Large DIS Exercises - 100 Entities Out Of 100,000

Steven D. Swaine and Matthew A. Stapf
McDonnell Douglas Training Systems
McDonnell Douglas Aerospace
St. Louis, Missouri

ABSTRACT

Distributed Interactive Simulation (DIS) is being promoted as a tool to aid in design, prototyping and manufacturing of weapon systems, development of joint doctrine, mission planning, after-mission reviews and historical analysis. Future networked exercises promise hundreds of thousands of users interacting in a single "seamless battlefield". To achieve this goal, the evolving DIS standards have addressed the protocol for the data which must be exchanged between participants, and they embody a data reduction method known as "dead reckoning" to help reduce network bandwidth utilization. But this is only part of the solution. Connecting a simulator to a large exercise has been likened to "drinking from a firehose"; and most simulations, especially legacy simulators, simply cannot process the envisioned number of external entities. This paper discusses techniques for managing large quantities of entities, by filtering, organizing and prioritizing the DIS data for presentation to the simulation host.

ABOUT THE AUTHORS

Steven D. Swaine received his M.S. degree in Electrical Engineering from Washington University, St. Louis, Missouri in 1993 and a B.S. in Electrical Engineering from the University of Missouri at Rolla in 1986. His primary responsibility is in the area of research and development related to low cost simulation systems, particularly in the areas of microprocessor systems and communication protocols. He has served as the Principal Investigator for the MASS program which pioneered low-cost networked aircraft simulations and is the chief developer of the MDTS DIS Interface Unit. He also has been involved in the development of the DIS network standard.

Matthew A. Stapf received a B.S. in Computer Engineering from the University of Illinois Urbana-Champaign in 1985. After graduation, he came to work for McDonnell Douglas Flight Simulation Laboratory where he was a systems engineer responsible for the system design, installation, and maintenance of high-speed ethernet networks to support real-time traffic. Mr. Stapf is currently the Principal Investigator for the McDonnell Douglas Training Systems Simulation Networking research program.

Large DIS Exercises - 100 Entities Out Of 100,000

Steven D. Swaine and Matthew A. Stapf
McDonnell Douglas Training Systems
St. Louis, Missouri

INTRODUCTION

Most problems associated with large-scale simulation networking can be divided into two categories. The first category defines the minimal data set needed to communicate between nodes, its format and delivery mechanisms. These problems are addressed by application protocols, physical network architectures, data compression and communication services. They have been the primary focus of the DIS community to date and have resulted in the Standard for Distributed Interactive Simulation -- Application Protocols [1] and related documents. The primary concern for large exercises has been the amount of network bandwidth required. The DIS standard has employed a mechanism for bandwidth reduction known as "dead reckoning", which is essentially a lossy compression algorithm relying on statistical multiplexing. In addition, it is generally assumed that technical advances in communications hardware will eventually provide enough bandwidth for the very large DIS exercises envisioned in the future.

The second category, and focus of this paper, relates to data processing at the receiving nodes. These problems are addressed by network interfaces and involve data filtering, prioritization and organization. Computational technology is advancing quickly, but at a slower rate than communications technology, and most workstations and operating systems are not optimized to handle the data rates associated with large DIS exercises. In fact, the packet arrival rate usually becomes a problem for most DIS implementers today long before the available network bandwidth is fully utilized.

We begin by considering the problems associated with designing an interface for large exercises and then outline some generic mechanisms which aid in reducing the amount of network data presented to the host. Next, a specific implementation of a DIS network interface which utilizes many of these mechanisms is described. And finally, an example implementation is outlined detailing how a legacy fighter simulator could be interfaced to a DIS network.

INTEROPERABILITY CONSIDERATIONS

What does it mean to say that a simulation can interoperate with 100,000 network-supplied entities? Stated as such, it generally means very little. Perhaps the 100,000 entities are static bridges and the simulation is a submarine trainer. The trainer probably has no mechanisms for handling bridges and therefore expends very few resources in processing them. If the external entities are all enemy submarines located nearby, it is likely that resources will not be available to process all the entities and some portion will have to be rejected. If some are rejected can we still say the simulation is interoperable? Maybe, if entities are rejected that would be unavailable to (or rejected by) all the actual sensors then, as far as the operator is concerned, the situation is perceived as it would be in the real world and we have perceptual interoperability. If some of the information is not presented to an operator that should be, the simulator may still be partially interoperable. For example, if a tank operator sees 1000 enemy tanks come over a hill instead of 1001 the reaction of the operator may not differ, so we can still claim reactional interoperability. Of course there are other factors which must be included in judging host/exercise interoperability, such as the accuracy of the models used and the fidelity of the sensor representations (including visual). But, this paper focuses on the network interfaces and here we are concerned primarily with the processing of network data for the host system.

From the standpoint of the network interface, interoperability is less qualitative. Each host must dictate to the interface what network information can be rejected, or filtered while retaining the required level of interoperability. In addition to filtering out data not needed by the

host, the interface may also have to prioritize the data to support a lower-level of interoperability during overload conditions. It is useful then to characterize the capabilities of an interface based upon its ability to process host requests and its ability to process network data (in terms of the network bandwidth, packet arrival rate and total numbers of entities). These two groups of metrics provide an insight into the extent of interoperability of an interface and can be used to compare the capabilities of various interfaces.

