Πανεπιστήμιο Μακεδονίας
ΠΜΣ Πληροφοριακά Συστήματα
Τεχνολογίες Τηλεπικοινωνιών & Δικτύων
Καθηγητές: Α.Α. Οικονομίδης & Α. Πομπόρτσης

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

Θέμα: Μη εμπορικό Λογισμικό για Προσομοιώσεις Δικτύων

Subject: Free Tools for Network Simulations

Γιοβανάκης Γιάννης
ΑΜ 2406

Θεσσαλονίκη 2007

Giovanakis Jiannis
AM 2406

Thessaloniki 2007

Abstract

Network simulation is broadly used today by a number of people working on the field of computer networks. Academicians or professionals, individual or companies use several network simulation software tools for purposes such as: research (on topologies, protocols, etc.), education, assessment of propriety or effectiveness and many others.
There are many network simulators, both commercial and non-commercial. Ussually the non-commercial ones are open–source programs developed by universities for research purposes and free distributed. I will present in this paper five of such open source simulators in an attempt those five tools to be the most indicative and represantative of the kind.

Περίληψη

Η προσομοίωση δικτύων χρησιμοποιείται σήμερα από ένα μεγάλο πλήθος ανθρώπων που ασχολούνται με τα δίκτυα υπολογιστών. Αυτοί οι άνθρωποι μπορεί να είναι ακαδημαϊκοί ή επαγγελματίες, άτομα ή εταιρείες που χρησιμοποιούν διάφορα εργαλεία λογισμικού για προσομοίωση δικτύων, με σκοπούς όπως: η έρευνα (πάνω σε τοπολογίες, πρωτόκολα, κλπ), η εκπαίδευση, η αξιολόγηση καταλληλότητας ή αποτελεσματικότητας και πολλοί άλλοι.
Υπάρχουν πολλοί προσομοιωτές δικτύων, τόσο εμπορικοί όσο και μη εμπορικοί. Συνήθως αυτοί που είναι μη εμπορικοί είναι προγράμματα ανοιχτού κώδικα που αναπτύσσονται από πανεπιστήμια για ερευμητικούς σκοπούς και διανέμονται δωρεάν. Σ’ αυτή την εργασία θα παρουσιάσω πέντε τέτοια εργαλεία προσωμοίωσης προσπαθώντας αυτά τα πέντε να αποτελούν τα ενδεικτικότερα και πιο αντιπροσωπευτικά του είδους.

Περιεχόμενα

Contents

1. Introduction

Simulation is one of the most widely used quantitative approaches to decision making. It is a method for learning about a real system by experimenting with a model that represents the system. The simulation model contains the mathematical expressions and logical relationships that describe how to compute the value of the outputs given the values of the inputs. Any simulation model has two inputs: controllable inputs and probabilistic inputs.
Simulation is not an optimization technique. It is a method that can be used to describe or predict how a system will operate given certain choices for the controllable inputs and randomly generated values for the probabilistic inputs.[1]

In computer network research, **network simulation** is a technique where a program simulates the behavior of a network. The program performs this simulation either by calculating the interaction between the different virtual network entities (hosts/routers, data links, packets, etc) using mathematical formulas, or actually capturing and playing back network parameters from a real production network. Using this input, the behavior of the network and the various applications and services it supports can be observed in a test lab. Various attributes of the environment can also be modified in a controlled manner to asses these behaviors under different conditions. When a simulation program is used in conjunction with live applications and services in order to observe end-to-end performance to the user desktop, this technique is also referred to as **network emulation**.

Network simulators are used to predict the behavior of networks and applications under different situations. Researchers use network simulators to see how their protocols would behave if deployed. It is typical to use a network simulator to test routing protocols, MAC (Medium Access Control) protocols, transport protocols, applications etc. Companies use simulators to design their networks and/or applications to get a feel for how they will perform under current or projected real-world conditions.

The simulator (or network simulator) is the program in charge of calculating how the network would behave. They may be distributed in source form (software) or provided in the form of a hardware appliance. Users can then customize the simulator to fulfill their specific analysis needs.[2]
Properties of a network simulator, which can be used as well for categorizing criteria, may be the following:

1. Operating system that is running on. The most often case is Linux, FreeBSD or some sort of Unix but also Solaris, Mac, Windows, HP-UX, MS, MS-dos can be met.
2. Availability of source code. A simulator can be distributed with its source code for any user interested in further development of the application.
3. Free of any charge or not. Although an open source simulator most of the times means free of charge the opposite is not always true. It can be distributed for free but no code available.
4. Discrete event or continuous time/space modeling. Most of the computer network simulators use discrete event modeling which is most appropriate for representing networks.[3],[4]

[1] An introduction to management science. by D.R.Anderson, D.J.Sweeney, T.A.Williams, Chapter 13, page 586, THOMSON SOUTH – WESTERN ed.11th.
2 http://en.wikipedia.org/wiki/Network_simulation
3 http://en.wikipedia.org/wiki/Discrete_event_simulation
4 Free tools for network simulation by Voultiou Efthimia of MIS University Of Macedonia, 2006

5. Software development environment (framework) or install and ready to go package.
6. Broad range of networks, protocols, network components, etc. covered or specific for e.g. wireless networks, ATM protocol, transport layer, etc.
7. Simulation, emulation software or both. Simulation s/w simulates offline a network, real or imaginary, that all of its entities are virtually operating in the model. Emulation s/w makes use of both parts of a real network (servers, PCs, links) and virtual parts (entities). Thus emulation must, at least partially, operate online with a real network in order to be fed from it with its required inputs.[5]

The following network simulators presented in this paper are: Imunes, NS-2, NCTUns, DESMO-J and deX .

2. Imunes

Imunes is an Integrated Multiprotocol Network Emulator / Simulator framework based on the FreeBSD 4.x OS kernel. Kernel is partitioned into multiple lightweight virtual nodes, which can be interconnected via kernel-level links to form arbitrarily complex network topologies.
The modified FreeBSD kernel allows multiple network stack instances to simultaneously coexist within a single kernel. Each network stack instance can act as an independent virtual node (router or host), connected to other virtual nodes via simulated links, or directly to the outside world via standard network interfaces.
This allows for complex emulated IP network configurations to be constructed on a single machine.[6]
The concept of using virtual nodes inside a kernel for fast network emulation is not entirely new, yet previously published work generally advocated the implementation of kernel-level virtual nodes with capabilities limited to only certain simple functions, such as passing of network frames from one queue to another based on a static precomputed path.
The development of Imunes is based on a thesis that virtual nodes, which could offer the identical rich set of capabilities as the standard kernel does, can be implemented very efficiently by reusing the existing OS kernel code.
The model therefore not only provides each node with an independent replica of the entire standard network stack, thus enabling highly realistic and detailed emulation of network routers; it also enables each virtual node to run a private copy of any unmodified user-level application, including routing protocol daemons, traffic generators, analyzers, or application servers. Furthermore, in later development phases it is expected to enable each virtual node to support multiple network protocols concurrently, such as both IPv4 and IPv6, which would bring Imunes a step closer to allowing for emulation of true multiprotocol networked environments.

