An Introduction to the Internet Networking Environment and SIMNET/DIS

by John Locke
(Internet: jxxl@cs.nps.navy.mil)
Computer Science Department, Naval Postgraduate School

1. THE PHYSICAL DESIGN OF THE INTERNET

The Internet is the world's largest data network. It was originally developed by Bolt Beranek and Newman, Inc. (BBN) in the late seventies under the auspices of the Defense Advanced Research Projects Agency (DARPA). The basic operation of the network is *packet-switching*--the sending host divides a block of data into packets, transmits each packet individually and the receiving host reassembles them. In between, the packets share the same network bandwidth with packets from other hosts. Note how this differs from a *circuit-switched* network like the phone system. During a phone call, caller and receiver share exclusive use of a range of the network bandwidth. So there is no overlap between the phone system and the Internet, the cabling is separate, the operation is different. However, the two systems do interface where Internet-connected systems have modems and provide services, like electronic mail transfer, to non-Internet hosts.

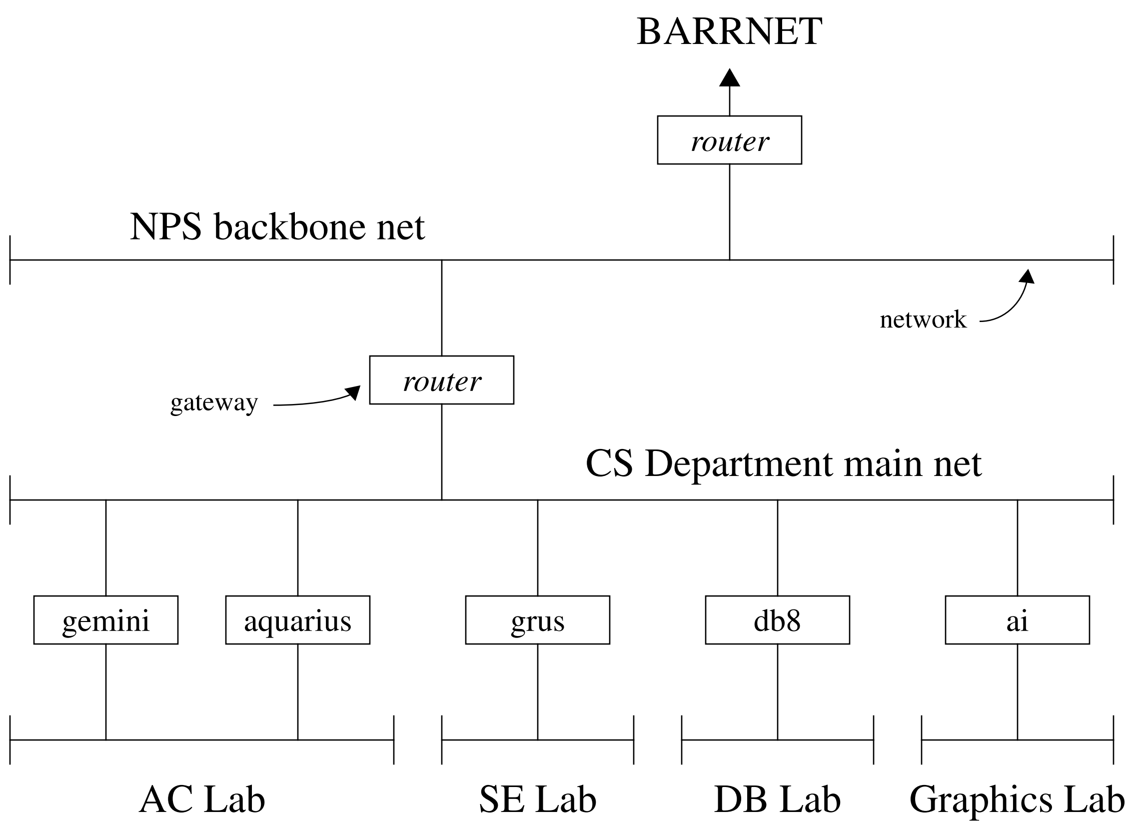
The Internet was so named because it drew together a number of separate networks into a single cohesive system. Among others, these included the MILNET, the NSFNET, and the original test bed, the ARPANET. Because of the Internet's distributed design, new networks can be added to the system with minimal effort. NPS, for example, is part of the Bay Area Research Network (BARRNET), which primarily services the San Francisco Bay Area, but extends as far south as Monterey and to Davis, California to the northeast. Within these networks are locally administered entities like NPS. In turn, we are subdivided into local-area networks (LANs) connected to a campus backbone network. Individual networks, large and small, are connected by gateways, often special-purpose hardware called *routers*, whose purpose is to transfer packets from one network to another, if it gets them closer to their destination.