Note that network utilization and the packet arrival rate are not merely functions of the quantity of entities in a given DIS exercise. The characterization of a DIS exercise must also include information about the dead-reckoning algorithms, error thresholds and minimum send times used as well as the mix of entity types and their associated behaviors. Some work has been done in simulating DIS networks themselves to determine parameters of interest such as maximum transport delay, network utilization, packet arrival rates, etc. for a given network topology based on these exercise characterizations [2]. Thus, for a given mix of entities with known behaviors and a given network architecture, it is possible to characterize an exercise and precisely define the requirements for a given interface and a given host. However, the real challenge is to design a generic interface which can support a wide variety of hosts and can be used in broad range of exercises.

DESIGNING A INTERFACE FOR LARGE EXERCISES

The primary role of the network interface is to off-load the overhead of DIS related processing and input/output, and to regulate, under dynamic host control, the amount of data presented to the host. In addition, the network interface may perform other services such as data conversion, radio modeling, etc.. Since, for very large exercises, there will generally be many more external entities than internal entities, the network interface must focus primarily on the received data.

Most simulations have a limited amount of resources dedicated to processing external entity information; fortunately, most simulations also have finite perception capabilities. The process of removing data which is not of interest to the host is referred to as filtering or clipping. If the host is operating in an environment which exceeds its external entity processing capabilities, the interface must use prioritization techniques to decide what data should be returned to the host.

Filtering, a special case of prioritization (priority of zero), is best done as soon in the receiving process as possible to eliminate data from consideration with the minimal impact on system resources. For this reason, a filtering hierarchy, similar to a computer memory hierarchy, is often employed. It is important to note that the data movement from the low-level interface must be kept to an absolute minimum or else the time spent moving the data will dominate the reception process. This is a crippling limitation of many DIS implementations on workstations, which often must read an entire received packet into virtual space even if the packet is then immediately rejected.

Prioritization involves multiplying weighting factors by a set of entity parameters and adding together the results to achieve an overall priority index for each entity. These parameters include, but are not limited to, entity type, relative location and velocity, and allegiance. For example, a fighter pilot will be much more interested in a enemy aircraft 10 miles away headed towards him and within his field-of-view than in a friendly guided missile targeting a site 500 miles away.

In the next few paragraphs, we consider the transmission of Protocol Data Units (PDUs) from an issuing simulation as they pass through a network and finally are made available to the receiving host processor. Figure 1 summarizes these concepts. Not detailed, but pervasive throughout the hierarchy, will also be numerous checks to attempt to eliminate erroneous data using various hardware and software mechanisms.

Send-Time and Network Operations

Send-Time Filtering. The first opportunity to eliminate a PDU from the data stream is in the originator of the PDU itself. One example is the proposed aggregation/deaggregation protocol. Under this protocol, a simulator may represent a group of entities as a single aggregate entity

until deaggregation is requested. Dead reckoning itself is another example of send filtering. Adaptive network loading has been demonstrated as a technique which involves dynamically varying the dead reckoning error thresholds as a function network utilization.

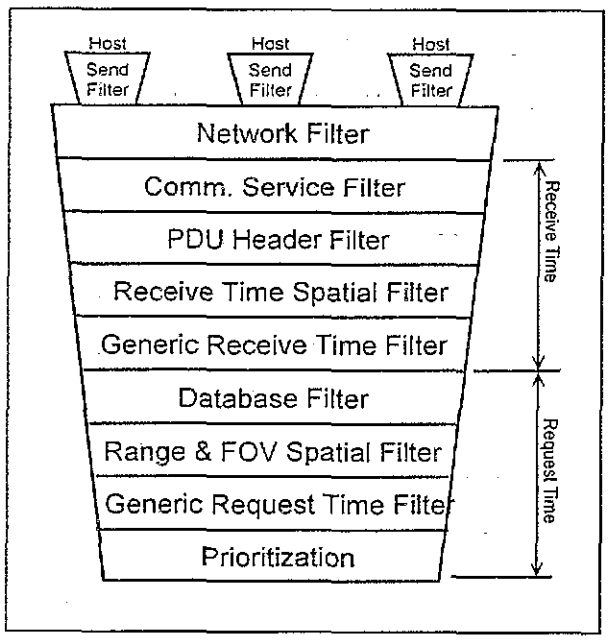

Figure 1. Filter Architecture

Network Filtering. Network filtering utilizes intelligent routers to prevent data from passing to sections of the network where it is not needed. Most research in this area involves the use of dynamic multicast addressing and battlefield gridding. Each grid square in the gaming area is assigned a unique multicast address which is used as the destination address in PDUs issued from entities contained within that grid square. Each host configures its network interface to only allow accept PDUs from entities in grids within its range of perception. This signaling information is then propagated out from the receiver to intelligent routers.

