the problem is how to study and possibly predict the general behavior of the complex stochastic dynamic systems considered in BISON, as well as the performance of the biologically-inspired solutions that we are going to propose to solve optimization and control tasks on these same systems.

Stochastic discrete-event simulation is the main tool used to study the characteristics and predict in some extent the behavior of communication networks, and, more in general to study complex stochastic dynamic systems modeling real-world situations of practical interest.

Simulation is an extremely powerful and flexible tool, since it potentially allows the study of a large number of system configurations whose levels of complexity, stochasticity and dynamicity can be sampled in a controlled way from the universe of the possible system configurations. Clearly, to exploit these intrinsic potentialities, the simulation system must be designed to be able to capture the distinctive traits of the real system under consideration along all these axes. The exact definition of which these *distinctive traits* are and which is the *level of detail* appropriate to draw conclusions that are at the same time statistically sound, meaningful and of practical interest, are major issues for the selection of the precise characteristics of the simulation environment. These issues are extensively discuss in the following of this document in relationship to the BISON's objectives.

It could be argued that simulation is not the only tool available for the study of the systems considered in BISON. However, from a recent survey [38] it results that far more the 50% of the research results published during the last years in the major journals/proceedings in the field of telecommunications are obtained through the use of simulation. This situation can be explained both by the continually increasing computing power made available to the researchers to run extensive simulations and by the fact that *theoretical analysis* and *direct experimentation* are unfortunately of limited application. In fact, from one side, direct experimentation is costly

and difficult to realize in a realistic way and without affecting the normal activities of the users. On the other side, meaningful theoretical analysis can be mainly carried out either for limit cases or after making working assumption which are seldom met in real-world conditions.

However, considered the intrinsic complexity of any communication system, which is typically made of a number of tightly interacting components of different nature (e.g., users, transmission links, processing/routing nodes, communication protocols), none of the available tools (simulation, theoretical analysis, and direct experimentation) taken in *isolation* is in principle sufficient in order to carry out studies which are are the same time meaningful and of practical interest. On the contrary, results coming from the application of the different tools should be integrated in order to have a possibly better picture of the ensemble of the characteristics associated to the network system under consideration.

This is the general approach that we intend to follow in BISON. We expect to heavily rely on the use of discrete-event stochastic simulation but, at the same time, for each specific sub-goal under consideration we will select the investigation tool which appears more appropriate, integrating in a unified framework of analysis the results obtained from all the different sources. Moreover, when possible, we expect to use different investigation tools at the same time, mixing microscopic and macroscopic levels of detail. Clearly, to have this possibility we need simulation platforms able to possibly accommodate *different levels of detail and modeling* in the same simulation.

Purpose and organization of the document

This document is aimed at presenting the characteristics of the simulation environment to be used in BISON. The precise characteristics of this environment depend on several factors, ranging from the stated targets of our research, to specific issues associated to the intrinsic properties of the real-world systems to be studied, up to the "common practices" for simulation studies that can be observed in the different research areas included in BISON.

We are going to discuss in two separate sections the characteristics of the simulation environment to be used for the case of *mobile ad hoc networks* and those of the simulation environment to be used for the case of *overlay networks*.[1] In fact, we have realized that the requirements for simulation of mobile ad hoc networks are completely different from those originated by overlay networks. First of all, the scale to be considered differs for several order of magnitudes; in particular, overlay networks may be composed of millions of nodes, while mobile ad hoc networks are usually supposed to be much smaller. Furthermore, as mentioned in Deliverable D01, the same networking infrastructure is different. In the case of overlay networks, message routing is understood as part of the networking infrastructure and considered relatively cheap, while in the case of mobile ad hoc networks routing is at the core of the system and it is considered as very difficult and expensive. This makes the two cases radically different. For overlay networks the more interesting part is the structure of the application because routing makes the infrastructure practically fully connected. For mobile ad hoc networks the more interesting part is the dynamics obtained by the tight and recursive interaction between routing decisions and traffic and mobility patterns.

[1] With the term overlay, following the terminology used in Deliverable D01, we mean distributed decentralized systems using the Internet as networking infrastructure; these include structured and unstructured peer-to-peer networks and grid systems.

The consequence of this is that the required level of simulation detail is completely different in the two cases. Overlay networks can be reasonably represented simply as graphs whose nodes have a bounded set of connections with other nodes, but can potentially communicate with every other node, once they discover their identifier or address. Nodes and communication channels are subject to failures and present different properties, but this can be easily represented on graphs as weights associated to nodes and edges. On the other hand, any serious study of the dynamics in mobile ad hoc networks requires modeling in some detail most of the layers of the OSI protocol stack, as well as the users mobility and the physical characteristics of the environment in which the users interact.

For these reasons, *two independent simulation environments* are adopted in BISON. One is specialized for simulation of mobile ad hoc networks, and is described in Section 2, while the other is specialized for simulation of overlay networks and is described in Section 3.

In the case of mobile ad hoc networks, the choice is to use an *already available* simulator as a simulation framework of reference. This simulation framework will be the harness on top of which the complete simulation architecture for mobile ad hoc networks will be built by adding new models, protocols, algorithms, and by adapting the existing simulator's components to the BISON's objectives. On the contrary, in the case of overlay networks the choice is to implement *from scratch* a custom simulator tailored to the specific BISON's needs for the study of overlay networks. This differentiation, not only between the two research domains but also between the strategies followed in the selection of the simulation environment, is reflected in a substantial differentiation between the contents of the section devoted to mobile ad hoc network and that devoted to overlay networks. In the case of mobile ad hoc networks the focus is on the identification of meaningful criteria to rank the different available simulators and on the detailed description of their properties with respect to these criteria, as well as in the perspective of expanding/modifying the simulator. On the other side, in the case of overlay networks, the focus is on the discussion of the characteristics of the custom simulation environment, while other existing simulators are only quickly reviewed.

2 Simulation of Mobile Ad Hoc Networks

Generally speaking, *simulation* (e.g., [5, 4]) can be defined as the process of designing a model of a real system and conducting experiments with this model for the purpose of understanding the behavior of the system and/or evaluating various strategies for the operation of the system. Thus it is critical that the model be designed in such a way that the model behavior mimics the response behavior of the real system to events that take place over time.

In particular, our interest is on *discrete-event* simulations, according to the intrinsic discrete nature of the events (e.g., start/end of user sessions) happening in a communication network. A discrete-event simulation model is such that the model's state variables change only at those discrete points in time at which events occur. Events occur as a consequence of activities and delays. Simulation entities may compete for system resources, possibly joining queues while waiting for an available resource. A discrete-event simulation model is conducted over time (*run*) by a mechanism that moves simulated time forward. The system state is updated at each event along with capturing and freeing of resources that may occur at that time.

There are several logical and practical steps that have to be followed in any simulation study [5]. Without entering into details, three major phases of the process of building and using a simulation model can be identified: (i) the *definition of the model* of the real system, (ii) the *collection of data* from the real system itself, and (iii) the definition of the *experimental design* and the actual *run* of the simulation using the defined model.

Definition of the characteristics of the simulation model. The first phase concerns the definition of the characteristics of the simulation model in relationship to: (a) the characteristics of the chosen abstract *representation* of the real system under investigation, (b) the objectives of the study. Aspect (a) plays the role of a *meta-model*. There is no a unique or even a "more natural" way of representing a whatever system. The experimenter implicitly or explicitly has to decide the depth of the resolution at which the real system has to be considered, possibly as a function of the ultimate objectives of its study. The meta-model defines which are the atomic parts of the system in terms of entities, interactions and structures. Informally speaking, the meta-model is how the reality is seen through the lenses of the experimenter.

The simulation model is in turn designed on the basis of the adopted meta-model. The simulation models that we are mainly considering here are *microscopic* (or *mechanistic*) ones, not *macroscopic* (or *phenomenological*) ones. In general, models at multiple scales, ranging from microscopic to macroscopic, can serve to investigate different properties of the same real system. Clearly, it is important to be able to understand which is the right level of detail necessary for a specific study. The inclusions of unnecessary details can at the same time hide interesting aspects and waste computing resources. On the contrary, with a too low level of detail the drawn conclusions can easily bear no relationship with reality.

Collection of input/output data from the real system. The second phase concerns the practical acquisition of *input* and *output data* from the real system. Input data are necessary to set the main parameters of the model (e.g., inter-arrival rates of traffic sessions, processing times, failure rates), while output data are necessary to *validate* the significance of the obtained results.

Definition of the experimental design and run of the simulations. The third, final phase, addresses the issue of the definition of the whole experimental design and of the other necessary operational details: the number of runs, the duration of the simulations, the measures of effectiveness to be used, the data that have to be collected, the characteristics of the random number generators, etc. The experiments must be designed such that the observed results can provide *statistically sound* answers to the stated objectives of the study. Clearly, the computing resources available must be adequate to practically run the selected number of simulations under the defined experimental conditions.

It is apparent that, in general, it is a non-trivial task to design and run a simulation model which is in turn able to provide results which are significant for the real system under consideration. This fact is even more true for the case of telecommunication networks, given the intrinsic complexity and variety of telecommunication systems, as well as the major role played in these systems by the users, with their rather unpredictable and ever changing behaviors and

expectations. In some sense, it is reasonable to expect that slightly different conceptual models of the same physical network can easily generate dramatically different simulation results. In general, this is an intrinsic characteristic of highly non-linear complex systems. Clearly, such a situation asks for simulations which are carefully designed in all their aspects, in particular for those aspects concerning with statistical soundness, comparison issues and overall validation. Unfortunately, in [38] it is claimed that this does not happen so often as it would be required, and the issue of the general *credibility* of the final results of simulation studies is raised up.

Anyhow, in spite of the many, obvious, difficulties, simulation is still the primary tool in the study of communication networks and we are going to heavily rely on it. However, it is important to be aware of the *limitations* and of the problems related to the simulation approach, in order to avoid the most common methodological errors, when possible. This is the attitude we want to have in BISON.

In the case of mobile ad hoc networks, the limitations and the problems related to the use of simulation are even amplified with respect to the cases of other types of networks (e.g., wired networks). In the following sections we discuss the characteristics of a generic mobile ad hoc network, pointing out the fact that there is a really wide spectrum of possibilities in building up microscopic models for such networks. This derives both from the intrinsic characteristics of mobile ad hoc networks, made of a number of different components, and from the practical *absence* of real-world implementations of mobile ad hoc networks. There are no real-world *reference* models that could limit the range of possible alternatives. This means also that the step in the simulation process concerning the acquisition of input/output data from the real system, as well as, the final step concerning the validation, cannot be carried out *properly*, if not at all. Clearly, this situation is a bit embarrassing, from a methodological point of view.

Anyhow, in spite of these difficulties, some simple reference models have *emerged* in the current literature, in the sense that most of the research works make use of the same, similar, models. This fact is mainly the effect of the use of the same few simulators (NS-2, GloMoSim) which offer a limited set of pre-built models, as well as the explicit choice of study simple, even if not realistic, situations. In practice, it is common to consider networks with only few tenth of nodes moving quite slowly according to uncorrelated decision schemes in open flat space areas.

