AN ENVIRONMENT FOR MODELLING AND SIMULATING COMMUNICATION SYSTEMS APPLICATION TO A SYSTEM BASED ON A SATELLITE BACKBONE

Atika COHEN and Radouane MRABET
Université Libre de Bruxelles
Service Télématique et Communication
Bd du Triomphe CP 230, B-1050 Brussels, Belgium
Tel : 32-2-650-57-04    Fax : 32-2-629-38-16
cohen@helios.ihe.rtt.be and mrabet@helios.ihe.rtt.be
C=be; ADMD=rtt; PRMD=iihe; O=helios; S=cohen (or S=mrabet)

ABSTRACT :
The availability of good software tools is vital to simulation practitioners. The objective of the OSISIM project (Open System Integrated Simulator) is to set up an environment which integrates facilities and tools to build a communication system, and to study its performance. The environment is called AMS : Atelier for Modelling and Simulation. Using it, the network architect, i.e. the user of the atelier, can quickly construct, in a graphical environment, a concise system from models of several standard networks available in a specific library, and execute the system in an efficient manner. Starting with an outline of the components of the atelier, we describe the AMS prototype, and the internal structure of basic models to be included in it. Moreover, a concrete example of a communication system including a satellite backbone is provided to illustrate how AMS is used to simulate a such system.

Key-words : communication system, simulation, modelling, satellite backbone, performance evaluation.

1. INTRODUCTION

The goal of the OSISIM project [1, 2, 3] is to set up an atelier for modelling communication systems and analysing their performance. The simulation is now considered to be an integrated part of the design process of communication systems. The design process also includes evaluation and optimisation of these systems. Simulations constitute the only possible way to obtain detailed information in this field. The objective of the atelier, named AMS (Atelier for Modelling and Simulation), is to make simulation a creative task. In other words, instead of having to spend most of his time on writing simulation code, which is a tedious task, the user will be able to devote more time to thinking about how to improve his system.

The user of the atelier is not expected to be a specialist in modelling or performance analysis, nor has in-depth knowledge of the simulation techniques; however, he should be a communication system designer, a network architect or a network manager. He will use the atelier to build and validate an architectural choice, and/or to compare several possible ways of solving a problem.

In the literature, we find the description of different toolkits dedicated to this field, such as OPNET (Optimised Network Engineering Tool) [4], which is based on extended finite state machines and C language, or BONeS (Block Oriented Network Simulator) [5], which is based on block-oriented modelling paradigm, written in C++. Unlike these approaches, ours is based on queueing networks.
Actually, OPNET is a simulation tool for analysing communication networks. The model description is done hierarchically. The user of OPNET specifies graphically the topology of his network which consists of nodes and links. Each node includes processors, queues, and traffic generators. He also has to describe the data flow between components in a node. Finally, the behaviour of each process is described using state diagrams. In general, attributes of various building blocks are specified using pop-up menus. When the editing of the model is terminated, OPNET generates a simulation program written in C. OPNET seems to be a powerful tool and the choice of UNIX/C is thus a guarantee of portability, although C is not a language for modelling and simulation.

Regarding BONeS, it is a simulation system for studying communication network models. The modelling principle of this product is the hierarchical decomposition; the network topology is described using blocks or sub-models. A graphical editor is provided for the user to construct his model. The blocks are written in the C++ language at the lowest level of abstraction, and are stored in a library. Once the model description is done, a C program is produced for realising a discrete event simulation. To add a new block to the library, the user constructs it from other blocks or can write the corresponding C code. The main drawback of BONeS is the usage of C as a language of simulation.

The modelling paradigm behind AMS is the queueing networks theory. Actually, queueing network models have been used extensively as a modelling paradigm for deriving analytical as well as simulation based performance measures. Queueing networks are especially effective in modelling computer communication systems, including point-to-point communication, broadcast systems, distributed multiple access systems, etc. The simulation package used by AMS is QNAP2 (Queueing Network Analysis Package 2) [6], which is a package for describing, handling and solving large and complex discrete event flow systems such as data communication networks and computer systems.

