

AFIT/GCS/ENG/96D-09

Dynamic Relevance Filtering  
In  
Asynchronous Transfer Mode-Based  
Distributed Interactive Simulation Exercises

THESIS

David T. Hightower, Captain, USAF

AFIT/GCS/ENG/96D-09

19970206 160

DTIC QUALITY INSPECTED 3

Approved for public release; distribution unlimited

The views expressed in this thesis are those of the author and do not reflect the official policy or position of the Department of Defense or the United States Government

**AFIT/GCS/ENG/96D-09**

Dynamic Relevance Filtering  
in  
Asynchronous Transfer Mode-Based  
Distributed Interactive Simulation Exercises

THESIS

Presented to the Faculty of the Graduate School of Engineering  
of the Air Force Institute of Technology  
Air Education and Training Command  
In Partial Fulfillment of the  
Requirements for the Degree of  
Master of Science (Computer Systems)

David T. Hightower, B.S.  
Captain, USAF

December, 1996

**Approved for public release; distribution unlimited**

## Acknowledgments

If ever there was a test of determination, it would have to be this thesis. Were it not for the unwavering support of Major Keith Shomper I am not sure this would have been completed. At the heights of enthusiasm and the depths of frustration his optimistic attitude proved invaluable.

Thanks also to Lieutenant Colonel Martin Stytz, the miracle worker who was able to grease the wheels and make sure I had at least a minimum development configuration.

My heartfelt appreciation to Captain Rick Raines, Network Guru and all-around problem-solver. His calm attitude in the face of adversity and continual faith in the light at the end of the tunnel kept me going through the delays and anxiety. Cells in Frames? IP over ATM? No problem! See the next few slides.

To Dr. Edward Powell of the Army's Joint Precision Strike Demonstration team I owe a debt of gratitude as well. His assistance and explanations of matters made the concepts of this work possible. All via e-mail, yet.

Finally, thanks also to Steve Sheasby for his help in making my work useful for AFIT's DIS exercises. Perhaps there will be a legacy in this after all.

David T. Hightower

## Table of Contents

Acknowledgements .....	ii
List of Figures .....	vii
List of Tables .....	ix
Abstract .....	x
1. INTRODUCTION .....	1-1
<i>1.1 DIS Limitations</i> .....	1-2
1.1.1 Limitations on computing resources .....	1-2
1.1.2 Limitations on network resources .....	1-3
<i>1.2 Problem Statement</i> .....	1-4
<i>1.3 Assumptions</i> .....	1-4
<i>1.4 Thesis Presentation</i> .....	1-4
2. BACKGROUND .....	2-1
<i>2.1 Asynchronous Transfer Mode (ATM)</i> .....	2-1
2.1.1 History .....	2-1
2.1.2 ATM .....	2-2
2.1.2.1 ATM Adaptation Layers (AALs) .....	2-2
2.1.2.1.1 AAL1 .....	2-3
2.1.2.1.2 AAL2 .....	2-4
2.1.2.1.3 AAL3/4 .....	2-4
2.1.2.1.4 AAL5 .....	2-4
2.1.2.2 Virtual Paths and Channels .....	2-4
<i>2.2 Data Partitioning</i> .....	2-5
2.2.1 RITN .....	2-6
2.2.2 JPSD .....	2-7
2.2.3 NPSNET .....	2-9
<i>2.3 IP Multicast</i> .....	2-9
<i>2.4 Conclusion</i> .....	2-10

3. APPROACH.....	3-1
3.1 <i>Network Protocols</i> .....	3-1
3.1.1 Local Area Network Emulation (LANE) .....	3-1
3.1.2 IP-Over-ATM.....	3-3
3.1.3 Native ATM .....	3-5
3.2 <i>Dynamic Partitioning</i> .....	3-8
3.3 <i>Component Hierarchy</i> .....	3-9
3.3.1 Daemon Manager (DM) .....	3-10
3.3.2 Partition Manager (PM).....	3-11
3.3.3 Battlespace Manager (BSM) .....	3-13
3.4 <i>ATM-Specific Issues</i> .....	3-13
3.4.1 Quality of Service (QoS).....	3-14
3.4.2 Maximum Transmission Unit (MTU).....	3-15
3.5 <i>Conclusion</i> .....	3-15
4. DESIGN.....	4-1
4.1 <i>Overview</i> .....	4-1
4.2 <i>Common Features</i> .....	4-1
4.2.1 Shared Memory Arenas.....	4-1
4.2.1.1 Shared Data Structures .....	4-2
4.2.1.1.1 The buffers Data Structure.....	4-3
4.2.1.1.2 The ATMinfo Data Structure.....	4-4
4.2.1.1.3 The fdlist Data Structure.....	4-4
4.2.2 Process Flow.....	4-4
4.2.2.1 Parse Command-line Options.....	4-5
4.2.2.2 Set Signal Vectors .....	4-6
4.2.2.3 Parse Configuration File.....	4-6
4.2.2.4 Initialize Shared Memory and Semaphores.....	4-7
4.2.2.5 Start SendDaemon.....	4-7