During the initial development phase, which is supported by the Croatian Ministry of Science and Technology research grant IP-2003-143, a network emulator prototype will be developed with basic support for virtual Ethernet, point-to-point and Frame Relay links, dynamic RIP and OSPF routing, together with an integrated GUI environment for specification and management of simulation scenarios.
IMUNES comes with a simple Tcl/Tk based graphical user interface console, which operates in two modes: *edit* mode and *exec* mode.

[5] http://www.topology.org/soft/sim.html
[6] http://www.tel.fer.hr/imunes/

Edit mode.
One can use symbols on the left side of the window in order to create network of desired topology. Meaning of icons is following: select, delete, link, router, LAN switch, server node, pc node, RJ45 (interface to the outside world via a real / physical Ethernet interface). One can edit default parameters by double click of the mouse on the node/ link you want to edit. Almost every parameter is editable, from node labels to link bandwidth

Figure 2.1 User interface layout

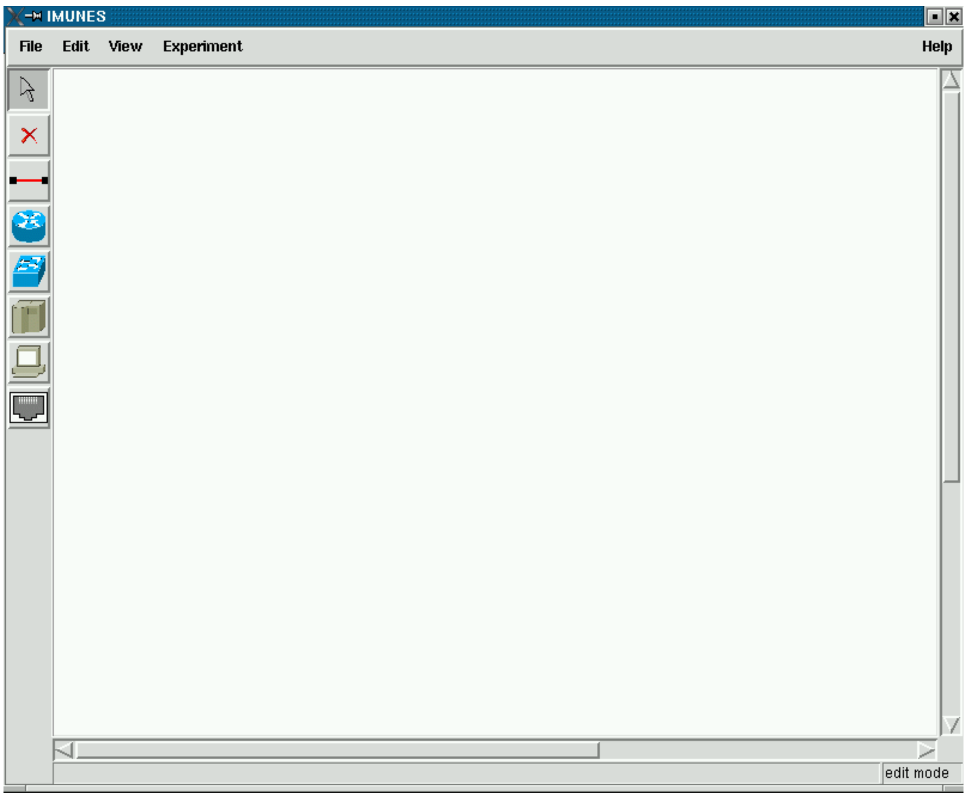
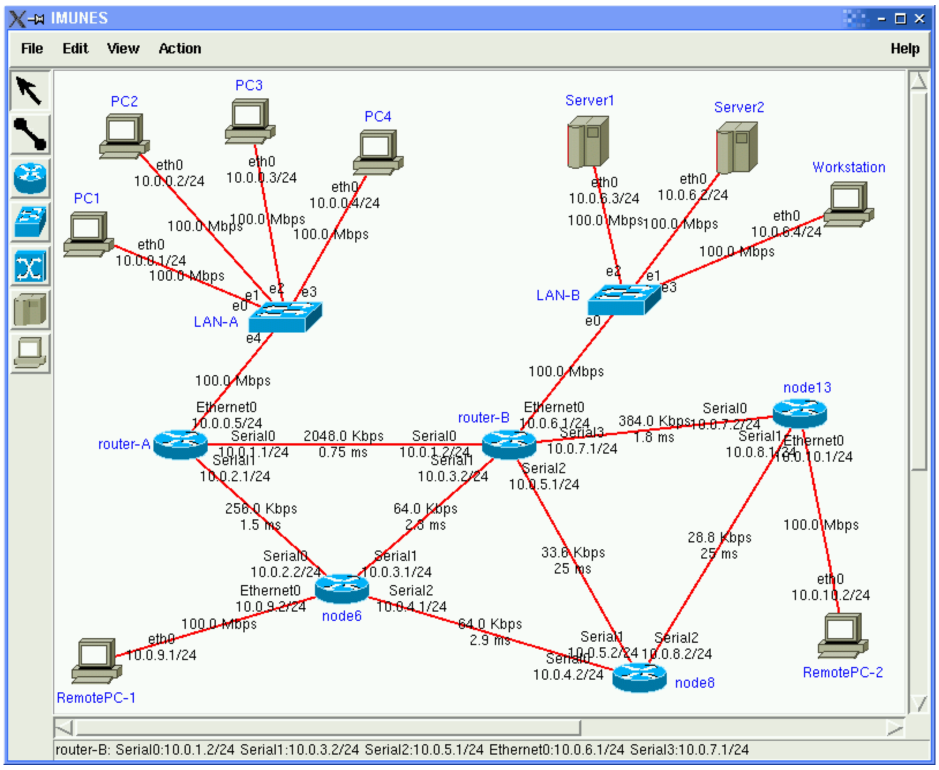


Figure 2.2 An example of a simulated network topology



router-B: Serial0:10.0.1.2/24 Serial1:10.0.3.2/24 Serial2:10.0.5.1/24 Ethernet0:10.0.6.1/24 Serial3:10.0.7.1/24

Exec mode.
In exec mode the researcher executes the set up experiment, performing starting and termimating of the experiment. For all other action the user must work from a command console.[7]

