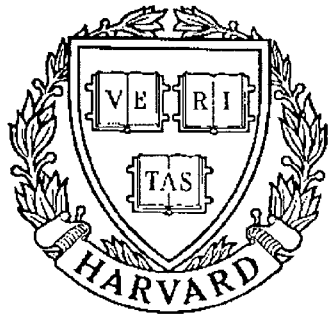
TECHNICAL RESEARCH REPORT



SYSTEMS RESEARCH CENTER



Supported by the National Science Foundation Engineering Research Center Program (NSFD CD 8803012), Industry and the University

Distributed Network Simulation Testbed Status Report

by M.S. Corson

TR 89-84

Distributed Network Simulation Testbed Status Report

M. Scott Corson

October 1989

Abstract

The following report describes the facilities, personnel, and simulation effort underway in the Communications and Signal Processing Laboratory (CSPL) of the Systems Research Center (SRC). The intent of the effort is to create a useful simulation and modeling environment to aid in the performance analysis of communication systems. In particular, we describe a distributed network simulation testbed under development to allow rapid prototyping and performance evaluation of distributed network communication protocols. The testbed's design is modular thus allowing it to be built in a piecemeal fashion with each sub-project contributing software modules to the testbed. We first describe the facilities and personnel involved in the simulation effort and then describe the current testbed status and future architecture.

1 Introduction

The CSPL facility was originally established in 1985 to perform research on speech and image processing. Since 1987 its scope has been expanded to include among other activities the development of a test-bed for communication network simulation and protocol performance evaluation. Initial funds for this expansion were provided from an Office of Naval Research Grant, industry contributions and matching funds from the Engineering Research Center and the Electrical Engineering Department. Subsequently, additional support was provided by SRC. A key to the facility's development was the collaboration with the University of Calgary and, then, Jade Simulations International from whom the JADE software system was acquired and with whom there is an ongoing cooperation.

2 Facilities and Personnel

2.1 Hardware

As of the Fall of 1989, the hardware platform consists of a local area network (LAN) of Sun Workstations residing in the CSPL. The breakdown of the LAN is as follows: 5 Sun 3/50's, 2

Sun 3/140's, 2 Sun 3/160's, and 1 Sun 4/110 RISC machine with one of the Sun 3/140's acting as the network file server controlling 862 megabytes of memory. Also, a Masscomp is located in the lab which is used primarily for remote file storage and D/A conversion.

In addition, 5 Sun 4 SPARC workstations (each with 16 megabytes of RAM) and 940 megabytes of disk capacity are on order to complement the existing network. Future acquisitions are also under discussion.

2.2 Software

The software environment consists of two distributed simulation packages; both of which currently reside on the Sun 3 workstations.

The first is the Jade[1] Inter-Process Communication (JIPC) Protocol. The protocol allows individual UNIX processes to enter a special JIPC subsystem. In so doing, the processes each acquire a JIPC process ID which is unique among all processes within the subsystem. This subsystem may reside on a single machine or span the entire Sun Network. Once in the subsystem, the processes are free to communicate with any other processes in the subsystem regardless of whether or not they reside on the same machine. In this way, the network resembles a large virtual machine. This package is the foundation of the current testbed described in Section 3.2.

The second package, which will serve as the basis for the future testbed described in section 3.3, is the Sim++ Programming Language[2]. Sim++ is a process-oriented, discrete-event simulation language embedded in the object-oriented programming language C++. Sim++ is designed so that programs are able to execute either sequentially on a single processor or in parallel in a multi-processor environment. Sim++ currently runs only on Sun 3 Workstations; however, JSI plans to support the Sun 4 architectures in the near future.

The relation of the simulations to their hardware and software platforms can be seen in Fig. 1.

[1] The JIPC software package was developed by the Department of Computer Science at the University of Calgary.

[2] Sim++ was developed by Jade Simulations International (JSI), Calgary, Alberta.

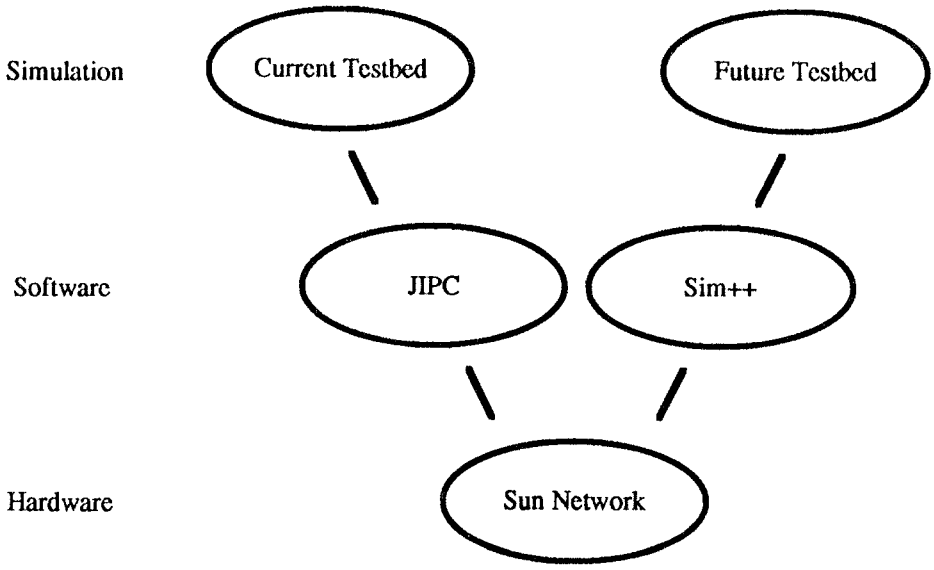


Figure 1: Software/Hardware Configuration

2.3 Personnel

The following people are involved as of the Fall of 1989 in developing the simulation testbed.