We plan to use these same models as main reference, while, at the same time, we keep open the possibility of extending and/or modifying them in order to test situations which are in our opinion more realistic. Clearly, at this aim, we need to have a simulation environment flexible enough for testing scenarios that can differ in terms of both dimensions, and structural and dynamic characteristics.

2.1 General Characteristics of the Network Model and Methodological Choices

2.1.1 The Components of a Mobile Ad hoc Network in the Real World

In order to study the application of a whatever function (basic or advanced, in the BISON terminology) on a mobile ad hoc network, it is customary first to define what we mean by an *instance of a mobile ad hoc network* . We need to define our abstract representation (the meta-model, using the previous terminology) of such class of networks, making explicit all the components that can be selected to be included in the simulation model. Our representation includes the

following major components.

- A set of mobile devices, called *nodes*, with processing and wireless transmission capabilities powered by on-board batteries The number of nodes participating to the network is in general not constant, at any time nodes can leave or join the network. In addition to the transmission capabilities, nodes can be also equipped with location devices (e.g., GPS) or other optional devices of different nature.
- According to the fact that nodes are communication devices, their internal architecture is defined in terms of the *OSI protocol stack*. This means that node communications are regulated by the characteristics associated to the OSI protocol layers: physical, data link, medium access control (MAC), network, transport, session, presentation, application.
- Communications happen through a *wireless* interface, therefore, the whole issue of the physical and technological aspects of *radio emission, propagation and reception* has to be taken into account. While emission and reception are strictly related to the physical characteristics of the node device, radio propagation is affected by the physics of the environment in which the network is placed.
- The network is seen as embedded in an *environment* with well defined physical characteristics which affect both the propagation of the radio signals and the mobility patterns of the nodes.
- Every node is associated to a single user which moves inside the environment. Therefore, the node *mobility patterns* depend on the characteristics of both the users and the environment.
- Node *energy power* relies on the use of on-board batteries. Every action involving the use of the radio channel, as well as the same fact that the device is switched on, has an energetic cost which affects the available energy power.
- Users generate *data traffic* in terms of open sessions between pairs or groups of users. Traffic patterns can in general assume arbitrary forms. The same node can open several traffic sessions during the length of the observation period. The generation of the traffic patterns is seen as a non-terminating process [10], in the sense that the network operations do not have "natural" starting and ending points.

2.1.2 The Simulation Model of a Mobile Ad hoc Network Adopted in BISON

It is well understood that in general terms all the physical and logical components listed in our abstract representation are equally important to determine the overall dynamics of the network system. Therefore, we admit that all these components should be also included in our simulation model. However, the open question is: which is the right *level of detail* and which are the core specific characteristics that shall be used to model inside the simulation architecture the different network components? For example, do we need to implement all the OSI layers and let every packet passing through them at each network node, or, is it sufficient to limit ourselves to the MAC, network, and session layers? How faithful has to be the implementation

of each selected OSI layer? Is it a meaningful choice to model the environment simply as a small open flat area? Must the mobility of the nodes reflect some precise model of human behavior, or, can it be modeled in terms of little more than a random walk? Do the transmission range and the modality of access to the medium have to be adapted to the values of devices currently available on the market, or, can it be envisaged the use of smarter and more powerful technologies? Should the input traffic patterns be defined in the generic terms of negative exponential distributions, or, should they be close to the patterns usually encountered in GSM or other wireless networks? And so on . . . The "appropriate" choice always depends on several factors, starting from the specific objectives of the research.

Wide experience with the important components of wired networks over the last 30 years allows nowadays significant abstraction. For example, point-to-point links are often represented simply by bandwidth and delay with a queue; framing, coding, and transmission errors are simply ignored or mathematically modeled. On the other side, the younger field of mobile ad hoc networks provides less guidance on what abstractions are appropriate. Low-level details can have a large effect on performance, but detailed simulations can be very expensive (e.g., radio propagation). Again, the absence of practical real-world implementations complicates the whole issue. Finding a good trade-off between *accuracy*, *statistical soundness* (which needs the ability of running the simulations several times) and *practical/scientific interest* is not a trivial task.

2.1.3 Selection of the Right Level of Detail

Several arguments can be raised pro and against the use of detailed versus abstract simulations (good discussions on the subject can be found in [22, 15]). Clearly, the use of detailed/realistic models in principle can truly help to understand the phenomena under study. In particular, since telecommunication networks are artificially built, it is possible to do reverse engineering and to bring inside the model the correct functioning behavior of the single hardware and software components. In some sense this is a fortunate situation with respect to the modeling of natural systems. However, it is pretty clear to practitioners in the field that a *fully realistic* microscopic simulation is not possible.

A simple example to stress the importance of the use of a high level of detail is that concerning the well-known *hidden terminal effect*, which cannot be evidenced without considering concurrent transmission and a realistic implementation of the access strategy to the transmission medium. On the other side, there are several reasons for intentionally choosing a high level of abstraction. Distillation of a research question to its essence can provide insights not colored by arbitrary details of specific proposed solutions. For example, although multiple resource reservation and quality-of-service protocols have been proposed, the study in [7] only make use of a very abstract service model to focus on the central issue of the benefits of reservations and to draw meaningful conclusions. When exploring a new area where many issues are unclear, as it is the case for mobile ad hoc networks, the need to quickly explore and compare a variety of alternatives can be more important than a detailed result for a single specific, realistically designed, scenario. A more abstract simulation can also make the effects of a change in the algorithm distinct, where they would be obscured by other effects in more detailed simulations. Moreover, as it is intuitive, omission of details can improve computational performance by multiple orders of magnitude (e.g., [24]). This is a critical argument, since better computa-

tional performance can allow simulations to *scale*, as well as to consider more realistic scenarios. Considering that the relative performance of ad hoc routing protocols seem to differ with the increasing of the number of nodes [16], the issue of scaling simulations appears even more fundamental. Clearly, on the other side, the lack of detail can also cause answers which are either simply *wrong* or *inapplicable* (technically correct but providing an answer to part of the design space that may not be sensible or relevant).

In the literature there are several studies [22, 15, 23, 25, 8] concerning the impact of different levels of details and of different models for the different components of the simulation. For example, in [22] five different cases are studied, considering the impact of detail in relationship to the characteristics of the energy consumption, radio propagation, and data visualization models. In [8] the performance evaluation of ad hoc routing protocols are evaluated in function of different possible models of node mobility. The effect of different modeling design in the MAC layer, as well as, in the setting of the related parameters, is extensively studied (e.g., [49, 6]). If all these examples refer to the analysis of the impact of the modeling detail for one single aspect (either energy or radio propagation or mobility or MAC protocol), there are also studies comparing the answers provided by *different simulators*, that is, taking into account the *whole* set of aspects relevant for the study of mobile ad hoc networks. In [9], the performance of the same simple flooding algorithm are compared using three different simulators widely in use (NS-2, OPNET and GloMoSim). All these studies confirms the fact that little differences, at any level of modeling, might result in different behaviors, and, consequently, in possibly different drawn conclusions. However, this does not imply that the use of simulation is in some sense meaningless. Simulation is still the most powerful tool to investigate the practical behavior of communication networks. Although, extreme care must be adopted selecting/implementing the characteristics of the simulator and at the moment of drawing conclusions.

2.1.4 The methodological approach followed in BISON

From all the previous discussions it clearly results that if it is true that simulation is probably the major tool to study mobile ad hoc networks, it is also true that the number of problems related to the modeling and validation aspects call for an extreme care in designing and using simulations, as well as for an extreme caution at the time of drawing conclusions. We judge as an important preliminary achievement the fact that we have got fully *aware* of the existence of these troublesome issues concerning both the investigation tools that we are going to heavily use to generate data and the scientific methodology that we are going to use to analyze these same data in order to draw conclusions and to plan further studies.

According to the BISON's objectives and to the discussed characteristics of the still young research field of mobile ad hoc networks, it is clear that we have to look for a simulation architecture which is flexible enough to easily accommodate different, selectable, levels of detail. We intend to study the systems at possibly different scales of detail, and we want to left open the possibility to use, for similar levels of detail, different sets of operational models for the single components of the mobile ad hoc network. In this sense, the overall characteristics of the simulation environment must *match* the specific set of different *objectives* of our research. Since in BISON we intend to span from issues which are strictly related to the topological and dynamic properties of the network, to issues concerned with more high-level services that can be delivered by the network as a whole, it is clearly necessary to have a simulation architecture

flexible enough to allow us to conduct such studies at multiple scales.

In this perspective, we have to look for an *open* and *modular* simulation architecture. We ruled out the choice of developing from *scratch* a new custom simulator. A satisfactory and well done job in this sense would take some time, which is not worth to spend since a number of good network simulators, both commercial and public domain, are already available from the Internet and from software telecommunication companies. We do *not* need "yet-another-simulator"from scratch. We can conveniently use an available simulator as the *initial platform* on which we can build the complete BISON simulation architecture for mobile ad hoc networks. It is in this sense that we need a truly open and modular system. BISON addresses specific issues and has specific requirements that we will bring inside the chosen simulation platform, expanding and modifying it.

Broadly speaking, discrete-event simulators for mobile ad hoc networks can divided in two classes. Those which are designed to simulate anything that can be mapped to active components that can access local resources and communicate by passing messages, and those which are specifically designed as network simulators. These latter usually have pretty detailed and hard-coded concepts about nodes, agents, protocols, links, packet representation, network addresses etc. Moreover, they usually come with a set of important components (e.g., routing or transport protocols) already implemented. If this is clearly an advantage, at the same time it is usually more difficult to modifying hard-coded aspects to adapt them to more specific needs. Again, a good trade-off must be found between flexibility, modifiability and availability of tools/components ready to use. In the following we discuss the characteristics of a set of four simulators which we have selected as possible candidates. These simulators are either the simulators most widely in use in the telecommunication community or new simulators which are attracting an increasing interest in the community because of their particularly good software design and performance.

Before discussing and comparing the four candidate simulators, it is appropriate to introduce a well defined set of features which are relevant for BISON's purposes and which can be therefore used to eventually score or rank the examined simulators.

2.2 Selection Criteria: Network Simulator Features that are Relevant for BISON

Types of simulations supported

We are mainly interested in *discrete-event* simulations. However, when possible, it might be interesting to carry out *trace-driven* simulation, that is, simulations using as input a time-ordered record of events (also called a "trace") from a real system.

More in general, since we have stated the intention to run studies at possibly different scales of resolution, it could be very useful in practice to have a simulator offering the possibility of switching off most of the microscopic details (e.g., the OSI chain inside the nodes) in order to carry out statistical-mechanics-like studies involving a large number of elements and a strong stochastic component. In the following we refer to this form of simulation as *generic Monte Carlo* simulation.[2]

[2] There is a clear abuse of the term Monte Carlo [45, 43] here, in fact, also in the case of the use of discrete-event or trace-driven simulation we expect to heavily use a stochastic component in the generation of traffic and mobility

Another possibility that can be envisaged is the *injection of live traffic* into the simulator and/or the injection of traffic from the simulator into the live network. Actually, these characteristics refer more to *emulators* than simulators. Some experiments in this sense for mobile ad hoc networks have been carried out in the framework of the Monarch Project at Carnegie Mellon University [25].