The following figure illustrates the relationship between networks and gateways in the Computer Science Department. Note that because of the uniform design of the Internet, any part of it is a microcosm of the whole. The horizontal lines represent Ethernet cables, a common, but not required, LAN medium. To attach to an Ethernet, a host must have an Ethernet interface (card) and a short piece of cable that can tap into the Ethernet through a *transceiver*. Each interface has a 48-bit id number burned into its ROM by the manufacturer that is guaranteed to be unique, thus hosts can always be uniquely identified on a LAN. Hosts are uniquely identified *within the Internet* by a 32-bit number assigned by local system administrators, thus facilitating communication between any two hosts on the Internet. All NPS hosts are numbered within the 131.120.*xxx.xxx* block (dotted notation is standard).[1] In the figure, gateways have two Ethernet interfaces, each attached to a dif-

1. The Internet faces an impending shortage of numbers. Why? For example, NPS's address space is large enough for 2^16 hosts--most will go unused.

ferent network. Currently in the department, ordinary Sun servers with routing software act as gateways. The host marked *router* is a dedicated Cisco gateway with an Ethernet on its campus backbone side.

__**CS Department Gateways and Networks**__



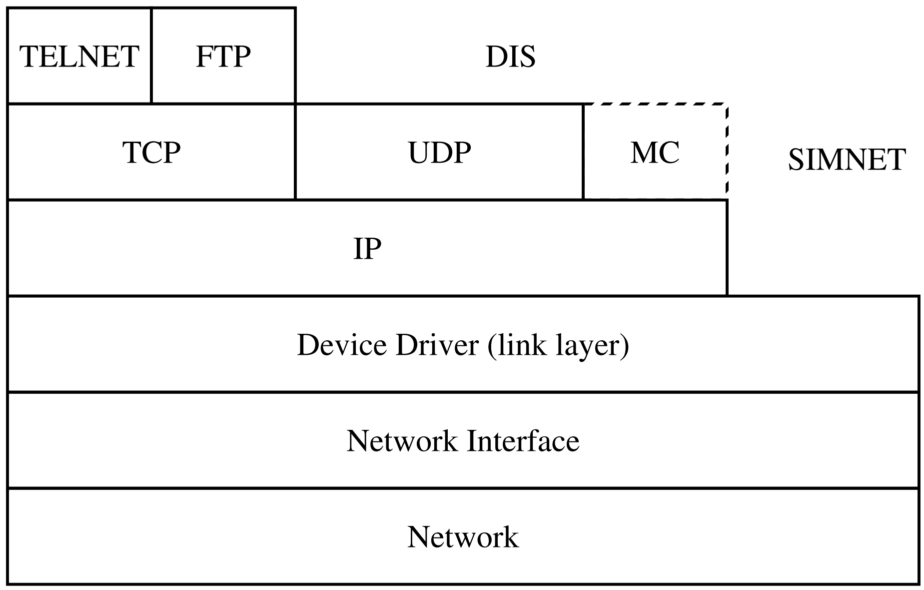
Ethernet hardware consists of a coaxial cable that provides a single communication channel and transceivers that send and sense signals. When a packet is put out on the cable, every host reads the data through its transceiver. The Ethernet card examines the destination address of the packet. If it is for that host, the packet is passed to the operating system, otherwise it is discarded. When two packets are put on the cable at once, so that their signals corrupt each other, the transceiver detects a *collision*. Both hosts wait a random interval and retransmit the packet. The maximum data rate Ethernet can handle is 10 Mbps. Practically speaking, optimal performance is at about 60% of that since at higher saturations the collision rate becomes too high to increase throughput.