- M. Scott Corson (Faculty Research Assistant)
  - Developed current testbed to perform performance evaluation of distributed routing algorithms as part of M.S. Thesis.
  - Responsible for overall design and implementation of future testbed.
- Amelia I. de la Rosa (Research Assistant)
  - Will contribute towards testbed development by writing software modules directly and indirectly related to her research.
- Douglas Oard (Judith Resnick Fellow)
  - Will contribute towards testbed development by writing software modules directly and indirectly related to his research.
- Nina Srinath (Graduate Student)
  - Will contribute towards testbed development by writing software modules directly and indirectly related to her research.

The following people have utilized the hardware facilities as part of their research.

- Leandros Tassiulas (Systems Research Center Fellow)
  - Performed performance evaluation of a hopfield neural network implementation of a channel access scheduling algorithm.
- Partha Bhattacharya (Systems Research Center Fellow)
  - Performed performance evaluation of different queueing system service strategies.
- Eytan Modiano (Naval Research Laboratory Fellow)
  - Performed performance evaluation of secure communication protocols.

3 Distributed Network Simulation Testbed

3.1 Scope

We now describe the current status of the distributed network simulation testbed, its proposed future design, and the short term implementation goals. The purpose of the testbed is to provide a vehicle for prototyping and analyzing the performance of communication protocols. The testbed is intended to be flexible and will allow modeling of both point-to-point and broadcast networks with time-varying or static topologies. Consequently, the testbed can be tailored to model static point-to-point data networks, local area networks, mobile radio networks including packet and cellular, satellite networks or general abstract networks of any sort. We first describe the present testbed status, then detail the future testbed architecture, and finally list the near term software development objectives.

3.2 Current Testbed Design

The current testbed is written in the C Programming Language and is presently tailored to modeling mobile radio networks. It is extensible to modeling other types of networks, although not as readily as the architecture proposed in section 3.3.