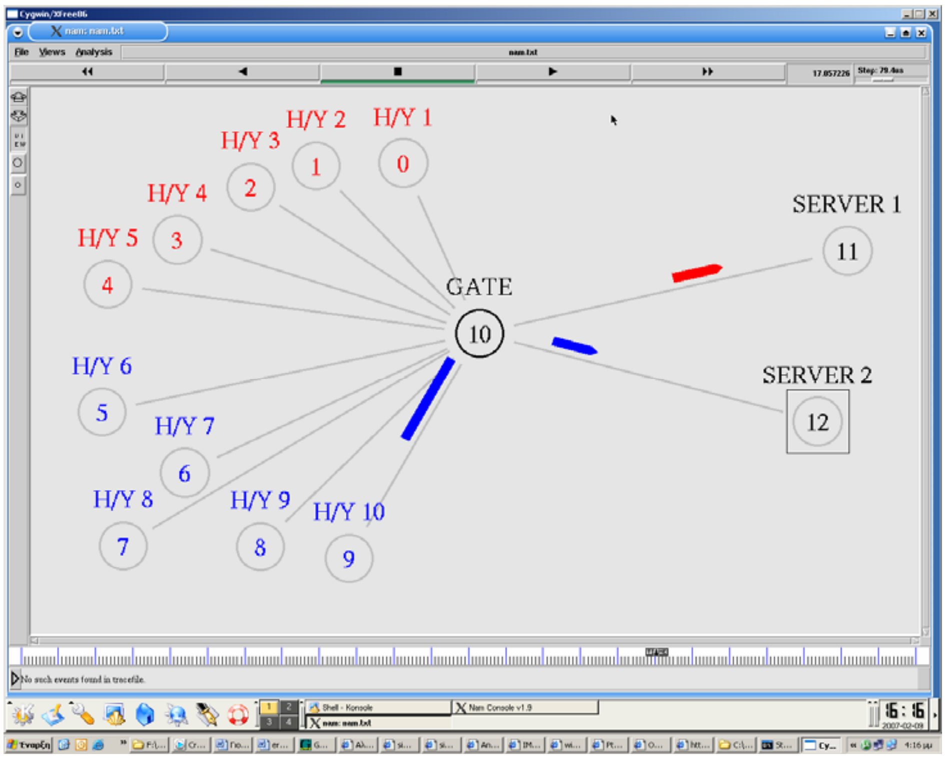
3. NS-2 (Network Simulator 2)

NS2 is an open-source simulation software that is part of a research project, called VINT, funded by DARPA.
The aim of VINT is to build a network simulator that will allow the study of scale and protocol interaction in the context of current and future network protocols. VINT is a collaborative project involving USC/ISI, Xerox PARC, LBNL, and UC Berkeley.[8]
NS2 is a discreet event simulator 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.
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

[7] http://www.tel.fer.hr/imunes/dl/imunes.pdf
[8] http://www.isi.edu/nsnam/vint/

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

Figure 3.1 An example of a simulated network topology in NAM graphic environment



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.
In order to verify some aspects of the protocol to be simulated, NS includes some validation tests distributed with the simulator.
Ns as well includes capabilities to make the simulation topologies dynamic although this latest point is still somewhat experimental.

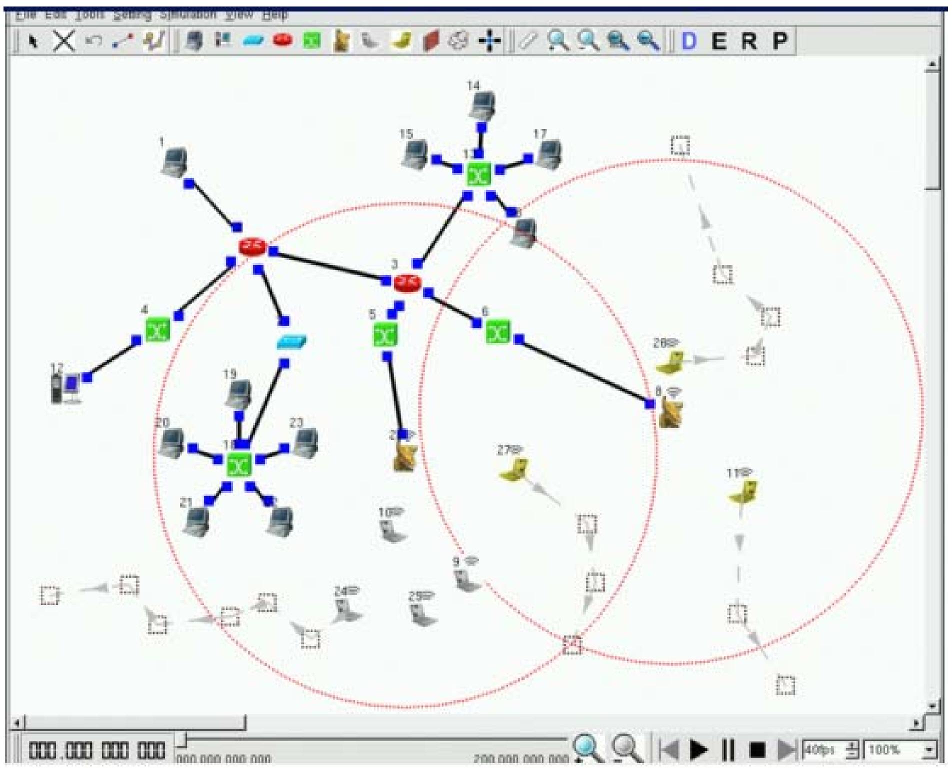
The simulation engine is extensible, configurable and programmable. The current implementation is single-threaded (only one event in execution at any given time). It does not support partial execution of events nor pre-emption.
Events are described by a firing time and a handler function. The type of 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.
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.
A graphical interface of NS2 to setup network simulations is NAM that supports a drag-and-drop user interface.
Nam is a Tcl/TK based animation tool for viewing network simulation traces and real world packet traces. It supports topology layout, packet level animation, and various data inspection tools.
Nam began at LBL. It has evolved substantially over the past few years. The nam development effort was an ongoing collaboration with the VINT project. Currently, it is being developed at ISI as part of the SAMAN and Conser projects.
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.[9],[10],[11]

4. NCTUns

The NCTUns is a high-fidelity and extensible network simulator and emulator capable of simulating various protocols used in both wired and wireless IP networks. Its core technology is based on the novel kernel re-entering methodology invented by Prof. S.Y. Wang when he was pursuing his Ph.D. degree at Harvard University. Due to this novel methodology, NCTUns provides many unique advantages that cannot be easily achieved by traditional network simulators such as ns-2 and OPNET.

Figure 4.1 Screenshot of topology editor



The NCTUns network simulator and emulator has many useful features listed below:

- **It can be used as an emulator**. An external host in the real world can exchange packets (e.g., set up a TCP connection) with nodes (e.g., host, router, or mobile station) in a network simulated by NCTUns. Two external hosts in the real world can also exchange their packets via a network simulated by NCTUns. This feature is very useful as the function and performance of real-world devices can be tested under various simulated network conditions.