2. THE INTERNET PROTOCOL STACK (TCP/IP)

A protocol can be defined as the format of the data that hosts impart to one another and the order that it is imparted in--how communication is initiated, how it proceeds, how it is terminated. As there are higher-to-lower levels of programming languages, higher levels satisfying broad concepts and lower levels accounting for hardware specifics, so there are higher-to-lower levels of protocols, with higher levels defined by user applications like file transfer and electronic mail, and lower levels managing network hardware. As higher level languages simplify application programming, so do higher levels protocols simplify network programming.

In the Internet environment, we're concerned with a set of protocol layers commonly called TCP/IP. The following figure illustrates the conceptual layering:

The Internet Protocol Stack and Network Layering



Going from bottom to top, here is a brief explanation of the layers:

- Network - the cable itself, and associated hardware.
- Network Interface - the Ethernet card, or some other host network device.
- Device Driver - the software component that handles direct operations on the Network Interface; in a UNIX system the driver is part of the kernel.
- Internet Protocol (IP) - manages the delivery and receipt of individual IP packets.
- User Datagram Protocol (UDP) - unreliable (not guaranteed) packet delivery between host processes; maximum message size is one IP datagram; often used for *broadcasting*, communication from one host to every other host on a LAN.
- Transmission Control Protocol (TCP) - reliable stream delivery of data with flow control for optimizing the use of available bandwidth; TCP breaks large blocks of data into sequenced datagrams, keeps track of acknowledged receipts, and retransmits lost or corrupted packets.
- Multicast (MC) - communication from one host to a subset of the hosts on a network; not yet an official standard, although it has been implemented in SGI's IRIX and Sun's Solaris operating systems.
- TELNET, FTP, etc. - application protocols that provide useful functionality to the network; the implementation of network services.

• SIMNET, DIS - simulation protocols; to be discussed in later sections.

3. HOST NAMES AND THE DOMAIN NAME SYSTEM (DNS)

The key identifier of hosts on the Internet is the IP address, e.g. 131.120.1.13. But since numerical identifiers are difficult to remember, hosts are given symbolic names. The name for 131.120.1.13, for example, is "taurus.cs.nps.navy.mil". That long name completely locates host taurus within the *Domain Name System*, a distributed database application that, primarily, provides host name to IP address translation. Do not be misled by the dotted notation, though. The DNS provides a logical grouping of names; there is no physical location implied or routing data provided by DNS. At NPS, our host names and physical subnets do correspond, providing a parallel between the naming and numbering systems, but that is only because our facility is not geographically dispersed. If we had offices in another part of the country, our host names could continue to have logical consistency even though the hosts are physically distant on the network.

There is a lot of confusion about what DNS does and doesn't do. When you use a host name with a command, like "ftp taurus.cs.nps.navy.mil", before doing anything else the command queries the DNS and receives, if successful, an IP address for the host. All interaction with the host is done on the basis of the IP address. If the command returns the message "Host unknown" the address resolution has failed; any other problems with connecting to the host are likely not the fault of DNS. For example, the address can be resolved, but if the host is down no connection can be made. The sure test to find out whether DNS is failing is to use the command with an IP address, like "ftp 131.120.1.13". If it connects where using the name did not, then DNS is the culprit.

4. INTERPROCESS COMMUNICATION (IPC)

It is too broad to think of hosts talking to each other; as a practical matter, the machines have nothing interesting to say. The substantive communication goes on between *processes*. A process, of course, is a program that is running, a useful distinction on a multi-processing machine. A process can communicate with another process on the same host, or with a process on another host, in which case a network links the two.

Various modes of interprocess communication--pipes, for instance--have long been part of UNIX. TCP/IP functionality was first added to UNIX at UC-Berkeley and is now a standard inclusion in UNIX-based workstations. TCP/IP is not a program proper, but a collection of code compiled into the kernel. Like any system resource, programmers access TCP/IP through the use of either system calls or library routines (which invoke system calls).