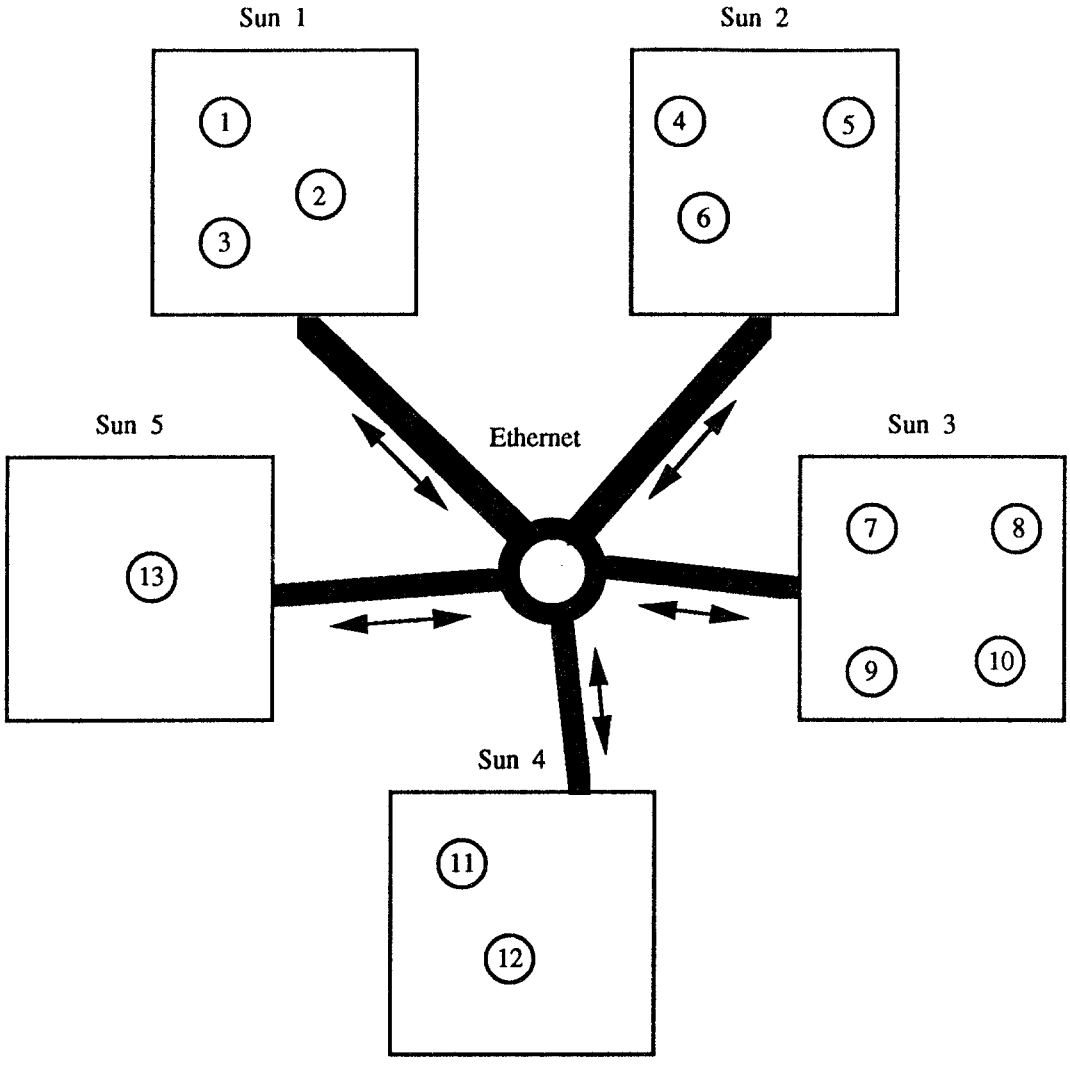
3.2.1 Overall Architecture

The testbed is designed as a multi-process simulation which can operate in either a single machine or multi-machine environment. In the single machine case, all processes reside on the same workstation. In the multi-machine mode, the processes may be distributed over the network to the various workstations in any manner desired. Both cases are shown in Fig. 2. In either case, regardless of the mapping of processes to processors, the same set of inputs and a given random seed will produce the same pseudo-random simulation run so that the simulation outputs are deterministic and reproducible.

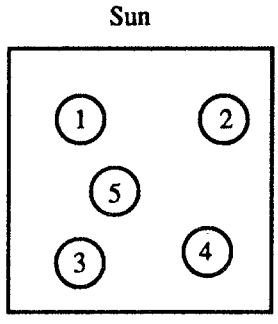
The processes composing the distributed simulation communicate via message passing according to the rules of the JIPC Protocol described earlier. The simulation consists of primary processes and their corresponding ear processes. The primary processes consist of a central process and a separate process for each node of the mobile radio network to be simulated. The ear processes exist only to receive and buffer messages bound for their primary processes. The ear processes are necessary because JIPC is a blocking protocol in that the sender of a message is "blocked" until the receiver receives and acknowledges the reception. Therefore, when one primary process wishes to communicate with another, it sends a message to the corresponding ear process which receives the message and acknowledges it immediately so that the sender is not blocked for long. Then, when the receiving major process is able to accept a message, it queries its ear and retrieves the message from the ear's buffer. This message flow is illustrated in Fig. 3.

The simulation is modeled as a discrete event system. The event list and simulation clock are managed by the central process which, after each event, consults the current clock time and event list to determine the next event. The node processes spend the majority of their time sleeping, awaiting events. When the central process determines that a node is ready to process an event, the node is awakened and informed of the current simulation time and event. After processing the event, the node determines its next event time and sends this time and the event type to the central process.

It is possible for more than one node to need to process an event at the same time. In such cases, it is be possible to achieve some gain in computational speed through parallel processing if the node processes reside on different machines in a multi-machine environment.



(a) Multi-machine mode



(b) Single machine mode

Figure 2: JIPC Process Distribution over Sun Network

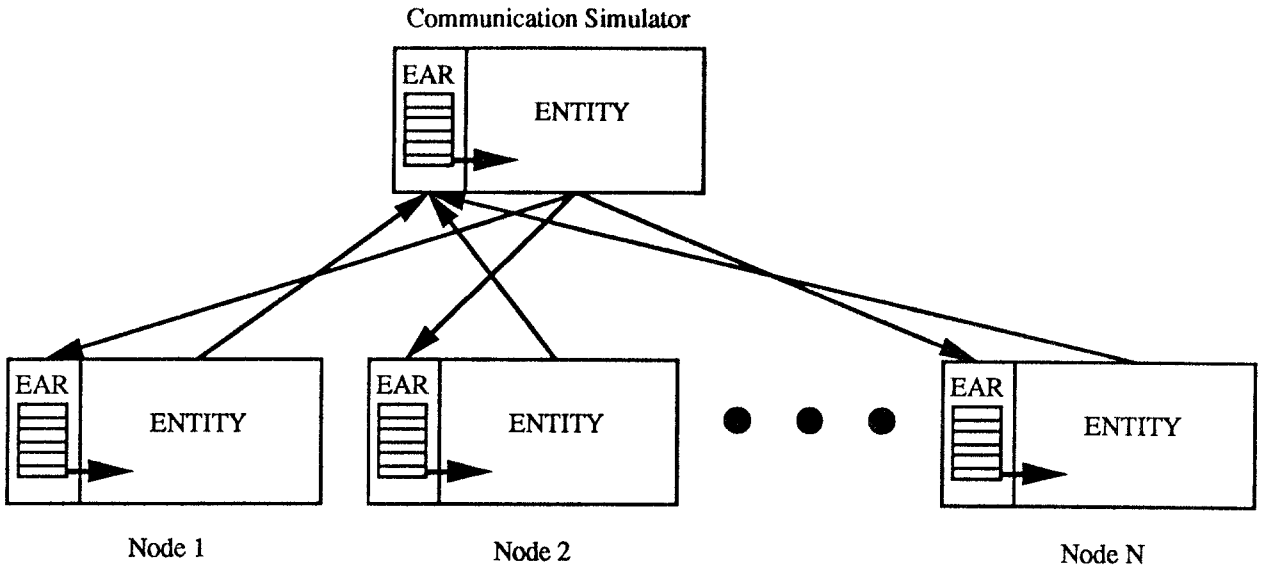


Figure 3: Inter-process Message Flow

However, in general, the multi-machine simulation is slower than the single machine case because the gains resulting from parallel execution are overshadowed by the communication overhead incurred by sending messages over the LAN. The major advantage of the distributed simulation is that since the network appears as a large virtual machine, the total amount of available RAM is much greater thus permitting the simulation of much larger networks than would otherwise be possible.