Computational platforms supported

According to some heterogeneity among the BISON's members in terms of used platforms, it would be helpful to have a simulator running on different platforms, in particular on both *Linux* and *Windows* systems.

Since we expect heavy computational loads associated to the simulations, *distributed*, *parallel* or *multi-threading* processing capabilities could be possibly exploited if available.

Support for the generation of topologies

All simulators have some mechanism for the *generation of network topologies*. Among the several possibilities, there are the use of special *script* or *configuration languages*, manually edited *numerical files*, and *graphical interfaces*. Some simulators do not easily permit the creation of *hierarchical topologies*, admitting only flat topologies. In some fortunate cases a special tool for the automatic controlled generation of *random topologies* is provided with the simulator.

Support for the generation and management of traffic profiles

In order to generate data traffic, it is in general necessary to use *generators of data* according to some specific target distribution (e.g., Poisson or derived from the observation of real traffic). Good simulators usually come either with a wide set of such traffic generators.

On the other side, it is also important to have tools to *profile the generated traffic* and to compute the *statistics* necessary to draw conclusions.

Monitoring support

During the execution it can be extremely useful to *monitor* the network dynamics on a per-flow, per-node, or, more in general, on the basis of some aggregation criteria. Monitoring can helped by the availability of *graphical* interfaces.

The results of monitoring can be also dumped on files to generate *traces* that can be used for further comparisons or for re-play the simulation for more careful studies of the generated dynamics.

Implemented modules for the OSI layers, mobility models, and radio propagation

A simulator coming with a rich set of already implemented and tested *modules* is preferable to a simulator coming with no implemented modules. However, according to the previous

patterns, and of other various aspects of the simulation.

discussions on the extreme sensitivity of the obtained results to the model characteristics, it is clear that not only the number but also the characteristics (quality) of the implemented modules are equally important.

The OSI modules that are really interesting for us are primarily those relative to *routing* protocols. It is also very important to have a reliable implementation of the *MAC* layer. The *physical* and *link* layer, if considered in the perspective of the radio emission/reception are also important. At the *application* or *session* layer it is probably enough to have simple FTP-like or CBR modules. For what concerns the *transport* layer, only a *best-effort* (unreliable) UDP-like service is probably sufficient. A TCP-like service would be in principle very useful in the context of mobile ad hoc networks, but unfortunately there are a number of difficult problems related with this issue. Therefore, we will likely not use any form of reliable transport protocol.

Modules implementing *mobility models* are usually rather simple to implement. However, it can be helpful to have some mobility module coming with the simulator, because it is likely that such module is being used also by other researchers, making comparison studies easier. This argument is valid in general. Modules coming with a simulator are likely to be used by several research groups. Therefore, they "automatically" become basic models of reference.

Radio wave propagation models coming with simulators are usually straightforward implementation of basic standard results from electromagnetism studies. An accurate propagation model should take into account the morphology and the physics of the environment. This would be overwhelming expensive in computational terms and incredibly complicate.

Flexibility, modifiability, extensibility and scaling issues

It has been already stressed that we want an architecture such that it is possible to select a desired level of detail, components can be easily switched on and off, current models can be substituted or modified, new models can be included. Shortly, the simulator must a truly *open modular* architecture.

In particular, scalability is an critical issue. In principle all simulators allow to add more nodes and to increase the computational burden associated to the traffic patterns and to the other dynamic aspects of the network (mobility, appearing/disappearing of nodes, broken paths, etc.). However, in practice some simulators scale much better than others, that is, some simulators have some practical limitations that must be evidenced, since we might consider to study networks with large number of nodes.

General usability

The *software design* of the simulator, as well as the *programming tools* required to use the simulator greatly affect the *usability* of the simulator. It is preferable to have a simulator requiring the use of tools and concepts already in the knowledge background of the BISON members.

The availability of *graphical interfaces* can speed up the operations on the simulator and are therefore seen as a general positive aspect.

Last but least, the overall quality of the available *documentation* and *technical support* are important factors to be able to quickly and effectively learn how to use and control the simulator.

Level of acceptance of the simulator by the network community

According to the discussed sensitivity of the obtainable results to the specific simulator used, it is clear that choosing a simulator widely in use and accepted by the community allows to produce results which are easier to compare with those present in the literature, and, in some sense, which are more easily "accepted" by the scientific community.

Type of software license

It is pretty obvious that *public domain* simulators are in principle more "convenient". However, *commercial* simulators must also considered and selected if they offer a strategical and clear set of advantages over the available public domain ones.

Table 1: Features of simulation environments for mobile ad hoc networks which are considered relevant for BISON's purposes and which are used to select the simulation environment.

Feature relevant for BISON	**Possible realizations/alternatives/indicators**
Supported simulation types	Discrete-event, trace-driven, Monte Carlo
Computational platforms	Linux, windows, parallel and distributed systems
Admitted topologies	Flat, hierarchical, random
Editing of topologies	Script languages, data files, graphical interfaces
Data traffic generation	Sampling from probabilistic distributions, real data
Traffic profiling	Online data collection and statistical analysis tools
Monitoring support	Graphical interfaces, trace generation
Modules for the OSI layers	Routing algorithms, MAC, physical and link layers
Mobility models	Random walk, random waypoint, gauss-markov
Models for radio propagation	Open space with ground reflection, shadowing effects
Modifiability and extensibility	Open and modular software design
Scaling	Efficient management of memory and CPU resources
Ease of use	Programming tools, graphic interfaces, documentation
Scientific acceptance	Number of publications using the simulator
Type of software license	Commercial, public domain

2.3 State-of-the-art of Network Simulators for Mobile Ad hoc Networks

We identified in *OPNET, GloMoSim/QualNet, NS-2* and *Omnet++* the possible candidates to build up our simulation environment. In some sense, we can consider these simulator as particularly representative of the state-of-the-art in the field of simulators for mobile ad hoc networks. In the following of this section we discuss one by one the characteristics of these simulators and their general "compliance" to the selection criteria reported in the previous Section 2.2 and summarized in Table 1.

2.3.1 OPNET

OPNET (Optimized Network Engineering Tools) is a commercial tool from MIL3 Inc. (for more information see the web site at `http://www.mil3.com/home.html`) for modeling and simulation of communications networks, devices, and protocols. It features graphical editors and animation. It has being developed for almost 15 years. It is widely held to be the state-of-the-art in network simulation. OPNET is used by large companies. It is available for *free* for North-American universities (though this free does not include any technical support).

It can simulate all kinds of wired and several wireless networks. An IEEE 802.11 compliant MAC layer implementation is also provided. Although OPNET is rather intended for companies to diagnose or reorganize their network, it is possible to implement one's own algorithm by reusing a lot of existing components. Most part of the deployment is made through a *hierarchical graphic user interface*. Network with several *hundreds of nodes* can be managed.

The software comprises several tools and is divided in several parts: *OPNET Modeler* and *OPNET Planner*, the *Model Library*, and the *Analysis tool*. Features included in this generic simulator are an event-driven scheduled simulation kernel, integrated analysis tools for interpreting and synthesizing output data, graphical specification of models and a hierarchical object-based modeling.

The internal architecture

The modeling methodology of OPNET is organized in a hierarchical structure. At the lowest level, which is the one really customizable, `Process models` are structured as a *finite state machines*. State and transitions can be graphically specified *graphically* using state-transition diagrams, whereas conditions that specify what happens within each state are programmed with a C-like language called *Proto-C*. Process models, and built-in modules in OPNET (source and destination modules, traffic generators, queues, wireless, ...) are then configured with menus and organized into data flow diagrams that represent nodes by using the graphical `Node Editor`. Using the graphical `Network Editor`, nodes and links are selected to build up the topology of a communication network. The `Analysis Tool` provides a graphical environment to view and manipulate data collected during simulation runs. Results can be analyzed for any network element.

`OPNET Planner` is an application that allows administrators to evaluate the performance of communications networks and distributed systems, without programming or compiling. Planner analyzes behavior and performance by discrete-event simulations. Models are built using

a graphical interface. The user only chooses pre-defined models (from the physical layer to the application) from the library and sets attributes. The user can define new models, but he/she has to contact the MIL3's modeling service.

The `Modeling libraries` are included with OPNET Modeler and OPNET Planner and contain protocols and environments (e.g., ATM, TCP, IP, Frame Relay, FDDI, Ethernet, IEEE 802.11, support for wireless), link models such as point-to-point and bus, queuing service disciplines such as First-in-First-Out (FIFO), Last-In-First-Out (LIFO), priority non-preemptive queuing, shortest first job, round-robin or preempt and resume.

Specific support for mobile ad hoc networks

From the point of view of predefined components specific for mobile ad hoc networks, the *Wireless* (local, GSM and ad hoc) module of OPNET comes with most of all the necessary components in terms of mobility and radio propagation models, as well as in terms of full protocol stack, including MAC (802.11) and popular routing algorithms like Adhoc On-demand Distance Vector (AODV) and Dynamic Source Routing (DSR). Featuring an open-source Longley-Rice model, the *Terrain Modeling* module can replicate a number of known atmospheric or topological conditions and their combined impact on signal propagation. The Wireless module features also *parallel simulation* capabilities.

General summary of OPNET's features

OPNET is a well-established and highly professional product. Support for mobile ad hoc networks is essential but of high-quality and sufficient to carry out serious studies. However, the results reported in [9] are in this sense a bit puzzling. In the paper a study of the behavior of a simple flooding algorithm is carried using OPNET, NS-2 and GloMoSim. While the results provided by NS-2 and GloMoSim are rather similar between them, the results provided by OPNET are significantly different. It is definitely unclear which is the "right" behavior...

The graphical interface simplifies most of the routine operations, while the development of new models always requires the definition of a finite state machine. If this fact can be seen as a difficulty at the beginning, it is also true that finite state machine are undoubtedly an expressive and effective tool for modeling. In OPNET, differently from NS-2 and GloMoSim, it results quite easy to describe an application that bypasses part of the protocol stack. This aspect can be quite important to speed up and/or to reduce the level of unnecessary detail during the simulations.

The performance seem to scale quite well, but there are not enough data in the current literature to make a more precise statement in this sense in the context of mobile ad hoc networks.

2.3.2 GloMoSim/QualNet

GloMoSim is a scalable simulation *library* designed at UCLA Computing Laboratory to support studies of *large-scale network* models, up to millions of nodes, using *parallel* and/or distributed execution on a diverse set of parallel computers (with both distributed and shared memory). It has been designed for wireless and wired networks systems, but it currently supports only

protocols for purely *wireless* networks. GloMoSim is a library for the C-based parallel discrete-event simulation language *PARSEC* (`pcl.cs.ucla.edu/projects/parsec`). One of the important distinguishing features of PARSEC is its ability to execute a discrete-event simulation model using several different asynchronous parallel simulation protocols on a variety of parallel architectures. PARSEC is designed to cleanly separate the description of a simulation model from the underlying simulation protocol, sequential or parallel, used to execute it.