Basic networking programming is surprisingly easy in a UNIX environment. To open a channel of communication, the programmer must specify whether the processes to communicate will be on the same or different hosts and, if on different hosts, whether TCP or UDP is preferred. The result of this initialization is to open a *socket*, defined as a special case of a file descriptor[2], an end-point of communication. Once obtained, data is transmitted simply by using the system calls for writing and reading, as if the socket were an ordinary data file. From that point, the programmer's task is to implement some higher-level protocol, i.e. I connect to you, I offer you data and wait for your reply, you accept the offer and wait for the data, I send the data, you acknowledge successful transmission, I terminate the connection.

Note that on a multi-processing machine, the hardware or IP host address is insufficient for channeling data to its final destination. Therefore, we use the concept of *port number*s to identify communicating processes. (It should not be confused with hardware ports that are used in low-level programming.) Port numbers from 0 to 1023 are reserved for well-known services[3], e.g. TELNET connections are always made on Port 23. Along with the protocol definition, this allows anyone to implement network applications for common services. Port numbers from 1024 and above are available for user applications, and must be agreed upon by programmers designing separate applications around a common protocol.

5. SIMNET

5.1. General Description

SIMNET is a standard for distributed interactive simulations developed under DARPA auspices beginning in 1985; functional systems were running within three years. A few words on terminology: A computer simulation could model any real-world activity; SIMNET, not surprisingly, is oriented toward entities on a battlefield (tanks, planes, projectiles, etc.). *Distributed* means that processing of the simulation can take place on different hosts on a network. *Interactive* means that the simulation can be dynamic (as opposed to scripted), guided by human operators. A simple but representative simulation might be two graphics workstations, both displaying a single battlefield from different points of view, both allowing human operators to guide their own tanks. SIMNET, of course, is more ambitious--a realistic simulation would include thousands of entities; controlling input for the entities could come from human operators, the computer itself, or even actual entities transmitting their locations as they move.

The purpose of SIMNET is to facilitate early phases of training at a cost far below the expense of operating real vehicles or conducting real battlefield exercises. The range of SIMNET applications goes from training tank drivers with a console that models the interior, controls, and external views provided in a real tank to tactical planning sessions for battlefield commanders. The role of the Computer Science Department is to use a standardized simulation environment as a test bed for study in areas such as graphics and artificial intelligence so that our research might ultimately serve the practical needs of the Department of Defense.

2. UNIX treats all devices or accessible hardware as files to allow uniformity in programming. Device drivers implement the specifics of opening, reading, writing, and performing other basic operations on the "file."

3. Protocol standards, including port numbers, are defined by Requests for Comments (RFCs), the official Internet design documents.

5.2. SIMNET Protocol Data Units (PDUs)

Having said all that, we can now refer to SIMNET as a network protocol, knowing that it is much more. As a practical matter, the data shared among hosts in a SIMNET simulation is defined by the protocol standard. A host can only know what it is told. If a vehicle controlled by one host moves or shoots, that information must be transmitted on the network so that all the other hosts can keep their displays, or at least their state, current. The SIMNET standard defines Protocol Data Units (PDUs), each of which is the format of the data in the packet describing a type of event. Following is a list of all the SIMNET PDUs [POPE]:

PDU	Purpose
Activate Request	- Introduce a vehicle to the simulation
Activate Response	- Accept or reject Activate Request
Deactivate Request	- Withdraw a vehicle from the simulation
Deactivate Response	- Accept Deactivate Request
Vehicle Appearance	- Update vehicle state, i.e. location, appearance, operational status
Radiate	- Report use of radar
Fire	- Report firing of projectile
Impact	- Report impact of projectile
Indirect Fire	- Report impact of indirect fire from projectile
Collision	- Report of collision by moving vehicle
Service Request	- Request transfer of munitions, fuel or ammo, from supplier
Resupply Offer	- Transfer munitions to requesting entity
Resupply Received	- Acknowledge receipt of some or all of offered munitions
Resupply Cancel	- Abort transfer of munitions
Repair Request	- Request repair
Repair Response	- Report completion of repair
Minefield	- Describe emplaced minefield
Breached Lane	- Describe path cleared through minefield
Marker	- Announce marking of minefield with flags

Within each PDU are a number of fields which take actual or coded values. Actual values consist of location coordinates, entity speed, or somesuch; coded values consist of booleans or defined constants to describe state conditions.

