Network Performance Requirements for Real-Time Distributed Simulation

Duncan C. Miller, Sc.D.
dmiller@ll.mit.edu

Abstract

The use of large-scale, real-time, multi-player simulations is growing rapidly, and is becoming an important network application. This paper provides some historical background on how such simulations emerged within the U.S. Department of Defense, outlines the architectural principles on which they are based, and identifies some of the network services and performance characteristics required to support very large scale applications.

SIMNET is a real-time, soldier-in-the-loop battlefield simulation developed by ARPA, in partnership with the U.S. Army, between 1983 and 1990. This program demonstrated that it is possible to link hundreds or thousands of simulators together to create a consistent, virtual world, in which all participants experience a common, logical sequence of events. In this world, the causal connections among these events, from the individual crew station to the battalion command post, are clear and easily inspectable. The SIMNET architecture and protocols have evolved into the Distributed Interactive Simulation (DIS) Standard Protocols (IEEE 1278-1993).

Under policies recommended by the Defense Science Board and now being adopted by the U.S. Department of Defense, these simulation techniques are being introduced into every stage of DoD system development, doctrinal development, training, and rehearsal for field operations. Military agencies in other countries, including Germany and the United Kingdom, are exploring the use of this technology. Non-military agencies, such as the U.S. Federal Aviation Administration, are also beginning to adopt DIS protocols.

I. A Brief History of SIMNET

The Distributed Simulator Networking (SIMNET) program was initiated by DARPA in 1983, with substantial support from the U.S. Army. The emphasis of the program from the outset was on tactical team performance: it was assumed that crew members were already proficient in their individual specialties.

After the basic design philosophy and system architecture were developed , the first conceptual demonstration was conducted in December,1984. The first demonstration involving real-time, out-the-window graphics was conducted in November, 1985. The first platoon-level system, incorporating custom image generation capabilities designed to optimize performance in an environment with large numbers of moving objects, was installed at Ft. Knox, Kentucky, in April, 1986. The first helicopter simulators were installed at Ft. Rucker, Alabama, in late 1987.

At the conclusion of the program, a suite of approximately 250 simulators, installed at nine operational training sites and two developmental sites, were transitioned to the U.S. Army. Two mobile platoon sets, installed in containers mounted on flatbed trailers, were delivered to the Army National Guard.

The communications protocols by which the simulators transmit essential information to one another are now incorporated in the Distributed Interactive Simulation (DIS) Standard Protocols (IEEE 1278-1993) [1]. The next generation of this standard is currently under development. The next DIS semi-annual workshop will be held in September, 1994, in Orlando, Florida.

II. Key Design Principles

The success of the SIMNET approach [2], and the fact that all of its essential elements have been adopted into the DIS draft standard after intense scrutiny by a highly competitive industry, results from a few key design principles and architectural decisions that were made early in the program and were progressively refined. These principles are summarized in the following paragraphs.

II.A. Object/event architecture

The first key decision was to model the world as a collection of objects, which interact with each other through a series of events. The basic terrain and cultural objects (buildings, bridges, etc.) are assumed to be known to every other object. This paradigm permits the possibility of an event that changes one or more of these objects, e.g., blowing up a bridge, or digging a pit at a certain location, though these capabilities were not implemented under the SIMNET program.

II.B. Autonomous simulation nodes

All events are broadcast on the simulation network, and are available to all objects that may be interested in them. The simulation nodes that initiate an event do not need to keep track of what other nodes may be affected by that event; these calculations are the responsibility of the receiving nodes.

In SIMNET (and DIS), there is no central control process that schedules events or resolves conflicts among contradictory versions. Instead, each simulation node is completely autonomous; the communications algorithms even permit a simulation node to join or leave an exercise in progress without disrupting the interactions among the other nodes.

