University of Macedonia
Master Information Systems
Networking Technologies
Professors: A.A. Economides & A. Pomportsis

Subject: Free tools for Network Simulation

Voultsiou Efthimia

A.M.: 23/05

Thessaloniki, 2006

Πανεπιστήμιο Μακεδονίας
Μεταπτυχιακό Πρόγραμμα στα Πληροφοριακά Συστήματα
Τεχνολογίες Δικτύων
Καθηγητής: Α.Α. Οικονομίδης & Α. Πομπόρτσης

Θέμα: Free tools for Network Simulation

Βούλτσιου Ευθυμία

Α.Μ.: 23/05

Θεσσαλονίκη, 2006

Abstract

Broadly defined, simulation is a way to model a representation of a system, so that the system's properties and behaviour can be inferred. Because the model represents a system, it can be manipulated in way that would be difficult, expensive, or impossible to perform on the actual system. Thus, simulation is an excellent way to pretest proposed system or to evaluate existing system.

There is a large number of tools for network simulation, both commercial and non-commercial. The aim of this paper is to overview some of the most important and most representative free tools. It gives a description of the simulator, its use and target, the tools provided with it and tries to describe the simulation engine and the function of the network unit to be simulated.

Περίληψη

Είναι ευρέως γνωστό ότι η προσομοίωση είναι ένας τρόπος παρουσίασης ενός συστήματος, έτσι ώστε οι ιδιότητες και η συμπεριφορά του συστήματος να ερευνηθούν. Επειδή το μοντέλο αντιπροσωπεύει ένα σύστημα, μπορεί κανείς να το χειριστεί με τρόπο που θα ήταν δύσκολος, ακριβός ή ακόμη και αδύνατος να εκτελεστεί το πραγματικό σύστημα. Κατ' αυτόν τον τρόπο, η προσομοίωση είναι ένας εξαιρετικός τρόπος να ελέγξουμε εκ των προτέρων ή να αξιολογήσουμε το υπάρχον σύστημα.

Υπάρχει ένας μεγάλος αριθμός εργαλείων προσομοίωσης δικτύων, εμπορικών και μη εμπορικών. Σκοπός αυτής της εργασίας είναι να παρουσιάσει μερικά από τα πιο σημαντικά και πιο αντιπροσωπευτικά εργαλεία που διατίθενται δωρεάν (free tools). Δίνεται μια περιγραφή του προσομοιωτή, της χρήσης του και του σκοπού του, των εργαλείων που παρέχονται μαζί με αυτόν, καθώς και του τρόπου λειτουργίας της μονάδας δικτύου που θα προσομοιωθεί.

Contents

Περιεχόμενα

3. Introduction

Different researchers use different definitions of simulation depending on the way they conceive and use it [1]. In the context of computer networks, we can say that simulation imitates behaviour of a real world or hypothetical future system by a computer program for the purpose of study.

Properties of simulation include the following:

- Simulation creates a model of a system, which is used to explain system behaviour and to see how the system performs under varying conditions to design the system with desirable performance characteristics
- Simulation usually represents the system on a smaller scale for easier study when compared to the full scale
- Simulation allows to study a system in well-defined and well-known conditions, repeatability is necessary in order to understand events
- In discrete simulations variables change at discrete time intervals between discrete values, in continuous simulations variables change anytime between any values, computer networks are by nature discrete and discrete simulations are used to represent them.

Some people find network simulations useful to conduct experiments on models of large-scale networks with many nodes and links, which could not be conducted on real large-scale networks. Study of routing protocol behaviour is an example of such experiments.

Other researchers use simulations to explore individual components of a computer networks, such as individual OSI-model layers. It includes study of properties of physical-layer transmission media end devices, such as optical amplifiers. Another example is study of behaviour of transport-layer congestion control mechanisms under varying conditions.

There are numerous network simulation free tools on the market today. Many research centres have designed simulators for their own purpose and specifically designed for the simulation of a particular protocol or problem. This results in a plethora of simulators sometimes based on a more general one or a common initial framework that evolved differently over time, which were not intended to evolve and are unsuitable for other simulators [2]. Moreover, the documentation available is often

sparse and poor, or confidential, bugs are not fixed, later versions are not made available and the simulators have not evolved with the protocol were intended to simulate at the time being. As a result, some simulators are better described than the others in this paper.

Some of the most important or most representative free tools for network simulation are OMNeT++, REAL, NS-2, cnet, C++SIM, J-Sim, SSFNet, Class, Smurph [3][4][5].

4. OMNET++

OMNeT++ is an object-oriented modular discrete event simulator for modeling communication networks, multiprocessors and other distributed systems [6]. OMNeT++ is open-source, free for academic and non-profit use.