Note that send filtering and network filtering have the added advantage of reducing network utilization, since data is not passed to sections of the network where it is not required. One drawback to network filtering is that entities with large ranges of perception, such as threat environments, environmental servers, data loggers, network monitors and plan view displays present obvious problems.

Receive Time Operations

Communication Service Filtering. Communication service filtering generally falls in the Data Link, Network and Transport Layers of the OSI/ISO reference model. The first opportunity for filtering is examination of the network address. With ethernet this would imply a simple examination of the destination address of an incoming packet to check for a multicast or individually addressed packet meant for the host system. The advantage of these filters are that they generally are implemented in hardware and require no run time software cost. In general, these filters eliminate any non-DIS data which shares the physical network media. Other examples of communication service filters which exist further up in the Network and Transport Layers include the Internet Protocol (IP) type and IP address and the User Datagram Protocol (UDP) port number. These filters also help eliminate some of the non-DIS traffic which may be on the physical network. If additional information, such as exercise identifiers, PDU protocol family or originator location can be encoded into the port number then PDUs can be eliminated without any interface processing. Even further, if this information were encoded into the network address (via multicast addressing) then the PDU can be eliminated again at the hardware level. Data which passes through the communication services filters should then be DIS PDUs which must be processed to determine their value to the host simulation.

Protocol (or PDU Header) Filtering. If not eliminated by any of the communication services filters, the header of the DIS PDU can be examined for relevance. Some basic fields would be the exercise ID, protocol family and protocol version. Note that some interfaces may support multiple DIS versions.

Receive-time Spatial Filtering. Receive-time spatial filtering involves the removal of entities that the host simulation could not detect by any means due to the physical geometry between the two. To determine the relative geometry required between the two entities for perception, the host simulation must define its perception volume. The host's perception volume is often defined by a Center-Of-Interest (COI) and the

maximum range of all the sensors on the receiving system (visual, electromagnetic, radar, etc.). Usually, the COI will be the location of the host entity; however, on some systems, such as Semi-Automated Force (SAF) systems, the perception volume is not always so easily defined. On these systems an entity centroid or the entire gaming area could be specified so as to accept entities only within a broad geographical area.

It is tempting to eliminate Entity State PDUs representing entities which are currently outside the Field-Of-View (FOV) of any host sensors. However, due to DIS dead reckoning, the network interface must maintain entity information covering a much larger volume than the immediate FOV of the sensors (Figure 2). For example, if an Entity State PDU is received for an entity located behind the observer, this PDU should not be filtered because the observer can turn and look behind him in less than the DIS minimum send time. In addition, the rejection ranges must take into consideration the relative velocities of the entities and actually be larger than the fixed perception range. Equation 1 details this relationship.

$$R_r = R_{smax} + (V_{coimax} + V_{emax}) * (T_{smin} + T_{fconfig})$$

R_r -	Rejection range, all entities with *greater ranges are rejected*
R_{smax} -	Maximum range of any sensor (including visual)
V_{coimax} -	Maximum velocity of the COI
V_{emax} -	Maximum velocity of any external entity
T_{smin} -	Minimum send time for DIS Entity State PDUs
$T_{fconfig}$ -	Filter reconfiguration time

Equation 1. Rejection Range

Other Receive-Time Filtering. It is impossible to anticipate all the combinations of filtering operations that may be desired by all applications; therefore, most interfaces include some mechanism to allow the host to specify filters in a generic manner. One such mechanism is for the network interface to perform a series of logical operations on the results of comparisons between user supplied data and fields within incoming PDUs. Using such generic filter specifications, the application could, for example, filter Transmitter PDUs that do not match the host's current Crypto Key ID

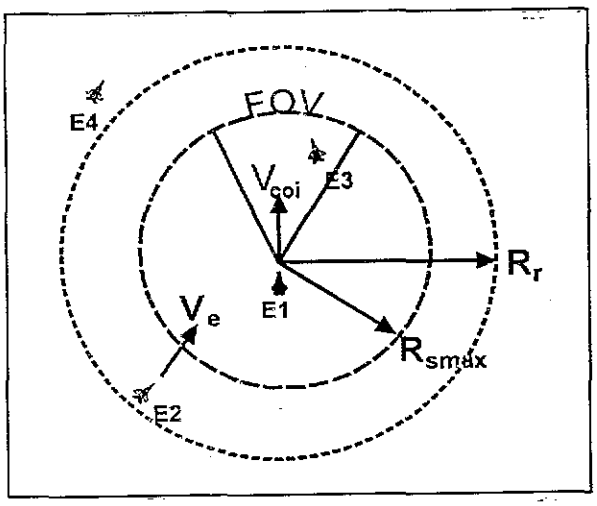

Figure 2. Rejection Range Illustration

Request-Time Operations

Data Organization. If a PDU is accepted by the network interface process, it must then be queued for processing by the host. It is often desirable to logically group data by PDU type or family so that independent host processes can address certain data types. The same generic mechanism used for receive-time filtering can also be used to partition the external entity database into logical groupings or "views". Later, when information is requested from the interface, it can be requested from one of these preconfigured views, greatly reducing the request processing time that would result from scanning the entire database for specific entity sets.