Each node is responsible for maintaining the state of at least one object in the simulated world, and for communicating to other nodes any events caused by its object(s). Each node is also responsible for receiving event reports from other nodes, and calculating the effects of this event on the object(s) they are simulating. If the effects cause other events to occur, then the node is responsible for notifying others of these events.

II.C. Transmission of "ground truth" information

Each node transmits absolute truth about the current state of the object(s) it represents and any events they have caused. It is the responsibility of the objects receiving an event message to determine whether they are able to perceive the event and whether (and how) they are affected by it. When information needs to be degraded before presentation to a human crew or an automated crew (e.g., on a radar display), it is the responsibility of the receiving objects to calculate and introduce this degradation.

II.D. Transmission of state change information

To minimize communications processing, nodes transmit state update information only when the object(s) they represent *change* their behavior. A change of behavior constitutes an event that may be of significance to other objects. This approach minimizes repetitive transmission of redundant information, which substantially reduces the processing load on other nodes.

II.E. "Dead reckoning" algorithms

Between state update messages, receiving nodes extrapolate the last reported states of remote objects that are of interest to their local object(s), and use this information in generating displays for human crews or detection probabilities for automated crews. The sending vehicles are responsible for generating a new state update message before discrepancies in the remote extrapolations become unacceptably large.

In effect, this algorithm depends on a "contract" among the nodes. Each node guarantees that it will transmit a state update event within one-fifteenth of a second of the time that the true position and orientation of any object(s) it represents diverges from the calculated values (based on an extrapolation of the last reported state update information) by more than an agreed-upon threshold. Obviously, this algorithm requires that each node maintain a dead reckoning model that corresponds exactly to the model(s) being used by the remote nodes.

Since each state update includes corrected position, as well as velocity and heading information, the dead reckoning algorithm is essentially self-healing. A node that fails to receive a state update message will, at worst, continue to extrapolate the previous state of a remote object for a few additional seconds. If the remote object continues to change its behavior, it will generate a burst of updates, and a new message is highly likely to arrive within a fraction of a second. This new message will correct any error that has accumulated and will initialize a new extrapolation.

In SIMNET, simple position, orientation, and velocity information was used for all vehicles. Manned vehicle simulators recalculated state information 15 times per second. Actual update transmission frequencies averaged 1 per second for ground vehicles and 3 per second for air vehicles, although individual vehicles often transmitted 15 updates per second during periods of intense activity. Tradeoff studies showed that the use of second derivative state variables reduced transmissions for air vehicles to about 1 per second as well [3].

III. Types of Simulations

Three principal types of simulations are incorporated in the SIMNET/DIS architecture. In addition, other nodes are used to support after-action review and data analysis.

III.A. Manned vehicles

Manned vehicle simulations are intended to provide realistic control/display interactions for each crew member. For example, an M1 tank simulator provides controls and out-the-window displays for the tank commander, gunner, loader, and driver. An

AH-64 helicopter simulator provides controls and displays for the pilot and the co-pilot/gunner. Each manned vehicle crew must navigate, avoid obstacles, maintain formation, detect and identify targets, employ weapons, and communicate via radio and other devices, just as they would on the battlefield.

III.B. Command post simulations

Command post simulations focus on decision-making and resource allocation, rather than on moving and shooting. They do not require out-the-window displays or detailed control/display interactions. Typically, command post simulations involve radio/telephone communication, a map display, and a workstation that supports the types of decisions and inputs that are normally made by each functional position. A Fire Support Officer, for example, uses a display that shows the location and strength of his artillery batteries, his remaining ammunition supply, and currently scheduled fire missions. These missions can be interrupted or modified, and new missions can be added, as circumstances require. A logistics support officer has a limited number of fuel and ammunition carriers, which he can dispatch to selected locations to rendezvous with and resupply units that are running low on expendables.