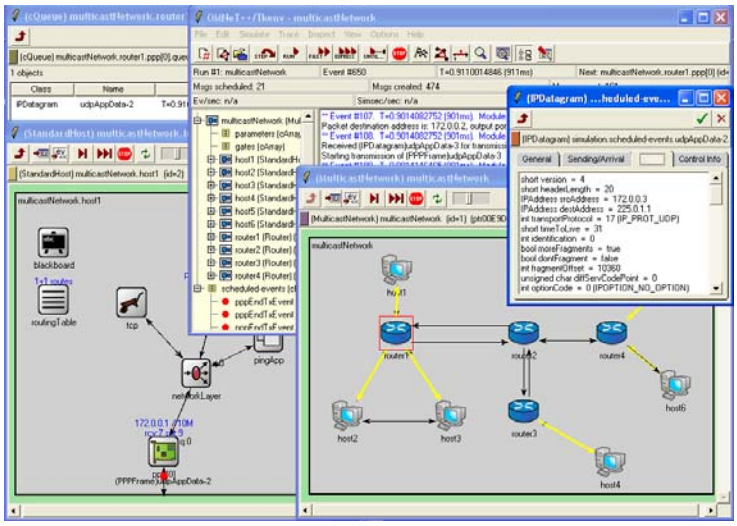
The name itself stands for Objective Modular Network Testbed in C++. The roots of OMNeT++ reach back to OMNeT, a simulation library written by György Pongor in Object Pascal at the Technical University of Budapest. The development started as a programming course assignment in 1992. The principal author was Andras Varga, but several people have joined the development for shorter periods. The purpose of the development was to give the internet community a powerful tool for discrete event simulation. The targeted audience is PhD students, university lecturers and other people from the education or research area. OMNeT++ has been available to the public since September 1997.

Its primary application area is the simulation of communication networks and because of its generic and flexible architecture [7], it has been successfully used in other areas like the simulation of IT systems, queueing networks, hardware architectures and business processes as well. OMNeT++ is rapidly becoming a popular simulation platform in the scientific community as well as in industrial settings.

The simulator can be used for [3]:

- traffic modeling of telecommunication networks
- protocol modeling
- modeling queueing networks
- modeling multiprocessors and other distributed hardware systems
- validating hardware architectures

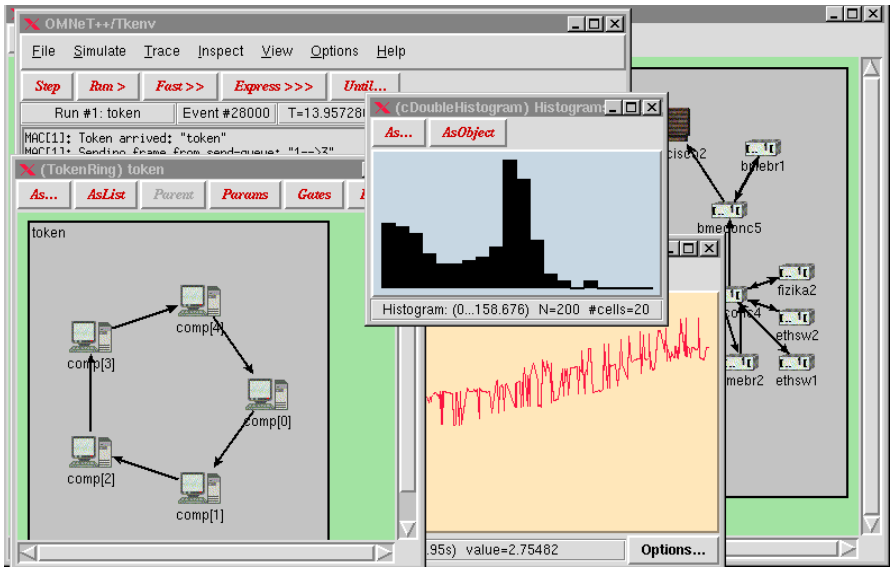
- evaluating performance aspects of complex software systems
- modeling any other system where the discrete event approach is suitable.



An OMNeT++ model consists of hierarchically nested modules. The depth of module nesting is not limited, which allows the user to reflect the logical structure of the actual system in the model structure. Modules communicate with 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.

Modules can have parameters which are used for three main purposes: to customize module behaviour; to create flexible model topologies (where parameters can specify the number of modules, connection structure etc) 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 does not need to learn a new programming language, but he is assumed to have some knowledge of C++ programming.



OMNeT++ simulations can feature different user interfaces for different purposes: debugging, demonstration and batch execution. Advanced user interfaces make the inside of the model visible to the user, allow him to start/stop simulation execution and to intervene by changing variables/objects inside the model. This is very important in the development/debugging phase of the simulation project. User interfaces also facilitate demonstration of how a model works.

The simulator as well as user interfaces and tools are portable: they are known to work on Windows and on several Unix flavours, using various C++ compilers. OMNeT++ also supports parallel simulation.

5. REAL

REAL (**RE**alistic **A**nd **L**arge) is a network simulator originally intended for studying the dynamic behaviour of flow and congestion control schemes in packet-switched data networks [8]. It provides users with a way of specifying such networks and to simulate their behavior. It was written at Cornell University by S. Keshav and based on a modified version of **NEST 2.5** [2]. Source code is provided so that interested users can modify the simulator to their own purposes.

REAL is written in C [9], and will run on Digital Unix/ SunOS/ Solaris/ IRIX/ BSD4.3/Ultrix /UMIPS systems on VAX, SUN, SPARC, MIPS, Alpha, SGI or DECstation hardware.

It provides around 30 modules that exactly emulate the actions of several well-known flow control protocols (such as TCP), and 5 research scheduling disciplines (such as FIFO, Fair Queueing, DEC congestion avoidance and Hierarchical Round Robin). The modular design of the system allows new modules to be added to the system with little effort.

The simulator takes as input a *scenario*, which is a description of network topology, protocols, workload and control parameters. It produces as output statistics such as the number of packets sent by each source of data, the queueing delay at each queueing point, the number of dropped and retransmitted packets and other similar information. It includes a graphical user interface (GUI) written in Java by Hani T. Jamjoom at Cornell University. The GUI allows users to quickly build simulation scenarios with a point-and-click interface. The GUI also allows scenarios to be simulated on a server generously loaned to Cornell University by Digital Inc. This makes it unnecessary for users to download, compile and build the simulator in order to run simulations. Instead, the scenarios, once created, are shipped to the server for execution, and the results are shipped back to the GUI for display.