4.2.2.6 Start ListenDaemon.....	4-8
4.2.2.6.1 ReceiveDaemons .....	4-9
4.2.2.7 Process.....	4-10
<b>4.3 Battlespace Manager .....</b>	<b>4-10</b>
4.3.1 Overview.....	4-10
4.3.2 Configuration File.....	4-10
4.3.3 Process.....	4-11
4.3.4 BSM Handling of ESPDUs.....	4-12
4.3.5 BSM Handling of Control PDUs .....	4-13
4.3.6 BSM Handling of Other PDUs .....	4-14
<b>4.4 Daemon Manager.....</b>	<b>4-14</b>
4.4.1 Overview.....	4-14
4.4.2 Configuration File.....	4-14
4.4.3 Process.....	4-14
4.4.4 DM Handling of ESPDUs .....	4-15
4.4.5 DM Handling of Control PDUs.....	4-16
4.4.6 DM Handling of Other PDUs.....	4-16
<b>4.5 Partition Manager .....</b>	<b>4-16</b>
4.5.1 Overview.....	4-16
4.5.2 Command-line Options.....	4-17
4.5.3 Configuration File.....	4-17
4.5.4 Process.....	4-17
4.5.5 PM Handling of ESPDUs.....	4-18
4.5.6 PM Handling of Control PDUs .....	4-19
4.5.7 PM Handling of Other PDUs .....	4-21
<b>4.6 Program Communication .....</b>	<b>4-21</b>
<b>4.7 Exercise Overview.....</b>	<b>4-21</b>
<b>4.8 Conclusion.....</b>	<b>4-26</b>

5. RESULTS .....	5-1
5.1 Overview.....	5-1
5.2 ATM Results.....	5-2
5.2.1 Run 1 .....	5-3
5.2.2 Run 2 .....	5-4
5.2.3 Run 3 .....	5-4
5.2.4 Analysis of ATM Results.....	5-5
5.3 Dynamic Partitioning Results .....	5-6
5.4 Conclusion.....	5-8
6. CONCLUSIONS.....	6-1
6.1 Summary of Problem.....	6-1
6.2 Contributions.....	6-1
6.3 Future work.....	6-1
6.3.1 Hybrid Partitioning.....	6-1
6.3.2 Merged BSM and PM .....	6-2
6.3.3 Signal-Driven ReceiveDaemons .....	6-2
6.3.4 AAL_NULL.....	6-3
6.3.5 Permanent Virtual Circuits .....	6-3
6.3.6 Modified DIS PDUs .....	6-3
6.4 Conclusions and Recommendations.....	6-4

## List of Figures

Figure 2-1: ATM Layer Hierarchy.....	2-3
Figure 2-2: Virtual Circuit and Virtual Path Switching.....	2-5
Figure 2-3: RITN System Architecture .....	2-6
Figure 2-4: JPSD System Architecture .....	2-8
Figure 2-5: Notional IP multicast group address hashing.....	2-10
Figure 3-1: Encapsulation of a 144-byte ESPDU using LANE.....	3-2
Figure 3-2: ForeThought implementation of TCP/IP.....	3-4
Figure 3-3: Encapsulation of a 144-byte entity state PDU .....	3-5
Figure 3-4: Encapsulation of a 144-byte entity state PDU .....	3-6
Figure 3-5: ICD ATM addressing scheme.....	3-7
Figure 3-6: Multicast rebroadcasting using an agent. ....	3-8
Figure 3-7: Component interaction .....	3-10
Figure 3-8: DaemonManager conceptual view .....	3-11
Figure 3-9: PM conceptual view.....	3-12
Figure 3-10: BSM conceptual view.....	3-13
Figure 4-1: Common process flow.....	4-5
Figure 4-2: SendDaemon process flow.....	4-8
Figure 4-3: ListenDaemon process flow.....	4-9
Figure 4-4: ReceiveDaemon process flow.....	4-10
Figure 4-5: Battlespace Manager process flow. ....	4-11
Figure 4-6: DM process flow.....	4-15
Figure 4-7: PM process flow. ....	4-18
Figure 4-8: PM geographic IE expansion. ....	4-20
Figure 4-9: Entity Registration .....	4-22
Figure 4-10: Interest Expression and MCAST_ADD Process. ....	4-23