III.C. Semi-automated forces (SAF)

Semi-automated simulations are designed to realistically mimic the externally visible behavior of opposing or supporting forces without requiring large numbers of manned simulators and personnel to operate them. A human commander exercises supervisory control over units that may include dozens of vehicles. These multi-vehicle simulations broadcast the same state update messages as manned simulators, so that their behavior is indistinguishable to the other simulators and (usually) to their crews. The semi-automated simulations of vehicles and small units are intelligent enough to provide basic route planning, obstacle avoidance, formation keeping, line-of-sight detection of nearby vehicles, target engagement, and so forth.

The human commander provides the goals and objectives for his subordinate units via input forms modeled after standard field orders and graphic overlays. He continually monitors the forces under his control, and can intervene to redirect their activities at any point. He may temporarily assume direct command of a lower-level subordinate unit (e.g., a tank platoon) in order to exercise finer control over their actions.

The SAF commander's workstation consists of (1) a task organization display, showing how the units currently under his command are organizationally related to each other, (2) an operations display for composing orders and requests for information, (3) a message log display, which displays recent radio reports from the units under his command, and (4) a military map display on which the latest position reports, enemy contact reports, etc., are automatically posted.

The SAF commander also maintains radio contact with command post staff and commanders of fully manned units.

III.D. Other simulation nodes

Other key elements in the simulation network are designed to observe exercises in progress and/or to collect data for later analysis and replay. The most important of these elements are the data collection and analysis system and the Observation Vehicle (or "Flying Carpet").

The data collection and analysis system captures, time stamps, and records every event message transmitted by every node. Using the timestamp information, any portion of any exercise can be replayed onto the network at a later time, and any simulator can be used to drive or fly freely throughout the battlefield, seeing every event that one would have seen at the time the exercise was conducted. In effect, this permits a "time travel" capability – the ability to view perfectly reconstructed events, which are not affected by the presence of the time-travelling simulator. The data, once recorded, can also be analyzed by a suite of statistical analysis tools to prepare reports regarding what took place.

The "Flying Carpet" is a simulator that provides both a situation (map) display and an out-the-window view of the battlefield. It is invisible to other simulators, and so can be used to observe an exercise in progress without affecting it, as well as for "time travel." The Flying Carpet can be attached to, and "towed" by any selected vehicle, so that the behavior of that vehicle can be monitored in detail.

IV. An example of DIS interaction

To appreciate the kinds of cooperative interactions involved in DIS simulations, consider a simulator that includes the following software processes**: (Refer to Figure 1)**

(a) A network interface, that sends and receives DIS PDUs on a Local Area Network (LAN).

(b) An "other vehicle state table" that records entity state information from other simulation nodes and carries out dead reckoning extrapolations based on their last reported states.