A simulation that thoroughly implemented all of the PDUs and all of the implied entity states would be very complicated (as well as creating an obvious performance bottleneck on current processors). A practical simulation implements a relevant subset of the PDUs. A simple simulation could be built around just the Vehicle Appearance PDU (activation would be implied by the first PDU for a vehicle).

5.3. Initial State of the Simulation

The above PDUs allow the developing events of a simulation to be shared among hosts. However, they do not describe the initial state of the simulation. This must simply be agreed upon by simulation participants beforehand. The most important part of the initial state is the terrain. This is a database of elevation measurements for the grid points of some real-world area. There tends to be a few of these databases that people use such as Ft. Hunter-Liggett, the National Training Center, Germany's Fulda Gap, or Northwestern Iraq.

The dead reckoning algorithm is another factor that must be known up front. It should be readily apparent that constantly updating a moving vehicle's position would consume a lot of network bandwidth. The solution is for hosts to *dead reckon* the position of vehicles controlled by other hosts, that is, to take the vehicle's speed and direction and calculate its likely position. The controlling host's responsibility, then, is to put out a Vehicle Appearance PDU when the vehicle's speed or direction changes. Because of the vagaries of terrain and atmosphere, dead reckoning is only accurate to some degree. Hosts must therefore dead reckon their own vehicles and put out a Vehicle Appearance PDU when the dead reckoned position diverges too much from the actual position. The reason for initial agreement is that there are a number of dead reckoning algorithms, from simple to complicated, each reflecting some trade off between accuracy and computational load. But all hosts in a simulation must be using the same algorithm.

Other items to agree upon are the specific vehicles participating in the simulation and the way they are identified. Since a practical simulator implements a subset of the standard capabilities, then obviously participating hosts with different software must closely accord on capabilities in use. (Note that there is wide latitude, not specified by SIMNET, in the potential operation or quality of a simulation. Many issues are left in the hands of implementors. Graphical simulation display is dependent on hardware quality and the degree of realism desired, or afforded by CPU throughput. Methods of automating vehicles, whether through human or software guidance, is similarly open.)

5.4. Tools

SIMNET does not require the use of specific network hardware or lower-level protocols, although it does define certain performance attributes such as minimum throughput and reliability. Because Ethernet is so commonplace, SIMNET recommends an implementation. It uses the 48-bit Ethernet destination address as a multicast address, with certain bit fields identifying the multicast group and the simulation exercise. Unfortunately, this technique prevents the use of TCP/IP, but then TCP/IP currently has no multicast facility. The problem for us was to find some way of bypassing TCP/IP, manipulating the hardware destination address, and reading and writing packets to the Ethernet. Ultimately, different solutions were found for SGI and Sun systems owing to different facilities which happened to be in their respective UNIX implementations [LOCKE Mar92].

SGI implements IP Multicast, but that uses a special block of the 32-bit IP host addresses and is therefore incompatible with SIMNET's recommendations for Ethernet. However, another SGI enhancement, *snoop*(7p), a network monitoring protocol, provides promiscuous packet capture, treating packets as datagrams containing a link-layer header followed by data. Since the header encompasses the destination Ethernet address, the SIMNET-specified multicasting address can be examined or set by the user program, thus providing read and write capability. SGI also provides shared memory between processes through the use of an *arena*. Combining the snoop protocol with the shared memory capability suggested an architecture for a SIMNET network library. A daemon process monitors the network for SIMNET packets and maintains incoming packets on a queue in the arena. Client applications read PDUs on the queue and add outgoing PDUs. Client implementors do not need to understand the underlying scheme because a client library does all the work. The implementor simply manipulates the queue with calls like *net_read()* and *net_write()*, passing a PDU structure. The software works reliably but is hindered by the overhead of using an arena. Additionally, the daemon had to run with root access to read the network at the link layer; because of the limitations of the shared memory implementation, clients had to run as root to access the arena.