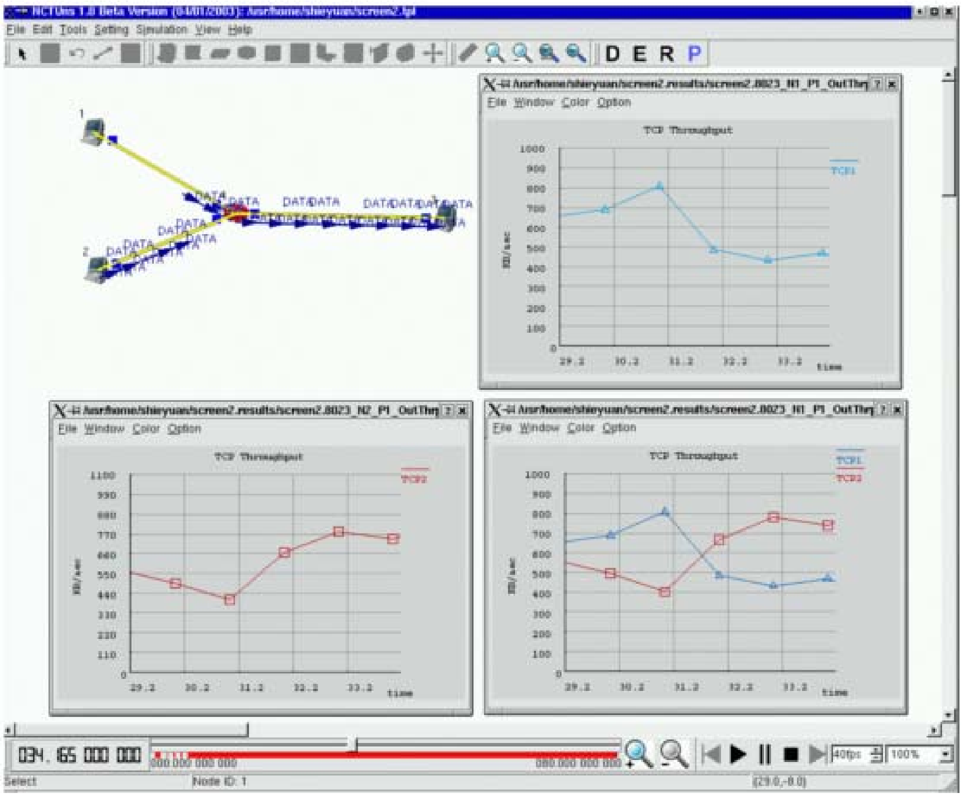
[9] The ns Manual, September 2005
[10] http://nsnam.isi.edu/nsnam/index.php/Contributed_Code#Documentation
[11] http://www.linuxjournal.com/article/5929

- **It directly uses the real-life Linux's TCP/IP protocol stack to generate high-fidelity simulation results.** By using a novel kernel re-entering simulation methodology, a real-life UNIX (e.g., Linux) kernel's protocol stack can be directly used to generate high-fidelity simulation results.
- **It can use any real-life existing or to-be-developed UNIX application program as a traffic generator program without any modification.** Any real-life program can be run on a simulated network to generate network traffic. This enables a researcher to test the functionality and performance of a distributed application or system under various network conditions. Another important advantage of this feature is that application programs developed during simulation studies can be directly moved to and used on real-world UNIX machines after simulation studies are finished. This eliminates the time and effort required to port a simulation prototype to a real-world implementation if traditional network simulators are used.
- **It can use any real-life UNIX network configuration and monitoring tools**. For example, the UNIX route, ifconfig, netstat, tcpdump, traceroute commands can be run on a simulated network to configure or monitor the simulated network.
- **Its setup and usage of a simulated network and application programs are exactly the same as those used in real-world IP networks.** For example, each layer-3 interface has an IP address assigned to it and application programs directly use these IP addresses to communicate with each other. For this reason, any person who is familiar with real-world IP networks can easily learn and operate NCTUns in a few minutes. For the same reason, NCTUns can be used as an educational tool to teach students how to configure and operate a real-world network.
- **It simulates many different and new types of networks.** The supported types include Ethernet-based fixed Internet, IEEE 802.11(b) wireless LANs, mobile ad hoc (sensor) networks, GPRS cellular networks, optical networks (including both circuit-switching and busrt-switching networks), IEEE 802.11(b) dual-radio wireless mesh networks, IEEE 802.11(e) QoS wireless LANs, Tactical and active mobile ad hoc networks, and 3dB beamwidth 60-degree and 90-degree directional antennas.
- **It simulates various networking devices.** For example, Ethernet hubs, switches, routers, hosts, IEEE 802.11 (b) wireless stations and access points, WAN (for purposely delaying/dropping/reordering packets), obstacle (block/attenuate wireless signal, block mobile nodes' movement, block mobile nodes' views), GPRS base station, GPRS phone, GPRS GGSN, GPRS SGSN, optical circuit switch, optical burst switch, QoS DiffServ interior and boundary routers, IEEE 802.11(b) dual-radio wireless mesh access point, IEEE 802.11(e) QoS access points and mobile stations, etc.
- **It simulates various protocols.** For example, IEEE 802.3 CSMA/CD MAC, IEEE 802.11 (b) CSMA/CA MAC, IEEE 802.11(e) QoS MAC, IEEE 802.11(b) wireless mesh network routing protocol, learning bridge protocol, spanning tree protocol, IP, Mobile IP, Diffserv (QoS), RIP, OSPF, UDP, TCP, RTP/RTCP/SDP, HTTP, FTP, Telnet, etc.
- **It simulates a network quickly**. By combining the kernel re-entering methodology with the discrete-event simulation methodology, a simulation job can be finished quickly.
- **It generates repeatable simulation results**. If the user fixes the random number seed for a simulation case, the simulation results of a case are the same across different simulation runs even if there are some other activities (e.g., disk I/O) occurring on the simulation machine.
- **It provides a highly-integrated and professional GUI environment.** This GUI can help a user (1) draw network topologies, (2) configure the protocol modules used inside

a node, (3) specify the moving paths of mobile nodes, (4) plot network performance graphs, (5) playing back the animation of a logged packet transfer trace, etc. All these operations can be easily and intuitively done with the GUI.