Channel Access Schemes

The current simulation only models two broadcast channels; however, this number can easily be increased. At present, all channels are noiseless. Two different channel access methods have been modeled and are available for general use. Both schemes are centralized and random access in nature. The first scheme is simple random access and allows as many nodes to transmit as possible, provided no two adjacent nodes transmit simultaneously. The second scheme is also simple random access and allows as many nodes to transmit as possible, provided no primary or secondary collisions occur. This scheme is the more realistic of the two in terms of the effect on throughput and delay.

Delay Model

Link propagation, node processing, and packet transmission delays are all modeled in the

simulation. Also, each node has a transmission queue for each channel and the effects of queueing delay are factored into the simulation results.

Topological Model

Presently, the testbed provides two ways to model abstract variable-connectivity radio networks. One method assumes the nodes are moving randomly in a 2-dimensional plane according to some motion model. A link exists between two nodes if the distance between them is less than a specified transmission radius. The other method assumes that the nodes' locations are fixed and that there exists a certain base topology. This topology is varied by randomly toggling the UP/DOWN status of the links.

3.2.2 Modeled Protocols

Three distributed routing protocols for mobile radio networks have been coded and tested via simulation in the testbed. They are (i) a flooding algorithm similar to that performed in the ARPANET algorithm [1] (ii) the Gafni-Bertsekas protocol described in [2] (iii) and a recent protocol by Corson and Ephremides described fully in [3].

Even in this early form, the testbed has already proven itself as an invaluable tool in evaluating the performance of communication protocols. Without the testbed, a realistic performance comparison of the above protocols would not have been possible as the problem is analytically intractable.

3.3 Future Testbed Design

3.3.1 Hardware and Software Basis

Hardware

The simulation will continue to reside on the Sun Network described earlier. Over the next few years, the expectation is to perform all the modeling and simulation in software. However, a long-term goal is to build a hardware channel emulator which will physically model the effects of noisy communication channels in real time. This will substantially increase the speed of simulation since the software will be relieved of the channel emulation burden. Furthermore, a hardware-based evaluation process will be applicable to high-speed

broadband networks of the future, such as optical LAN's.

Software

The future version of the testbed will be written in the Sim++ Programming Language described earlier. In Sim++, a program is broken down into objects or entities. These entities are loosely coupled, independently executing objects that interact by scheduling and receiving events. These entities may either all reside on a single processor or be distributed over multiple processors. In many ways, the entities in a Sim++ program are analogous to the processes of a JIPC-based simulation. The interacting entities form a distributed, object-oriented, discrete-event simulation whose synchronization is maintained by mechanisms built into Sim++, thus relieving the programmer of the burden of process synchronization.

While the CSPL Laboratory is currently a beta-test site for both the sequential and distributed 2.0 versions of Sim++, only the sequential version has been purchased. We are in the process of evaluating the speed-up potential of parallel execution for programs executed distributedly over the LAN. Should the speed-up be significant, we may consider eventually purchasing the distributed version. Otherwise, it will be more advantageous to simply buy additional hardware and run the programs sequentially.

3.3.2 Simulation Architecture and Capabilities

Overall Structure

The overall simulation architecture can be seen in Fig. 4. The middle block is the Communication Simulator which handles all aspects of communication modeling. It receives input from scenario configuration files which specify all initial information required for a particular run. In addition, input can be obtained during the simulation run either from the simulation preprocessor directly or from physical event lists precomputed and stored by the preprocessor prior to the simulation run. Any information computed by the preprocessor must be such that it is independent of any actions which might take place during a simulation run. For example, the random motion of nodes and the resulting dynamic topology might be precomputed prior to the simulation run provided the nodes' motion models are independent of simulation events. In this way, specific random topologies can be

computed once, stored in memory, and reused indefinitely.