Request-Time Spatial Filtering. As stated previously, the receive-time range rejection is usually best done using a spherical or rectangular volume. However, the volume specified at request-time may be tailored to the instance of the request. For example, the volume may be defined in terms of the FOV of a sensor, or even within a more generic polygonal bounding volume pre-defined by the host. In addition to removing entities not within the field-of-view and performing range rejection, other fidelity/processing decisions can be made at this point. For example, perhaps only the very nearest entities require smoothing, and entities

just within the rejection range may not require rotational dead reckoning. Figure 3 illustrates this concept.

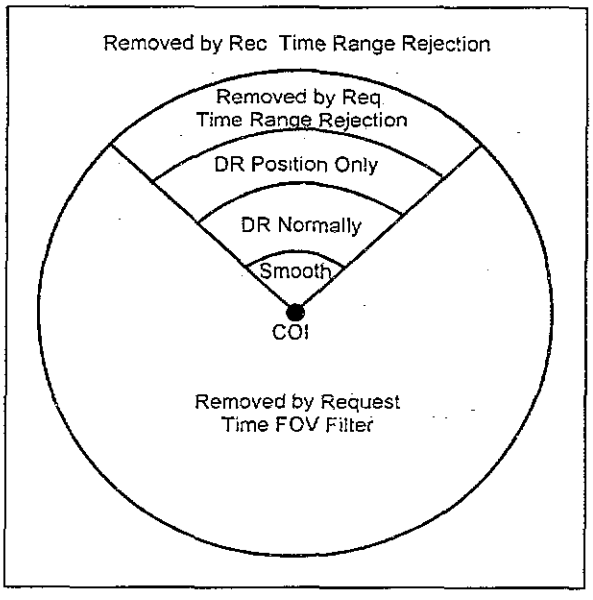

Figure 3. Range Based Fidelity Considerations

A simple dot product can be used to determine if an external entity is within a conical FOV. In the simplest case, a partial dot product can be used to eliminate entities located behind the observer (FOV=180°). The following equations illustrate the process of conical FOV filtering.

FOV	= conical Field-Of-View
alpha	= 1/cos2(FOV/2) (note, this can be calculated once, at initialization)
T	= location of external entity
P	= location of conical vertex (the ownship location is usually used)
N	= Normalized axis of the conical volume (look angle or viewport)

$d_X = T_X - P_X$, $d_Y = T_Y - P_Y$, $d_Z = T_Z - P_Z$
$D = d_X * N_X + d_Y * N_Y + d_Z * N_Z$
if $D < 0$
then the target is behind the viewpoint, stop and reject here (done if FOV=180°)
if $D*D*alpha >= d_X*d_X + d_Y*d_Y + d_Z*d_Z$
then it is in the FOV, accept

Equation 2. FOV Inclusion Calculation

Other Request-Time Filtering. The same generic mechanisms for receive-time filtering can also be employed at request time. By doing so, the application has even more control in reducing the data that is presented to the host. This is especially useful when more than one application shares a network interface, because each application can have independent filter sets which are applied to a common entity database. Meaning, all the data on the network which is of interest to either of the applications which share the interface must pass the receive-time filter stage; however, separate request-time filtering can be imposed by each host.

Request-Time Prioritization. If, after all filtering is complete, there is still more data then the host can process, some data must be eliminated. This is most simply accomplished by truncation, but usually sorting based on range from the COI gives a more desirable result. Note that because sorting is at best a k log n algorithm (the closest k entities selected from a list of n) it is imperative that as many entities as possible are filtered before this final step. Other prioritization options include giving preference to enemy entities or to entities approaching the host's entity.

AN EXAMPLE IMPLEMENTATION OF A GENERIC DIS INTERFACE UNIT

We now outline a specific implementation of a DIS Interface Unit (DIU) which utilizes many of the mechanisms outlined thus far.

In the development of the DIU, speed was sacrificed only when the generic design or platform independence would have been compromised. In general, any complexities in the interface due to the emphasis on performance are abstracted out by the interface libraries into user-friendly calls. While C++ or Ada would have been well suited for this application, C was used due to portability and speed considerations. Interface libraries can be developed in any language, the preferable choice would be that of the host or intermediate translator process. Figure 4 illustrates the software architecture of the DIU. The communication mechanisms between the layers and the functions of each individual layer are described in the next sections.

Hardware Interface Package

The interface to the DIS network and all other platform specific interfaces are entirely encapsulated in the Hardware Interface Package (HIP) which allows the Engine, the heart of the DIU, to be completely independent of the hardware, operating system, and communication service. It is generally desirable to optimize the HIP for each platform, and it should be stressed that the performance of the entire DIU depends largely on the speed and capability of the HIP.