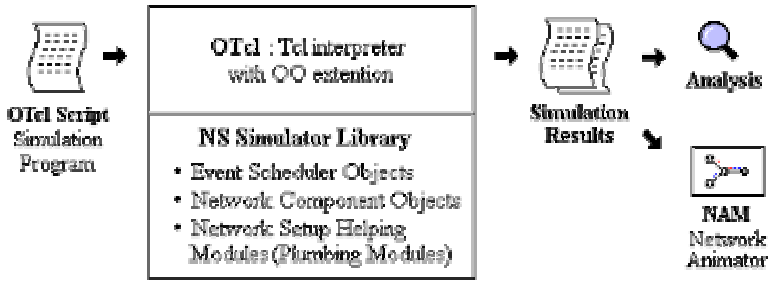
6. NS-2

NS2 is an open-source simulation tool that runs on Linux [10]. It is a discreet event simulator [11] targeted at networking research and provides substantial support for simulation of routing, multicast protocols and IP protocols, such as UDP, TCP, RTP and SRM over wired and wireless (local and satellite) networks. It has many advantages that make it a useful tool, such as support for multiple protocols and the capability of graphically detailing network traffic. Additionally, NS2 supports several algorithms in routing and queuing. LAN routing and broadcasts are part of routing algorithms. Queuing algorithms include fair queuing, deficit round-robin and FIFO.

Ns began as a variant of the REAL network simulator in 1989 and has evolved substantially over the past few years [12]. In 1995 ns development was supported by DARPA through the VINT project at LBL, Xerox PARC, UCB, and USC/ISI. Currently ns development is support through DARPA with SAMAN and through NSF with CONSER, both in collaboration with other researchers including ACIRI. NS2 is

available on several platforms such as FreeBSD, Linux, SunOS and Solaris. NS2 also builds and runs under Windows.

NS is written in C++. The package provides a compiled class hierarchy of objects written in C++ and an interpreted class hierarchy of objects written in OTcl (MIT's object extension to Tcl - Tool Command Language) which are closely 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.

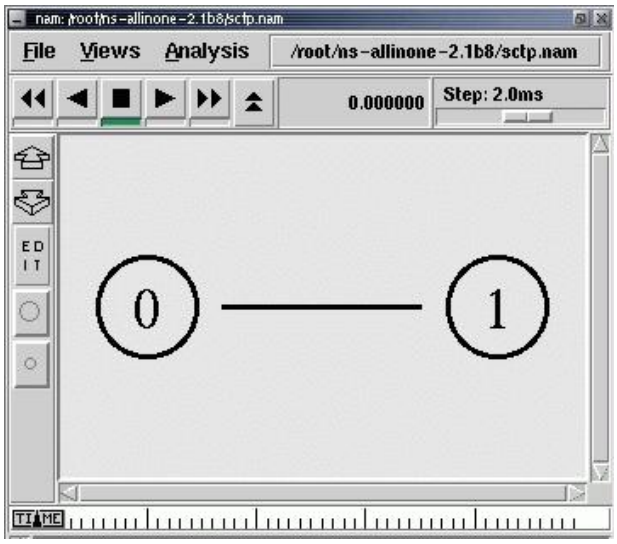


Tcl procedures are used to provide flexible and powerful control over the simulation (start and stop events, network failure, statistic gathering and network configuration). The Tcl interpreter has been extended (OTcl) with commands to create the networks topology of links and nodes and the agents associated with nodes.

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 type of components of a node: those model endpoints of the network where packets are fed or consumed. Users create new sources or sinks from the class Agent. NS currently supports various TCP agents, CBR, UDP, and others protocols, including RTP, RTCP, SRM. There is no mention of ATM protocols.



Links are characterized in terms of delay and bandwidth. They are built from a sequence of connector 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 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.

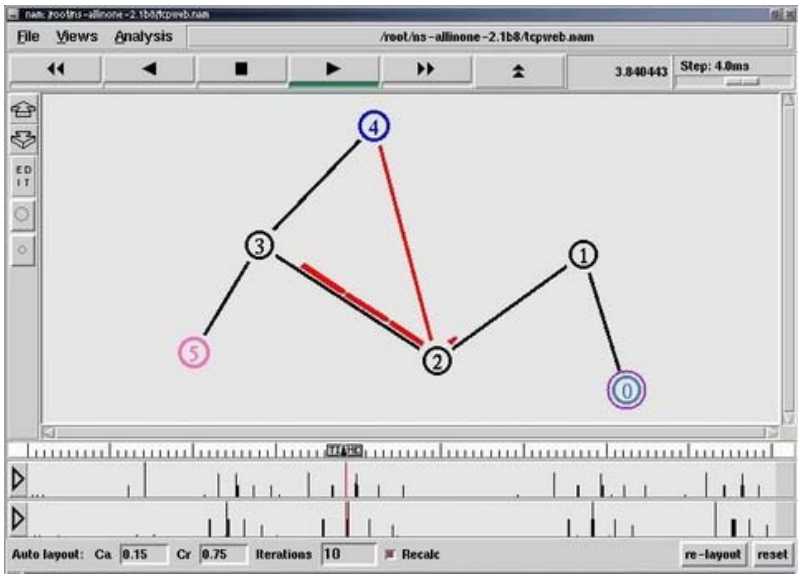
The output buffers attached to a link in a "real" router in a network are modeled by queues. In NS, queues are considered as part of a link. NS allows the simulation of various queuing and packet scheduling disciplines. C++ classes provided include drop-tail (FIFO) queuing, Random Early Detection (RED) buffer management, CBQ (priority and round-robin), Weighted Fair Queuing (WFQ), Stochastic Fair Queuing (SFQ) and Deficit Round-Robin (DRR).