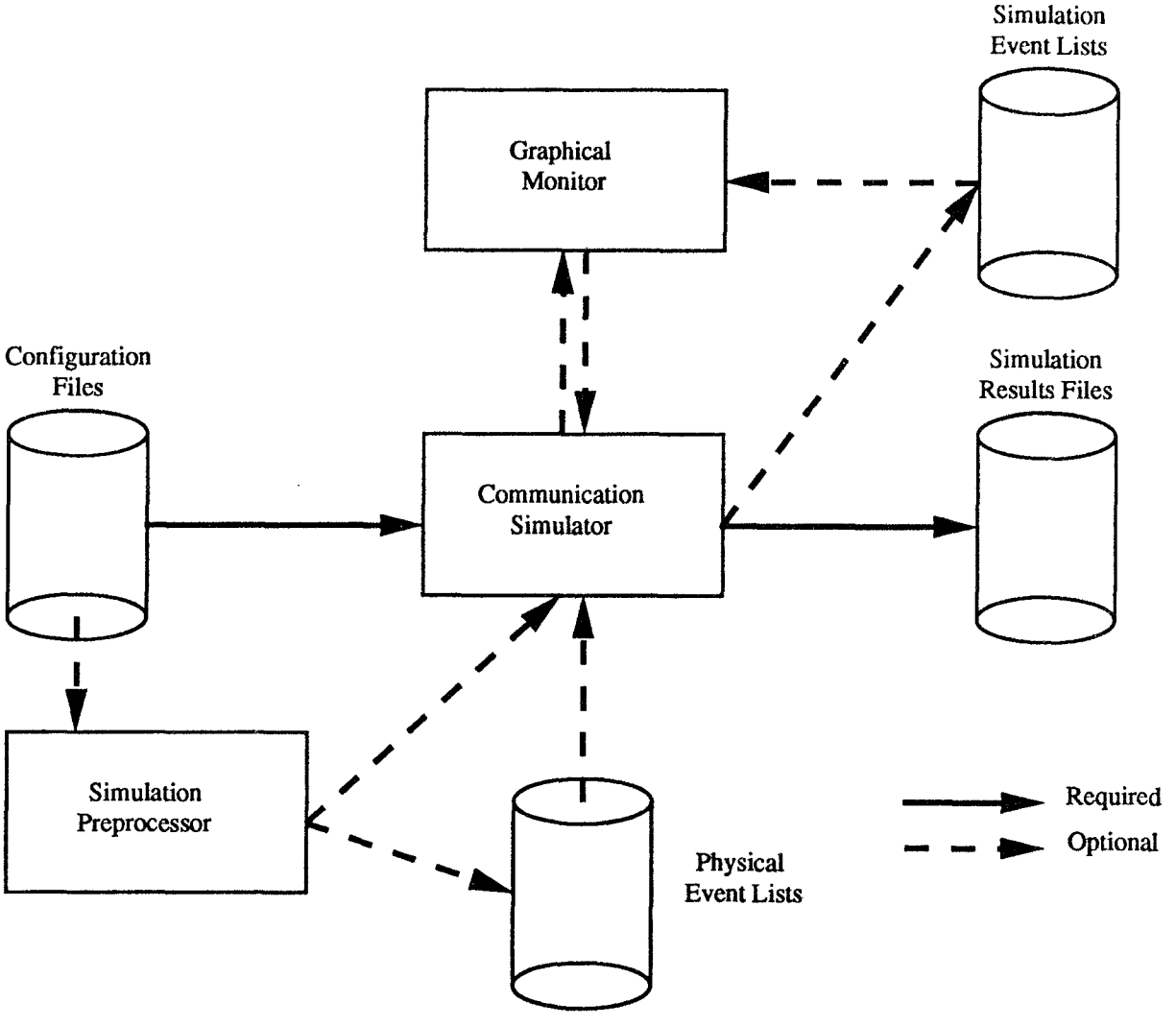


Figure 4: Overall Testbed Architecture

At the end of the simulation, the simulator stores performance parameters initially requested by the configuration files in simulation results files. Optionally, the simulator may also keep an event trace of the entire simulation run and store this trace in a simulation event list. This event list may later be examined manually or viewed graphically through the graphical monitor.

The graphical monitor is shown in the figure above the Communications Simulator and

serves three main purposes. First, the monitor may be used to view a previously run simulation which has been stored in an event list. Second, the monitor may be used to view a simulation as it runs thus receiving its input directly from the simulator. In both of these modes, the monitor is asynchronous and "parasitic" in that it only receives and displays information. In the third mode, the monitor is synchronized with the simulation and the speed of the simulation can be controlled through the monitor. This mode is useful for both demonstration and debugging purposes.

Communication Simulator

We now look more closely at the Communication Simulator. Fig. 5 shows the simulator composed of N node objects and L channel objects. Although not shown in the figure, node objects can be grouped into clusters; the significance of which is that all nodes in a cluster have the same velocity vector and thus move together. Node objects communicate with one another by transmitting messages through channel objects.

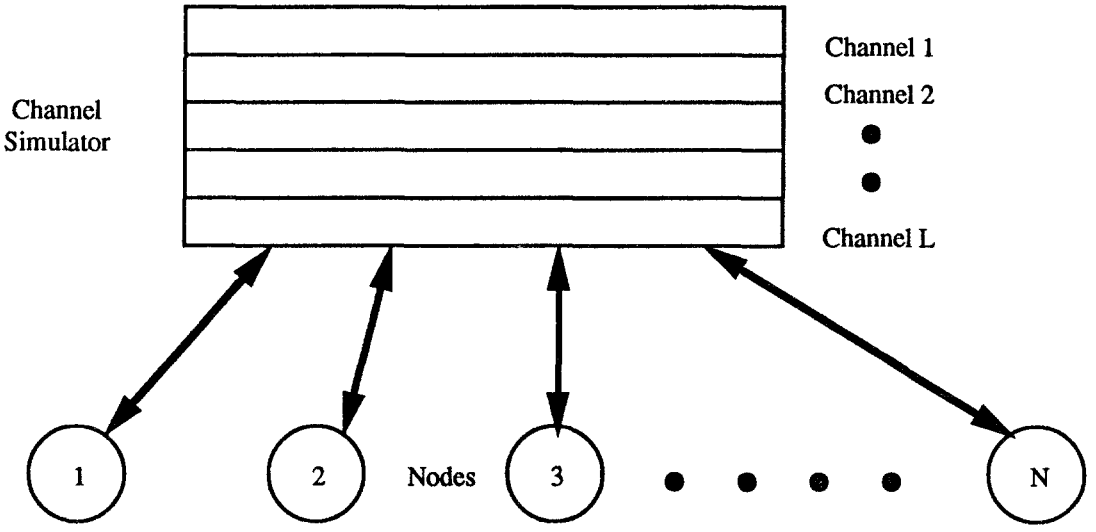


Figure 5: Communication Simulator

Fig. 6 shows the internal structure of a node object. The node is divided into layer objects according to the 7-layer ISO OSI reference model. Any communication protocol, when modeled, will have to be separable according to the ISO layering scheme. Each layer object communicates with adjacent layers according to a generic layer interface to which all protocols must adhere. The layers pass message events amongst themselves according to the

rules of the particular protocol which is being modeled. The Data Link Control (DLC) layer is subdivided into a number of smaller DLC objects. Each DLC object communicates with a transmitter and receiver object as shown in the figure. These transmitter and receiver objects may be allocated or tuned to a particular channel object for message transmission and reception.