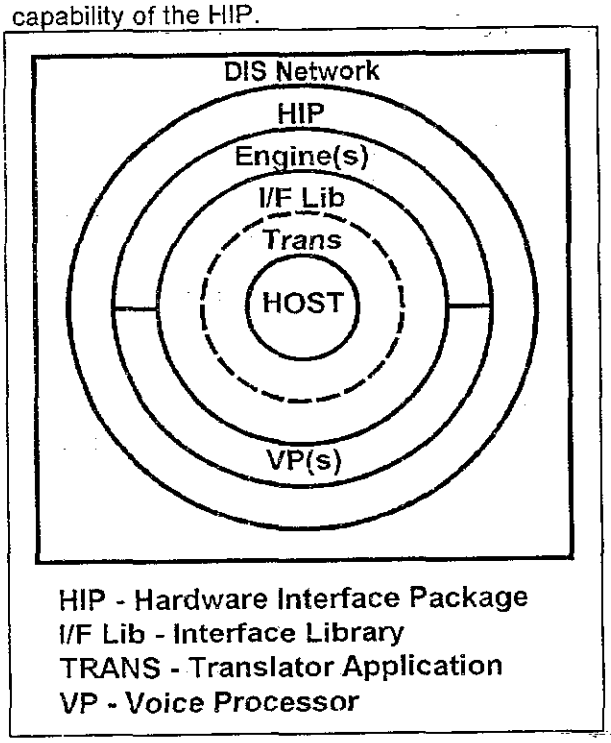

Figure 4. DIU Software Architecture

Many workstations are crippled in that the interface must move the entire packet from the physical network queue into user space even if it will be immediately filtered[1]. Note that if the HIP has access to the hardware, it is possible to dynamically re-allocate network queue buffers so that packets can even be accepted and filed into the interface with no data movement.

Another limitation of most workstations is the difficulty accessing high-resolution timers, and accurately determining the time of packet arrival is often completely impossible. In addition, with most operating systems it is not possible to determine the size of the packet on the network backbone, which compromises the network statistics gathering capabilities of the interface[2].

DIU Engine

The heart of the DIU is the Engine. It consists of several modules which are linked with a Hardware Interface Package to provide an executable DIU. The Executive Unit is the Engine's main loop. It controls operations within the Engine. The Receive Unit is responsible for processing incoming PDUs. It performs receive-time filtering, network variance compensation or transport delay compensation when absolute time is available. The Transmit Unit is responsible maintaining the extrapolated position/orientation of each host entity and sending state data only when necessary. The Statistics Unit accumulates information about network and DIU loading and events. This information is then available to the host or to a network monitor. The Command Unit is the primary application interface, handling all configuration and run-time data requests. It is responsible for entity extrapolation, smoothing and request-time filtering.

Engine Send Process. The DIU Engine maintains separate output "slots" for each internally simulated entity. This allows the Engine to accumulate statistics at the entity level, and allows the host to allocate PDU space within the Engine itself. The advantage of keeping the memory within the Engine is that the host is only required to fill the whole PDU once at initialization, then during real-time, only the dynamic fields need to be modified before instructing the Engine that the data is ready to

[1] Berkeley Standard Distribution (BSD) does support the capability to "peek" into a received message; however, the data must still be moved to a user buffer, it is just left in the queue for the next call to read again. To check the header information, the peek function could be used with a small buffer specified so that the entire packet would not be moved unless it passed the protocol (header) filters. This technique will not work for generic filters, however, since the test fields could be located anywhere within the PDU. One approach (when forced to use BSD) is to keep track of the percentage of PDUs being rejected based on header filters, and use the peek method only when it becomes more efficient.

[2] Silicon Graphics IRIX™ operating system does allow timestamp of received packets and size on backbone, but still must move all received data into user space. Silicon Graphics' raw "snoop" protocol allows the setting of "filters" but, unfortunately, do not support offsets greater than the size of a UDP header.

transmit. Considering that as much as 40 percent of an Entity State PDU is static or padding, this results in a significant time savings.

Engine Receive Process. After an incoming PDU passes the network filters, it will be loaded into the input queue by the network interface hardware. When possible, the HIP will also time-tag the packet at its reception. If present, the operating system will perform communication service filtering; otherwise, the HIP will perform these functions. Once this process is completed a pointer to the PDU itself is then made available to the Receive Unit on the Engine.

The Receive Unit then performs remaining receive-time filtering operations including removing PDUs with unrecognized DIS version fields and, if so directed by the application, PDUs with out-of-range exercise identifiers. Applications may specify unique receive-time generic filter objects to be applied by PDU type. The mechanism for the specification of these filters is further described later. If the PDU is an Entity State PDU, network variance or transport delay compensation can be performed and the entity accepted into the State Database. If this is the first PDU received for this entity, then a number of additional steps are performed. The entity is subjected to all the view filters which have been loaded by the application, and if it is found to be a member of a view, it is entered into the linked view list for that view. View zero is reserved and always represents the entire State Database. An entity may be a member of multiple views.

Views may also be specified as ranged-views. Ranged views are configured with a maximum number of entities or a maximum range from a given COI and the background task will constantly update the view membership of each entity. This further eliminates entities from consideration and since the background check is much faster then the DIS minimum send time, the range can be less then the receive-time rejection range.

