NEST: A NETWORK SIMULATION & PROTOTYPING TESTBED

A. Dupuy, J. Schwartz, Y. Yemini
450 Computer Science, Columbia University, NY, NY 10027
D. Bacon
IBM, T. J. Watson Research Center

CUCS-478-89

NEST: A NETWORK SIMULATION & PROTOTYPING TESTBED*

A. Dupuy, J. Schwartz, Y. Yemini
450 Computer Science, Columbia University NY, NY 10027
D. Bacon
IBM, T.J. Watson Research Center

ABSTRACT

This paper describes Nest, an graphical environment for distributed networked systems simulation and rapid-prototyping. Nest users can develop and test distributed systems and protocols (from crude models to actual system code) within simulated network scenarios. Nest represents an environment-based approach to simulation. Users view Nest as an extension of their standard Unix™ environment. Nest offers the generality of language-based simulation techniques and the efficiencies of model-based techniques. Users interact with Nest through standardized graphical interfaces. Nest permits the users to modify and reconfigure the simulation during execution. Thus, it is possible to study the dynamic response of a distributed system to failures or burst-loads. Nest is organized as a simulation server, responsible for execution of complex simulation scenarios and client monitors responsible for simulation control. The client/server model permits distribution of Nest over a network environment. This permits migration of simulations to powerful remote computational servers as well as development of a shared multi-site simulation/integration testbed. Nest is portable and extensible. It has been ported to virtually all Unix variants and distributed since 1987 to over 150 sites worldwide. It has been used in scores of studies ranging from communication protocols, to distributed databases and operating systems as well as distributed manufacturing systems.

* Research supported by DARPA contract #F-29601-87-C-0074 and by the NY State CAT contract #NYSSTFCAT(89)-5

† UNIX is a trademark of AT&T.

Nest pursues a different approach to simulation studies: extending a networked operating system environment to support simulation modeling and efficient execution. This *environment-based* approach to simulation shares with language-based approaches the generality of its modeling power. Nest may be used to model arbitrary distributed interacting systems. Nest also shares with the language-based approach an internal execution architecture (see below) that accomplishes very efficient scheduling of a large number of processes. However, unlike language-based approaches, Nest does not require the user to master or use a new/separate simulation language facility and the processes of design/development and simulation are integrated. The user can study the behavior of the actual system being developed (at any level of detail) under arbitrary simulated scenarios. The routing protocol designer, for example, can attach the very routing protocol designed (actual code with minor adjustment) to a Nest simulation and pursue study of the actual system behavior. As the system changes through the design process, new simulation studies may be conducted by attaching the new code to the same simulation models. Nest can thus be used as an integral part of the design process unified with other tools (*e.g.*, for debugging).

In similarity to model-based approaches, Nest is specifically targeted towards a limited scope of applications: distributed networked systems. Nest supports a built-in customizable communication network model. However, this scope has been sufficiently broad to support studies ranging from low-level communication protocols to distributed transaction processing systems, avionic systems and even manufacturing processes.

The environment-based approach to simulation offers a few important attractions to users:

1. simulation is integrated with the range of tools supported by the environment
   - the user can utilize graphics, statistical packages, debuggers and other standard tools of choice in the simulation study
   - the user can integrate simulation as an integral part of a standard development process
2. users need-not develop extensive new skills or knowledge to pursue simulation studies
3. standard features of the environment can be used to enhance the range of applicability
   - Nest simulation is configured as a network server with monitors as clients. The client/server model permits multiple remote accesses to a shared testbed. This can be very important in supporting a large scale multi-site project.

In what follows we describe the architecture of Nest (section 2), illustrate its use through a simple example (section 3), describe some aspects of Nest implementation (section 4) and provide a few examples of Nest applications.