Figure 4-11: Entity Transition..... 4-25

Figure 4-12: PM Split..... 4-26

Figure 5-1: IE overlap..... 5-7

List of Tables

Table 2-1: B-ISDN service classes.....	2-2
Table 3-1: Overhead associated with LANE.....	3-2
Table 3-2: Performance Figures for ATM adapters.....	3-3
Table 3-3: Overhead associated with IP-over-ATM. ....	3-5
Table 3-4: Overhead associated with native ATM.....	3-6
Table 3-5: Bandwidth reservation parameters. ....	3-14
Table 4-1: buffers data structure.....	4-3
Table 4-2: Send/receive buffer data structure.....	4-3
Table 4-3: ATMinfo data structure. ....	4-4
Table 4-4: fdlist data structure. ....	4-4
Table 4-5: Signals vectors changed.....	4-6
Table 4-6: Common configuration file entries.....	4-7
Table 4-7: BSM configuration file entries.....	4-11
Table 4-8: DM configuration file entry. ....	4-14
Table 4-9: PM configuration file entry.....	4-17
Table 5-1: ATM test results.....	5-3

Abstract

As Distributed Interactive Simulation (DIS) exercises continue to grow in scale, the need to support a large number of players has become apparent. The demands on the network and the simulation hosts in large exercises, though, have proved to be prohibitive, requiring significant computational overhead to filter through the information and extract what is relevant to a particular simulation. Some mechanism is needed to reduce irrelevant network traffic received by a system, while increasing the bandwidth available for the DIS exercise. Previous research efforts in this area have centered primarily on fixed geographic partitions of the battlespace to reduce the traffic at a given host. This geographic partitioning cannot adapt to the changing battlespace, and requires relatively significant pre-exercise setup and coordination. Our research has been to implement a DIS exercise system using native ATM interfaces, and to determine if a dynamic partitioning system is feasible and will provide a sufficient reduction in network traffic to allow DIS exercises to scale to the target 100,000 entities. A support infrastructure for DIS over ATM was developed and tested with current AFIT DIS applications, and a prototype dynamic partitioning system using geographic criteria was implemented.

## *1. Introduction*

Modern warfare has become increasingly complicated as technology has moved onto the battlefield. Gone are the days when the "citizen soldier" could be relied upon to provide national defense; the weapon systems employed by today's military forces have progressed to the point where extensive training is required. Traditionally, the United States Armed Forces have relied on field training exercises to gain this training [25]. While these exercises serve a valid purpose, they are not without their drawbacks:

- 1) Field training exercises are extremely expensive, both in terms of money and materiel; weapons and equipment wear out faster when employed in real-world training exercises.
- 2) Assembling all the players necessary to conduct a successful training exercise is difficult, particularly given the increased mission requirements on each unit caused by the drawdown.
- 3) Potential exercises must be weighed against any environmental concerns.
- 4) In some cases, it is impossible to duplicate the parameters of a required scenario with enough realism to be useful.

Recognizing these limitations, in 1983 the Defense Advanced Research Project Agency (DARPA) initiated an effort to harness the increasing capabilities of computer-based simulation. The goal was to develop simulators and training systems that would be capable of providing the desired level of tactical coordination training at a minimum cost. The United States Army had already developed stand-alone simulators and training systems for major weapons system purchases [16]; what made this effort unique was its emphasis on interconnected simulations. The result of this research was the Distributed Simulator Networking (SIMNET) protocol [1]. SIMNET allowed information produced by a simulator (such as an entity's location and armament) to be transmitted over a computer network via "messages" known as Protocol Data Units (PDUs) and received by other simulators participating in the same exercise [17, 18].

While highly successful in its purpose, SIMNET was originally designed for use with United States Army simulators only. This limitation proved difficult to overcome as other services began entering the simulation arena. For example, the SIMNET Vehicle Appearance PDU field that describes object types only allowed for army vehicles; an attempt was made to add other vehicles, but with little success [3]. To address these deficiencies, in 1989 DARPA and the Army Project Manager for Training Devices (PM TRADE) began the process of developing a "follow-on" to the SIMNET protocol [16]. This protocol became the Distributed Interactive Simulation (DIS) protocol, and incorporated all the essential elements of SIMNET.