Traffic generation in NS looks rather basic in the current implementation. For the purpose of TCP, only FTP and Telnet traffic can be generated; otherwise, NS provides an exponential on/off distribution and it allows generating traffic according to a trace file.

In order to analyze results, NS provides classes to trace each individual packet as it arrives, departs or is dropped, and to record any kind of counts, applied on all packets or a per-flow basis. The trace can be set or unset as desired by the user.

The user has to specify the routing strategy (static, dynamic) and protocol to be used. This is done with a procedure in the class simulator. Supported routing

features include asymmetric routing, multipath routing, Distance Vector algorithm, multicast routing.



Other features of NS include error models where the unit could be packet, bit or time based, and mathematical classes for the approximation of continuous integration by discrete sums and for random number generation (implementation of the minimal standard multiplicative linear congruent generator of Park et al). In order to verify some aspects of the protocol to be simulated, NS includes some validation tests distributed with the simulator. Is as well includes capabilities to make the simulation topologies dynamic although this latest point is still somewhat experimental.

The simulation engine is extensible, configurable and programmable. The current implementation is single-threaded (only one event in execution at any given time). It does not support partial execution of events nor pre-emption.

Events are described by a firing time and a handler function. The type of event scheduler used to drive the simulation can be chosen among the four presently available: a simple linked-list (default), heap, calendar queue, and a special type called real-time. Each one is implemented using a different data structure.

The simple linked-list scheduler provides a list of events kept in time-order, from the earliest to the latest. This requires scanning the list to find the appropriate entry upon insertion or deletion. The entry at the head is always executed first. Entries with the same simulated time are extracted according to their order in the list.

The heap scheduler code is borrowed from the MARS-2.0 simulator (that itself borrowed the code from MIT'S NETSIM). This implementation is superior to the linked list scheduler when the number of events is large. Insertion and deletion times are in O (logn) for n events.

In the calendar queue scheduler implementation, events with the same "month/day" of multiples "year" are recorded in one "day".

The real-time scheduler is still under development and is currently a subclass of the list scheduler. It is well suited when events arrive with a relatively slow rate. Execution of events should occur in real time.

NS has scaling constraints in terms of storage requirements of the routing tables: each node maintains a route to all other nodes in the network, resulting in aggregate memory. This grows with the number of nodes in the network. VINT proposes to replace NS's flat addressing conventions by implementing efficient hierarchical routing table look-ups in the nodes objects.

NS represents an underlying transmission link and its corresponding scheduling and queuing algorithms in a single object, but the VINT simulation framework will require their separation in order to flexibly combine different scheduling algorithms with different underlying link technologies. In the current implementation of NS, each module that implements a scheduling discipline need be changed when adding modules to support a new link type.

Another modification is the performance improvement to the event scheduler. The linear search insertion algorithm should be replaced with a heap or a calendar queue. Coexistent scheduling algorithms can be derived from base class abstraction and are easily implemented in NS due to its C++ implementation.

Since the simulation is likely to generate enormous amounts of data, visualization will have to play a key role and will be added to all phases of the simulation process, from inputs to outputs.

A tool that generates random topologies according to user-specified parameters will help in describing very large topologies; a tool for the manipulation of data packets at the bit level allows detailed tests.

For the study of protocol interaction and behaviour at significantly larger scale, the simulator will provide two levels of abstraction. The detailed level simulator is the one currently built. It allows a fine abstraction of the distinct modules of the simulation and will later include an emulation interface that will allow incorporating a

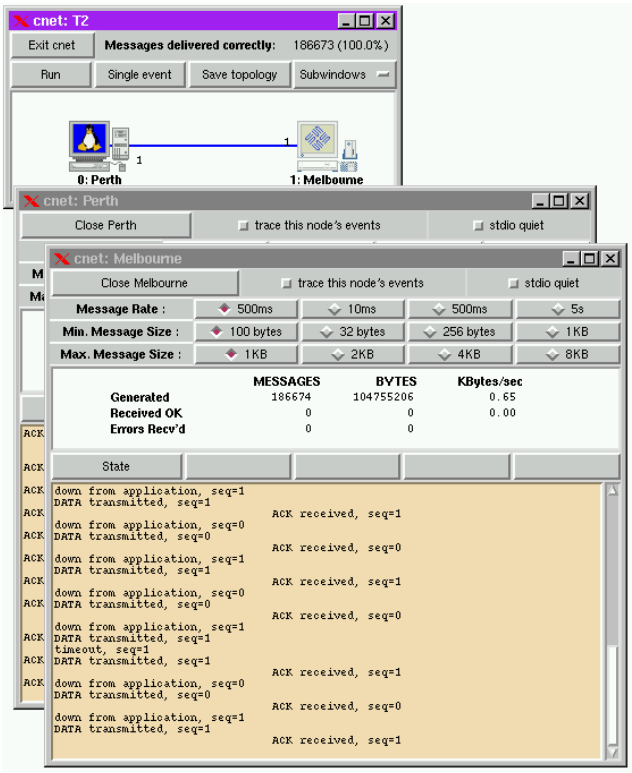
real network node as a component of the simulation. The session level simulator will give a coarse-grain abstraction. Data packets will be represented by flows instead of individual packets. This will reduce the number of events and state required, at the cost of lost detail in the simulation. As an instance, instead of tracing each packet through each router and link, the simulator only calculate the time for that packet to be received by the sink according to the path used.