Furthermore, our atelier is built around a library of basic models which includes the most standard networks such as LANs (Ethernet, Token Ring, FDDI, etc.), WANs (X25, TCP/IP), satellite (TDMA, FDMA, etc.) and radio networks. It is obvious that the usefulness of the atelier heavily depends on the number of available models necessary to make up transmission systems and networks.

The basic models are to be constructed in a very modular fashion in order to allow more flexibility. So, we have to build basic models, which will make up other models of more complex systems. Each basic model implements uniform and well-defined set of functions, while having clearly specified interfaces.

In addition to the library, the atelier is composed of several modular components which are categorised in four essential phases, namely the editing phase, the ADL (Architectural Description Language) phase, the simulation phase, and finally the presentation phase.

The description of the system to be studied can be made graphically or textually by using the specific language ADL [7] designed for this purpose. ADL is an object-oriented language which provides a means for modular and hierarchical description of network architectures. So, the role of the ADL phase is to define a clear border between the editing and the simulation phases. The portability of the atelier to other platforms becomes possible.

In order to be operational in a reasonable time, AMS has been designed on the basis of existing and proved packages rather than starting from scratch. QNAP2, GSS [8] (Graphical Support System), MODLINE [9] and S-PLUS [10] are the main packages that are used to operate AMS. Based on those tools, a prototype of the atelier has been implemented.
This paper is organised as follows. Section 2 focuses on the components of the atelier. The processes behind those components are described in the same section.

Section 3 describes the prototype of AMS. Section 4 focuses on the design of a basic model [11], and its internal structure. Section 5 briefly describes new concepts used by AMS to check the coherence of a network architecture[12].

Section 6 is devoted to explain how AMS can be used to study a satellite based communication system. The whole simulation model process is executed on a well-known architecture of this system. A conclusion is drawn in the last section.

2. STRUCTURE OF AMS

AMS is designed in a modular manner. The main modules constituting AMS are structured in four phases, each of them is handled by one or several processes. Figure 1 illustrates this structure. Processes are accessible by the user in a graphical and/or a textual manner. Furthermore, the atelier is based on the library of basic models constructed in a very modular fashion to allow more flexibility. This library is the core of AMS.

A brief description of those phases, as well as the library, is given before describing in more details, the processes composing the atelier.

The editing phase : In this phase, the user constructs a communication system graphically and hierarchically using the building detailed basic models from the AMS library. The editing phase is managed by a dedicated process called the graphical editor. A text-based version of this editor is also provided for a user without graphical capabilities; in this case, the user has to describe his architecture directly using the ADL language. On the contrary, when the user works in a graphical environment, he does not need to know the ADL syntax. Indeed, the editor process is to generate the ADL code corresponding to the edited architecture. It is obvious that, using a graphical environment will decrease the time and effort for the users.

The ADL phase : This phase mainly consists of translating the description written in ADL language into a QNAP2 simulation model. In addition, this phase allows us to fulfil several objectives :

- The definition of a clear border between the editing phase and the simulation phase. This separation enforces the independence between the interactive part of the atelier represented by the graphical editor and the computational part represented mainly by the simulator. This phase permits the design of the editor separately from the design of the computational part of the atelier, so that, modifications in either tend not to cause changes in the other.
- On the other hand, it will improve the portability of the atelier. Namely, several
graphical editors can be developed for the atelier, depending on the platform where the atelier will be implemented. These platforms will run different operating systems and graphical kernels.

- This phase allows a user to simulate communication architecture even if he has no graphical editor at its disposal. In this case, he will describe textually his communication architecture directly in the ADL language.

- Finally, the person who implements a graphical editor is not obliged to know the simulation language which is in our case QNAP2.