GloMoSim is build according to the *OSI layered* approach. A *common, standard, API* between every two neighboring models on protocol stacks is predefined to support their composition. These APIs specify parameter exchanges and services between neighboring layers. This allows the rapid integration of models developed at different layers by different people. Actual operational code can also be easily integrated into GloMoSim (e.g., this has already been done importing the BSD implementation of the TCP suite inside the library). If all protocol models obey the strict APIs defined at each layer, it is be feasible to simply swap protocol models at a certain layer (say evaluate the impact of using CSMA rather than 802.11 at the MAC protocol) without having to modify the models for the remaining layers in the stack. That is, the modular implementation enables consistent comparison of multiple protocols and of different levels of detail at a given layer.

To run GloMoSim, the PARSEC compiler is clearly needed. To develop new protocols in GloMoSim, the user should have some familiarity with PARSEC, but it is not required an expert-level knowledge. For most protocols it is sufficient to write purely C code with the addition of few PARSEC functions.

A number of protocols have been developed at each layer, and models of these protocols or layers can be deployed at different levels of granularity and detail. are currently available in GloMoSim. Telnet, ftp, CBR, HTTP, replicated file system for what concerns applications. While at the transport layer the following models are available: TCP (FreeBSD), UDP, TCP (Tahoe). For unicast Routing the available algorithms are: AODV, Bellman-Ford, DSR, Fisheye, Flooding, LAR, NS DSDV, OSPF, WRP. The models available for the MAC layer are CSMA, IEEE 802.11, FAMA, MACA. The radio interface uses omni-directional antennas with and without capture capability. Radio wave propagation can be modeled according to Free space, Rayleigh, Ricean, or SIRCIM models (including obstructions and building-type situations). Mobility models include different variations of the basic Random waypoint, Random drunken, ECRV, BBN, Path-loss matrix, and group mobility.

A platform-independent tool coded in JAVA is available for both real-time and trace-based debug, monitoring and statistics report.

The internal architecture

GloMoSim has a layered *architecture*. Simulation is a collection of network nodes, each with its own protocol stack parameters and statistics. In turn, each layer, is an object with its own variables and structures. Messages can be exchanged between *nodes* and *layers* at any desired level of grouping and granularity. The synchronization among the events is insured by self-scheduled (timer) messages. The library comes with a complete suite of simple functions, based on standard APIs, for the creation, transmission and manipulation of messages. Clearly, new types of events, as well as of messages and node/layers can be created in a rather straightfor-

ward way. Adding a model or protocol to a layer requires: (i) an *initialization function* to allocate and initialize model specific data, (ii) a *finalization function* that defines the output statistics from the simulation run for the model, and (iii) a *simulation event handling function* that performs simulation actions when scheduled with an event. New user-defined function can be easily plugged into every layer by means of a skeleton interface code file to be modified by the user. In this way, users do not have to keep reinserting local changes in GloMoSim files for every GloMoSim version, and it makes easy to customize the interface file to call the new functions.

The internal architecture of GloMoSim has been designed with the specific aims to develop a *modular* simulation environment capable of *scaling* up to networks with thousands/millions of heterogeneous nodes and easy to port on a *parallel/distributed* computing environment. The requirements of scalability and modularity make the library design a challenging issue. GloMoSim assumes that the network is decomposed into a number of *partitions* and a *single* entity is defined to simulate a single layer of the complete protocol stack for all the network nodes that belong to the same partition. Interactions among the entities obey the corresponding APIs. Syntactically, the interactions may be specified using messages, function calls, or entity parameters as appropriate. This method supports modularity because a PARSEC library entity representing a layer of the protocol stack is largely self-contained. It encapsulates the complexity of a specific network behavior independently from other ones. This method also supports scalability because node aggregation inside one entity will be able to reduce the total number of entities, which has been found to improve also the sequential performance other than the parallel ones.

Efficient *parallel simulators* must address three general sets of concerns: efficient synchronization to reduce simulation overheads; model decomposition or partitioning to achieve load balance; efficient process to processor mappings to reduce communications and other overheads in parallel execution. GloMoSim offers several possibilities in this sense, but since at the moment we do not plan to use a parallel machine, we do not enter into details in this sense (see [52, 3] for results concerning the parallel performance of GloMoSim).

QualNet: the commercial version of GloMoSim

GloMoSim is not public domain, but it is freely available without fee for education, or research, or to non-profit agencies. However, the documentation shipped with GloMoSim is quite poor (there is even no a user manual), as well as the set of tools available to speedup the generation of topologies, to monitor the system behavior, to analyze the post-simulation results, and, more in general, to design the characteristics of and interact with the whole simulation environment.

QualNet alleviates most of the GloMoSim's flaws. QualNet (`www.scalable-networks.com`) is a commercial product from Scalable Network Technologies which is derived from GloMoSim. However, QualNet has loads of additional features with respect to GloMoSim. QualNet comes with an extensive suite of models and protocols for *both* wired and wireless networks (local, ad hoc, satellite and cellular), as well as extensive documentation and technical support. Three libraries are available: a `Standard` library which offers most of the models and protocols necessary for both research and business-oriented activities in the field of wired and wireless networks; a `MANET` library which provides additional and very specific components for ad hoc networks other than those already present in the standard library; and, a `QoS` library which in-

cludes specialized protocols for quality-of-service. To our knowledge, QualNet seems to be the most complete network simulator, in terms of available protocols, models and tools for what concerns mobile ad hoc networks. It is hard to think of missing components. Clearly, being a commercial product, all model libraries are available in binary code format: their behavior can be configured through configuration files but their core functionality cannot be modified.

The `QualNet library` is the basis for the `QualNet Developer` software suite which includes five different modules: `Animator`, which is a graphical tool for experiment setup (e.g., topology, layer protocols, traffic) and animation (e.g., watching of traffic flows and specific performance metrics, re-play of the simulation); the `Designer`, a finite state machine tool for custom protocol modeling: a state-based visual tool is used to define the events and processes of a new protocol model; `Analyzer`, a statistical graphing tool to report the statistics associated to hundreds of pre-defined or custom performance metrics; `Tracer`, which allows packet-level visualization for viewing the contents of a packet as it goes up and down the protocol stack; `Simulator`, the simulator itself, designed to accommodate high-fidelity models of networks of 10's of thousands of nodes with heavy traffic and high mobility. In addition, QualNet offers also `Parallel Developer` for managing and running parallel executions.

General summary of GloMoSim/QualNet's features

GloMoSim has several nice features. It comes with a rich suite of models, it is the only simulator among the considered ones that seems able to scale up to thousands of nodes, its highly modular design is such that it appears quite straightforward to modify and/or extend the basic functionalities. The ability to scale up and the definition of standard APIs for the modular protocol stack, opens the possibility to flexibly investigate the effect of multiple models at dramatically different levels of detail.

Analysis and visualization tools are basic but sufficient for general studies. Documentation is quite poor.

GloMoSim is likely the second most used simulator, after NS-2, in the research community. Therefore, it is widely accepted from a scientific point of view.

QualNet is based on GloMoSim but it dramatically expands its capabilities in terms of contributed models and protocols, graphical tools for experiment planning, analysis and visualization, as well as, in terms of available documentation and technical support. QualNet supports also some form of network emulation. In some sense, QualNet is quite complete as simulator.

2.3.3 NS-2

The NS network simulator (`www.isi.edu/nsnam/ns`), from U.C. Berkeley/LBNL, is a object-oriented discrete event simulator targeted at networking research and available as *public domain*. Its first version (NS-1) began in 1989 as a variant of the REAL network simulator (`www.cs.cornell.edu/skeshav/real/overview.html`), and was developed by the Network Research Group at the Lawrence Berkeley National Laboratory (LBNL), USA. Its development was then part of the VINT project (`www.isi.edu/nsnam/vint/index.html`), supported by DARPA, at LBNL, Xerox PARC, and UCB, under which NS version 2.0 (NS-2) was released, evolving substantially from the first version. The aim of the VINT was not to design

a new network simulator, but to unify the effort of all people working in the research field of network simulation. The result is that NS-2 is widely used in the networking research community and has found large acceptance as a tool to experiment new ideas, protocols and distributed algorithms. Currently NS-2 development is still supported through DARPA. NS has always included substantial contributions from other researchers, including wireless code for both mobile ad hoc networks and wireless LANs from the UCB Daedelus and CMU Monarch projects and Sun Microsystems.

At the time being, NS-2 is well-suited for packets switched networks and wireless networks (ad hoc, local and satellite), and is used mostly for small scale simulations of queuing and routing algorithms, transport protocols, congestion control, and some multicast related work. It provides substantial support for simulation of TCP, routing, and multicast protocols over wired and wireless networks.

NS-2 is suitable not only for simulation but also for *emulation*, that is, it is possible to introduce the simulator into a live network. Special objects within the simulator are capable of introducing live traffic into the simulator and injecting traffic from the simulator into the live network.

NS-2 play an important role in the research community of mobile ad hoc networks, being a sort of reference simulator. NS-2 is the most used simulator for studies on mobile ad hoc networks, and it comes with a rich suite of algorithms and models.

In this perspective, NS-2 would be the natural candidate to be used in BISON too. Unfortunately, its software architecture is such that adding new components and/or modifying existing ones is not a straightforward process. That is, in terms of *ease* to implement/test new algorithms or scenarios, NS-2 scores poorly with respect to other candidates. Moreover, NS-2 does *not scale* well in terms of number of nodes and it is reported to be in general quite slow from a computational point of view.

NS-2's internal architecture

The NS-2 architecture closely follows the *OSI model*. The *network model* represents the interconnection of network elements. It consists of nodes and links. Single or multiple traffic generators, including statistical generators and other typical generators such as FTP and Telnet, can be attached to any node. In addition, network and transport protocol behavior are simulated by attaching the appropriate agents to the interested nodes.

NS-2's code source is split between C++ for its core engine and OTcl, the MIT's object extension to Tcl (`ftp.tns.lcs.mit.edu/pub/otcl/README.html`), for configuration and simulation scripts. The combination of the two languages offers a sort of compromise between performance and ease of use. The package provides a compiled class hierarchy of objects written in C++ and an interpreted class hierarchy of objects written in OTcl related to the compiled ones. The user creates new objects through the OTcl interpreter. New objects are closely mirrored by a corresponding object in the compiled hierarchy. Tcl procedures are used to provide flexible and powerful control over the simulation (start and stop events, network failure, statistic gathering and network configuration). The OTcl interpreter provides commands to create the networks topology of links and nodes and the agents associated with nodes.

Implementation and simulation under NS-2 consists of 4 steps: (1) implementing the protocol by adding a combination of C++ and OTcl code to NS-2's source base; (2) describing the simu-

lation in an OTcl script; (3) running the simulation and (4) analyzing the generated trace files. Implementing a new protocol requires adding C++ code for the protocol's functionality, as well as updating key NS-2 OTcl configuration files in order for NS2 to recognize the new protocol and its default parameters. The C++ code also describes which parameters and methods are to be made available for OTcl scripting.