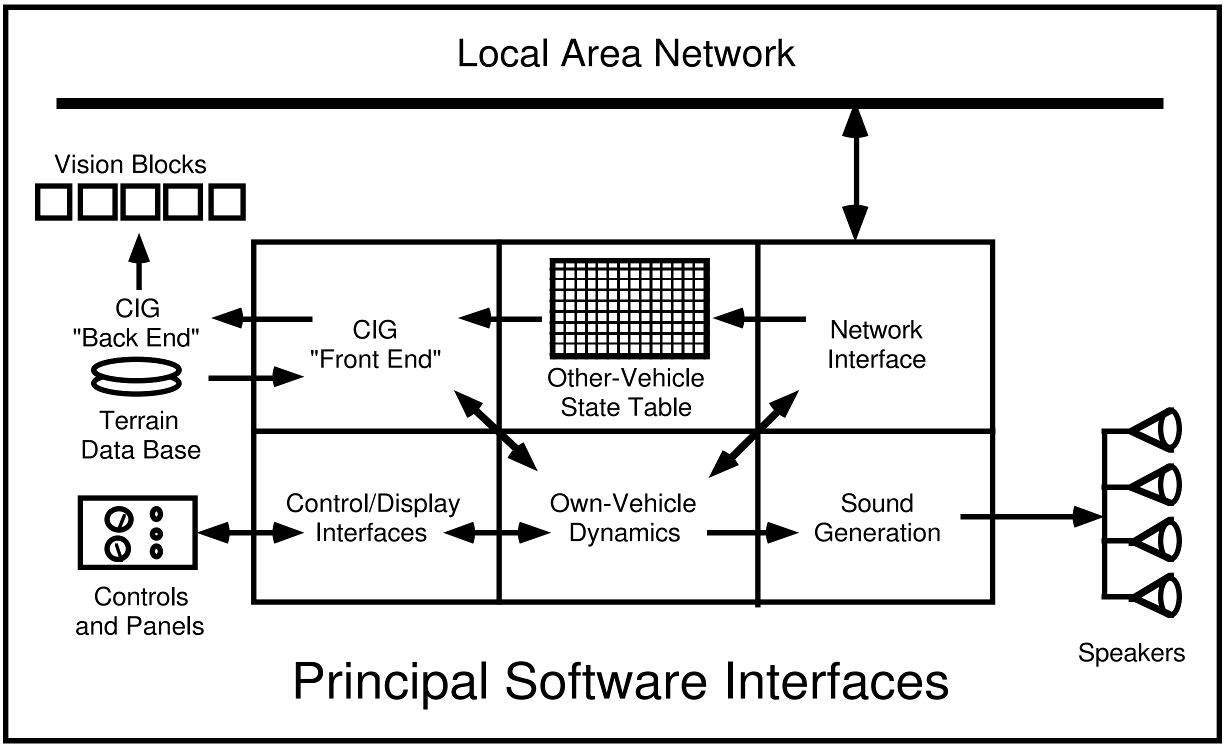

Figure 1 Software Modules in a Typical Simulator

(c) A Computer Image Generator (CIG) front end, which combines the non-changing information from the local copy of the terrain data base and the dynamic information from the other-vehicle state table, calculates the potentially visible terrain and dynamic object polygons, and transfers them to special-purpose CIG back end hardware. This specialized hardware filters, clips, rotates, and scales visible polygons, applies calculated texture patterns, and sends them to the proper vision block displays for viewing by the crew members.

(d) An own-vehicle dynamics model that computes (in whatever detail is necessary) the state of the entity being simulated.

(e) A control/display interface process that reads control positions and drives meters, gauges, and similar displays.

(f) A sound generation process that drives a set of speakers within the simulator.

Now imagine two such simulators, and let's step through the sequence of events that occurs when simulator A fires at, and hits, simulator B.

(Refer to Figures 2 and 3).

(1) First, the control/display interface detects that the gunner has pulled the trigger.

(2) The sound generation software is notified that a local main-gun sound effect is required.

(3) The image generator front end is notified that a local muzzle flash effect is required, which will obscure the gunner's and commander's vision blocks for a second or so.

(4) The network interface process is notified that an event has just occurred that has changed the vehicle's appearance, and a new Entity State PDU is transmitted onto the network. This PDU is received by all other simulation nodes, and the updated state information is transferred into the other-vehicle state tables.

(5) During the next CIG frame recomputation, the updated state information for simulator A is incorporated, and (unless some intervening object blocks his view) any crew member looking in the direction of simulator A will see a tank with a large fireball at the end of its gun tube.

Note that at this point no calculation has yet begun as to where the round that was fired is going. We will assume for the purposes of this explanation that the round follows a predictable ballistic trajectory. For a guided munition, the story is more complicated and beyond the scope of this paper.

Well in advance of the simulation, flyout trajectories have been computed for each type of ammunition that can be fired by a given simulator.