- **Its simulation engine adopts an open-system architecture and is open source**. By using a set of module APIs provided by the simulation engine, a protocol developer can easily implement his (her) protocol and integrate it into the simulation engine. NCTUns uses a simple but effective syntax to describe the settings and configurations of a simulation job. These descriptions are generated by the GUI and stored in a suite of files. Normally the GUI will automatically transfer these files to the simulation engine for execution. However, if a researcher wants to try his (her) novel device or network configurations that the current GUI does not support, he (she) can totally bypass the GUI and generate the suite of description files by himself (herself) using any text editor (or script program). The non-GUI-generated suite of files can then be manually fed to the simulation engine for execution.
- **It supports remote and concurrent simulations.** NCTUns adopts a distributed architecture. The GUI and simulation engine are separately implemented and use the client-server model to communicate. Therefore, a remote user using the GUI program can remotely submit his (her) simulation job to a server running the simulation engine. The server will run the submitted simulation job and later return the results back to the remote GUI program for analyses. This scheme can easily support the cluster computing model in which multiple simulation jobs are performed in parallel on different server machines. This can increase the total simulation throughput.
- **It provides complete and high-quality documentations.** The GUI user manual has 118 pages while the protocol developer manual has 320 pages. NCTUns also provides over 50 example simulation cases and their demo video clips to help a user easily understand how to run up a simulation case.

Figure 4.2 Screenshot of performance monitor



- **It is continuously supported, maintained, and improved.** New functions and network types are continuously added to NCTUns to enhance its function, speed, and capability. For example, WiMax wireless networks (including PMP and mesh modes) and GEO satellite networks are being under development and will be released in NCTUns 4.0.[12]

NCTUns supports the following types of network links and devices:

Device	Acronym	Technology Specification
Point-to-Point Ethernet-like Link	PPL	For wired networks
Point-to-Point WDM Optical Link	WDMLINK	For optical networks
Host	HOST	For wired networks
Hub	HUB	For wired networks
Switch	SWITCH	For wired networks
Router	ROUTER	For wired networks
WAN cloud	WAN	For wired networks. Used to purposely drop, delay, and reorder passing packets.
Subnet	SUBNET	For wired networks. Used to insert a group of hosts that are all connected to a central switch.
External Host	EXTHOST	This is for emulation purposes. Used to represent a host in the real world.
External Router	EXTROUTER	This is for emulation purposes. Used to represent a router in the real world.
External IEEE 802.11 (b) Mobile Node (ad hoc mode)	EXTMNODE	This is for emulation purposes. Used to represent an IEEE 802.11 (b) mobile node (ad hoc mode) in the real world.
External IEEE 802.11 (b) Mobile Node (infrastructure mode)	EXTMNODE_INFRA	This is for emulation purposes. Used to represent an IEEE 802.11 (b) mobile node (infrastructure mode) in the real world.
QoS DiffServ Boundary Router	BROUTER	For wired network. Used as a boundary router in a QoS Diffserv network.
QoS DiffServ Interior Router	IROUTER	For wired network. Used as an interior router in a QoS Diffserv network.
Optical Circuit Switch	OSWITCH	For optical network. Used as an optical circuit switch.
Optical Burst Switch	OBSWITCH	For optical network. Used as an optical burst switch.
GPRS Base Station	GPRSBS	For GPRS cellular network. Used as a base station in a GPRS network.
GPRS Phone	GPRSPHONE	For GPRS cellular network. Used as a phone in a GPRS network.
GPRS SGSN	GPRSSGSN	For GPRS cellular network. Used as a SGSN in a GPRS network.
GPRS GGSN	GPRSGGSN	For GPRS cellular network. Used as a GGSN in a GPRS network.
GPRS Pseudo Switch	GPRSPS	For GPRS cellular network. Used as a pseudo switch in a GPRS network.

[12] http://nsl10.csie.nctu.edu.tw/

Wall (wireless signal obstacle)	WALL	For wireless networks (to purposely block wireless signal propagation)
IEEE 802.11 (b) Mobile Node (ad hoc mode)	MNODE	For mobile wireless networks. Used as an IEEE 802.11 (b) mobile node (ad hoc mode).
IEEE 802.11 (b) Mobile Node (infrastructure mode)	MNODE_INFRA	For mobile wireless networks. Used as an IEEE 802.11 (b) mobile node (infrastructure mode).
IEEE 802.11 (b) Access Point	AP	For mobile wireless networks. Used as an IEEE 802.11 (b) access point (infrastructure mode).
IEEE 802.11(b) dual-radio wireless mesh access point running OSPF routing protocol	MESH_OSPF_AP	For mobile wireless mesh networks. Used as an IEEE 802.11 (b) dual-radio mesh access point (infrastructure mode) running the OSPF routing protocol.
IEEE 802.11(b) dual-radio wireless mesh access point running SPT routing protocol	MESH_SPT_AP	For mobile wireless mesh networks. Used as an IEEE 802.11 (b) dual-radio mesh access point (infrastructure mode) running the spanning tree protocol.
IEEE 802.11 (e) QoS Mobile Node (infrastructure mode)	QoS_ MNODE_INFRA	For mobile wireless networks. Used as an IEEE 802.11 (e) mobile node (infrastructure mode).
IEEE 802.11 (e) QoS Access Point (infrastructure mode)	QoS_AP	For mobile wireless networks. Used as an IEEE 802.11 (e) access point (infrastructure mode)

NCTUns supports the following network protocols:

Physical Layer

Protocol	Acronym	Technology Specification
Simple BER model for point-to-point link	PHY	For wired networks
Simple BER model for wireless signal (i.e., using the transmission and interference ranges only)	WPHY	For wireless networks
More realistic (advanced) BER model for wireless signal (i.e., considering the used modulation scheme, received bit power, and noise level)	AWPHY	For wireless networks
3dB beamwidth 60-degree and 120-degree steerable directional antennas	AWPHY	For wireless networks

Data Link Layer

Protocol	Acronym	Technology Specification
Ethernet	Ether	IEEE 802.3
IEEE 802.11 (b) Wireless LAN (ad hoc mode)	WLANAD	IEEE 802.11b
IEEE 802.11 (b) Wireless LAN (infrastructure mode)	WLANIN	IEEE 802.11b
Learning Bridge Protocol (used in the switch)	LBP	IEEE 802.11d
Spanning Tree Protocol (used in the switch)	STP	IEEE 802.11d
Address Resolution Protocol	ARP	RFC-826

2F-BLSR Optical Ring Protection Protocol	2F-BLSR	2F-BLSR
IEEE 802.11(b) wireless mesh network MAC	MESH	IEEE 802.11b (dual-radio access point)
IEEE 802.11(e) QoS wireless LAN MAC	802.11eWLAN	802.11e