The simulation is configured, controlled and operated through the use of interfaces provided by the OTcl class `Simulator`. The class provides procedures to create and manage the topology, to initialize the packet format and to choose the scheduler. It stores internally references to each element of the topology. The user creates the topology using OTcl through the use of the standalone classes `node` and `link` that provide a few simple primitives.

The function of a *node* is to receive a packet, to examine it and map it to the relevant outgoing interfaces. A node is composed of simpler `classifier` objects. Each `classifier` in a node performs a particular function, looking at a specific portion of the packet and forwarding it to the next `classifier`.

Agents are another important type of components of a node: they model endpoints of the network where packets are constructed, processed or consumed. Users create new sources or sinks from the class `Agent`. NS currently supports various TCP agents, UDP, and other general protocols, including RTP, RTCP, SRM.

Links are modeled either as simplex- or duplex-links with a predefined capacity, delay, and queuing discipline. In addition, links can be torn down or restored at any point in time during the simulation,simulating link failures. Links are built from a sequence of `connectors` objects. The data structure representing a link is composed by a queue of connector objects, its head, the type of link, the ttl (time to live), and an object that processes link drops. `Connectors` receive a packet, perform a function, and send the packet to the next connector or to the `drop object`. Various kinds of links are supported, e.g. point-to-point, broadcast, *wireless*. The *queues* are considered as part of a link. NS-2 allows the simulation of various queuing and packet scheduling disciplines. Provided C++ classes include: drop-tail (FIFO) queuing, random early detection (RED) buffer management, CBQ (priority and round-robin), weighted fair queuing (WFQ), stochastic fair queuing (SFQ) and deficit round-robin (DRR).

The user has to specify the routing strategy (static, dynamic) and protocol to be used. Supported routing features include asymmetric routing, multi-path routing, link-state and distance vector algorithms, multicast routing, and several ad hoc algorithms. *New protocols* can be added by specifying new agents in C++ and exposing the relevant parameters to the Tcl interpreter. In particular, OTcl is assumed to be advantageous for quick prototyping purposes.

Various types of *applications* can be simulated. Among them are FTP, Telnet, and HTTP, which use TCP as the underlying transport protocol, and applications requiring a constant bit rate (CBR) traffic pattern, which use the UDP transport protocol.

For the purpose of *traffic generation* NS-2 provides an exponential on/off distribution and it allows also to generate traffic according to a trace file.

Packet losses are simulated by buffer overflows in routers, which is also the dominant way packets get lost in the Internet. Other packet losses are related to *error models* where the unit could be packet, bit or time based.

Arbitrary network topologies, composed of routers, links and shared media can be defined either

by listing the network nodes and edges in a topology file or using some network generators developed for NS-2 (Tiers and GT-ITM).

For collecting output or trace data on a simulation NS-2 uses both *traces*, records of each individual packet as it arrives, departs, or is dropped at a link or queue, and *monitors*, record counts of various interesting quantities such as packet and byte arrivals, departures, etc., that can be associated to both packets and flows. The `Network Animator` (NAM), is a Tcl/TK based animation tool that can be used for viewing NS-2 trace files for *post-processing*, *analysis* and *replay* of simulations (actually NAM can be used with any simulator, as long as data formats are respected).

Specific support for mobile ad hoc networks

The class `MobileNode` extends the basic capability of the `Node` object class by adding functionalities of a wireless and mobile node like ability to move within a given topology, ability to receive and transmit signals to and from a network interface with an antenna, etc. In addition, a `MobileNode` is not connected by means of `Links` to other nodes or mobile nodes. `MobileNode` is a split object. The mobility features, including node movement, periodic position updates, maintaining topology boundary etc. are implemented in C++, while plumbing of network components within `MobileNode` itself (like `classifiers`, MAC, `Channel`, etc.) are implemented in OTcl.

The *network stack* for a mobile node consists of: (i) a *link layer*, (ii) an *address resolution protocol* (ARP) module connected to the link layer, (iii) an *interface priority queue* which gives priority to routing protocol packets and supports running a filter over all packets in the queue, (iv) a *MAC layer* implementing the specifications of the standard IEEE 802.11 (as well as a single-hop preamble-based TDMA MAC protocol), (v) a `Tap Agent` which receives, if allowed, all the packets from the MAC layer before address filtering is done, (vi) a *network interface* which serves as a hardware interface used by the mobile node to access the wireless channel. The network interface is subject to collisions and to the *radio propagation model*, which, in turn, receives the packets transmitted by node interfaces to their wireless channel. The interface stamps each transmitted packet with meta-data related to the transmitting interface, like the transmission power, wavelength etc. This meta-data in the packet header is used in turn by the propagation model in receiving network interface to determine if the packet has minimum power to be received and/or captured and/or detected (carrier sense) by the receiving node. The model approximates the DSSS radio interface (Lucent WaveLan direct-sequence spread-spectrum).

The *radio propagation model* uses a *shadowing model* which, to take into account multi-path propagation effects, represents the received power in terms of a random variable. near distances the model uses an approximation of the Friss-space attenuation ($O(1/r^2)$), while far distances from the emitting node a Two Ray Ground ($O(1/r^4)$) model is used. The approximations assume specular reflection off a flat ground plane.

Antennas used by the mobile nodes are assumed to be omni-directional and with unity gain.

Four ad-hoc *routing protocols* are currently supported: Destination Sequence Distance Vector (DSDV), AODV, DSR, and Temporally ordered Routing Algorithm (TORA). While the Zone Routing Protocol (ZRP) and the Optimized Link-State Routing (OLSR) can be found on the Internet.

Nodes are designed to *move* in a three dimensional topology. However the third dimension is not used, therefore, the nodes are assumed to move always on a flat terrain.

General summary of NS-2's features

NS-2 is the most popular simulator used in the research field of mobile ad hoc networks. NS-2 comes fully equipped of protocols, models, algorithms and accessory tools, and it is for free. Therefore, in terms of scientific acceptance, number of tools/modules and cost, NS-2 would be a sort of ideal choice.

On the other hand, from the short given description of its architecture, it is quite clear that NS-2 is a rather complex software. Adding new components and/or modifying existing ones necessarily involves writing several software modules (in both C++ and OTcl), taking into account several parameters and multi-step data flows. As a net result, NS-2 is not so easy to use in the BISON perspective of contributing new models, protocols, and studying different scenarios at different levels of detail. Moreover, there is a lack of graphical tools that could greatly help code development.

Unfortunately, the provided documentation does not help from this point of view. In fact, it is often limited and out of date with the current release of the simulator. Most problems must be solved by consulting the highly dynamic newsgroups and browsing the source code.

In general, it is admitted that the *learning curve* for NS-2 is steep and *debugging* is difficult due to the dual C++/OTcl nature of the simulator. In some sense, being NS-2 the result of a rather long process of development, which has incorporated contributions from several different sources, the software design is quite poor with respect to current standards. If it is rather easy to use the simulator, it is it not that easy to learn how to add new components or modify existing ones.

Also in terms of (graphical) tools to describe simulation scenarios and analyze or visualize simulation trace files, NS-2 cannot compare positively with commercial tools like OPNET and QualNet.

A more troublesome limitation of NS-2 is its large memory footprint and its lack of scalability as soon as simulations of a few hundred to a few thousand of nodes are undertaken. This is a major impediment according to the BISON's objectives. Only small sized, but at the same time fine-grained simulations can be undertaken.

2.3.4 OMNeT++

OMNeT++ (`whale.hit.bme.hu/omnetpp`) is an *open-source*, component-based simulation package built on C++ foundations. It offers a C++ simulation *class library* and GUI support (graphical network editing, animation).

The simulator can be used for: traffic modeling of telecommunication networks, protocol modeling, modeling queuing networks, modeling multiprocessors and other distributed hardware systems, validating hardware architectures, evaluating performance aspects of complex software systems, and, more in general, modeling any other system that can be mapped to active components that communicate by passing messages. In some sense, and differently from the other simulators discussed so far, OMNeT++ is not specifically designed for telecommunica-

tion networks, but it is far more general. Clearly, this fact, together with the fact that OMNeT++ is still a young software product (its development started in 1998), determines that OMNeT++ comes with far less pre-built modules and protocols than the other (network) simulators considered so far. On the other side, OMNeT++ has been carefully designed from the software point of view, resulting in a product which is much better organized, flexible and easy to use than the other simulators, which are all based on older software products and design concepts.

OMNeT++'s internal architecture

An OMNeT++ model of a system (e.g, a network) consists of *hierarchically nested modules*. The depth of module nesting is not limited, allowing the user to reflect the logical structure of the actual system in the model structure. Modules communicate via *message passing*. Messages can contain arbitrarily complex data structures. Modules can send messages either directly to their destination or along a predefined path, through *gates* and *connections*, which have assigned properties like bandwidth, delay, and error rate. Modules can have parameters which are used to customize module behavior, to create flexible model topologies, and for module communication, as shared variables.

Modules at the lowest level of the module hierarchy are to be provided by the user, and they contain the algorithms in the model. During simulation execution, simple modules appear to run in parallel, since they are implemented as *coroutines* (sometimes termed lightweight processes). To write simple modules, the user is requested to use C++ programming. OMNeT++ has a consistent object-oriented design. One can freely use concepts of object-oriented programming (inheritance, polymorphism etc.) to extend the functionality of the simulator.

OMNeT++ simulations can feature different user interfaces for different purposes: debugging, demonstration and batch execution. *Graphical* user interfaces make the inside of the model visible to the user, allowing him/her to start/stop simulation execution and to intervene by changing variables/objects inside the model. The *GUI library* is linked with the debugging/tracing capability into the simulation executable. This choice enables every user-created object being visible (and modifiable) in the GUI via inspector windows.

OMNeT++ also supports *parallel simulation* through the use of either MPI or PVM3 communication libraries. Moreover, OMNeT++ has also Akaroa[3] support for MRIP, and includes the same synchronization methods used in PARSEC. In general, OMNeT++ bears some similarities with PARSEC but it is a much more flexible, versatile and rich environment.

It is possible to create any form of *hierarchical topology* by using a human-readable and expressive textual topology description format (the `NED` language). The same format is used by a graphical editor (`GNED`). It is also possible to build a network at run-time by program. Both modules and connections among modules can be also dynamically created/destroyed during the simulation run by using library calls.

OMNeT++ comes with an extensive set of container classes for data structures (e.g., queues,

[3] The Akaroa research project is aimed at improving the credibility of results from quantitative stochastic simulation using automated sequential analysis, and speeding up such simulations using Multiple Replications In Parallel (MRIP) to harness the computing power of a network of inexpensive workstations. See the description of the Akaroa's project at the web site:
`www.cosc.canterbury.ac.nz/research/RG/net_sim/simulation_group/akaroa`.

arrays), data collection classes, probability and statistics classes (e.g., histograms, exponential distributions) transient detection and result accuracy detection classes. Output files are text files in a format which, after some simple processing, can be read into standard mathematical and statistical packages. In this sense, OMNeT++ does not provide any "duplicate" of such standard packages.