It is the size of the State Database which dictates the number of external entities that the Engine maintains available for immediate request. On a 8MB MVME187 single-board computer, the State Database can be sized for over 10,000 external entities without using virtual memory. Note that this is not the same as the number of entities on the network since many entities will have been removed by receive-time filtering. Only the entities which might be of interest within the minimum send time need to be in the database.

If the PDU is a transient PDU (not an Entity State PDU), then it is optionally subjected to the database filter. Here, the PDU may be removed if it relates to an entity which is not currently in the State Database based on a fixed set of rules for each PDU type. For example, if the emitting ID in a Electromagnetic Emissions PDU is not found in the State Database, then the PDU is discarded, otherwise it is placed into a transient receive buffer. These buffers are also maintained on the Engine so that only the data actually used by the host need be moved from the Engine. There are two sets of double buffers, one for PDUs of the audio family and one for remaining PDUs. This allows the audio data stream to be processed separately and at a different rate than the remaining data. There is little advantage in further segregating the PDUs at this point, since the all PDUs not of interest to the host have been removed.

Interface Libraries

The interface library links with the host (or intermediate translator) application to provide a consistent API to the Engine. Interface libraries can be written in any language as required using object-oriented paradigms and extensive error checking. As previously stated, the primary role of the library is to abstract the complexities of the shared memory interface which was optimized for speed and efficiency and not ease-of-use. It also contains many utilities for allocating and filling PDU structures and building the generic filter specifications which are used for generic receive and request-time filters and in determining view membership.

A generic filter consists of one or more "terms", which specify a logical operation between a given value and a field within an incoming PDU. A term consists of a PDU field offset, a field type, an operation and a constant test data element. Each term also includes a global "and" or "or" operation which dictates how it logically relates to the previous term. The general form of the filter specification, selected for run-time efficiency, must match the following logical form:

(term [&& term ...]) [|| (term [&& term ...]) || ...]

where the brackets enclose optional terms and global operations. In words, a filter specification consists of one or more logical terms "and'd" and "or'd" together, with the "ands" taking precedence. Parenthesis are not supported, for run-time efficiency, but could be incorporated at the interface library level and removed before passing the data to the Engine. A syntax for generic filters has been developed and incorporated into the interface library which allows the application to easily specify filters in terms of ASCII strings. The Backus-Naur Form (BNF)[3] grammar of this syntax is shown in Figure 5.

```
<filter>      := "<series>"
<series>      := <spec> | <series>&&<spec>
                 | <series>||<spec>
<spec> :=     <natural> <type> <oper>
                 <number>
<type> :=     c | s | l | uc | us | ul | f | d
<oper> :=     == | != | < | > | <= | >=
<number> :=   -<natural> | <natural>
                 | -natural>.<natural>
                 | <natural>.<natural>
<natural>     := <digit> | <natural>|<digit>
<digit> :=    0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9
```

Figure 5. Backus-Naur Form Specification

As an example, consider a filter object that will be loaded as the Collision PDU receive-time generic filter which will reject Collision PDUs when the colliding site is not 100 or the colliding application is not 200. The associated filter string is:

"18 us != 100 || 20 us != 200"

Which can be interpreted: "reject this packet if the unsigned short at offset 18 (the colliding site) is not equal to 100 or the unsigned short at offset 20 (the colliding application) is not equal to 200." It is sometimes the case that fewer terms are necessary, or filters can be processed faster, if a filter is specified in terms of when to accept a PDU rather then when to reject it. Both methods are supported by the Engine. The interface library also supports generic filter production utilities which allow the application to build and load filters without having to know the offsets of the fields.

ENGINE PERFORMANCE ON THE MVME197 SINGLE BOARD COMPUTER

We now consider the performance of the DIU Engine hosted on the Motorola MVME197. The MVME197 is a single-slot VME board available with either one or two 88110 processors, embedded ethernet, serial and SCSI interfaces. The dual processor board is particularly well suited for DIS applications since one processor can be dedicated to the Engine and the other processor can perform any translator and host interface functions. The MVME197's on-board ethernet interface was used as the DIS network interface for the following benchmarks. The Hardware Interface Package was developed on top of the bare board -- no operating system was used. For purposes of these timings, the host is considered to be the translator. Of course, if the host is further removed, say another processor in the VME or a separate computer with a VME interface, then these times would increase. When not explicitly stated otherwise, timing represents nominal times. To simplify comparisons, dead reckoning algorithm two (FPW) was used and Entity State PDUs had zero articulation parameters. Also, no data conversion or smoothing was requested of the Engine.

Send Statistics

Once the translator has loaded the PDU information into the Engine, a single call to an interface library routine instructs the Engine to issue the PDU. The Engine, in turn, then submits the PDU to the HIP which returns immediately. Because the data is never moved from the buffer, but instead is accessed directly by the ethernet hardware, the buffer is marked "in-use" until the packet has been issued.