Network Layer

Protocol	Acronym	Technology Specification
Internet Protocol	IP	RFC-791, RFC-792, RFC-826 (Both unicast and subnet broadcast are supported.)
Internet Control Message Protocol	ICMP	RFC-792
Open Shortest Path First Routing protocol	OSPF	RFC-1247
Routing Information Protocol	RIP	RFC-1058
Fixed-Network God Routing Protocol	FNGRP	Automatically calculate the best routing paths for a fixed network (for research comparison purposes)
Dynamic Source Routing protocol	DSR	For mobile ad hoc networks
Ad hoc On Demand Distance Vector Routing protocol	AODV	For mobile ad hoc networks
Adaptive Distance Vector Routing protocol	ADV	For mobile ad hoc networks
Destination-Sequenced Distance-Vector Routing protocol	DSDV	For mobile ad hoc networks
Mobile-Network God Routing Protocol	MNGRP	Automatically calculate the best routing paths for a mobile ad hoc network (for research comparison purposes)
FIFO Packet Scheduling mechanism	FIFO	(Can be used in a switch as well)
Deficit Round-Robin Packet Scheduling	DRR	(Can be used in a switch as well)
Random Early Detection Buffer Management	RED	(Can be used in a switch as well)
Packet dropping, delaying, and reordering	WAN	This module is used in the WAN node to purposely drop, delay, and reorder passing packets according to a specified statistic distribution.
QoS DiffServ EF, BE, CL, PHB	DS_I	This module is used in an interior router in a QoS DiffServ network to perform the specified per-hop-behavior packet scheduling methods.
QoS DiffServ Traffic metering, conditioning, and shaping	DS_TC	This module is used in a boundary router in a QoS DiffServ network to perform traffic metering, conditioning, and shaping.
GPRS cellular network's protocol stacks	GPRS	Many modules are provided to construct the protocol stacks used in the GPRS network.
Optical Burst Switching Protocol	OBS	Several modules are provided to construct the protocol stacks used in the OBS network.
Mobile IP	MIP	Several daemon programs are provided to implement the home and foreign agents used in an Mobile IP network. RFC-2002, RFC 2003.
IEEE 802.11(b) wireless mesh network OSPF routing protocol	MESHOSPF	The OSPF routing protocol is run on the wireless mesh network.
IEEE 802.11(b) wireless mesh network SPT (Spanning Tree) protocol	MESHSTP	The spanning tree protocol is run on the wireless mesh network.

Transport Layer

Protocol	Acronym	Technology Specification
Transmission Control Protocol	TCP	RFC-791, RFC-792, RFC-826
User Datagram Protocol	UDP	RFC-768, RFC-1122
Real Time Transport Protocol, Real Time Control Protocol	RTP/RTCP	RFC-1889, RFC-1890
Session Description Protocol	SDP	RFC-2327

Application Layer [13]

Protocol or applications	Acronym	Technology Specification
HTTP		
FTP		
Telnet		
Tcpdump		
Traceroute		
Ping		
Stcp/rtcp		Greedy TCP traffic
Ttcp		Greedy TCP/UDP trafic
Stg/rtg		Greedy TCP/UDP traffic, trace driven, self-similar traffic, on-off, constant-bit-rate, and various packet streams.
Magent1~5		Tactical agent programs for tactical and active mobile ad hoc network simulations

13 All existing real-world network application programs and tools can directly run on a simulated network. The above represent just a few of them.

5. DESMO-J

DESMO-J is an object-oriented framework targeted at programmers developing simulation models. The acronym "DESMO-J" stands for "Discrete-Event Simulation and MOdelling in Java". This longer name highlights DESMO-J's two significant properties:

- DESMO-J supports the discrete-event simulation paradigm. In models of this type, all system state changes are supposed to happen at discrete points in time. Between such events the system state is assumed to remain constant. Discrete-event simulation is therefore particularly suitable for systems in which relevant changes of state occur suddenly and irregularly, like queueing networks for example.
- DESMO-J is implemented in Java. Using this framework to build simulation models ultimately results in writing a Java program.

Basic Features

The framework DESMO-J extends Java by adding features which greatly simplify the construction of discrete-event simulation models. It provides the modeller with

- ready-to-use classes for common model components like queues, stochastic distributions based on random number streams, and data collectors.
- abstract classes to be adapted to model-specific behaviour, like model, entity, event, and simulation process. Depending on the modelling style applied you will have to either define your own simulation processes or define your own entities and events.
- ready-to-use simulation infrastructure comprising scheduler, event list, and simulation time clock, all encapsulated in an experiment class. This experiment class also provides for the generation of reports and traces of a simulation run.

DESMO-J supports the separation of model and experiment, a widely acknowledged requirement of good simulation software as it allows for performing the same experiment with different models which may represent competing system designs or alternative strategies as well as performing different experiments with the same model. In DESMO-J the model class handles all model components whereas the experiment class provides the simulation infrastructure. Both are explicitly connected during a simulation run.[14]

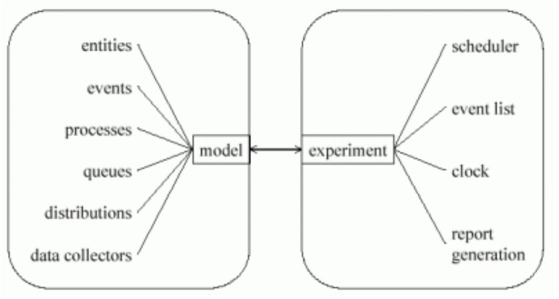


Figure 5.1 Separation of Model and Experiment

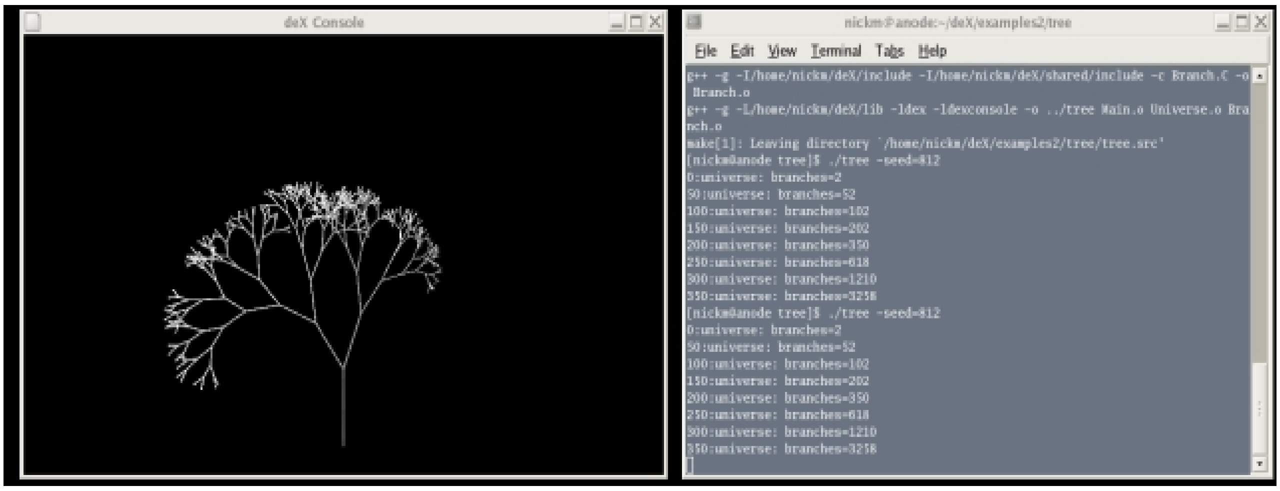
DESMO-J is based on a number of earlier DESMO systems, all developed at the University of Hamburg, Working Group Prof. Page, in the context of students' projects. The original DESMO, written 1989 in the programming language Modula-2, was inspired by DEMOS, a package for discrete-event simulation in Simula developed by Graham Birtwistle in 1979. Since 1999, when the core DESMO-J framework was completed, it has been extended in various aspects, e.g. to provide special components for the simulation of