The Sun implementation was completely different (although it provides an identical interface). Sun provides a driver called a Network Interface Tap (NIT) that allows link layer access to the network. Programmers at Columbia have developed a library of reliable access routines for the NIT, and this was employed. As with the SGI implementation the keys were the ability to manipulate the Ethernet destination address, and to read and write packets. Since there is no shared memory facility on the Suns, all capabilities were embodied in a client library. The defects in the implementation are serious. An interrupt handler could not be associated with incoming packets as they can with the equivalent TCP/IP drivers. Consequently, the client library itself cannot provide queuing in parallel with the operations of the client, and the client must poll using *net_read()* at the rate of incoming messages or lose the messages. The implementation is therefore of little practical use.

6. DISTRIBUTED INTERACTIVE SIMULATION (DIS)

6.1. General Description

DIS is a newer simulation standard than SIMNET. Many aspects of it are currently under development and refinement. DIS shares its goals and purposes with SIMNET, but is far more ambitious, allowing for greater complexity and realism. Examples: SIMNET uses a flat terrain; DIS accounts for the curvature of the Earth. SIMNET is oriented towards terrain and the sky above it; the DIS world encompasses all areas of potential military activity--earth, atmosphere, above and below the surface of the ocean, and space.

6.2. DIS Protocol Data Units (PDUs)

DIS defines its own set of PDUs, of which a number have SIMNET counterparts.

PDU (official)	Purpose
Entity State	- Update vehicle state, i.e. location, appearance, operational status
Fire	- Report firing of weapon
Detonation	- Report impact or detonation of munition
Collision	- Report collision of entity with another object

Service Request	- Request transfer of supplies
Resupply Offer	- Offer supplies to another entity
Resupply Received	- Acknowledge receipt of some or all of offered supplies
Resupply Cancel	- Cancel resupply service
Repair Complete	- Report completion of repair
Repair Response	- Acknowledge completion of repair

6.3. The Entity State PDU

To give a flavor of what a PDU contains, following is the Entity State PDU v2.0 and its fields (in C syntax) [IST Oct91]. Many of the fields are themselves structures, but the contents of these are not shown.

```
typedef struct {
   PDUHeader                entity_state_header;
   EntityID                 entity_id;
   ForceID                  force_id;
   unsigned char            num_articulat_params;
   EntityType               entity_type;
   EntityType               alt_entity_type;
   VelocityVector           entity_velocity;
   EntityLocation           entity_location;
   EntityOrientation        entity_orientation;
   unsigned int             entity_appearance;
   DeadReckonParams         dead_reckon_params;
   EntityMarking            entity_marking;
   unsigned int             capabilities;
   ArticulatParams          articulat_params[MAX_ARTICULAT_PARAMS];
} EntityStatePDU;
```

6.4. Tools

Unlike SIMNET, DIS does not require any non-standard network coding for Ethernet. PDUs are broadcast (UDP/IP) onto the LAN. Reflecting on the experience of the SIMNET implementation, it was decided to make a UNIX-portable DIS network library [LOCKE Oct92]. Sun and SGI both fully support the BSD interface to networking facilities, so the package only has to account for a few minor compilation differences between the two systems. As with the SIMNET implementation, a simple interface employing calls like *net_read()* and *net_write()* is used. PDUs are written immediately; incoming packets trigger an interrupt handler which maintains a queue of PDUs. A call to *net_read()* returns the oldest valid PDU on the queue.

After experience with a UNIX-portable DIS network library, we decided to eliminate linked lists to prevent memory allocation contention. The new architecture of the library has two circular PDU buffers. New PDUS from the net fill one buffer; *net_read()* consumes PDUs from the other. When *net_read()* has exhausted its buffer, and under other circumstances, the buffers are swapped.