The project also hopes to apply parallel network simulation techniques because the limits of a single-processor computational power will necessarily be stressed. The project will implement a distributed version of the simulator.

7. cnet

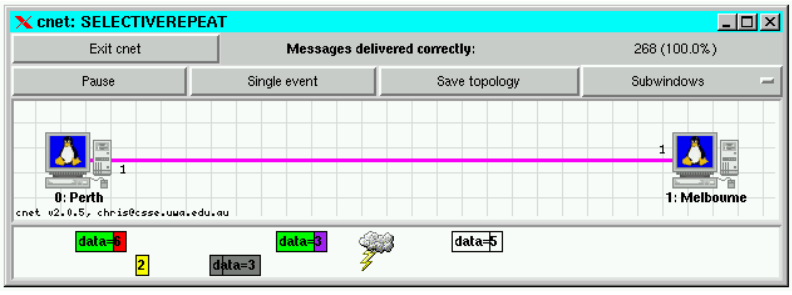
cnet is a network simulator which enables experimentation with various data-link layer, network layer, routing and transport layer networking protocols in networks consisting of any combination of point-to-point links and IEEE 802.3 Ethernet segments [13]. With reference to the OSI/ISO Networking Reference Model, cnet provides the Application and Physical layers. User-written protocols are required to “fill-in” any necessary internal layers and, in particular, to overcome the corrupted and lost frames that cnet's Physical Layer randomly introduces. In addition, advanced users may develop different Application and Physical Layers which exhibit varying statistical characteristics of message generation and data transmission. Simulation sizes may range from two to a few hundred nodes.

cnet either displays the entire network under Tcl/Tk or runs rather less visually on an ASCII terminal. Under Tcl/Tk, cnet provides a graphical representation of the network under execution and permits a number of attributes of the network to be modified while the simulation is running. Nodes may be selected with the mouse to reveal a sub-window displaying the output and protocol statistics of that node. Some of the node's attributes, such as message generation rates and sizes, may be modified while the network is running by selecting choice buttons. Similarly, the default attributes of all nodes in the network may be simultaneously modified by selecting and changing global attributes. From another menu, each node may be forced to reboot, (impolitely) crash, (politely) shutdown and reboot, pause and (hardware) fail.

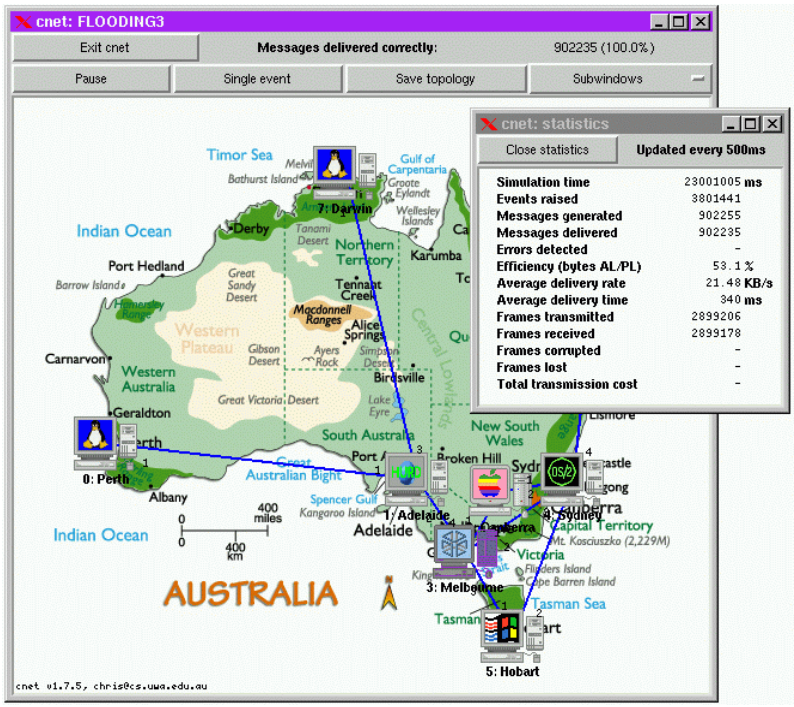


cnet is able to automatically detect and report a wide variety of errors commonly made by undergraduate students when developing their protocols. These include invalid arguments passed to functions, an incorrect specification of links, addresses and timers, undetected message corruption or loss, messages passed to incorrect destinations, out-of-sequence or duplicated messages, etc. Errors are reported in a popup-window describing the error and its location in the student's source code.

cnet can present a limited visualization of data frames traversing the Physical Layer. Using simple rectangles, colours, and short strings, it is possible to display both data and acknowledgment frames, and the contents of some of their fields. In combination, these features may be used to debug implementations of Data Link Layer protocols.



cnet requires network protocols to be written in the ANSI-C programming language and supports their execution within a single UNIX or Linux process. A standard compiler, preferably gcc, is used to compile the user-written protocol code. The compiled code is then dynamically linked at run-time with the cnet simulator. Protocols are invoked directly by cnet itself - they are not interpreted. By design, the protocols do not need to contain any windowing code.