In terms of *contributed models* for telecommunications, OMNeT++ comes with an Internet protocol suite containing IP, TCP, UDP, RTP and some support for basic QoS, while, concerning wireless networks, there is a module for the simulation of lower layers of GSM networks, and some preliminary modules for node mobility, radio propagation, routing algorithms (AODV) and general wireless communication.

General summary of OMNeT++'s features

The overall design of OMNeT++ is pretty nice and fully based on the driving ideas of object-oriented design. In this respect, OMNeT++ is probably the best software product among the considered ones. The use of the simulator, as well as the definition of new custom models and protocols is quite easy when compared to the other simulators, and to NS-2 in particular. Moreover, it is reasonable to expect a good scalability of OMNeT++, even if there are no real data concerning the simulation of large networks. Therefore, OMNeT++ would perfectly fit BISON's needs for studies at different levels of resolutions and scale, as well as for what concerns rapid prototyping of new models and protocols. Moreover, the nice graphical interface offers good potentialities for monitoring and analysis of the simulations.

Unfortunately, some important network-specific models and protocols are definitely missing. This fact, in spite of some interest that OMNeT++ is widely attracting, dramatically limits the use of OMNeT++. In our case, the use of the current release of OMNeT++ could be limited to very high-level studies and when a quick prototyping of new ideas is required. The choice of OMNeT++ as the primary tool for our studies would necessarily ask for a custom implementation of modules related to the MAC layer, the radio propagation, the application layer, and the node mobility. Moreover, some of the most popular routing algorithms for mobile ad hoc networks (e.g., AODV and DSR) should be definitely implemented.

2.4 The Choice of QualNet as Simulation Framework

In the light of the review of the candidate network simulators, and according to the previously selected criteria to rank the different simulators, we have decided to use *QualNet* as the environment for development and simulation. None of the reviewed simulators seem to really possess the whole set of characteristics required by BISON's research plans and objectives. However, QualNet appears as the best compromise in terms of number of pre-built components, modularity, scalability, and modifiability. In this sense, we see QualNet as an effective simulation framework on top of which we can build the complete BISON's simulation architecture for mobile ad hoc networks, by adding new models and protocols, and by adapting/modifying the existing QualNet's components to BISON's specific needs.

QualNet has been chosen since it met most of the requirements discussed in Subsection 2.2. In particular, it comes with: (i) an extensive set of pre-built models, protocols and algorithms,

(ii) a good level of acceptance from the scientific community, (iii) an excellent scalability, (iv) a rather good, highly modular, software design, (v) a satisfactory level of usability, modifiability and expandability, (vi) advanced graphical and mathematical tools for experiment building, monitoring and post-processing, (vii) good documentation, (viii) possibility of parallel and/or distributed implementations.

The presence of an extensive set of models, protocols and algorithms will dramatically shorten our startup time, since we will have to worry only about the implementation of *our* algorithms and of possible modifications/extensions to existing modules. Also, OPNET and NS-2 present this important feature, but not OMNeT++.

The scalability to up to thousand of nodes will leave us the possibility to carry out studies over a wide range of detail and situations. This fact, jointly with the highly modular nature of the system can in principle allow also statistical mechanics-like simulations. Among the other considered simulators only OMNeT++ could enjoy this possibility.

In terms of ease to use/modify/extend, QualNet seems to be more than satisfactory. In this sense it is probably comparable to OPNET, while OMNeT++, with its good object-oriented design seems to be the best simulator in this respect. NS-2's scores poorly.

The quality of the additional tools for design, monitoring, and analysis is in general at an acceptable level in GloMoSim, and definitely good in QualNet. From this point view GloMoSim/QualNet compares favorably and/or at the same level with respect to the other simulators.

We plan to start to build our algorithms and simulation architecture by using QualNet. In spite of the fact that GloMoSim and QualNet have rather similar features and QualNet is not cost-free, we have preferred QualNet over GloMoSim since QualNet's additional tools for design and analysis seem to be key-features to speedup algorithms' implementation and facilitate the in-depth understanding of their behavior. Moreover, GloMoSim is not anymore supported by its authors, while QualNet, being a commercial product, is shipped with a full online support for any sort of technical and implementation questions.

We leave also open the possibility to use either NS-2 or OMNeT++ in the future, if we realize that the use of QualNet is in some sense limiting for our research. We also expect that we can migrate our code to GloMoSim with only minor changes (QualNet and GloMoSim share a similar structure, however, function calls might have different names and arguments). More realistically, according to the previous discussions on the problems inherent to the use of simulation and simulators, we might consider the possibility to test our models and algorithms using simulators other than our, in order to carry out analysis at different levels and to obtain stronger statistical validation.

3 Simulation of Overlay Networks

The simulation of overlay networks poses special requirements on the simulation environment, that are discussed in Section 3.1. Section 3.2 presents a survey about simulation techniques and tools used currently to evaluate novel overlay network and protocols, and discusses the applicability of these techniques for the particular requirements described in the previous section. Section 3.3 presents the architecture of our simulation environment for overlay networks.

Finally, Section 3.4 discusses some additional techniques that may exploited to evaluate performance of overlay networks and protocols.

3.1 Simulation Requirements for Overlay Networks

Overlay networks (including peer-to-peer and grid systems) differ from other distributed system technologies for several important aspects. Some of them, such as decentralized control, high autonomy of participating nodes and heterogeneity in terms of shared resources, do not have any particular repercussions on the simulation environment, as most of existing simulators can support them without problems. On the other hand, other properties like *large scale* and *dynamicity* (with respect to nodes joining and leaving the system, and to failures to both nodes and connections) pose important requirements on the simulator.

- Regarding scalability, overlay networks are expected to incorporate several thousands of nodes, or even more: at the time of writing, more than 4 millions of users are connected to the Kazaa network [17, 29, 21]. The reason for which overlay networks scale toward such huge sizes can be found in the desire to exploit as much user resources as possible, in order to provide a better service.

  Clearly, such scalability requirement poses limits to the accuracy (in terms of level of detail) that can be reached by such simulations. A simulated overlay network composed of millions of nodes can be represented just as a graph, with nodes capable to communicate with each other with message exchange, without having the possibility to consider all low-level details, e.g. the overhead associated to the communication stack (e.g. TCP or UDP). Furthermore, in the more extreme cases, even details such as network latency or bandwidth can be dropped, in particular in those cases where we want to study topological properties of the protocols adopted.

- Regarding dynamicity, in existing file-sharing applications, the connection time of each node composing the network is generally short; it has been proved that most of the nodes of a Gnutella networks stay connected for less than one hour [47]. This has to be reflected in the simulator, that must provide specific tools for representing such dynamicity.

3.2 Related Work

The picture we have obtained from surveying existing literature about peer-to-peer, overlay and grid networks is highly fragmented: most of the current projects base their results on home-made simulators, specially tuned for the particular protocols developed by those projects. At the best of our knowledge, with the single exception of Tapestry [53], none of the existing projects make use of complex simulators like NS2 [34], Omnet++ [35], Opnet [36] [36], etc. This choice is due to the high costs associated with packet-level simulations that pose very restrictive limits to scalability.

In spite of the fact that most of the research performed in the overlay network area does not make use of any existing tools for simulation, recently some simulation environments customized for peer-to-peer, overlay and grid networks have started to appear. These tools in-

clude Neurogrid [46], 3LS [50], Query-Cycle Simulator [48] and Anthill [2]. We survey them in the next sections.

3.2.1 State-of-the-art of Simulation in Overlay Networks

By surveying the existing literature about peer-to-peer [37], overlay [14, 41, 53, 44, 32, 26], and grid systems [18], it is possible to note that most of the published papers in this area make use of "home-made" simulators. For example, Freenet [31], one of the first peer-to-peer systems to be presented in a scientific publication [13], was associated with Aurora [1], a simulator written in C++. An additional simulator running the same protocol has been written in Java and supports the evaluation of different caching algorithms[39]. GnutellaSim is a Java simulator of the Gnutella Network [20], created by Limewire, one of the most important Gnutella servent.

Looking at structured overlay networks like Chord [14], Koorde [26], Viceroy [32], CAN [41], Pastry [44] and Tapestry [53], the situation is not different. Viceroy and Koorde provide only analytical results. Chord and Pastry are associated to their own simulators, respectively Chord-Simulator [11] and FreePastry/SimPastry [19], respectively written in C and Java/C#. Chord-Simulator is an event-driven simulator that takes a simulation description file as input, describing the simulated network and the events to be presented to the Chord nodes. FreePastry is an open-source implementation of the Pastry protocol in Java. The settings of the simulator parameters, such as the number of nodes to simulate and the number of events to generate, is done by providing the values in the command line upon starting the local simulators. Details about the performance of these simulation CAN is also simulated using an home-made simulator, for which details are not provided. All these simulators provide very high performance with respect to the number of nodes simulated. For example, the networks simulated for CAN are composed 10^6 nodes, while networks simulated for Chord are composed of 10^5 nodes.

The only notable exception is Oceanstore/Tapestry [30, 53], where part of the simulation studies are performed using a packet-level simulator like NS2 [34]. In this case, the simulated networks are limited to 4096 nodes, and in some cases only few hundreds of nodes are simulated.

3.2.2 Overlay and P2P Network Simulators

3LS

3LS [50] (3-Level-Simulator) is an open-source simulator for overlay networks. It claims to be designed to overcome the problems of existing simulators, namely extensibility, usability and level of detail. The simulator design is based on three architectural levels, namely the *network level*, the *protocol level* and the *user level*. Communication can only happen between directly connected levels. The network level simulates the underlying network, represented as a two-dimensional matrix storing the distance values between the nodes. The protocol level represents the actual protocol to be simulated. Finally, the user level represents the actual input coming from users; this input can be fed into the simulator through a graphical interface or a file.

The implementation of the 3LS simulator is not completely satisfactory, as every event executed by the simulator is maintained in main memory to allow a easier reproduction through

the graphical interface. This limit the system to very small number of nodes (less than one thousand on a machine with 2 GB). For this reason, 3LS is more useful as a debugger than as a simulator. No support for dynamicity is mentioned in the paper.

Query-Cycle Simulator

The Query-Cycle Simulator [48] is a simulator specialized in file-sharing simulations. It includes realistic models for content distribution (for both data and types), query activity, download behavior, uptime, etc. The simulation engine is based on the following model. The simulation proceeds in query cycles. In each query cycle, a peer may be actively issuing a query, inactive, or even down and not responding to queries passing by. Upon issuing a query, a peer waits for incoming responses, selects a download source among those nodes that responded and starts downloading the file. The query cycle finishes when all peers who have issued queries download a satisfactory response. Statistics may be collected at each peer, such as the number of downloads and uploads of the peer.