The simulation phase: The main objective of this phase is to assess the performance of the system previously edited. This phase includes making scenarios, executing simulations, and post-processing the rough results. This phase is handled by three processes which are the simulator, the experimenter and the post-processor.

The presentation phase: This phase is meant to allow different presentations of the simulation results. It offers the user the possibility of supervising the simulation by visualising its execution. In other words, it offers a high level animation of showing, for instance, messages passing through the simulated architecture. This phase is handled by three processes which are the plotter, the animator and the reporter.

2.1. The library

The core of AMS is the library of basic models. Each of them corresponds to a communication entity, and has to be studied separately, verified and validated before its inclusion into the library. Each model is represented by an icon, an ADL description, as well as a QNAP2 code. This implies that the library has different representations depending on the phase where it is used. For the editing phase, and in the case of a graphical environment, the library is seen as a set of icons. At the ADL phase, the components of the library are described in ADL language where the description is limited to the main attributes of the models. Finally, at the simulation phase, the library is seen as a set of models written in QNAP2 language.

2.2. The Editor Process

AMS provides a Graphical Editor that corresponds to the world view of the network designer, with icons that represents rings, buses, bridges, protocols, etc. To edit the architecture, a user can create several instances of basic models, and connect them by links. The editor has the ability to provide substantial user interaction for either simple parameter changes or major reconfiguration of systems, and the provision of hierarchical modelling capability for extremely large models.

Each basic model is represented by a simple icon. Besides, the user has two facilities to edit the architecture: generic icon display facility and sub-model icon editing facility.

A generic icon also represents a basic model such as a bridge, but has several displays. In other words, the display that appears in the window edition depends on some attributes and/or on the basic model to which it is connected. For instance, the display of a bridge is different if it connects two Ethernets, or two Token-Rings.

The second facility is the capability to represent an entire graph as a node within another graph. This hierarchical modelling approach brings the powerful notions of data abstraction and modular construction into model construction. In other words, a sub-model icon will represent a whole sub-model which is a part of the system under study. For instance, when several local area networks are to be interconnected by a satellite backbone, each local area network can be represented by a sub-model icon. In general, a sub-model icon is constructed by combining several simple or generic icons. It can also include other sub-models.

The editor process, internally, uses a graphical database, which contains all information about
icons, pop-up menu and editing rules, allowing the user to edit graphically his communication system. That database is necessary in the editing phase, and is not used directly by the user.

In order to reduce the number of errors made at this step, the user is assisted by a help menu, and some predefined restrictions on the available choices depending on the models used and the way they have to be linked. The Editor generates, in a manageable manner, the description of the edited system following the ADL syntax.

2.3. The ADL Process

This phase is handled by a process called the ADL process. The ADL description produced in the previous phase constitutes the input of this process.

This process includes three main tasks which are compiling the ADL description, validating the corresponding architecture, also called checking task, and generating the QNAP2 code.

The role of the checking task is to verify the correctness of the description, i.e.:

- completeness of the architecture; namely if the communication system architecture contains what is required for the system to function properly;
- local connectivity; namely if a relationship has been defined between two basic models, a relationship instance may be established between their instances in the network architecture. Otherwise, we cannot couple their instances together;
- full connectivity; namely if all the elements created by the user are linked in such a manner as to avoid having two or several portions not connected to each other.

The checking task is done before the QNAP2 code generation. The concepts used to carry out the checking task will be presented in section 5.

2.4. The Experimenter Process

The experimenter goal is to help to formalising a proper experimental plan to obtain the maximum information with the minimum number of experiments, also called scenarios. The experimenter allows the user to concentrate on the desired conditions for model execution and the treatment of results. A scenario is defined mainly by varying the values of some parameters, called free variables. Each free variable is assigned either a set of values, or a range defined by the lower bound, the upper bound and a step. The user can also define a scenario using the notion of parallel step variation, which means that, two or several free-variables will change their values simultaneously.