8. C++SIM

C++SIM is an object-oriented simulation package written in C++ [14]. It provides discrete event process-based simulation similar to SIMULA's simulation class and libraries.

C++SIM was originally developed by members of the Arjuna project to aid in the building of fault-tolerant distributed systems. The package has been extensively used for distributed systems simulation, and continues to aid in that area and simulation in general.

Sun workstations running SunOS 4, Solaris 2, HP9000s300 & HP9000s700 workstations using HPUX, Windows 95 and Windows NT (3.5.1 and 4.0), and i386

compatible machines using Linux. The thread packages currently supported include: Sun's lightweight processes (lwp) library, Solaris threads, various flavours of Posix threads, NT threads, the C++ Task Library, C threads, and the Gnu Rex lightweight processes library.

A complete list of the facilities provided follows:

- the core of the system gives SIMULA-like simulation routines, random number generators, queuing algorithms, and thread package interfaces
- entity and set manipulation facilities similar to SIMSET
- classes allow "non-causal" events, such as interrupts, to be handled
- various statistical gathering routines, such as histogram and variance classes
- debugging classes

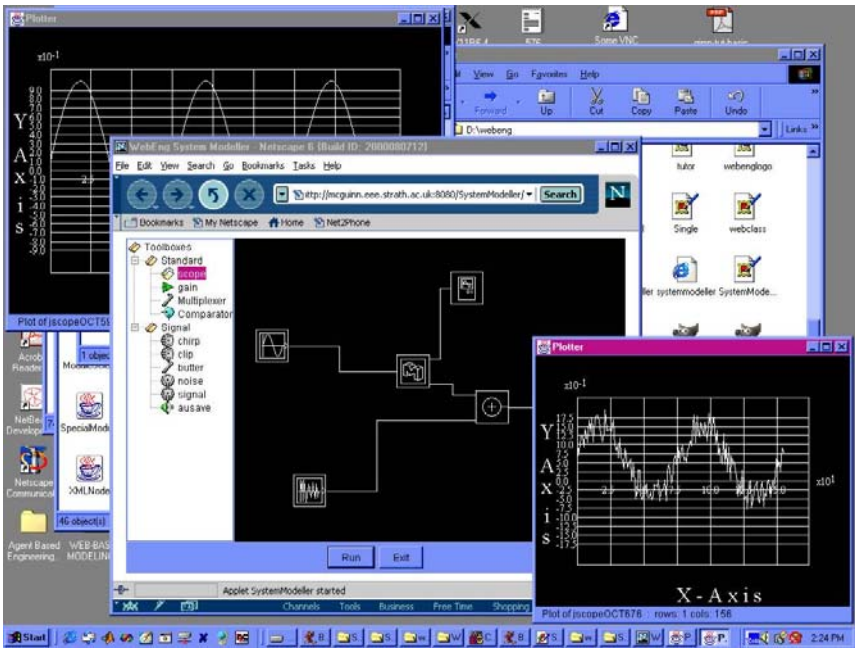
The system also comes with complete examples and test routines which illustrate many of the issues raised in using the simulation package.

9. J-SIM

J-Sim (formerly known as JavaSim) is a component-based, compositional simulation environment for discrete-time process-oriented simulation [15]. Its main application area is queueing network simulation, however, the range of its use can be very wide – almost any system where object states change discretely can be modelled using J-Sim. J-Sim is a Simula-like simulation environment written in Java. J-Sim has been tested on Solaris 8, OS/2 Warp 4, eComStation 1.1, Windows NT 4, Windows XP, and Linux.

It has been built upon the notion of the autonomous component programming model [16]. The basic entity in J-Sim is components. The components in J-Sim are autonomous and are realization of software ICs.

The autonomous component architecture mimics the IC design architecture in the closest possible way. The behaviour of J-Sim components are defined in terms of contracts and can be individually designed, implemented, tested, and incrementally deployed in a software system. A system can be composed of individual components in much the same way a hardware module is composed of IC chips. Moreover, components can be plugged into a software system, even during execution.



On top of the autonomous component architecture there is a generalized packet switched network model. The model defines the generic structure of a node (either an end host or a router) and the generic network components, both of which can then be used as base classes to implement protocols across various layers. Although the model is derived by featuring out the common attributes of network entities in the current best-effort Internet, it is general enough to accommodate other network architectures, such as the IETF differentiated services architecture, the mobile wireless network architecture, and the WDM-based optical network architecture.

J-Sim is a truly platform-neutral, extensible, and reusable environment. It provides a script interface to allow integration with different script languages such as Perl, Tcl, or Python. Similar to ns-2, J-Sim is a dual-language simulation environment in which classes are written in Java (for ns-2, in C++) and "glued" together using Tcl/Java. However, unlike ns-2, classes/methods/fields in Java need not be explicitly exported in order to be accessed in the Tcl environment. Instead, all the public classes/methods/fields in Java can be accessed (naturally) in the Tcl environment.

The salient features of J-Sim are:

- Loosely-coupled, component-based programming model
- Real-time process-driven simulation

- Implementation of an (almost) complete suite of Internet Integrated, Differentiated, and Best Effort Services protocols
- A dual-language environment that allows auto-configuration and on-line monitoring
- Implementation of generic interface classes for trace-driven simulation
- Partial realization of network emulation.

10. SSFNet

The SSFNet is a mature network simulation tool started in 1998 developed by SSFNet project (SSFa) [17]. Most SSFNet components are licensed under the GNU General Public License [5].