2. ARCHITECTURE OF NEST

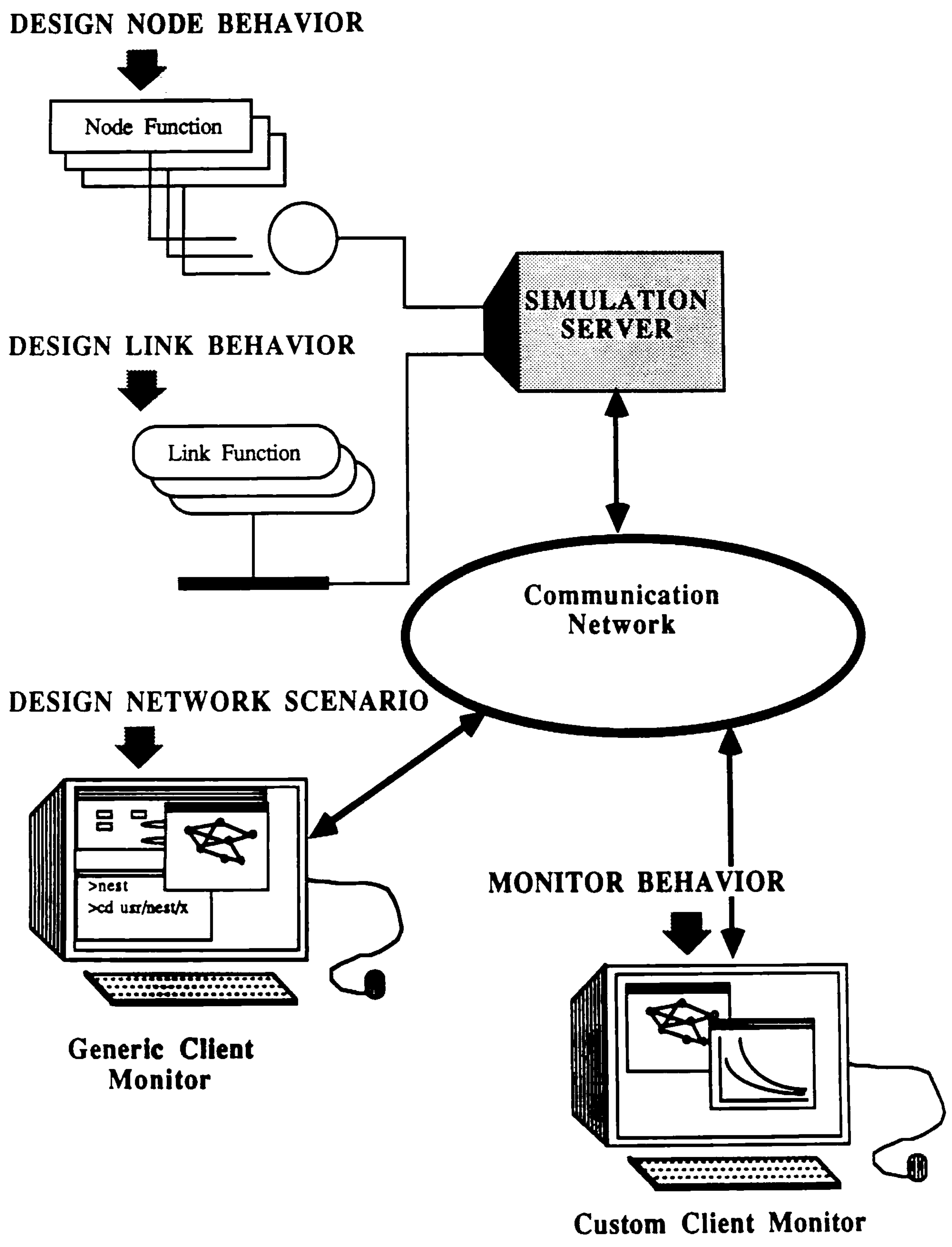


Figure 1: Overall architecture of Nest

2.1 OVERALL SERVER/CLIENT STRUCTURE

The overall architecture of Nest is depicted in the figure (1) above. Nest consists of a simulation server and client monitors. The simulation server is responsible for the execution of a simulation run. The generic client monitors are used to (re)configure a simulation model and control its execution. The custom clients are used to monitor a simulation behavior and display the results. Clients can (and typically will) reside on separate machines from the server.This allows dedication of a computing server to execute a cycle-consuming simulation while delegating presentation and control functions to remote workstations.