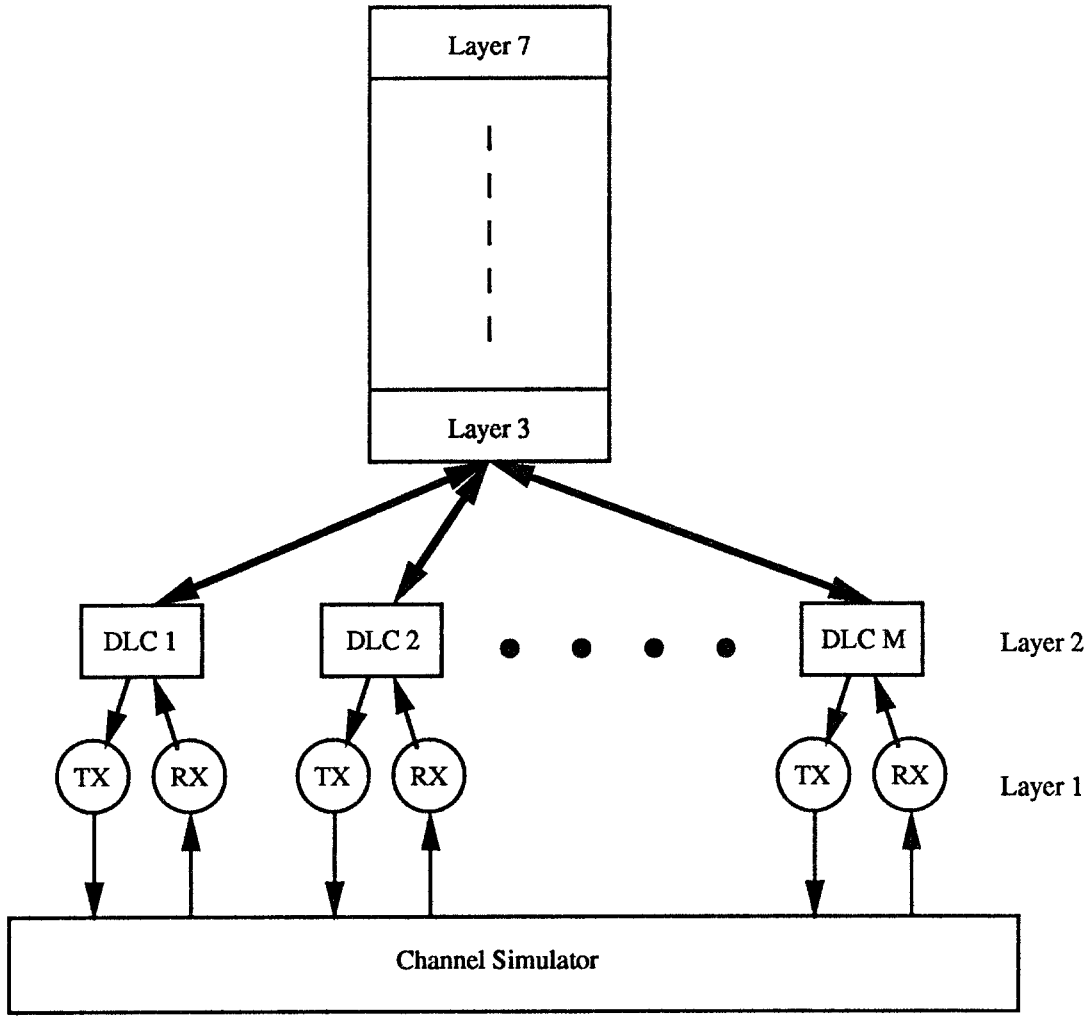


Figure 6: Node Internal Structure

A channel object is composed of a number of link objects; each linking a particular transmitter-receiver object pair within the channel which is "connected" in some sense by the model (eg. either physically via hardware or within transmission range). The channel can model both interfering and non-interfering links which are either bi-directional or uni-directional.

In point-to-point channel models, there is a separate receiver object for each incoming link and a separate transmitter object for each outgoing link as shown in Fig. 7b. In broadcast channel models, there is only one receiver object and one transmitter object per node. In these models, multiple link objects can be connected to individual transmitter or receiver objects thus permitting transmission interference. Also, when a node has a transmitter and receiver allocated to the same broadcast channel, the transmitter output is fed into the receiver input through a suitably modified link object as seen in Fig. 7a. In such cases, the receiver object is "blinded" by the transmitter object when it is transmitting.

A link object is shown in Fig. 8. It has a transmitting end assigned to a transmitter object, a receiving end similarly assigned to a receiving object, a link emulation object which simulates the effects of propagation loss, propagation delay, and noise.