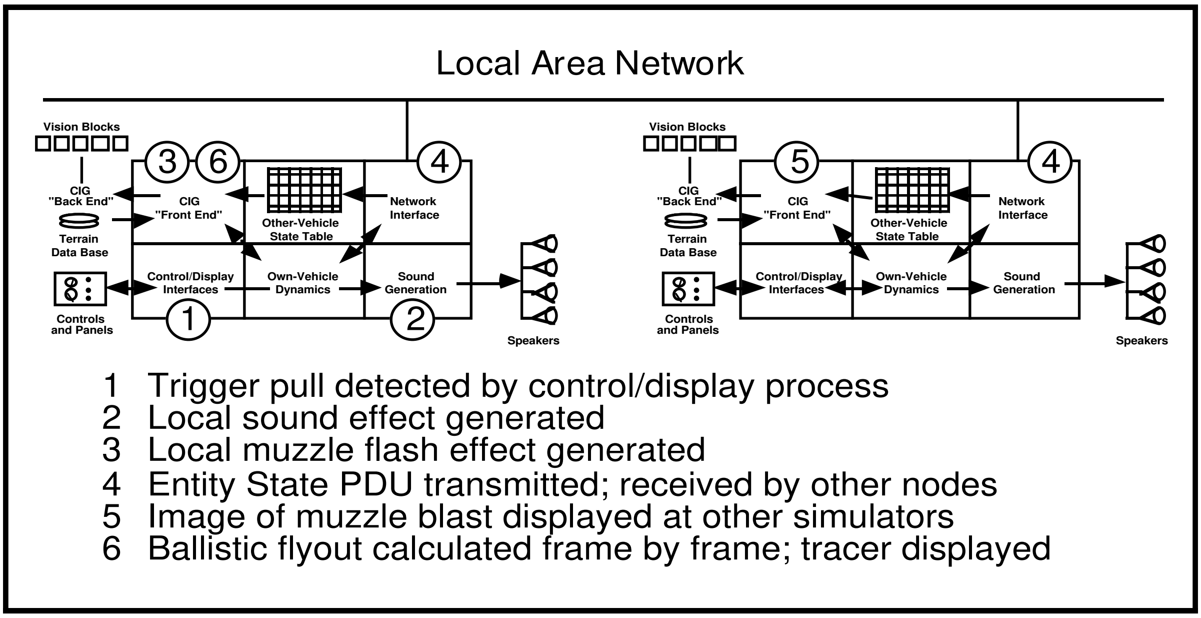

Figure 2 Typical sequence of events when Simulator A fires at Simulator B

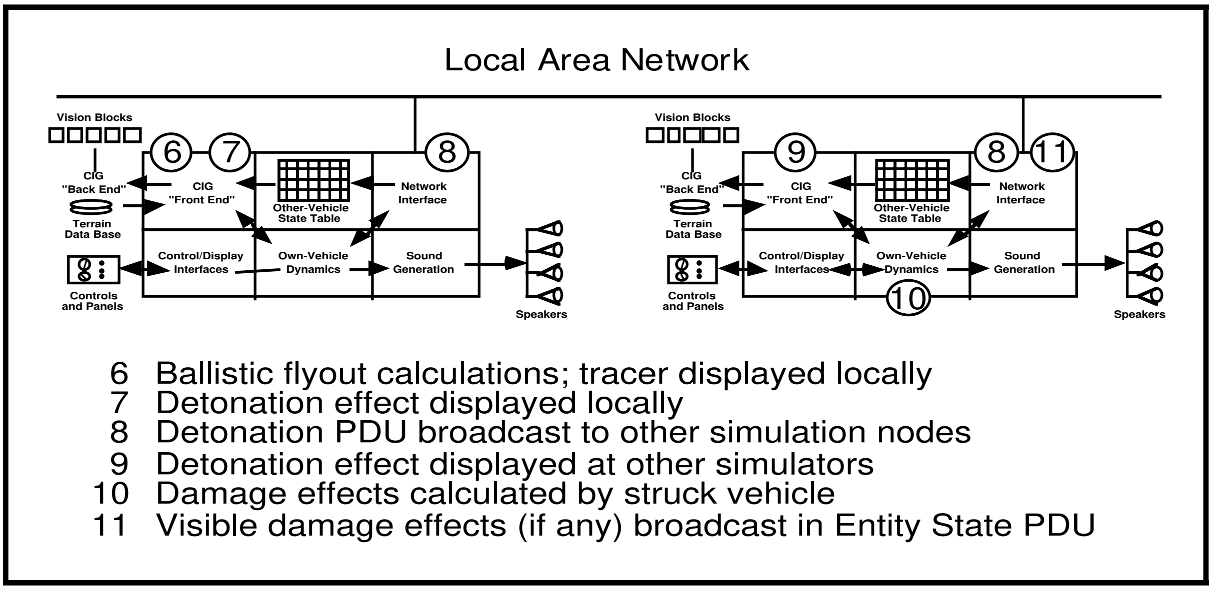

Figure 3 Typical sequence of events (continued)

Each flyout trajectory is then divided into short chord segments that correspond to the distance traveled by the projectile within one frame interval.

(6) When the round is fired, the gun tube vector is used to initiate a flyout calculation. As the CIG computes each new out-the-window view, it retrieves the next sequential chord segment from the flyout vector. It computes whether this chord segment intersected any polygon in its static or dynamic object list. If not, it displays a tracer image for the crew of the firing simulator. If so, it returns the x,y,z coordinates at which the intersection occurred and the object with which the polygon is associated. Note that this approach ensures that a vehicle that inadvertently drives or flies into the path of a projectile will be hit.

(7) Once an impact has been determined, the CIG displays an appropriate impact effect to the crew of the firing simulator. The effect displayed for a kinetic energy round striking the ground is quite different from the effect of a high-explosive round striking armor.

(8) The network interface software transmits a Detonation PDU, which is received by all other

simulation nodes. This information is inserted into the other-vehicle state table for incorporation in the next out-the-window image.

(9) The detonation is displayed. Crews of any vehicles in the vicinity that have an unobstructed line-of-sight to the detonation point will see the round impact effect. For a simulator that does not represent an entity struck by the round, this is all that is required.

(10) If, however, the simulator finds that the impacted polygon is part of an entity it represents, it is responsible for determining what damage, if any, it suffered as a result of the impact. Just as the simulator representing the firing entity is responsible for determining the point of impact, the simulator representing the struck entity is responsible for determining the effect of the impact. The simulator typically uses probabilistic damage tables based on ammunition type, where it was struck, the angle of incidence, and the firing range. Damage results can range from none at all to a catastrophic kill.