Host time to pack data: (varies based on the amount of data updated in the PDU)
position, attitude and DR fields only: **10μsec**

Host time to command a send:
Engine time to pre-process the PDU and decide to send it: **60μsec**

Engine time to pre-process the PDU and decide not to send it: **25μsec**

Total time until the data area is again available on a lightly loaded network (packet send complete): **225μsec**

Receive Statistics:

Interrupt Handler (per packet):	**18μsec**
HIP processing (per packet):	**9μsec**
Processing time for a PDU rejected by exercise ID:	**5μsec**
Processing time for a PDU rejected by third term of generic filter:	**9μsec**
Processing time for a accepted Entity State PDU:	**25μsec**
Processing time for a accepted transient PDU:	**20μsec**

Request Statistics

Again the focus is on Entity State PDUs related to entities which are currently in the requested view in the State Database.

Return 100 entities from a view of 100 (No sorting):	**3.1msec**
Return 100 entities from a view of 300 (Sorted and FOV filtering enabled):	**3.9msec**
Return 100 entities from a view of 1000 (Sorted and FOV filtering enabled)	**8.5msec**
Return 100 entities from a view of 5000 (Sorted and FOV filtering enabled):	**25.2msec**

For each of the previous cases where FOV filtering was used, the FOV filter first reduced the number of candidates to approximately one-third of the total in the view. The remaining entities were sorted to return the closest 100. Clipping plane filtering and request-time generic filters were not used in these measurements.

AN EXAMPLE IMPLEMENTATION TO A LEGACY SIMULATION

In this section we consider the problem of interfacing a legacy fighter simulation to a large DIS exercise using the DIU. The elementary frame rate of the simulator is 20 Hertz and all Engine calculations are completed within a single frame. The simulator has the following sensor restrictions due to the host simulator hardware, software and/or actual avionics system.

Air-to-air (A/A) radar - 100 air entities within 120 degrees FOV and out to a range of 80nm.

Visual system - 100 entities within a 45 degree FOV out to 15nm from the ownship.

Air-to-ground (A/G) radar - 100 land entities within a geometric volume approximating the scan width of the radar during a single frame out to a range of 40nm. This volume is a five degree "wedge" defined by two vertical planes intersecting at the fighter. By using this technique, thousands of entities can be displayed on the radar.

Based on this information, the maximum external entity density can then be calculated for each sensor. To simplify this analysis, a uniform entity distribution is assumed. A better method would be to use a non-uniform distribution (perhaps a Poisson distribution). The degree of interoperability is then characterized in a probabilistic fashion (i.e. the simulator has a certain probability that it will be fully interoperable in a given exercise). For density calculations, the perception volumes have been projected onto a plane and the area is calculated. This admittedly greatly reduces the number of calculated entities which the interface must maintain; however, most of the entities in a very large exercise will generally be ground-based or close to the ground. With ground-based entities it is reasonable to assume that there will not be more than one entity in any vertical projection. Even with air entities, it is usually more intuitive to convey density information in terms of area (i.e. 1000 aircraft within a 60nm by 60nm area) rather than in terms of a volume.

Entities Densities Used:

A/A radar -	0.015 Air Domain Entities/nm2
Visual -	1.132 Total Entities/nm2
A/G radar -	1.432 Gnd Domain Entities/nm2

Next, we calculate the maximum rejection range, which will determine the subset of entities accepted into the State Database (View 0). The maximum velocity of the ownship is assumed to be 1000 knots, and the maximum velocity of any entity of interest is 1000 knots. The network will have a minimum send time of thirty seconds and the receive-time range rejection filter reconfiguration time on the DIU

Engine is negligible. Therefore, based on the formula for receive-time range rejection (Equation 1), we must add an additional 17nm to the A/G radar range of 40nm to calculate the final receive-time rejection range for ground-based entities of 57nm from the ownship. Similarly, for air platforms, the rejection range is 97nm.

Next we define the rejection range for all of the views. Recall that the DIU Engine will check each received entity in the background for migration into and out of ranged views. The worst case time for this check is five seconds, which is the filter reconfiguration time. Since the data is immediately available in the database, the effective minimum send time is zero and the view rejection range must only be increased by 2.8nm in each view.

Using these results, the number of entities that the interface must support in each view can be calculated. For example, the air view must support 322 entities (.015 entities/nm^2 × π × 82.8 nm^2)(see Figure 6).

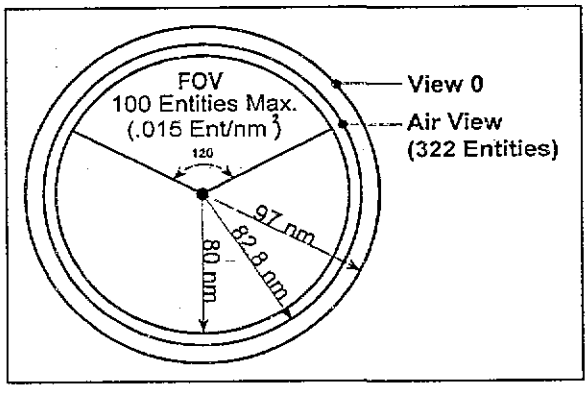

Figure 6. Air View

Calculated Maximum Entities Per View:

A/A radar	- 322
Visual	- 1127
A/G radar	- 8243

1) A/A Radar View

filter - accept only air domain platforms
COI - ownship position
range - 83nm

2) Visual View

filter - none
COI - ownship position
range - 18nm

3) A/G Radar View

filter - Accept only ground domain platforms
COI - ownship position
range - 43nm

Once per host cycle, request are then made as follows:

1) Request A/A radar information from the A/A radar view, sorting to return the closest 100 entities to the ownship within the radar's FOV.