The client-server communications require relatively low communication bandwidth. This permits the server to serve remote clients over a wide-area network. Thus, it may be necessary sometime to utilize the power of a remote supercomputer to execute a complex simulation study. Similarly, it may be useful to retain a shared simulation testbed for integration of systems developed by a multi-site project. For example, in the design of a complex communication network different sites may be responsible for different protocols and subsystems. A shared testbed permits both simplified integration of these subsystems as well as testing of individual subsystems in relations to each other under standardized scenarios.

The interaction of users with Nest are depicted via the shaded arrows. Users provide node and link functions which are linked with the simulation server to form a simulation testbed. These functions are coded in C and include calls upon the Nest library. Node functions are used to model distributed communicating processes running at network nodes (*e.g.*, protocols, database transactions, manufacturing cells). Nest executes the node processes and their communications calls using Nest-provided primitives for sending, broadcasting or receiving packets.

A simulated link has an associated stack of link functions. The motion of a packet over the link is simulated by passing it through the link functions, which act as a stack of filters. Link functions are used to model the behavior of communication links (*e.g.*, packet loss, link jamming, support of a standard protocol stack). Link functions are also used to monitor and collect performance statistics of link traffic (*e.g.*, number of control packets, link delay).

The simulation server integrates the node and link functions to form a single simulation process. The simulation process schedules the execution of the node and link processes to meet the specifications of delay and timing set by the users. The user can control the timing of events and the delays associated with communications through a collection of Nest-supported timing control functions. These functions simulate standard Unix timing control (*e.g.*, sleep) and support full user control over simulation time.

2.2 NEST USER INTERFACES

Nest users control and manage a simulation through graphical monitoring tools. Nest provides two kinds of monitors: generic monitor and custom monitors. The generic monitor provides a complete environment to create edit and (re)configure simulation scenarios. A typical generic monitor screen is depicted in figure (2) below. The user creates and modifies a network description using a mouse to draw it; clicking on the mouse generates nodes; dragging the mouse between nodes creates links. Node and link pop-menus offer a range of editing features to configure the respective simulated objects. Simulation parameters may be set via respective panels at the top. Once the user defined a simulation scenario, it is sent to the simulation server where it is loaded and executed.

One of Nest's key features is the ability to reconfigure a scenario during the simulation run. This is particularly important for studies of complex dynamic system behaviors: how does the system respond to a node/link crash? how will it handle addition of new nodes or links? what transient behaviors occur as a result of such critical changes? how long will certain transients last? how probable are they? These type of questions are typically difficult to answer through analytical studies or model-based simulation and require significant experimentation. Nest support such experimentation with varying scenarios. Users may delete/add nodes/links or change their features while the simulation is running. The impact of these changes on the system behavior may be instantly observed and interpreted.