2.5. The Simulator Process

This process compiles and executes the generated code which is QNAP2 code. At this stage the QNAP2 simulator is invoked. The execution depends on the scenarios defined by the user. At the end of the execution, the desired results are produced.

2.6. The Post-Processor

The recorded results of the simulation can be statistically analysed with the post-processor, so that aggregate measurements of interest can be reported. The post-processor provides several routines that perform statistical analysis. Thus, it allows the computation of statistical information in order to draw direct cause-and-effect inferences about system operation.

2.7. The Plotter Process

This process is to represent the results provided by the simulator in different ways depending on the choices of the user. The results can be visualised in the form of charts, tables, lists, or a combination of them.
2.8. The Animator Process

Animation serves many purposes. It can serve as a powerful validation device, particular with experts who “know” the “look and feel” of a system and can assess it more easily visually than statistically.

The animation consists in visualising the execution of the simulation. It can depict messages flowing between nodes, and dynamic values of objects attributes. The animator process can be executed in parallel with the simulator process to show the progress of the simulation on-line. In this case, we can speak of a synchronous animation. On the other hand, the animator can work asynchronously, namely this process will be activated, after the simulation is finished. In this case, the process needs a specific file containing the trace of the simulation.

2.9. The Reporter Process

The reporter make the management of documentation easier and allows the development of tailored reports. Actually, intermediate reports can be produced on a model, on a run, on a scenario, or on an analysis. Thus, the user has the capability to choose the kind of information he wants to include in its final reports.

3. AMS PROTOTYPE

In order to be operational in a reasonable time, AMS has been designed on the basis of existing and proved packages rather than starting from scratch. QNAP2, GSS, MODLINE and S-PLUS are the main packages that are used to operate AMS. Based on those tools, a prototype of the atelier has been implemented. It is operational on workstations and it is made available through a graphical user interface, known as the workbench, and by mouse and menu-driven operations on iconic objects.

QNAP2 comprises an object oriented specification language, based on PASCAL and SIMULA, for the description of the models under study and the control of their resolution. Efficient resolution algorithms are available, including discrete event simulation, exact and approximate mathematical methods. QNAP2 is a very powerful and easy to use tool, and has been coded in FORTRAN77 for portability. Nevertheless, the QNAP2 package suffers from bad management of the memory allocation when running the simulator. For example, the fragmentation ratio increases when the simulation time is long; this happens when the simulation model needs an extensive size of memory.

Concerning GSS, it provides generic graph editing facilities to make the development of graphical performance modelling tools easier. A GSS graph consists of a collection of nodes, links and comments. GSS displays the graph in a window, and then the user can create, move, modify or delete objects under control of the mouse, or use a form like interface for changing the values of object’s attributes.

MODLINE is a Unix-based modelling environment. It consists of a set of (mostly) graphical tools connected to each other by a typed object system, i. e., each object has a type definition including its attributes and possible operations. Objects may, for example, be models, results, reports or collections of input values. MODLINE provides functions to create and delete certain objects, and to establish relational links between them.

S-PLUS is a statistical analysis package, offering an interactive graphical environment for visually exploring and thoroughly analysing data. Using graphical displays and various other data analysis tools, the user can obtain an understanding of the structure of his data, their distribution, and more. In addition, S-PLUS includes an object oriented programming language and a unified paradigm for developing statistical models which will be used to create AMS post-processor routines.

Figure 2 shows the available tools in the prototype and on which packages they are based. Furthermore, the graphical user interface (GUI) of the prototype is the same as the GUI of MODLINE. The simulator is based on QNAP2, the post-processor is based on S-
PLUS, the graphical editor is based on GSS, the textual edition of an architecture can be done using ADL. The processes Experimenter, Plotter, Animator, and Reporter are based on MODLINE.