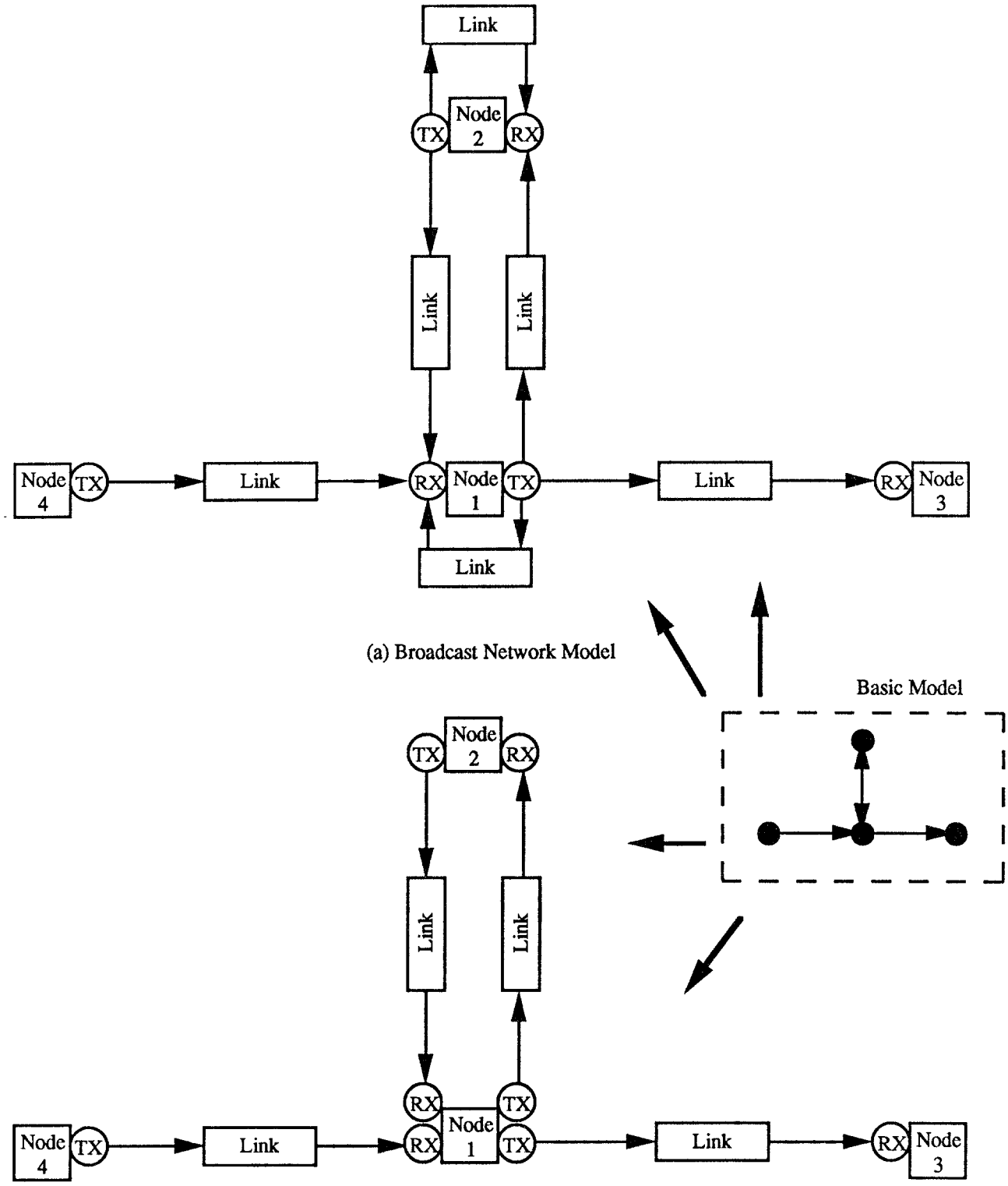
3.4 Software Design Methodology and Development Strategy

3.4.1 Design Methodology

From the preceding description, it should be apparent that the simulation is object-oriented and hence, very modular. The division of objects into smaller objects continues below the level which has been presented here. For instance, the link emulation object actually consists of two objects; one to compute the physics parameters (i.e. propagation loss and delay) and another corresponding to the link noise model. Given this modularity, different noise models can be easily swapped in and out of the simulation provided they all adhere to a common physics-object/receiver-object interface. Similarly, software libraries of receiver and transmitter models can be maintained and reused indefinitely. Object-oriented design permits a very modular, flexible, and reusable software system.

3.4.2 Development Strategy

Fig. 9 shows the software hierarchy as it has been presented here. The simulation is divided into three major components: Simulation Preprocessor, Communication Simulator, and Graphical Monitor. These components, although not functionally independent, interact to a limited degree and once their interaction has been defined, the three components can be developed individually in parallel.



(a) Broadcast Network Model

(b) Point-to-point Network Model

Figure 7: Channel Internal Structure

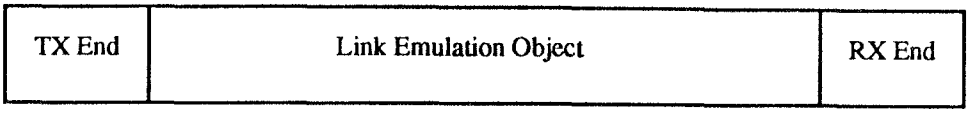


Figure 8: Link Internal Structure

The development of each component will proceed in a top-down fashion. Upper level objects and their external interfaces are defined. These objects are then subdivided into smaller, less abstract objects and the process is repeated. The subdivision process continues until a point is reached where each lowest-level object has a single, clear task to perform.