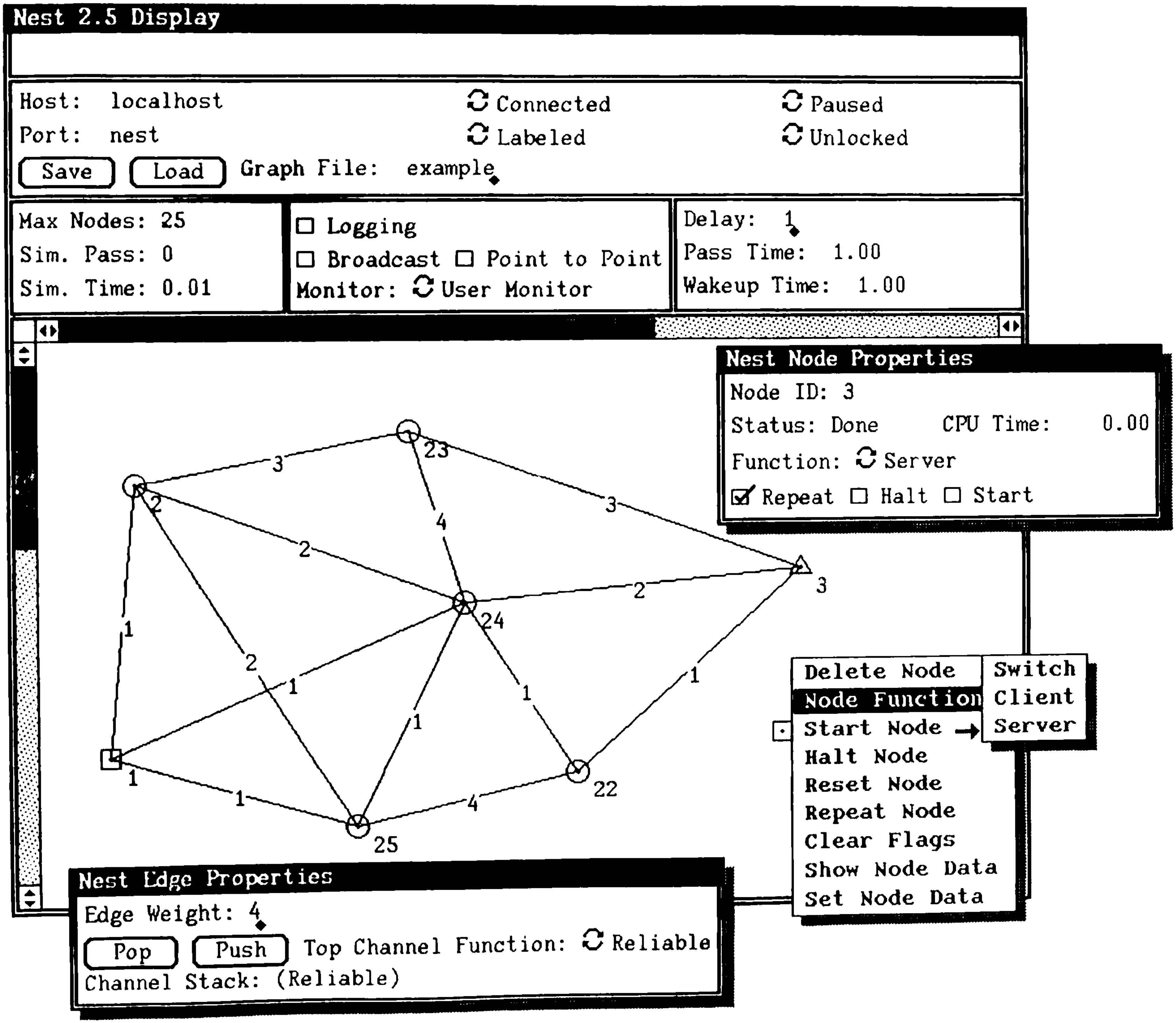
Nest's custom monitors offer tools to display the results of a simulation. A user may view the status and data associated with different nodes or performance statistics of interest. The custom monitors may be used to animate the dynamics of the simulation behavior and represent the evolution of local partial views of the system state. This is particularly useful in the study of complex dynamical behaviors of distributed systems.

2.3 THE SIMULATION SERVER AS AN EFFICIENT LIGHT-WEIGHT PROCESS MODEL & SCHEDULER

Nest supports an efficient light-weight process model to facilitate simulation of complex distributed systems. A process typically models the behavior of a node. A process is provided with an appropriate context including configurational information (*e.g.*, respective node and incident links), simulation scheduling information (*e.g.*, pending messages) and execution information (*i.e.*, pointer to its run-time stack). Nest simulation server manages the appropriate scheduling of processes execution and the context switching. Multiple threads of execution are supported from within a single Unix process. The overhead associated with context-switching is thus significantly reduced. Therefore, Nest can support large simulations (scores to hundreds of nodes) in a workstation environment and hundreds to thousands of nodes within a more powerful server environment.