4. DETAILED BASIC MODELS
Each basic model is to be detailed enough as to reflect the exact behaviour of its corresponding target component. Therefore, we have to specify the functions performed by the target component as exactly as possible. This is why basic models are called Detailed Basic Models (DBMs). The main advantages of this approach are to facilitate the validation of the model, to have accurate measurements, and to seek the largest possible number of parameters to characterise the target component. In order to facilitate carrying out these models, their interconnection and their maintenance, an internal structure is proposed [11]. This structure has to be followed by each DBM, which is being considered to be included in the library. Figure 3 shows this structure; it consists of three blocks : Behaviour Engine Block (BEB), Interface Block (IB) and Measurement Blocks (MeB).

![Figure 2: AMS Prototype](image)

![Figure 3: Internal structure of a DBM](image)
4.1. The Behaviour Engine Block

The behaviour of a target component is modelled within the BEB of its associated DBM. The BEB is an open network of stations, where each station includes a queue with limited or unlimited capacity, and one or several servers. The network of stations is open because it receives (respectively sends) from (respectively to) the outside different messages. The configuration of the network of stations and the services offered by each station are left to the responsibility of the modeler. The complexity of the network of stations heavily depends on the complexity of the target component.

There are parameters accessible by the user of AMS, which may influence the performance of the system where the DBM is instanced. These parameters make the DBM flexible from the user's point of view. The default values of those parameters are specified by the modeler, and can be modified by the user, according to the range within which the parameters can vary.

Besides, the DBM may have other parameters which are not accessible by the user. The values of these parameters (called transparent parameters) depend on the architecture in which the DBM is used, namely on the values of other DBM parameters. For instance, the DBM IP fragments the segments it receives; the size of the fragments depends on the kind of DBM it is connected to: the size of the fragment for an Ethernet is up to 1500 bytes, while it does not exceed 1024 bytes for a Token-Ring.

4.2. The Interface Block

The interface block is an important part of a DBM for several reasons. The modularity aspect of a DBM is reinforced by means of its presence. It allows the modeler to develop a DBM independently of any system of which it can be a component. Well-defined interface block also promotes the reuse of the whole model. Finally, it frees the BEB of the messages exchange with the outside world (i.e. another interface belonging to another DBM).

A DBM can have several interfaces, N in figure 3. This number N can either be a fixed value known during the modelling phase or can vary. In the latter case, the modeler can progressively add new interfaces to the DBM.

The validity of the interconnection between several DBMs is not checked by the DBMs themselves but, by the ADL process. Figure 4 shows how an interface transmits messages from its BEB to the outside world and vice versa.

4.3. The Measurement Block

With regard to measurements, the user can be satisfied with the default list of measurements presented by the atelier. Otherwise, he can make his choice from the exhaustive list of measurements related to that particular DBM.

The measurement block contains two types of measures. The first type reflects the behaviour of a DBM, and it is associated with the BEB. The second type is associated with the interfaces, and it mainly shows the data flow entering and leaving the DBM.

The measurements must be meaningful to a user who is not specialised in the field of queueing networks. All the aspects related to this field are transparent. These measurements must be related to some metrics currently used in the field of communication systems, e.g. throughput, response delay, etc.

They are managed by a set of ON/OFF switches. Namely, each measurement is associated with a switch. If the switch is set to ON, the associated measurement is computed, otherwise it is ignored. This policy has two advantages:

1. The user is not submerged by a long list of measurements which may not all be useful for his study.
2. The simulation time is reduced if some measurements are not computed.
5. CONCEPTS FOR ARCHITECTURE VALIDATION

To have a more flexible interconnection between DBMs, the *facet* and *superfacet* concepts will be introduced first, and then, we will try to use the well known *service user/service provider* concept [13]. These concepts lead to a high degree of abstraction to validate the connections during the ADL phase.

A DBM can be connected to a variety of other DBMs. We say that this current DBM (denoted C-DBM) has several facets. A *facet* represents a DBM that can be connected to C-DBM.

Let us notice that to describe a network architecture it may happen that not all these facets are involved. Consider the following example illustrated in figure 5. This figure shows that C-DBM can be connected to DBM1 and DBM2. So, it has two facets which are DBM1 and DBM2.