[14] http://asi-www.informatik.uni-hamburg.de/themen/sim/forschung/Simulation/Desmo-J/overview.html

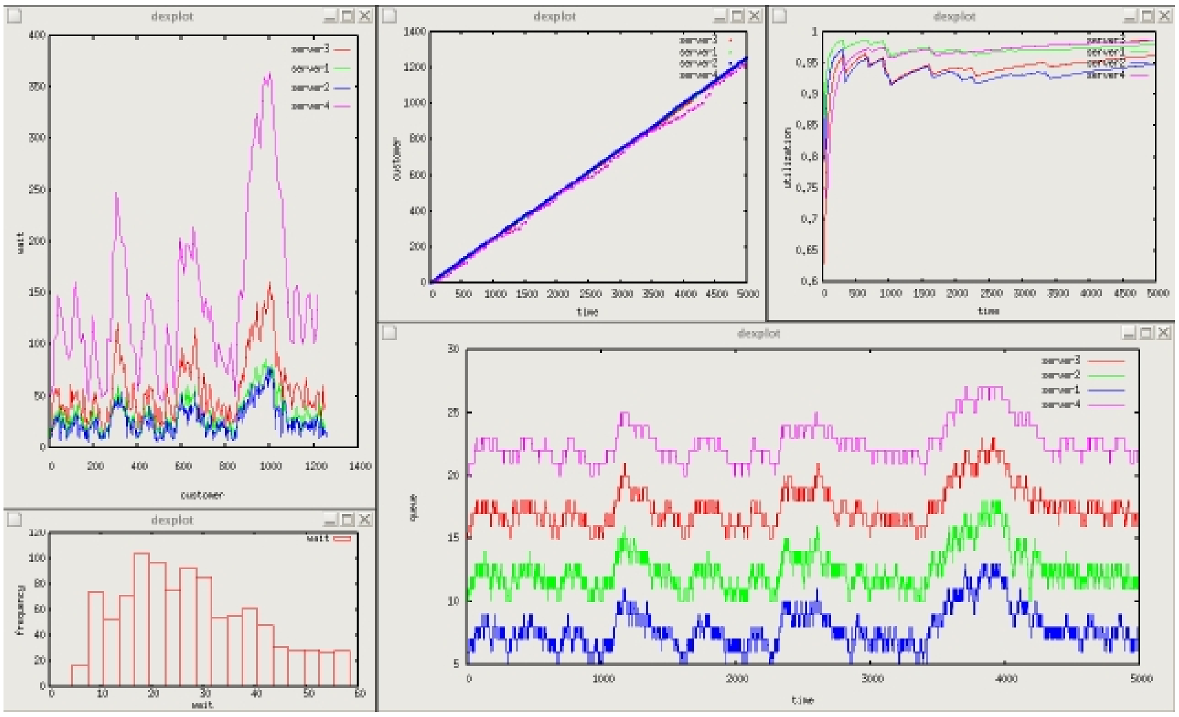
production systems or harbour logistics. DESMO-J is maintained by a team of developers headed by Prof. Page.

6. deX

deX (Distributed Execution & Dynamic Experimentation Simulation Framework & Modeling Language) aims to provide a fast, flexible, and easy-to-use platform for developing, analyzing, and visualizing dynamic agent-based and multi-body simulations. deX programs run efficiently on a single processor but more demanding programs are easily adapted to run in parallel through high-level constructs utilizing shared memory/multithreading, distributed TCP/IP communication, MPI, or any combination of the three.
deX provides a versatile abstraction layer for managing parallelism, synchronization, and event communication that facilitates continuous, discrete-event, and hybrid simulation and allows complex programs to be constructed with a minimal amount of coding. deX is built on top of a high-performance simulation engine designed to handle a large number of entities with high levels of event communication. deX is an object oriented C++ framework which may be optionally used in combination with dML (deX Modeling Language): a domain-specific language based on C++, built on top of the core deX framework, and designed specifically for modeling, simulation, and rapid program development. deX includes tools for batch execution, optimization, distributed job execution, GUI controls, and real-time plotting/3D visualization using OpenGL.


deX Console
nickm@anode:~/deX/examples2/tree
File Edit View Terminal Tabs Help

```
g++ -g -I/home/nickm/deX/include -I/home/nickm/deX/shared/include -c Branch.C -o
 Branch.o
g++ -g -L/home/nickm/deX/lib -ldex -ldexconsole -o ../tree Main.o Universe.o Bra
nch.o
make[1]: Leaving directory `/home/nickm/deX/examples2/tree/tree.src'
[nickm@anode tree]$ ./tree -seed=812
0:universe: branches=2
50:universe: branches=52
100:universe: branches=102
150:universe: branches=202
200:universe: branches=350
250:universe: branches=618
300:universe: branches=1210
350:universe: branches=3258
[nickm@anode tree]$ ./tree -seed=812
0:universe: branches=2
50:universe: branches=52
100:universe: branches=102
150:universe: branches=202
200:universe: branches=350
250:universe: branches=618
300:universe: branches=1210
350:universe: branches=3258
```



Simulation is a thoroughly established field and there are hundreds of existing high-quality simulation-related software tools, so what does deX have to offer? A small subset of the available simulation tools are complete simulation frameworks or simulation languages. A smaller portion in turn are freely downloadable and available for Linux. UNIX is a powerful platform in scientific computing, offering, among other things, the best of both worlds in GUI and command-line flexibility. Linux is a highly accessible and widely supported UNIX platform with a large base of high-quality, free, and open-source software and libraries, many of which are installed as standard on a modern Linux distribution.