Scheduling of simulation events is made complex by the mixture of real and simulated events. It is necessary for Nest to manage simulated time that is, in-part, controlled by real events. Simulated events requiring synchronization with the simulation clock include attempts by a processes to receive a message (*i.e.*, it is necessary to suspend the process until all messages that should have been delivered by its simulation time are available) or explicit requests by processes to control their timing (*e.g.*, a request to be suspended for certain simulation time).

Nest permits the simulation clock to be controlled by internal and external events occurring in real-time. Thus, since a user may wish to study the execution of a real code (for node processes) the actual execution time of a particular segment of a process may be used to evaluate the respective simulated time and synchronize with the simulation clock. Furthermore, the users too can be a source of real-time events that are mapped to simulated time. Users can interact with the simulation directly (*e.g.*, changing the network configuration during a run). A change by the user redirects the evolution of the simulation. It is thus necessary to assure that the user real-time interaction is properly managed in simulation time. This implies that a user event can only take place after all simulated processes have been synchronized to an appropriate simulation time. (Otherwise it would be possible for some processes to continue and execute in the past relative to the changes introduced by the user).



Nest 2.5 Display

Host: localhost

Port: nest

Graph File: example

Max Nodes: 25

Sim. Pass: 0

Sim. Time: 0.01

□ Logging

□ Broadcast □ Point to Point

Monitor: User Monitor

Delay: 1

Pass Time: 1.00

Wakeup Time: 1.00

Nest Node Properties

Node ID: 3

Status: Done CPU Time: 0.00

Function: Server

☑ Repeat □ Halt □ Start

Nest Edge Properties

Edge Weight: 4

Top Channel Function: Reliable

Channel Stack: (Reliable)

Figure 2: The generic monitor

These complex mappings of real-time into simulated time are managed by Nest through a simple adjustable scheduling policy. Nest passes through all its processes using a round-robin scheduling. During each pass all processes are provided with a quantum of simulation time to be executed. Processes that are suspended (*e.g.*, for reception of messages that have not yet been delivered or through direct requests) only require a simple advance of their clocks and potential reactivation if the respective time is arrived at. Changes of configuration through user interaction or program control can only be executed between passes.

The user can adjust the temporal duration of a Nest pass. In the limit when the pass duration is very long, the simulation schedule is entirely controlled by simulated events (*e.g.*, communications), as real-time interactions are not permitted. Processes will be permitted to execute until they reach a synchronizing event when they are suspended. The simulation clock will be advanced at a maximum rate possible as it is not necessary to anticipate external interactions and advance the clock in locked-step. If significant level of external control is desired, a shorter duration for the pass can be selected leading to finer locked-step execution of the simulated processes.

3. AN EXAMPLE OF NEST APPLICATIONS

Having looked at the structure of Nest, we will turn to the programmer interface, *i.e.*, what someone writing a simulation, or prototyping a distributed system, needs to know in order to use Nest). In the classic Unix tradition, we will use a variant of the famous *"hello, world"* program to demonstrate the basic features and usage of Nest.

The program in Figure 3 is a complete Nest program in C. It can be compiled and linked with the Nest library to create a Nest simulation program. The first thing you may notice is the absence of a *main*() routine. While the programmer can supply a *main*() routine for the emulation program if desired, the Nest library contains a generic main routine which initializes the simulation with *node_main*() as the main routine for each simulated node. This main routine for each node takes a single argument, the node id assigned to it by Nest. Node ids are used by Nest to uniquely identify each node, and are used whenever a node needs to be specified for a Nest function.

An important part of a network simulation is communications between nodes. An example of this can be seen at the beginning of the *node_main*() routine. The first thing the routine does is to broadcast a message to all its neighbors. A message in Nest consists of two parts, usually called the key and the data pointer. While these are both just 32 bit quantities, they are conventionally used in different ways. The key is typically used to identify messages, either by type or by number. The data pointer is usually a pointer to the data of the message, in some format determined by the type of the message. In the example, the key is the defined constant HELLO, and the data pointer is just a pointer to the null-terminated string "hello, world". This simple message structure of key and data pointer is extremely flexible, since the data pointer can point to any sort of data, from character strings to complex linked data structures, such as trees and lists.