A *superfacet* of a C-DBM is a sub-set of its facets for which C-DBM has to react identically. The aggregation of several facets into what we called a superfacet is possible because the service concept in the world of protocols allows a protocol to provide a particular service which can be used by several other protocols. Clearly, several protocols can be above another one which behaves in exactly the same way whatever protocol is asking for the service. For instance, the IP protocol reacts identically whether it is used with TCP (Transmission Control Protocol) or UDP (User Datagram Protocol).

Regarding the concept of *service user/service provider* defined in OSI RM, it is exclusively used by the atelier to validate the connectivity of an architecture. Each facet of C-DBM can be seen either as a service user and/or a service provider of C-DBM:

A. If C-DBM represents an application such as FTP, it is considered as a service user of its facets (e.g., TCP), while its facets are seen as service providers (figure 6.A).

B. Otherwise, C-DBM is, on one side, service user of a part of its facets, and on the other side, service provider for the rest of its facets (figure 6.C). There are facets seen as service providers, and facets seen as service users.

For points A. and B., C-DBM has to be connected to at least one of its facets, while for point C., it has to be connected to at least two facets, each one belonging to a different part of its set of facets.
Other concepts, such as the mutual exclusion, are also used to validate complex architectures. All these concepts are structured as a set of rules which have to be applied to the edited architecture.

It should be noted that even if the edited architecture complies with all the constraints previously defined, its components do not necessary interoperate together. The mere fact that a sequence of wireline or wireless links exists between some pair of network devices does not mean that they can interoperate. We must also be able to ensure meaningful communication between network devices by examining their ability to exchange data. For example, if a desktop computer runs only an OSI protocol stack, it cannot interoperate with an FTP server running only an Internet protocol stack.

6. EXAMPLE

AMS should allow the user to study examples such as the communication system based on satellite backbone shown in figure 7. This system is intended to offer communication services through the efficient use of satellite networks. It is composed of geographically scattered Customer Network Facilities (CNF) interworking through a backbone satellite which will allow direct communication paths between the earth stations located at the CNFs. The CNF consists of a LAN connecting several workstations. The LANs could be high speed LAN such as FDDI or medium speed LANs such as Ethernet. The editing could be done in a top-down manner. Indeed, the system can be broken down into three main components. On one hand, there is a satellite backbone component, and on the other hand, many CNF components. Each CNF is connected via a gateway component to the satellite.

Our primary objective in this paper is to show that AMS is operational. Hence, we will limit our illustration to a concrete and classical example \[14, 15\] which is a simplified architecture (figure 8). It consists of an interconnection network via a satellite backbone for which the network performances are obtained in \[14\]. Yang et al., evaluated the performance of that system both by analytical and simulation methods based on the QNAP2 package. Let us remind that QNAP2 is the kernel of AMS simulator.
In this section, two fundamental steps are illustrated through the example: how the editing of this simplified architecture can be done using the atelier, and how results can be obtained from the atelier. In order to make the comparison of some results easier, the performance results we present, are those found in [14]. The objective of simulating this architecture is to study the congestion of the system. Indeed, congestion can dramatically influence the throughput of the network, the response time, etc.

6.1. Editing the architecture

The simplified architecture is composed of a satellite backbone, and two Ethernet networks. Each LAN is connected via a bridge to the satellite, and supports TCP/IP protocols, upon a MAC layer. FTP is the only application that is handled by the workstations associated with each LAN.

The user can edit such an architecture in the main window using either the simple icons and the generic icons, or in a hierarchical manner, using the sub-model editing facility. In figure 8, all the available facilities are used. The simple icons are the Ethernet MAC, IP, TCP, FTP and the Satellite repeater; the generic icon is represented by the Ethernet-Satellite bridge; and the sub-model icon is represented by a stack of protocols which is identical to the stack edited in the main window.