The SimP^2 Simulator

The SimP^2 simulator [28] is designed to provide support to the analysis of ad-hoc p2p networks. The analysis is based on a non-uniform random graph model, and is limited to studying basic properties such as reachability and nodal degree. Simulation is employed to assess more detailed performance characteristics such as queuing delays and message loss. The simulator can take any sort of graph input, such as a set of network instances created using the same network construction approach as the analytical model. SimP2 ignores transient nodes and other dynamic aspects of the network for the purpose of tractability.

Anthill

Anhtill [2] is an agent-based peer-to-peer simulator, developed by one of the partners of BISON (Bologna). Its aim is to provide a "testbed" for studying and experimenting with multi-agent peer-to-peer systems in order to understand their properties and evaluate their performance. One of the goals of Anthill is to enable a simple migration of algorithms and protocols tested in the simulation environment to a realistic run-time environment, based on JXTA. This feature limits the scalability of the system, as several details needed to enable the execution of agents in a distributed systems increase the overhead for each simulated node.

NeuroGrid

NeuroGrid [46] is a Java-based overlay simulator specialized in simulation of search protocols for file-sharing systems. The NeuroGrid simulator was originally created to support comparative simulations between the FreeNet [31], Gnutella [27] and NeuroGrid [46] protocols. Unlike many other simulators that were designed for a specific P2P system, however, the NeuroGrid simulator was explicitly designed with extensibility in mind. Several protocols are now simulated using NeuroGrid, including a distributed DNS and a distributed e-mail protocol.

The simulator is a single-threaded discrete event simulator, capable to read properties files that enable the user to modify the parameters for a simulation run. The parameters that a user can specify include the type of protocol to simulate, the characteristics of the network and the number of searches to be executed. Statistics (e.g. number of messages parsed, number of successful queries, etc.) can be saved into files for later analysis.

Among the simulators analyzed in this document, NeuroGrid is one of the most promising. The simulator has recently been improved by adding full support for dynamicity, with nodes being removed and added to the network. Furthermore, the simulator has been used to simulate networks composed of 10^5 nodes.

3.2.3 Discussion

Although the simulators listed in the previous section claim to be specialized for overlay and peer-to-peer systems, none of them is completely satisfactory for simulations in BISON. 3LS [50] and Anthill [2] present important scalability limitations, that make them not suitable for large-scale overlay networks. From this point of view, NeuroGrid [46] is more promising; still, it is still not suitable to simulate networks with millions of nodes. Query-Cycle [48] do not make any claim about scalability, but is limited to the file-sharing area, while BISON has a larger set of application functions to be implemented. None of them present a sufficient support for dynamicity.

On the other hand, existing packet-level simulators like NS2 [34], Omnet++ [35] and Opnet [36] present similar scalability issues, as demonstrated by the lack of their adoption in the overlay and peer-to-peer research community.

For these reasons, one is tempted to adopt the approach followed in existing overlay and peer-to-peer publications, i.e. going toward highly specialized, hand-made simulators for each of the particular algorithm or protocol we intend to simulate. This approach would enable the BISON members to obtain the maximum possible scalability, as simulators could be highly optimized for the particular task to be simulated, without incurring in the expensive overhead generated by casting our solution in a general framework.

In order to avoid the drawbacks of existing approaches, we intend to build a light-weight, component-based framework for simulation of overlay networks. This framework will enable members of the BISON consortium to develop and use multiple highly-customized and optimized simulators, by reusing components taken from a component library. Components will be re-used only when appropriate, leaving to developers the possibility of implementing optimized versions when possible. The development of new simulations for the fulfillment of the BISON goals will enrich this library with new components.

This library will enable members of the BISON consortium to adopt different approaches to simulation, ranging from time-driven to event-driven simulators. Members will be able to re-build every component from scratch, if this fit their needs. Nevertheless, especially in the case of event-driven simulation, we intend to reuse existing code (including support for event lists, event scheduling, resource management, condition queues, wait queues, event interrupts, priority handling, etc.) in order to speed up the development process.

Details of the architecture of this simulation environment are illustrated in the next section.

```
simulation peersim.cdsim.Simulator
simulation.cycles 30
simulation.shuffle

network.size 100000

protocol.0 peersim.core.IdleProtocol
protocol.0.cache 20

protocol.1 aggregation.AverageFunction
protocol.1.linkable 0

init.0 peersim.init.WireRegularRandom
init.0.protocol 0
init.0.degree 20

init.1 example.aggregation.PeakDistributionInitializer
init.1.value 1

dynamics.0 peersim.dynamics.DynamicNetwork
dynamics.0.add -100
dynamics.0.minsize 4000

observer.0 example.aggregation.AverageObserver
observer.0.protocol 1
```

Figure 1: Example of configuration file for a simple simulation

3.3 Architecture of a Simulation Environment for Overlay Networks

The main goal of our simulation environment is to enable the composition of complex simulators without incurring in excessive overhead both in terms of memory and time. In order to achieve this goal, the architecture of the simulation environment will be as light-weight as possible. The idea is to provide a configuration service that will enable the construction of simulators by "juxtaposing" together pre-existing or newly-developed components. Every component of the simulator will be interchangeable, in order to allow developers to write their customized and optimized version when needed. Components will be specified by object-oriented programmatic interfaces, whose methods will describe the expected behavior of a component. These interfaces will be as simple as possible, in order to avoid the burden for developers of implementing complex interfaces; and also, to avoid the memory overhead that can be associated with complex implementations.

The basic building block of our simulation environment will be the *configuration manager*. This module will be present in each simulation, and its task will be to read configuration files and/or command-line parameters, and compose a simulation starting from the classes listed in these configuration specifications. The flexibility offered by this mechanisms will enable developers to re-implement, when needed, every component of the system, with the freedom of re-using existing components for fast prototyping.

Figure 1 contains an example of configuration file. The example is neither complete nor exhaustive; it is meant to give an idea of the potentiality of the configuration manager. The components listed in the file represent some of the main elements that will be used in our simulators. Each simulation will be driven by a *simulation engine* over a simulated *network*. The characteristics of the simulation engine and the network that will be used in the simulation are described in the first lines of the configuration file. The main elements that will compose the dynamic behavior of the simulation are *protocols*, *initializers*, *dynamics* and *observers*. Each component will be responsible for loading the appropriate components needed for its execution; for example, a simulation engine will instantiate a network, the network will instantiate a set of nodes, each nodes will instantiate the appropriate protocols, and so on.

The flexibility of our simulation environment is twofold: neither the set of interfaces nor the set of implementations of those interfaces are fixed:

- For example, it will be possible for a developer to decide that a certain component should be organized as a collection of sub-components, and decide the interfaces for these sub-components. For example, a dynamic network can be composed by a data structure for maintaining the network information, and an "uptime" model that actually decides when a new node should join or an existing node should leave. This interface flexibility must be expressed programmatically, in the sense that the developer will have to write new code to support it.

- On the other hand, developers will be enabled to specify the actual implementation to be used in the simulator. For example, in the example, a static network (IdleProtocol) is selected; it is possible to declaratively modify the simulation by simply changing the configuration file (provided that an appropriate implementation is available, otherwise a new implementation will have to be written).

Due to this flexibility, it is clear that our simulation environment will grow over time. Thus, the set of components listed in Figure 1 is very limited and represent only a first approximation of the final model. In the next sections we describe them, and we provide some ideas about how to extend them.

3.3.1 Simulation Engine

As describe above, the simulation engine will take care of executing the simulation. In our intentions, two simulation engines are planned:

- The first one will be a very simple engine, performing time-driven simulations, whose task will be to enable the fast prototyping of promising protocols, and the analysis of the influence of different topologies on the studied functions. We are already using such simulator to evaluate the effects of random, small-world and other topologies for simple functions like aggregation and load-balancing.

- A second simulator will be a full-fledged event-driven simulator, that will integrate communication delays, dynamicity and event management.

3.3.2 Network Topology

As described in deliverable D01, overlay topologies and overlay topology management will be a fundamental research topics in BISON. For this reason, the concept of topology will constitute the basic building block of all simulations. Its basic description will be based on the concept of *graph*, composed of a set of *nodes*. Each node n will be associated with a non-empty list of *connections*, i.e. neighbors nodes that are known to n. Two basic operations will be possible: to modify the set of connections, and to obtain the set of connections. Connections will be used to send messages over them, or to directly invoke methods on the connected components.

In our simulation environment, overlay topologies will be constructed and will be used in different ways. For example, it will be possible to study what is the best topology for a given function, or for a protocol implementing that function. In order to pursue this goal, several well-known topologies, such as random, small-world, scale-free, superpeer-based will be implemented and made available to developers, that will use them to test and evaluate their protocols.

On the other hand, building and maintaining an overlay topology is a complex task. So, the simulation environment should enable developers to study also how to maintain the topologies identified above in a decentralized way.

3.3.3 Protocols

The domain of protocols is clearly the most open one; several different functions and algorithms will be implemented on our simulator. So far, we have implemented some basic prototypes for aggregation and load balancing. Since the protocols that can be implemented are out of the scope of the simulation environment, we will not discuss them any further. In the example of 1, two protocols are configured: the IdleProtocol, whose task is to maintain a static connected topology, and AverageFunction, an aggregation protocol whose task is to compute an aggregate function (the average) over a set of values maintained by peers participating in the computation.

3.3.4 Initializers

Initializers are run at the beginning of the simulation in order to initialize the state of the system. In the specific example, the two initializers create a static regular random topology using the idle protocol (protocol identifier 0) and initialize the values to be aggregated using a peak distribution (protocol identifier 1).

3.3.5 Dynamics

Dynamics are run during the simulation, and modify the state of the system according to environmental inputs (for example, by simulating the crash of some nodes or by modifying the value to be aggregated. In the specific example, 100 nodes are removed from the network at each cycle by the DynamicNetwork class.

3.3.6 Observers

Observer objects will be run periodically (in a time-driven simulator, potentially at every time step) to analyze the network, the nodes composing the network and the state of the protocols executed on them. The aim is to collect statistical information about the behavior of the simulated system.

Some of the observers will be highly customized for the particular protocol to be monitored. For example, in our load-balancing prototype, an observer object reports the standard deviation of the loads measured at each of the nodes, and eventually stops the simulation when some configured level of balancing is obtained.

Other observers will be more general, and applicable to different function domains. For example, we intend to develop a graph-theoretical library for analyzing interesting graph properties like diameter, clustering, conductance, etc.

3.4 Beyond Simulation

Simulation is not the only way to analyze and study overlay networks. We will follow also other approaches, like:

- Analysis and monitoring of existing deployed peer-to-peer networks, like Gnutella [33, 51, 42, 12, 47], in order to capture statistical information about the behavior of users of current peer-to-peer networks. This information constitute an essential input for both simulation and analytic modeling. Given the open-source characteristic of some of the existing P2P networks like Gnutella, it is very simple to develop crawlers for them. We plan to implement such a tool.