After communication, the next most important thing in a simulation is the passage of time. Since all the processes in a simulation are running in a single Unix process, and since the passage of time within the simulation is largely independent of real time, system calls such as *sleep*() and *time*() will not have the desired effect. Instead Nest provides alternate routines, such as *slumber*(), which is the next function called in the example. After a node broadcasts the hello message, it would like to receive messages from its neighbors. But it may take a certain amount of time for the messages to arrive. So slumber is called to suspend the node (in this case, for five seconds) to allow messages to arrive. The SLUMBER_NOWAKE parameter is a defined constant which tells Nest not to interrupt the slumber if messages arrive.

Once the node has waited a certain amount of time, it calls *any_messages*() to see if any messages are available to be received. This prevents the node from blocking indefinitely on a receive if there are no more messages which have been sent to it. While there are messages to be received, the node calls *recvm*() to receive the message. It passes three pointers to variables, which are set to the original destination, key and data pointer of the message. The original destination stored in *dest* is just the node id of the receiving node, or 0 (which is not a valid node id) if the message was broadcast. The nodeid of the sender of the message is returned by *recvm*().

Once the message has been received, the example prints a diagnostic message describing the message which has been received. Before it does, it calls the *hold*() function to prevent Nest from interrupting the printf call and giving control to another node. This ensures that the messages from different nodes will not be mixed together, as well as preventing any problems caused by non-reentrant implementations of the stdio library. The *release*() function is called afterward, indicating that the critical region has ended. The parameter to release indicates the number of nested *hold*() calls to be released.

Finally, the node replies to each received HELLO message with an acknowledgment. Since the acknowledgment is directed to the node which sent us the HELLO, *sendm*() is called instead of *broadcast*(). The first parameter to sendm is a destination node id; otherwise it is identical to broadcast.

Figure 3: Nest "Helo, World" Program

```
#include <nest.h>

#define HELLO 1
#define ACK 2

struct timeval five_seconds = { 5, 0 };

node_main (nodeid)                                                node_main
ident nodeid;
{
   char *message;
   ident dest, sender;
   int msgtype;

   broadcast (HELLO, "hello, world");                     /* broadcast hello message */

   slumber (&five_seconds, SLUMBER_NOWAKE);               /* wait for hellos */

   while ( any_messages ( ) ) {                           /* avoid blocking on recvm */

            sender = recvm (&dest, &msgtype, &message);

            hold ( );                                     /* begin critical section */

            printf ("%d received \"%s\" from %d via %s\n",
                        nodeid, message, sender,
                        dest == 0 ? "broadcast" : "sendm" );

            release (1);                                  /* end critical section */

            if (msgtype == HELLO)                         /* acknowledge hello message */
               sendm (sender, ACK, "isn't that a bit cliched?");
   }
}
```

4. IMPLEMENTATION OF NEST

The key goals of Nest implementation are: efficiency, portability and extensibility. Efficiency of simulation is of great importance in the study of large scale complex distributed systems. Certain phenomena occurring in such systems cannot be extrapolated from the study of small scale simplified versions. However, a simulation study of complex large scale systems may require significant computing resources and consume too long a time. Nest accomplishes significant efficiency through the use of single process multi-threaded execution model and through the use of an optimized scheduler. The single-process execution model involves a minimal amount of context switching overhead, significantly less that the overheads associated with multi-tasking implementation. Sharing memory among the different process threads permits the simulation to accomplish significant efficiency (*e.g.*, passing pointers instead of full messages). Finally, the scheduler, considered above permits the user to fine tune the execution runs to maximize efficiency or real-time reconfiguration and experimentation. Typically, a user will set the initial granularity of the round-robin passes to facilitate high degree of interaction and change. Once the scenario of interest has been defined, the user can reset the pass duration to allow for efficient uninterrupted execution of the simulation.