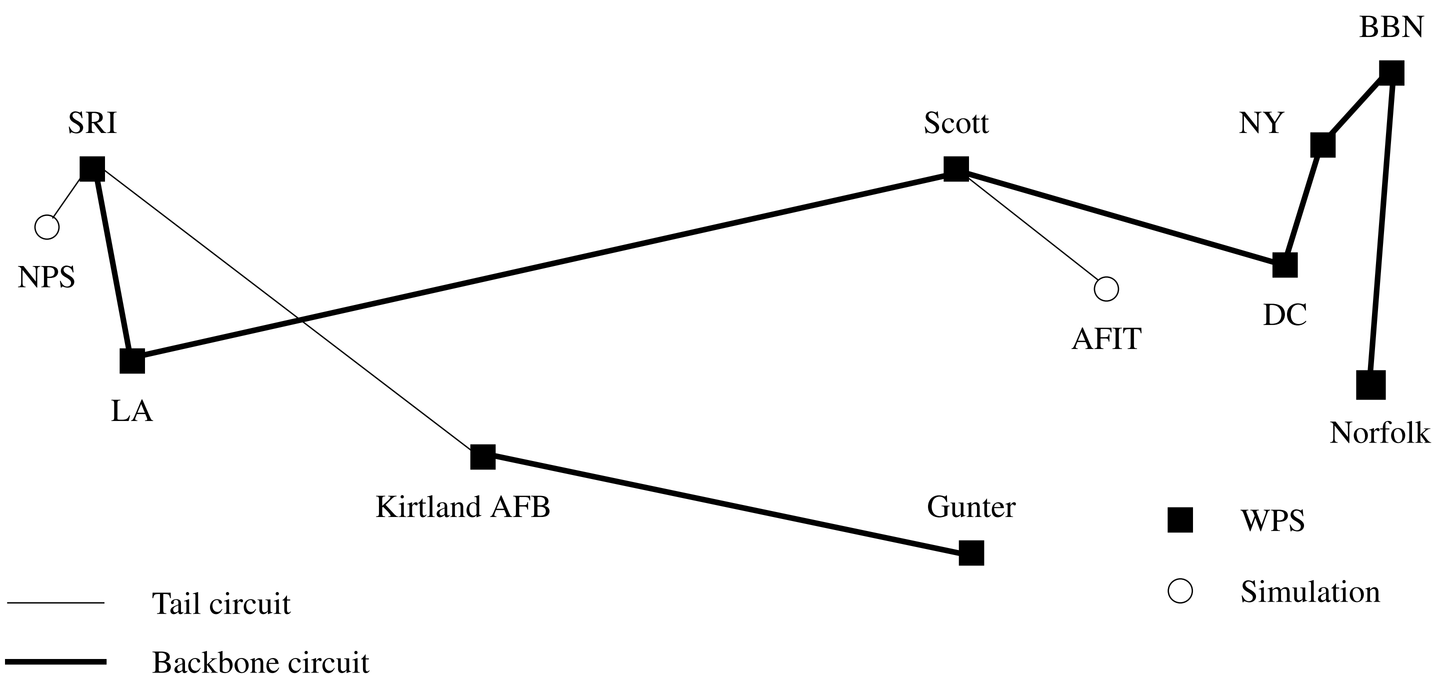
6.5. The Semiannual DIS Workshop

Twice yearly (March and September), the weighty issues of DIS development, from what data should go into a PDU to whether it can ever work, are discussed in Orlando, Florida. Attendees include about three hundred individuals from government, industry, and academia. Anyone wanting to know more about DIS or would like to help guide its development are encouraged to attend. Representatives from the Computer Science Department often attend and have presented a number of papers.

7. THE DEFENSE SIMULATION INTERNET (DSI)

BBN, under DARPA funding, has been rapidly building a dedicated nationwide simulation network called the DSI. NPS is one of approximately a hundred sites currently on-line, although the number is steadily rising. It runs TCP/IP traffic but is not considered part of *the* Internet. It works in a slightly different way. Use of the DSI is governed by the Network Operations Center (NOC) at BBN in Cambridge. Parties desiring to run a simulation schedule bandwidth on the network; they then have exclusive use of the bandwidth for the period scheduled. This avoids the network saturation that would result from multiple simultaneous simulations.