(11) In the case of a catastrophic kill accompanied by fire, the simulator representing the struck entity will begin broadcasting Entity State PDUs that reflect a burning vehicle at its current location.

V. SIMNET simulation of radio/telephone communications

One other instructive example is the SINCGARS radio simulation developed for the Army Research Institute Combat Vehicle Command and Control Project under the SIMNET program. This simulation involved the propagation of digital map data, as well as voice signals, over a standard radio channel.

First the voice and data signals were digitized at the simulated transmitter. Header information was added to the data packets to represent the transmitter location, frequency, power, and antenna orientation.

Using principles similar to those for the effects of projectiles, the receiver simulators were responsible for determining which signals were received by the crew. Each receiving simulator used its copy of the terrain data to compute the received signal strength, based on the known location of the transmitter and receiver and diffraction effects caused by intervening objects. A receiver characteristics model was used to determine which signals were captured; only the "winning" signal was decoded and mixed with noise at the calculated signal-to-noise ratio. This signal could be a jammer, as a result of either intentional or unintentional jamming.

The same signal source information could be used for navigational purposes (by friendly forces) or by radiation-seeking missiles (to disrupt enemy communications).

VI. Critical network requirements for DIS interactions

To support the interactions described above, the network must provide a certain level of assured services. In particular, it must exhibit

Low transport latency. DIS Guidance Documents specify less than 300 msec total end-to-end latency (from the trigger pull to the remote display of the muzzle flash in the example above) for "loosely coupled" interactions, and 100 msec total latency for "tightly coupled" interactions (e.g., formation flying).