Nest has proved very efficient in studies of both large scale and complex distributed systems. Exact comparative benchmarks of simulation tools are yet to be developed. Nest has been used in simulation studies (utilizing a Sun server) involving networks of hundreds nodes executing a standard routing protocol model. Similarly, Nest was applied in the development of a complex distributed transaction processing model where node functions involved over 20k lines of codes. In all cases the response time was very fast.

Portability has been accomplished by minimizing and localizing dependencies of Nest on specific hardware or even Unix variant characteristics. A typical port of the simulation server can be accomplished in a matter of a short few days. The client software depended initially on Sun window systems. A recent porting to the X-window environment can be expected to ease the portability of the user interfaces. Nest has been ported into a large number of workstations environments and exported to over 150 users sites worldwide.

Extensibility and customization by users are key elements of Nest's design. From a user's perspective Nest is perceived as an extension of the standard Unix environment. Nest simulation server and clients are simply Unix libraries of functions. The user can modify any of these functions or augment them with her/his own. A user simulation consists of expansion of the Nest libraries with user-provided node and link functions. This process of extension permits the user to adapt Nest and develop it into a custom environment for specific simulation studies of interest. Researchers at Northrop Corporation [10] pursued this approach to develop a simulation testbed for protocol research. Similarly, researchers at UC Berkeley [8] extended Nest into a complete environment to study and test TCP/IP internetwork designs (*e.g.*, gateway routing techniques). In both cases Nest was equipped with node functions modeling in details the respective protocol environments.

Incremental expandability is a key element in long term development of simulation studies. Typical simulation studies are designed as throw-away software and the enormous investment in their development is lost when the object of the study is completed. Nest simulations can be construed to retain a significant part of the investment through incremental expansion. Thus, a TCP/IP internet testbed may be used to support a significant number of relevant studies sharing the same testbed software. Users can share and port the respective libraries among different sites leading to important savings and cross-fertilization.

Finally, customization and expandability is also supported in the design of the Nest client monitor tools. All menus provide handles for simple adaptations and expansions to support user-defined options. This completes the range of flexibilities offered to users in developing testbed simulation studies.

5. SAMPLE APPLICATIONS OF NEST

5.1 IPLS - a Distributed Incremental Position Location System

Nest was initially developed as a tool to experiment with the design of a distributed position location system. Consider a network of packet-switched mobile radio units. Two radios within range of each other can measure the propagation delay between them and extract an estimation of their mutual distance from each other. Given these distributed observations of distances it is required to compute the location coordinates of the radio units. IPLS involves a few distributed algorithms that aim to address the range of problems arising from mobility, partial distributed measurements and possible errors. Nest became a key tool in the development of IPLS and in the the performance studies of these algorithms under varied dynamic scenarios.

5.2 Topology Recognition, ARPAnet and Internet Routing

Broadcasting connectivity tables is an important technique to accomplish topology recognition by nodes of a dynamic network. Topology recognition protocols are used within a number of communication networks. It is difficult to establish the dynamic behavior of such algorithms. Examples of instabilities and long response time (to failures) have been discussed in the literature [3]. However, the theoretical understanding of these dynamics is in its embryonic stages. Simulation studies are the only practical tool at this time in developing better understanding of these dynamics. Traditional queuing-network simulators typically address equilibrium behaviors. Nest became an important tool in the study of dynamic response behaviors. A few studies of topology recognition algorithms were conducted. Similarly, the response of the ARPAnet routing algorithm [3] to node and link failures was studied extensively [12].

In the examples above the respective protocols were coded directly as part of the study. In contrast, another Nest study attached parts of actual BSD code (with minor modifications) for IP gateway routing to a simulated network environment and demonstrated the occurence of loops, instabilities and long response time to topology changes [9].

5.3 Dynamic Load-Balancing, Distributed Transaction Processing

Other work [7] has used Nest to develop simulations of complex distributed systems. Microeconomic models of supply and demand, with bidding and auctions, were used to develop a dynamic load-balancing system. Processes were given a certain amount of "money", and would bid for communcations and CPU resources. This bidding behavior was implemented on the nodes in a Nest simulation and analyzed to find the relative performance of various bidding strategies and auction methods, and to compare them with traditional load-balancing methods.