The DSI Backbone and Selected Simulation Sites (as of 4 Jun 93)



The mechanics are as follows. DSI sites have a BBN T/20 gateway attached to a local Ethernet cable. Hosts on the Ethernet broadcast PDUs. The T/20 reads the packets and multicasts them on the DSI where they will be received by all other gateways, particularly those involved in the simulation.

8. Performance Issues

The viability of the DIS plan for large-scale simulations is a key controversy. The long-term goal is to conduct simulations with up to 10,000 entities. This raises serious questions. Will network bandwidth be able to handle the volume of traffic representing all the events that must be shared among hosts? Will processors be able to keep up with the demands of the simulation, which includes dead reckoning every mobile entity in addition to graphics generation and all other processing? Where will the bottleneck be?

We must also remember that actual DoD simulations may be classified since they reveal information about vehicle capabilities and tactical planning. These simulations must be run on secure networks. PDUs must be encrypted on the sending end and then decrypted at the receiving end. This is another potentially debilitating bottleneck (although not one that will affect our unclassified research in the Computer Science Department).

But DIS is a long-term project. Its ultimate success will depend on dramatic progress being made in the performance of processors and networks.

9. REFERENCES

SECHREST, STUART [August 1986], *An Introductory 4.3BSD Interprocess Communication Tutorial*, UNIX Programmer's Supplementary Documents, Volume 1 (PS1:7)

LEFFLER, SAMUEL J., et al. [August 1986], *An Advanced 4.3BSD Interprocess Communication Tutorial*, UNIX Programmer's Supplementary Documents, Volume 1 (PS1:8)

COMER, DOUGLAS E. [1988], *Internetworking With TCP/IP - Principles, Protocols, and Architecture*, Prentice-Hall, Englewood Cliffs, New Jersey

LEFFLER, SAMUEL J., et al. [1989], *The Design and Implementation of the 4.3BSD UNIX Operating System*, Addison-Wesley, Reading, Massachusetts

STEVENS, W. RICHARD [1990], *UNIX Network Programming*, Prentice-Hall, Englewood Cliffs, New Jersey

POPE, ARTHUR R. and SCHAFFER, RICHARD L. [June 1991], *The SIMNET Network and Protocols*, BBN Systems and Technologies Corporation Report No. 7627

IST [October 1991], *Military Standard: Protocol Data Units for Entity Information and Entity Interaction in a Distributed Interactive Simulation*, Institute for Simulation and Training, Orlando, Florida

IST [January 1992], *Rationale Document: Entity Information and Entity Interaction in a Distributed Interactive Simulation*, Institute for Simulation and Training, Orlando, Florida

IST [February 1992], *Standards Development Guidance Document*, Institute for Simulation and Training, Orlando, Florida

IST [February 1992], *Distributed Interactive Simulation Operational Concept*, Institute for Simulation and Training, Orlando, Florida

ZYDA, MICHAEL J., PRATT, DAVID R., MONAHAN, JAMES G. and WILSON, KALIN P. [February 1992], *NPSNET: Constructing A 3D Virtual World*, paper in Proceedings of the 1992 Symposium on Interactive 3D Graphics

LOCKE, JOHN, PRATT, DAVID R. and ZYDA, MICHAEL J. [March 1992], *Integrating SIMNET with NPSNET Using a Mix of Silicon Graphics and Sun Workstations*, paper presented to The Sixth Workshop on Standards for the Interoperability of Defense Simulations

NEVIN, BRUCE [May 1992], *T/20 Internet Packet Router (IPR) Operations Guide*, BBN Communications Operations Series

LOCKE, JOHN, PRATT, DAVID R. and ZYDA, MICHAEL J. [September 1992], *A DIS Network Library for UNIX and NPSNET*, paper presented to The Seventh Workshop on Standards for the Interoperability of Defense Simulations

WETZEL, REBECCA, et al. [October 1992], *The Defense Simulation Internet: User's Guide*, BBN Systems and Technologies Corporation Report No. 7765

IST [various], Summary Reports of the Workshops for the Interoperability of Defense Simulations, Institute for Simulation and Training, Orlando, Florida