SSFNet is made up of the following components:

- Scalable Simulation Framework (SSF): a discrete event simulation of large complex systems written in Java and C++. The SSF is open source.
- Domain Modeling Language (DML): a language for describing the model of your network you wish to simulate. The DML is open source.
- Integrated Development Environments (IDEs): a range of tools to make the process of building your model easier. Some of the IDEs are free, some are not.

11. CLASS

CLASS is a simulator for ATM networks that was developed by the Telecommunication Networks Group of Politecnico di Torino in co-operation with CSELT (Centro Studi E Laboratori Telecomunicazioni) under the research contract "Valutazione di Traffico nell'Interconnessione MAN/ATM" (Traffic Evaluation in MAN/ATM Interconnection) and with the Technical University of Budapest, Hungary [18].

CLASS is completely written in ANSI C language and works without modifications on several platforms. Currently it has been tested under OpenVMS

(VAX and AXP), Linux, HP-UX, Ultrix and MS-DOS (using DJGPP, the MS-DOS version of the GNU C compiler). Generally, it should compile on every system supporting an ANSI C compiler. It also requires the presence of YACC and FLEX. No graphical interface has been so far developed.

CLASS stands for Cell Level ATM Services Simulator and it is devoted to the simulation of connectionless traffic, basically the services offered by B-ISDN (Broadband Integrated Services Digital Network), on networks based on the ATM technology.

Simulations performed by CLASS are at the cell level, taking into account routing and switching functions, the allocation of the bandwidth to different connections, the possible presence in the network of special nodes devoted to the management of connectionless traffic and many other functions that may influence the performance on the network as a whole.

CLASS offers the opportunity of defining arbitrary topologies with a great freedom on the number of nodes within the network, the number of ports connected to each node, the characteristic of the traffic and so on. The topological description of the network, is done through a specialized formal grammar that allows the easy check of the topology definition

Results obtained during the simulation are all written onto ASCII files and can be easily post processed with standard data analysis tools. Results comprise several statistics about messages and cells generated during the simulation, and are referred both to the whole network and to each single link or traffic relation.

The main features of CLASS are:

- Several types of traffic sources (among which a DQDB-like and a TCP source)
- A GCRA-derived shaping function can be applied to the traffic produced by each source
- A GCRA-based policing function at the node input interface (both tagging and discarding cells)
- A dedicated language is used to describe the topology and the parameters of the network to be simulated
- A fixed priority or a Fair Queuing algorithm can be implemented at the output interface of the nodes
- Implementation of the ATM Forum ABR transfer capability

- Several ABR traffic control schemes (RRM, ER) implemented in the nodes
- Integration with ANCLES, the companion Call Level simulator

A special CLASS version has been developed to manage Wireless ATM Networks and the analysis of different handover strategies.

12. SMURPH

SMURPH (a System for Modelling Unslotted Real-time PHenomena) is an object-oriented software package for modelling discrete events in communication protocols [19] [20]. It can be viewed as both an implementation of a certain protocol specification language and a process-driven simulator oriented towards investigating medium access control (MAC) level protocols. In accordance with the above postulate, the structure of the simulator is completely invisible to the user. All the simulation-related operations, as creating and scheduling individual events, maintaining a consistent notion of time, etc., are covered by a high-level interface. A protocol description in SMURPH looks like a program that could be executed on a hypothetical hardware. SMURPH emulates this hardware and thus provides a realistic environment for executing protocols described in its specification language.

SMURPH descends from its predecessor, called LANSF, which was designed and implementation by the authors in 1987. LANSF was programmed in C and has evolved a number of times from its original version. It has been successfully applied to investigating the performance and correctness of a number of protocols for local area networks and other distributed systems.

SMURPH has been programmed in C++. The package is configurable. It is not a single interpreter for a variety of protocols, but it configures itself into a stand-alone modelling program for each particular application.

Although the protocol description language of SMURPH is essentially C++, the standard data types, objects, functions and macros provided by the package extend this language substantially. With this approach, the user gets the full power of C++ combined with the power of a realistic, emulated environment for programming and executed communication protocols.

Protocol execution in SMURPH can be monitored. One reason for monitoring the behaviour of a protocol is to investigate its performance by gathering empirical data. Although SMURPH does not purport to be a protocol verification system, it offers some tools for protocol testing. These tools, the so-called observers, look like programmable dynamic assertions describing legitimate sequences of protocol actions.

13. Conclusions and future work

There is a plenty of tools for network simulation today, both commercial and non-commercial. This paper tried to overview some of the most important and representative free tools. There are seven issues [3] are examined to get an overview about the network simulation tools:

- **Detail Level**: *Does the simulation tool have the necessary power to express details in the model?* In other words, can the user implement arbitrary new building blocks or he is confined to the predefined blocks implemented by the supplier? OMNeT++ is programmable by the user to this extent. Other network simulation tools like NS and CLASS fall into this category.
- **Available Models.** *What protocol models are readily available for the simulation tool?* On this point, non-commercial simulation tools cannot compete with some commercial ones which have a large selection of ready-made protocol models.
- **Defining Network Topology**. *How does the simulation tool support defining the network topology?* Is it possible to create some form of hierarchy (nesting) or only "flat" topologies are supported? Network simulation tools naturally share the property that a model (network) consists of "nodes" (blocks, entities, modules) connected by "links" (channels, connections). Many commercial simulators have graphical editors to define the network; however, this is only a good solution if there is an alternative form of topology description (e.g. text file) which allows one to generate the topology by program. On the other hand, most non-commercial simulation tools do not provide explicit support for topology description: one must program a "driver entity" which will boot

the model by creating the necessary nodes and interconnecting them (SMURPH, NS). A large part of the tools that do support explicit topology description supports only flat topologies (CLASS). Other tools (like OMNeT++) probably uses the most flexible method: it has a human-readable textual topology description format (the NED language) which is easy to create with any text-processing tool (like perl or awk), and the same format is used by the graphical editor. It is also possible to create a "driver entity" to build a network at run-time by program.