A more complex simulation used the microeconomic models to manage a distributed transaction processing system with replicated data. This simulation had upwards of 20,000 lines of C code running on each node, showing the ability of Nest to model complex behaviors.

5.4 Distributed Multiprocessor Operating Systems

Nest has also been used to simulate the behavior of an experimental multiprocessor operating system [1]. In this study, operating system code was run using Nest to see how various performance measures would be affected by adding additional processors. The results generated with Nest were later verified on real hardware and found to differ by only a few percent.

6. CONCLUSIONS

The study of Nest and its applications established a few important results:

Environment-based simulation tools can offer important attractions over language-or model-based approaches. The users do not require sophisticated expertise in the use of complex modelling tools or specialized languages. The simulation tools can be entirely integrated within the user standard development environment and offer a unified set of tools. While Nest was developed to support , the extension to support other standard languages (*e.g.*, Pascal, Fortran) is straightforward. Simulation can be conducted as an integral part of the design and implementation cycle. The actual code of the system developed (or a modification of it) may be used to model its behavior.

The environment can support simple modeling of arbitrary simulation scenarios and execute the actual system within the resulting simulation testbed.

It is often important to study through simulation the dynamic response of the system to changes. These changes may be introduced through user or program control. It is therefore useful to separate scenario modeling and control functions from the simulation code. The scenario may then be passed to the simulation as a parameter, allowing the simulation to adapt to new scenarios and respond to the respective changes.

Separation of scenario display and control from the simulation execution in terms of simulation server and monitoring clients can offer additional attractions. The simulation may be executed over a remote computational server permitting optimum utilization of the server cycles and faster response. The client monitors may provide effective remote (re)configuration and scenario controls. This permits users to access over a network substantial more computing capabilities than may be available to them locally and conduct extensive simulations studies. Additionally, as is often the case, the development of a complex distributed system often involves work by multiple sites. A common simulation testbed can support sharing of software and efforts as well as improved studies of the interactions between the different subsystems. Furthermore, it serves as an important factor in improving the integration processas it enforces certain standards over the disparate development efforts.

Finally, extensibility, customizability and portability enable users to extend and adapt Nest to become a total design environment for their systems. The word-of-mouth ad-hoc distribution of Nest to over 150 sites and its successful applications in scores of different projects provide a measure of success in accomplishing the results reported in this paper. The authors would be pleased to share Nest software and experiences with other interested sites and users. Please use the address above to pursue this further.

REFERENCES

[1] A. Barak, International Computer Science Institute, Berkeley, private communication, 1989.

[2] D. Bacon, A. Dupuy, J Schwartz, Y. Yemini, "Nest: A Network Simulation and Prototyping Tool", Proceedings of the Winter USENIX conference, February 1988.

[3] D. Bertsekas, "Data Networks", Prentice-Hall, 1987.

[4] A. Demers, S. Keshav, S. Shenker, "Analysis and Simulation of a Fair Queueing Algorithm", Submitted for Sigmetrics 89, September 1988.

[5] A. Dupuy, "Nest User Interface Manual", Columbia University, March 1988.

[6] A. Dupuy, "Nest User's Guide", Columbia University, March 1988.

[7] D. Ferguson, "The Application of Microeconomics to the Design of Resource Allocation and Control Algorithms", Ph.D Thesis, Columbia University, 1989.

[8] S. Keshav, "REAL: A Network Simulator", Technical Report No. UCB/CSD 88/472, University of California at Berkeley, December 1988.

[9] C. Maio, Project Report, Columbia University, 1988.

[10] M. Rose, "The Nest Simulation facility at NRTC", Technical Report, Northrop Research and Technology Center, 1987.

[11] C. H. Sauer, E. A. McNair, J. F. Kurose, "The Research Queueing Package: past, present and future", Proceedings of the National Computer Conference, Arlington, VA, 1982.

[12] B. Swinyer, Project Report, Columbia University, 1988.