Low latency variance. Low latency variance is important to minimize jitter, although within certain limits, required time stamps on Entity State PDUs can be used to correct for latency variance by inserting corrective changes in dead-reckoning extrapolations.

Reasonably reliable delivery. The dead reckoning algorithm was designed to be robust with respect to missing datagrams, within certain limits. If an Entity State PDU is missed, the receiving simulator will continue to dead reckon the entity along its prior trajectory until a new PDU is received. If the entity is maneuvering significantly, the next update will arrive within a fraction of a second, the entity's position will be corrected, and the new derivatives will be used to initiate a new extrapolation. A one or two percent datagram loss will present no problem, as long as the missing datagrams are randomly distributed and do not occur in correlated sequences.

For many military simulations, NSA-approved encryption processes are required. This currently represents a major bottleneck, with the encryption devices introducing both significant throughput limitations and latencies.

VII. Techniques to support very large exercises

The largest SIMNET exercise was conducted in March 1990 at five sites. At the peak of the exercise, approximately 850 entities were active, most of which were Semi-Automated Forces. Since SIMNET PDUs contain approximately 1000 bits and active entities average one PDU per second, the traffic volume at the peak was approximately 850 kilobits per second. DIS PDUs are larger than SIMNET PDUs (typically 1700 bits), so the same exercise in DIS would generate roughly 1.5 megabits per second.

If such an exercise were scaled up to involve 100,000 entities, as is currently being proposed as a goal, each simulation node would have to process over 175 megabits per second, extracting from this torrent the relatively small percentage of events that are of potential interest to the entities being simulated at that node. Such a flow would be beyond the capacity of most low-cost simulators, and in any case would represent a very inefficient use of computing and communication capacity. Fortunately, some promising techniques are available to improve this situation.

VII.A. Entity state PDU compression

The first technique takes advantage of the fact that DIS PDUs contain a substantial amount of redundant and/or invariant information. Each entity repeats a description of its characteristics in every Entity State PDU, even though these characteristics never change once the entity has been initialized. Each entity continues to broadcast entity state information periodically even if it is motionless, and even if it has been destroyed. Each entity broadcasts its entire state, even if only one state variable changes (e.g., a rotating turret on a stationary tank).

In each case, a substantial percentage of the PDU content does not change. Analyses indicate that 95% of the time, less than 10% of the bytes change in two successive PDUs from the same entity. This redundancy provides an opportunity for substantial savings from fairly straightforward byte-difference encoding techniques. A recent paper by the author and his colleagues [4] describes a protocol-independent compression algorithm (PICA), which has produced a 4:1 compression ratio in initial tests. Among the advantages of this algorithm are the fact that almost all byte difference PDUs are small enough to fit within a single ATM cell.

VII.B. Site-to-site filtering of PDUs

The next logical technique to explore is a screening algorithm that avoids sending PDUs to sites at which all entities are beyond possible interaction range. Simple geographic range filtering is probably not sufficient, however, since some entities may have long-range sensors that can detect other entities at a considerable distance. Radio signals need to be sent to sites at which one or more entities is tuned to the appropriate channels. Local agents of some kind can be used at each site to maintain lists of entities that are of interest to other sites, and vice versa.

This technique will probably require the use of multicast groups that can be formed and dynamically modified to minimize the transmission of irrelevant data. Studies of how best to do this are just beginning.

VII.C. Combined techniques

These two techniques can be combined, of course. Filtered sets of PDUs can be compressed and bundled for optimum transmission efficiency to each remote site. Compression and bundling also minimizes loads on encryption devices which, as previously noted, are currently a principal bottleneck.