2) Request the visual data from the visual view to return the closest 100 entities to the ownship within the visual FOV.

3) Request A/G radar information from the A/G radar view, using clipping planes to reject entities outside the current radar swath and return the first 100 entities encountered.

Processing Considerations

Recall that while the Engine is processing these requests, the host is free to perform other tasks, such as processing received transients or performing send operations. Therefore, these sequential requests actually represent the worst case path. Based on the performance data of the MVME197 processor board, it can be seen that the memory and processing capabilities exist to service hosts requests. The previous analysis shows that the fighter simulator would be fully interoperable and that the limiting factor is the host itself and not the network interface. If other sensors are required (such as FLIR), additional engines could be added in parallel to support them.

Turning attention to the receive process, since the visual view will include all ground-based entities, the limiting densities are .015 air platforms per square nautical mile over an area out to 97nm from the ownship and 1.132 total entities per square nautical mile over an area out to 57nm. This corresponds to about 12,000 entities which must be accepted into the State Database when operating in fully interoperable mode. If we assume the use of ethernet and UDP/IP and a minimum send time of 30 seconds, the data rate associated with 12,000 static entities with no articulated parts is much less than one megabit per second and the packet rate is only 400 packets per second. Of course, entities which send with a greater

frequency, or entities with articulated parts and transient PDUs will greatly increase these numbers. If we assume one percent of the entities issue Entity State PDUs at a rate of five PDUs/second and ten percent of the entities issue Entity State PDUs at a rate of one PDU/second while the rest are static, then the total rate for the data which will be accepted by the interface is about 2156 packets/second.

Since 100,000 external static entities will generate over 30 megabits/second of Entity State data, ethernet's 10 megabits/second bandwidth will be exceeded unless some sort of network filtering is employed. One solution is to use dynamic multicast addressing and separate multicast address groups for air versus all other entities. If we define database grids which cover 60nm by 60nm at the ground level, the interface needs to subscribe to 16 air grids and four general entity grids (which also covers a five second network filter reconfiguration time). Based on the worst-case densities, this corresponds to a receive load of about 17,000 entities. Using the previous ratios for network loading, this corresponds to a packet rate of about 3085 packets/second.

Experimental Comparisons

In order to validate the load on the DIU Engine in the previous example, an actual prototype implementation of this exercise environment was constructed. A SAF system was used to generate the 17,000 entities with the appropriate distribution to simulate the input from the network into the ethernet interface. The following processing statistics were gathered by the DIU Engine in the fighter simulator.

65%	-Host Requests for Entities
22%	-Network Processing
<1%	-Statistics Gathering
<1%	-Sending of Host PDU's

<89%	Total Engine Processing Load

CONCLUSIONS

Most networked, man-in-the-loop simulations strive to achieve a high level of reactional interoperability. This level of interoperbility mandates that the user reacts to given simulated situations in the same manner as they would in the real world. For large exercises, the role of the network interface is to filter and organize inforation for the host without compromising the required level of interoperability. Further, the interface should prioritize the data for graceful degradation when operating in overload conditions. As has been illustrated, by analyzing the simulator capabilites and making some assumptions on external entity distribution, the limits on external entity desities can be determined. This information also dictates the requirements of the network interface.

On the basis of the Engine processing loads detailed in the previous experiment, it is possible to attach a multi-sensor legacy simulation which only supports approximately 100 entities per sensor to a network of 100,000. In order to interface such a simulation with ethernet networks and current processing technology, dynamic multicast addressing should be used in combination with extensive filtering and entity database management techniques. Such an interface does not require costly hardware or removal of entities which would normally be perceived in a real world scenario. It merely requires the application of prior knowledge to help the network interface anticipate what information the host will be requesting. This application of prior knowledge allows the network interface to reject information outright which will not be of interest to the host and therefore greatly reduces processing requirements placed on the network interface.

REFERENCES

1. Standard for Distributed Interactive Simulation -- Application Protocols, Version 2.0, Fourth Draft, IST-CR-93-40, Institute for Simulation and Training, 3280 Progress Drive, Orlando, FL, 32836.

2. Dille, J. and S. Swaine, *"Discrete Event Simulation and Analysis of DIS Network Architectures"*, Proceedings of the 14th Interservice/Industry Training Systems and Education Conference, 1992.

3. Pagan, Frank P., "Formal Specification of Programming Languages: A Panoramic Primer", Prentice-Hall Inc., Englewood Cliffs, New Jersey, 1981.