deX avoids re-inventing the wheel and seeks to leverage the power of existing Linux libraries and programs whenever possible. Many simulation packages are cross-platform with ports to Linux often as an afterthought. deX was designed specifically for Linux and deX programs use powerful UNIX facilities such as commands, options, input/output redirection, and piping. deX follows the UNIX philosophy of small specialized programs that can be combined in useful ways. Some simulation packages provide an integrated environment using an elaborate GUI and editor for simulation development. deX prefers the use of a familiar editor such as emacs and a strong text-based foundation that does not rule out the possibility of creating a GUI on top of this foundation later. Design goals such as these make Linux an essential asset.
Many simulation frameworks use Java. Java as a language, with its extensive class libraries and built-in threading is an attractive platform for simulation. However, because of its interpreted nature, it is not a good choice in the performance-intensive and real-time visualization types of programs targeted by deX. Further, Java lacks a 3D library that is on par with OpenGL.
Many simulation packages do not include a simulation language. When they do, the language often differs greatly from familiar languages and the user must code in this custom

language in order to make use of the simulation package. Unfamiliar languages take time to learn, make maintenance difficult, and often do not interface with established generalpurpose languages. Simulation languages are often interpreted which can lead to poor performance when compared to a compiled binary. dML uses a familiar grammar based on C++ and is built on top of a simulation framework and translated to C++, allowing integration with existing C++ code and libraries. deX programs are compiled to binary using shared libraries for high performance. The framework and dML together allow the development of complex simulations with a minimal amount of coding.
Given these combined considerations, deX fills a unique niche in the available simulation software; deX:

• may be downloaded and used without restriction for free
• is designed specifically for Linux taking advantage of UNIX features
• leverages the power of existing Linux libraries and programs
• is easy-to-use with minimal coding
• offers hybrid simulation features
• supports real-time 3D visualization using OpenGL
• is based on a C++ foundation for integration and high performance
• provides an optional simulation language based on a familiar C++ grammar and built on top of a simulation framework.

deX is a versatile framework that supports continuous, discrete-event, and hybrid simulation and was designed especially for multi-body, agent-based, and general systems modeling.

• deX is built on top of a high-performance simulation engine designed to support a large number of entities with high levels of event communication. This is vital in nbody simulations when at each cycle each body must communicate with every other body. In this common scenario, event communication is O(n2) and quickly becomes the bottleneck.
• deX provides high-level constructs that ease the difficulties of creating parallel and distributed programs and make use of shared memory/multi-threading, TCP/IP, MPI, or any combination of the three.
• dML is designed specifically for modeling and offers many features that facilitate rapid-prototyping. The grammar is based on C++ making it easy to learn for anyone familiar with C++, Java, or popular procedural programming languages.
• dML code is translated to C++. In addition to the benefits of efficiency in execution, this approach allows for existing C/C++ code and libraries to be easily integrated with dML code. Conversely, code translated from dML to C++ may be integrated into existing general-purpose C++ projects. Using dML is optional. The core framework and utility classes may be used as a C++ framework.
• The console entity is an example of integrating dML code with existing C++ libraries.
The console entity makes use of OpenGL and SDL to provide a simplified interface to real-time 3D visualization facilities.
• The controls entity is another example of an entity written in C++. The controls entity interfaces seamlessly with dML providing an easy-to-use interface to the GTK library allowing the user to construct a GUI with buttons and sliders which emit events that may be received by other entities.
• deX includes an extensive library of continuous and discrete probability distributions that are useful in stochastic modeling.
• The dexplot command leverages the power of gnuplot allowing data to be filtered through regular expressions to provide real-time plotting.
• The dexc command allows for complex and frequently used commands to be associated with a project. dexc uses a configuration file that is similar to a Makefile.

• The dexbatch command coordinates batch executions and optimization. Dexbatch allows the user to run a series of executions on a linear combination of input parameters and varying in an interval. Each execution may be repeated a number of times with a unique random seed and the results averaged.
• deX provides a run-time error handling mechanism that greatly speeds development time and reduces the need to use a debugger.
• deX is freely available for x86-linux and x86 64-linux from http://dex.infini-x.org[15]

7. Conclusions and future work

Based on a bibliographic point of view it is quite difficult to discern among the plenty simulators the “best”, the “more complete” or the “most suitable” one. The field of network research or investigation and assessment is very wide and many issues remain unexplored given the situation of constant progress. Thus one can only sort the different simulators based on criteria like the ones on above pages 4, 5.
NS2 is the most well known and established simulator in students and researchers. It seems to deliver quite well what it claims it does on network simulation but also with the usual problems of debugging.
NCTUns is a very fresh simulator (first released on 11/2002) and has a very good presentation and documentation of its features. It is promoted as innovative and achieving things that NS2 and OPNET can not while referring to them as “traditional simulators”. His innovative feature is the kernel re-entering methodology for creating the entities of the simulation giving them in this way a rich set of attributes otherwise not available.
Imunes uses a similar methodology of reusing the OS kernel code in each virtual node as NCTUns does.
DESMO-J and **deX** are both object oriented frameworks with a broad range of simulation applications not only restricted on networks.
In later papers and after an efficient number of network simulators had been presented maybe it would be more useful for a more thoroughly examination of the most interesting of them.

[15] deX: Distributed Execution & Dynamic Experimentation Simulation Framework & Modeling Language, User’s Manual Version 1.5.0, Copyright c 2003-2006 infini-x.org, pages 8-9.

8. References

1. http://en.wikipedia.org/wiki/Network_simulation
   Definition of Network Simulation in wikipedia site, containing useful links on the subject

2. http://www.emulab.net/index.php3?stayhome=1
   Official web site of emulab, the distributed network testbed of University of UTAH.

3. http://www.tel.fer.hr/imunes/
   Official web site of imunes simulator framework

4. http://www.topology.org/soft/sim.html#DESS
   A web site with a very large list of simulators of all kinds.

5. http://www.isi.edu/nsnam/ns/
   Presentation and resources of NS-2

6. http://www.isi.edu/nsnam/vint/
   Presentation of the project VINT and correlated links

7. The ns Manual, September 2005

8. http://asi-www.informatik.uni-hamburg.de/themen/sim/forschung/Simulation/Desmo-J/index.html
   Presentation web pages and resources of the DESMO-J simulator

9. "An introduction to management science" by D.R.Anderson, D.J.Sweeney, T.A.Williams, THOMSON SOUTH – WESTERN ed.11th.

10. Free tools for network simulation, paper by Voultiou Efthimia of MIS University Of Macedonia, 2006

11. http://www.linuxjournal.com/article/5929
    Article in Linux Journal site about use of NS-2 to a specific project.

12. http://nsl10.csie.nctu.edu.tw/
    Web Site of SIMREAL the virtual company that promotes NCTUns simulator

13. http://nsl.csie.nctu.edu.tw/nctuns.html
    Web Site of the NCTUns simulator/emulator

14. deX: Distributed Execution & Dynamic Experimentation Simulation Framework & Modeling Language, User's Manual Version 1.5.0, Copyright c 2003-2006 infini-x.org

15. http://dextk.org/dex/index.html
    Web Site of the deX framework.

16. Epixeirisiaki diadiktyosi by G. Diakonikolaou, A. Ajiakatsika and H. Mpouras. Kleidarithmos 2004

17. http://www.lionhrtpub.com/orms/surveys/Simulation/Simulation.html
    Web page of Lionheart Publishing (USA) presenting its annual survey (2006) of commercial simulation softwares