- Deployment of selected peer-to-peer application in a geographically distributed setting, in order to collect realistic data about the behavior of such applications. Clearly, this approach is quite impractical; current peer-to-peer networks have thousands of nodes, and performing a software update for each of these nodes in order to test each novel P2P algorithm is impossible. Nevertheless, one of the BISON partner (Bologna) is member of the Planet-Lab consortium [40]. Planet-Lab is a project aimed at providing a testbed to researchers where to test their peer-to-peer algorithm. At the time of writing, Planet-Lab is composed of 160 production machines, two of them are hosted in Bologna. We plan to exploit this possibility to test the most promising peer-to-peer applications developed in WP2 and WP3.

4 Conclusions

The development of two independent simulation environments for ad-hoc networks and overlay networks is associated to Deliverables D12 and D13. Although these deliverables are due by Month 18 (July 2004), it is clearly not possible for BISON to make any substantial progress without the use of such simulation environments.

This document has discussed the choices that have been made in BISON to design and implement simulation architectures which are appropriate for the different BISON's objectives. Two preliminary practical realizations of such architectures, one for ad-hoc and one overlay networks, have been already completed and we are starting to gain experience with them.

References

[1] The Aurora Simulator. `http://cvs.sourceforge.net/cgi-bin/viewcvs.cgi/freenet/aurora/`.

[2] Özalp Babaoğlu, Hein Meling, and Alberto Montresor. Anthill: A Framework for the Development of Agent-Based Peer-to-Peer Systems. In *Proceedings of the 22th International Conference on Distributed Computing Systems*, Vienna, Austria, July 2002.

[3] L. Bajaj, M. Takai, R. Ahuja, R. Bagrodia, and M. Gerla. GloMoSim: A scalable network simulation environment. Technical Report 990027, UCLA Computer Science Department, 13, 1999.

[4] J. Banks, editor. *Handbook of Simulation : Principles, Methodology, Advances, Applications, and Practice*. Engineering and Management Press, 1998.

[5] J. Banks. Introduction to simulation. In P. A. Farrington, H. B. Nembhard, D. T. Sturrock, and G. W. Evans, editors, *Proceedings of the Winter Simulation Conference*, 1999.

[6] B. Blum, T. He, and Y. Pointurier. Mac layer abstraction for simulation scalability improvements. Technical report, Department of Computer Science, Charlottesville, VA, December 2001.

[7] L. Breslau and S. Shenker. Best-effort versus reservations: A simple comparative analysis. In *Proceedings of the ACM SIGCOMM*, pages 3–16, Vancouver, Canada, September 1998. ACM Press.

[8] T. Camp, J. Boleng, and V. Davies. A survey of mobility models for ad hoc network research. *Wireless Communications & Mobile Computing (WCMC): Special issue on Mobile Ad Hoc Networking: Research, Trends and Applications*, 2(5):483–502, 2002.

[9] D. Cavin, Y. Sasson, and A. Schiper. On the accuracy of MANET simulators. In *Proceedings of the Workshop on Principles of Mobile Computing (POMC'02)*, pages 38–43, Toulouse, France, October 30–31 2002. ACM.

[10] M. A. Centeno and M. F. Reyes. So you have your model: what to do next. a tutorial on simulation output analysis. In D. J. Medeiros, E. F. Watson, J. S. Carson, and M. S. Manivannan, editors, *Proceedings of the Winter Simulation Conference*, 1998.

[11] The Chord Simulator. `http://www.pdos.lcs.mit.edu/cgi-bin/cvsweb.cgi/sfsnet/simulator/`.

[12] J. Chu, K. Labonte, and B. Levine. Availability and locality measurements of peer-to-peer file systems, 2002.

[13] Ian Clarke, Oskar Sandberg, Branden Wiley, and Theodore W. Hong. Freenet: A Distributed Anonymous Information Storage and Retrieval System. In Hannes Federrath, editor, *Proceedings of the Workshop on Design Issues in Anonymity and Unobservability*, Berkeley, CA, July 2000.

[14] Frank Dabek, Emma Brunskill, M. Frans Kaashoek, David Karger, Robert Morris, Ion Stoica, and Hari Balakrishnan. Building Peer-to-Peer Systems With Chord, a Distributed Lookup Service. In *Proceedings of the 8th Workshop on Hot Topics in Operating Systems (HotOS)*, Schloss Elmau, Germany, May 2001. IEEE Computer Society.

[15] DARPA/NIST. DARPA/NIST Network Simulation Validation Workshop. Proceedings at `http://www.dyncorp-is.com/darpa/meetings/nist99may/index.html`, May 1999.

[16] S. Das, C. Perkins, and E. Royer. Performance comparison of two on-demand routing protocols for ad hoc networks. In *Proceedings of the IEEE Infocom*, pages 3–14, Tel Aviv, Israel, March 2000. IEEE Press.

[17] Fasttrack Home Page. `http://www.fasttrack.nu`.

[18] Ian Foster and Carl Kesselman, editors. *The Grid: Blueprint for a Future Computing Infrastructure,*. Morgan Kaufmann, 1999.

[19] The FreePastry/SimPastry simulators. `http://research.microsoft.com/~antr/Pastry/`.

[20] The GnutellaSim Simulator. `http://gnutellasim.limewire.org/servlets/ProjectHome`.

[21] Grokster Home Page. `http://www.grokster.com`.

[22] J. Heidemann, N. Bulusu, J. Elson, C. Intanagowiwat, K. Lan, Y. Xu, W. Ye, D. Estrin, and R. Govindan. Effects of detail in wireless network simulation. In *Proceedings of the SCS Multiconference on Distributed Simulation*, pages 3–11, January 2001.

[23] J. Heidemann, K. Mills, and S. Kumar. Expanding confidence in network simulation. Technical Report 00-522, USC/Information Sciences Institute, April 2000.

[24] P. Huang, D. Estrin, and J. Heidemann. Enabling large-scale simulations: selective abstraction approach to the study of multicast protocols. In *Proceedings of the International Symposium on Modeling, Analysis and Simulation of Computer and Telecommunication Systems*, pages 241–248, Montreal, Canada, July 1998. IEEE Press.

[25] D. B. Johnson. Validation of wireless and mobile networ models and simulation. In *Proceedings of DARPA/NIST Network Simulation Validation Workshop*, Fairfax and Virginia, USA, May 1999.
`http://www.dyncorp-is.com/darpa/meetings/nist99may/index.html`.

[26] M. Kaashoek and D. Karger. Koorde: A simple degree-optimal distributed hash table. In *Proceedings of the 2nd International Workshop on Peer-to-Peer Systems*, Berkeley, CA, USA, February 2003.

[27] Gene Kan. Gnutella. In Andy Oram, editor, *Peer-to-Peer: Harnessing the Benefits of a Disruptive Technology*, chapter 8. O'Reilly & Associates, March 2001.

[28] K. Kant and R. Iyer. Modeling and simulation of ad-hoc/p2p file-sharing networks. In *Proceedings of Tools*, May 2003.

[29] Kazaa Home Page. `http://www.kazaa.com`.

[30] John Kubiatowicz et al. OceanStore: An Architecture for Global-Scale Persistent Storage. In *Proceedings of the 9th International Conference on Architectural support for Programming Languages and Operating Systems*, Cambridge, MA, November 2000.

[31] Adam Langley. Freenet. In Andy Oram, editor, *Peer-to-Peer: Harnessing the Benefits of a Disruptive Technology*, chapter 8. O'Reilly & Associates, March 2001.

[32] D. Malkhi, M. Naor, and D. Ratajczak. Viceroy: A scalable and dynamic emulation of the butterfly. In *Proceedings of the 21st ACM Symposium on Principles of Distributed Computing*, Monterey, CA, USA, July 2002.

[33] Evangelos P. Markatos. Tracing a large-scale peer to peer system : an hour in the life of gnutella. In *Proceedings of the 2nd IEEE/ACM International Symposium on Cluster Computing and the Grid*, 2002.

[34] Network Simulator 2. `http://www.isi.edu/nsnam/ns/`.

[35] The Omnet++ simulator. `http://whale.hit.bme.hu/omnetpp/`.

[36] The Opnet Simulator. `http://www.opnet.com`.

[37] Andy Oram, editor. *Peer-to-Peer: Harnessing the Benefits of a Disruptive Technology*. O'Reilly & Associates, March 2001.

[38] K. Pawlikowski, H.-D. J. Jeong, and J.-S. R. Lee. On credibility of simulation studies of telecommunication networks. *IEEE Communications Magazine*, January 2002.

[39] Jeremy Pfeifer. Freenet Caching Algorithms Under High Load. `http://www.cs.usask.ca/classes/498/t1/898/W7/P2/freenet.pdf`.

[40] PlanetLab. `http://www.planet-lab.org`.

[41] Sylvia Ratnasamy, Paul Francis, Mark Handley, Richard Karp, and Scott Shenker. A Scalable Content-Addressable Network. In *Proceedings of the ACM SIGCOMM'01*, San Diego, CA, 2001.

[42] M. Ripeanu, I. Foster, and A. Iamnitchi. Mapping the gnutella network: Properties of large-scale peer-to-peer systems and implications for system design, 2002.

[43] C. P. Robert and G. Casella. *Monte Carlo Statistical Methods*. Springer-Verlag, 1999.

[44] Antony Rowstron and Peter Druschel. Pastry: Scalable, Decentralized Object Location and Routing for Large-Scale Peer-to-Peer Systems. In *Proceedings of the 18th International Conference on Distributed Systems Platforms*, Heidelberg, Germany, November 2001.

[45] R. Y. Rubistein. *Simulation and the Monte Carlo Method*. John Wiley & Sons, 1981.

[46] Sam Joseph and T. Hoshiai. Decentralized meta-data strategies: Effective peer-to-peer search. *IEICE Transactions on Communications*, E86-B(6):1740–1753, 2003.

[47] Stefan Saroiu, P. Krishna Gummadi, and Steven D. Gribble. A measurement study of peer-to-peer file sharing systems. In *Proceedings of Multimedia Computing and Networking 2002 (MMCN '02)*, San Jose, CA, USA, January 2002.

[48] Mario Schlosser, Tyson Condie, and Sepandar Kamvar. Simulating a File-Sharing P2P Network. In *Proceedings of the First Workshop on Semantics in P2P and Grid Computing*, December 2002.

[49] Y. C. Tay and K. C. Chua. A capacity analysis for the ieee 802.11 mac protocol. *ACM/Baltzer Wireless Networks*, 7(2):159–171, March 2001.

[50] Nyik San Ting and Ralph Deters. A Generic Peer-to-Peer Network Simulator. Technical report, Department of Computer Science, University of Saskatchewan, June 2003.

[51] Jean Vaucher, Peter Kropf, Gilbert Babin, and Thierry Jouve. Experimenting with gnutella communities.

[52] 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 Simulation*, pages 154–161, Banff, Alberta, Canada, May, 26–29 1998.

[53] Ben Y. Zhao, John Kubiatowicz, and Anthony D. Joseph. Tapestry: An Infrastructure for Fault-Tolerant Wide-Area Location and Routing. Technical Report UCB/CSD-01-1141, U.C. Berkeley, April 2001.