- **Programming Model.** *What is the programming model supported by the simulation environment?* Network simulators typically use either thread/coroutine-based programming or FSMs built upon a handleMessage()-like function. For example, SMURPH use FSMs (with underlying handleMessage()), C++SIM use threads and OMNeT++ supports both programming models.
- **Debugging and Tracing Support.** *What debugging or tracing facilities does the simulation tool offer?* Simulation programs are infamous for long debugging periods. C++-based simulation tools rarely offer much more than printf()-style debugging. Often the simulation kernel is also capable of dumping selected debug information on the standard output. Animation is also often supported, either off-line or in some client-server architecture, where the simulation program is the "server" and it can be viewed using the "client". Off-line animation naturally lacks interactivity and is therefore little use in debugging. The client-server solution typically has limited power because the simulation and the viewer run as independent operating system processes, and the viewer has limited access to the simulation program's internals and/or it does not have enough control over the course of simulation execution.
- **Performance.** *What performance can be expected from the simulation?* Simulation programs typically run for several hours. Probably the most important factor is the programming language.
- **Source Availability.** *Is the simulation library available in source?* This is a trivial question but it immediately becomes important if one wants to examine or teach the internal workings of a simulation kernel, or one runs into trouble because some function in the simulation library has a bug or it is not

documented well enough. In general it can be said that non-commercial tools are open-source and commercial ones are not.

There is no perfect free tool for network simulation as most of them have been designed by research centres for their own purpose and specifically designed for the simulation of a particular protocol or problem.

However, it is evident that the more networks are developed, the more the researchers have to focus on this particular issue. Due to network complexity, emphasis should be placed on network simulation and tool development which could not only solve problems but also prevent them from coming up.

References

[1] http://www.cesnet.cz/doc/techzpravy/2003/congestion/congestion.pdf
Article about "Experience with using simulations for congestion control research".

[2] http://www.inrialpes.fr/planete/people/ernst/Documents/simulator.html
A www site that includes notes about network simulators.

[3] http://whale.hit.bme.hu/omnetpp/manual/usman.html#toc_1
Omnet++ user manual.

[4] http://seacorn.ptinovacao.pt/sim_tools.html
The SEACORN (Simulation of Enhanced UMTS Access and Core Networks) web site. It refers to some of the most important commercial and non-commercial network simulators and it includes a summary of some articles about the simulation.

[5] http://www.openxtra.co.uk/articles/network-simulation.php
It includes an article about network simulation, its use and an introduction about some of the network simulators.

[6] http://www.ewh.ieee.org/soc/es/Nov1999/18/Begin.htm
Article about "Using the OMNeT++ Discrete Event Simulation System in Education". The intent of the paper is to contribute to the teaching of computer networks, parallel and distributed systems and discrete event simulation by presenting a simulation system that is ideally suited for educational use.

[7] http://www.omnetpp.org/
The OMNET++ community site. It includes the documentation of OMNET++, tutorials and other related links.

[8] http://minnie.tuhs.org/REAL/
WWW site that describes the REAL simulator. It also contains links for downloading the simulator and its manuals.

[9] http://www.cs.cornell.edu/skeshav/real/overview.html
An overview of the REAL simulator.

[10] http://www.linuxjournal.com/article/5929
The www site of the Magazine of the Linux Community. It contains an article about NS simulator, the process of installing NS and instructions about the way it is used.

[11] http://nsl10.csie.nctu.edu.tw/products/nctuns/NCTUnsReferences/Cradle.pdf
It contains an article that describes the Network Simulation Cradle and presents an analysis of differences between simulations using NS-2's and FreeBSD's TCP/IP implementation.

[12] http://www.isi.edu/nsnam/ns
It describes the Network Simulator NS-2.

[13] http://www.csse.uwa.edu.au/cnet/introduction.html
An introduction to the cnet network simulator.

[14] http://cxxsim.ncl.ac.uk/summary.html
A www site that describes the C++SIM network simulator. It includes information about its history and its environment.

[15] http://www.j-sim.zcu.cz/
A www site that contains general information about J-Sim network simulator.

[16] http://www.j-sim.org/
A very descriptive site about J-Sim simulator. It contains its documentation, its packages, tutorials and link for downloading the simulator.

[17] http://www.net.in.tum.de/software/Web-Workload-Generator-Bachelor-Thesis.pdf
An article about "Design and Implementation of a Web Workload Generator for the SSFNet simulator".

[18] http://www1.tlc.polito.it/class/
The www site of Telecommunication Networks Group. An overview of Class simulator for ATM networks.

[19] http://web.cs.ualberta.ca/~pawel/SMURPH/report.pdf
An article about SMURPH. The subject of the article is "An overview of SMURPH: an Object-oriented Configurable Simulator for Low-Level Communication Protocols".

[20] http://web.cs.ualberta.ca/~pawel/SMURPH/smurph.html
It describes the LanSF, SMURPH and SIDE simulators.