Most of the "work" in the simulation is done by these low-level objects. In performing their tasks, these objects will call upon many generic[3] system software routines and tools. In addition to these system supplied routines, generic software objects will have to be written in Sim++ to perform a variety of functions such as specialized statistical collection, random number generation, queueing and list management, and X windows-based graphic management. Therefore, in parallel with top-down development of the major simulation components, work can begin on generic low-level packages which will be required by the simulation.

3.5 Summary

The future testbed is intended to be a flexible, reusable software system capable of modeling any conceivable communication network architecture. Its modular, object-oriented design will permit simplified substitution of protocol, link, channel, receiver and transmitter models for purposes of protocol testing, evaluation, and verification.

The software development effort will proceed simultaneously along two avenues; the first being top-down specification and development of the three major system components and the second being bottom-up development of required software tools.

The initial version of the testbed has already proven itself as a valuable tool for protocol performance evaluation. The future version, with its added flexibility and capability, should prove even more so.

[3] A "generic" routine is one that has utility in any simulation (i.e. sin(), cos(), etc.).

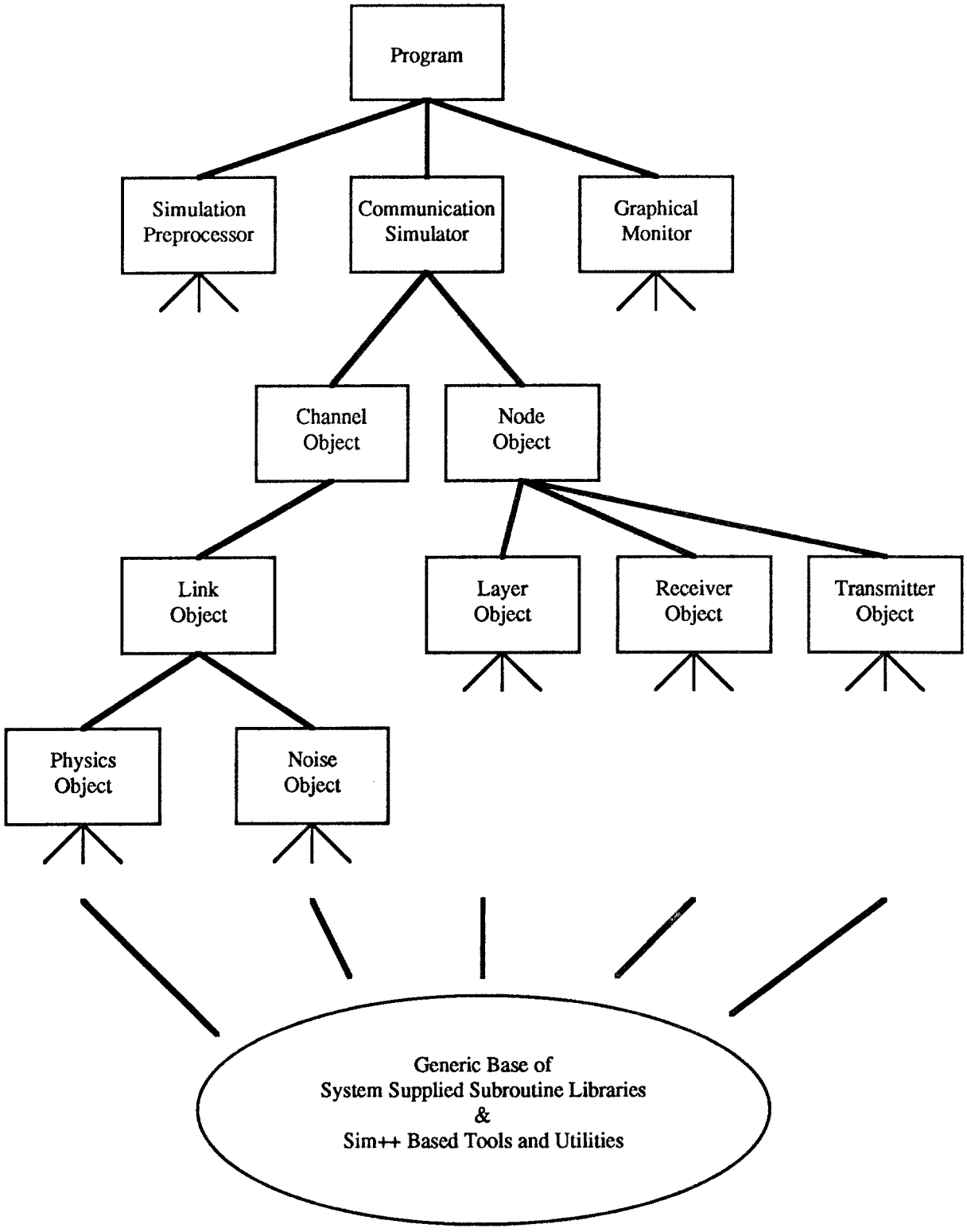


Figure 9: Software Hierarchy

References

[1] D. Bertsekas, R. Gallager, *Data Networks*, Prentice-Hall, 1987.

[2] E. Gafni, D. Bertsekas, "Distributed Algorithms for Generating Loop-free Routes in Networks with Frequently Changing Topology," *IEEE Trans. on Comm.*, January 1981.

[3] S. Corson, "A Distributed Routing Algorithm for Mobile Radio Networks and its Performance Evaluation," M.S. Thesis, University of Maryland, College Park, August 1989.