![Simple architecture](image)

Figure 8: Simple architecture
Each basic model is characterised by a set of parameters and a set of measurements. A dialogue box allows to modify parameters or to set measurements. The user may change the predefined parameter values, or make a sub-set of them to be free. Making a parameter free will allow the definition of different scenarios. The user has to choose symbolic names for these parameters. The symbolic names are called free variables.

In our case, two parameters will be associated to free variables. They are the congestion window of the TCP sender, and the bit error rate of the satellite repeater. The symbolic names of these two defined free-variables are respectively “congwin” and “ber”.

Figures 9 and 10 show the list of TCP parameters and the list of satellite repeater parameters. Each parameter is associated with either a predefined value or a free-variable. Besides, each measurement is associated with a switch initially set to OFF. To have some measurements, the user has to ask explicitly for their computation by setting to ON their corresponding switches. Figure 11 shows the list of TCP measurements, some of them are simple and others are more complex; namely, the mean, the variance, the maximum and the minimum can be processed if the user asks for their computation.

6.2. Performance results

Once the first step is successfully achieved, the QNAP2 code is generated corresponding to the edited architecture. The second step is about how to obtain results from AMS. As we said earlier, the user has to ask explicitly for the measurements he wants to obtain. Using the Experimenter, one or several scenarios using a form-like interface can be edited (figure 12). Either a set of values or a range is assigned to each free variable.
The Simulator will launch the appropriate number of simulations corresponding to the edited scenarios. For the example of figure 12, ten simulations will be executed.

The AMS Post-Processor will be used to calculate the throughput which is defined, in this case, to be the file size divided by the file transfer time. File transfer time is a measurement of FTP DBM. In our example, its corresponding switch has to be set to ON.

Finally, the Plotter will allow us to draw some significant charts, such as the throughput versus congestion window or the throughput versus bit error rate which are the most results found in [14].

The tuning of our system fulfils the conditions chosen by Yang, as follows: the propagation delay on the satellite backbone is equal to 257.6 ms with a bandwidth of 1.544 Mb/s, the bridges are modelled as learning bridges [16] with a packet switching time equal to 0.1 ms, and the Ethernet bandwidth equal to 10Mb/s. Furthermore, a workstation on one LAN is assumed to be actively sending a file with a size of 500 Kbytes to the other workstation on the other LAN via the satellite repeater.

Furthermore, TCP uses a traditional sliding window with a fixed size through the whole life of a connection, its segment size is equal to 1500 bytes, and the timeout is set to 1 second.

Figures 13 and 14 show the charts representing the throughput versus the congestion window for BER = 0 and $10^{-5}$. We include the results obtained by AMS and the results found in Yang’s paper. These figures depict that both results are almost identical. This validates on one side our models and on the other side prove that we gain confidence in our atelier AMS.
Figure 13: Throughput (Mb/s) Vs. Window Size (BER=0)

Figure 14: Throughput (Mb/s) Vs. Window Size (BER=10^{-5})

7. CONCLUSION
We have described the design of AMS, an atelier for modelling and simulating communication systems. The atelier has been developed inside the OSISIM project. This atelier is meant to be able to model and simulate complex systems. The atelier is made up of four independent phases, each phase involves one or several processes. The division adopted allows a high degree of modularity and hence the portability.

A prototype of the atelier is already operational. This has been proved in the previous section in which we have studied an example. The latter has shown how a communication system based on a satellite backbone is edited graphically, and how it is tuned before launching simulations meant to assess the performance of this system. The congestion issue is treated at the transport layer, and comparison is done with the results found in Yang’s paper.

8. ACKNOWLEDGEMENTS
We wish to express our gratitude to SAIT Systems with whom we have been working on the OSISIM project. Besides, we gratefully acknowledge the financial support granted by IRSIA. The authors are grateful to the two anonymous reviewers for their thoughtful and constructive comments for the improvement of the paper.
9. REFERENCES