VIII. Some caveats regarding network traffic prediction

It cannot be overemphasized that traffic flow volume is *highly* dependent on the specific exercise scenario being played out – which determines which entities are interacting in the virtual world with entities being simulated at other physical sites on the network. A "worst case" analysis, in which some entities from every physical site are closely interacting with entities from every other site, always requires sending all information everywhere.

To ensure that large exercises (relative to the available bandwidth and processing resources) will not overload these resources, proposed exercises should always be played out in a "simulation of the simulation" to check for an appropriate mapping of real-world simulations to virtual-world interactions and a reasonable balancing of network loads.

These considerations will only become greater as new efforts, to be funded soon, add additional classes of network traffic, such as command and control communication between higher-echelon commanders and their staffs, dynamic terrain effects (such as bridges and runways that accumulate damage), weather effects (rain, fog, smoke, etc.), and more elaborate exercise monitoring and management protocols.

IX. Conclusion

As applications of distributed simulation grow, it is increasingly important that efficient and economical network services be available to support them. Many of the characteristics of distributed simulation are unlike thoseof other "information superhighway" applications. In particular, the requirement for low-latency, real-time, continous-time interactions, combined with large numbers of dynamically changing multicast groups in which each simulation node is both a producer and consumer of information, is unique among foreseeable uses of high-bandwidth communication.

As the magnitude of such applications increases, and the virtual-world phenomenology grows richer, the need for intelligent data traffic management algorithms will be critical. Testbeds now under development will be important in evaluating candidate algorithms. Ideas and contributions are welcome.

References

[1] IEEE Standard for Information Technology – Protocols for Distributed Simulation Applications: Entity Information and Interaction. IEEE Std 1278-1993, IEEE Computer Society, New York, NY, 1993.

[2] Pope, Arthur R. and R.L. Schaffer, "The SIMNET Network and Protocols." BBN Report No. 7627, Cambridge, MA (Revised Version, June, 1991)

[3] Miller, Duncan C., A.R. Pope, and R.M. Waters, "Long-Haul Networking of Simulators." Proceedings: Tenth Interservice/Industry Training Systems Conference, Orlando, FL, December, 1989

[4] Van Hook, Daniel J., J.O. Calvin, and D.C. Miller, "A Protocol-Independent Compression Algorithm (PICA)," Project Memorandum No. 20PM-ADS-005, MIT Lincoln Laboratory, Lexington, MA, April 1994

Author Information

Dr. Duncan C. Miller currently leads a small group of experts at MIT Lincoln Laboratory specializing in Distributed Interactive Simulation architectural issues. This group facilitates cooperation and technical exchange across government programs regarding DIS standards and related issues.

From 1963 to 1993, Dr. Miller worked at Bolt Beranek and Newman Inc. in Cambridge, Massachusetts. In 1983, he formed and managed the group that developed the protocols and software for SIMNET, the innovative ARPA program that led to DIS. For the last five years of his tenure at BBN, he served as Vice President of BBN's Systems and Technologies Division, with responsibility for Advanced Simulation engineering activities in Cambridge, Massachusetts and Bellevue, Washington.

He has served on a number of advisory committees, boards, and panels,including the Naval Research Advisory Committee Panel on the Impact of Advancing Technology on Exercise Reconstruction and Data Collection(1990-91), the Defense Science Board Task Force on Simulation, Readiness,and Prototyping (1992), the Congressional Office of Technology Assessment Defense Modeling and Simulation Project Advisory Panel (1993-94), and the Air Force Science Advisory Board Joint Modeling and Simulation Systems Review Panel (1994).

Dr. Miller holds four degrees from MIT, including the Doctor of Science in Mechanical Engineering. His principal areas of study included control theory, human operator performance modeling, human factors, and perceptual psychology. He is a member of the Institute of Electrical and Electronics Engineers and the Society of American Magicians.

Note: The material presented in this paper is based on previous work conducted by the author at Bolt Beranek and Newman, Inc., Cambridge, MA, USA, under ARPA Contract MDA972-89-C-0060.