## **1.1 DIS Limitations**

As the DIS protocol began to proliferate in the simulation community, it became obvious that as the number of participants in a simulation increased, so too would the amount of traffic on the network. DIS requires that each entity in a simulation transmit information about itself as changes occur; however, even a quiescent entity must send a "heartbeat" PDU every 5 seconds, since an entity that has been quiet for 12 seconds or longer is considered to have left the exercise [12].

In its original implementation, DIS assumed the use of *broadcast* addresses, in which all simulation hosts receive all information from each other, and each receiving host is responsible for sorting out what information is relevant to the entities it supports. Similar to the SIMNET protocol, information sent to other hosts is encapsulated into a network packet known as a Protocol Data Unit (PDU). In any DIS exercise, however, only a subset of all DIS PDUs are relevant to a given receiver; only this relevant subset needs to be transmitted and processed [28]. Under a broadcast paradigm, though, each host must examine *each* PDU it receives to determine if it contains relevant data.

### ***1.1.1 Limitations on computing resources***

The main constraint in any DIS simulation system is the limited number of times per second a host (a computer participating in a simulation) can service network interrupts

(receive and process a PDU) and still be able to simulate its entities with reasonable fidelity [20]. Since each PDU arriving at a host must be examined for relevance, the obvious solution is to reduce the number of PDUs that each host receives. The problem becomes one of ensuring the host receives all PDUs that are relevant to the entities it is simulating (for example, all PDUs that are within a certain proximity to its entities), while reducing the number of non-relevant PDUs received.

A possible solution to this problem that has been presented is to segment, or *partition*, the traffic on a network into areas of interest [20]. Each host would then receive PDUs from only those partitions in which it was interested [26]. Work done to date on partitioning has concentrated on *static* partitions; the division of the battlespace is determined before the exercise begins, and remains constant throughout. Little effort has been directed toward the development of *dynamic* partitions—a division of the battlespace that adapts to the exercise in progress, changing partitioning criteria as necessary to optimize partitions and present the minimum number of excess PDUs to a host.

### ***1.1.2 Limitations on network resources***

The network bandwidth available for DIS exercises is also a limiting factor. It has been calculated that future planned DIS exercises will need to support network loads of 140,000 packets per second, with bandwidth requirements reaching 230 Megabits per second (Mbps) [4]. Most Local Area Networks (LANs) in use today provide bandwidth ranging from 10 Mbps (Ethernet) to 100 Mbps (FDDI/Fast Ethernet); Wide Area Networks (WANs) are typically 1.544 Mbps (T-1) to 44.736 Mbps (T-3) [24]. Clearly, existing network infrastructure will not support the requirements of expanding DIS exercises.

A possible solution to the network bandwidth limitations is the use of Asynchronous Transfer Mode (ATM). ATM is capable of speeds ranging from 51 Mbps (OC-3) to 9,953 Mbps (OC-192), and is available for both WAN and LAN applications.

## **1.2 Problem Statement**

I propose to create a program interface to allow existing AFIT simulations to take advantage of ATM's greater bandwidth. In addition, I seek to develop and evaluate a dynamic relevance filtering mechanism that will allow adaptive partitioning of the DIS battlespace.

## **1.3 Assumptions**

To successfully conduct this research, I assume the availability of an installed ATM network. The expected configuration will consist of 9 SGI workstations connected via OC-3 multimode fiber-optic strands to a Fore Systems ASX-1000 ATM switch. Each system should be running IRIX 6.2 or later, and have installed the software libraries necessary to allow program interface with the underlying network.

## **1.4 Thesis Presentation**

This thesis is divided into six chapters, with two appendices. Chapter Two presents a brief overview of ATM, as well as an outline of current research work in the area of relevance filtering for DIS exercises. Chapter Three presents the approach I took in development of the exercise system, with a discussion of the rationale behind some of the decisions made. Chapter Four describes the system that was implemented to meet the requirements set forth above. Chapter Five covers the results of this research, and Chapter Six details requirements not fulfilled as well as providing recommendations for future work. Appendix A lists the function prototypes for the DIS ATM library developed for this work. Appendix B covers the PDUs developed to support the partitioning infrastructure.

## **2. Background**

This chapter presents a brief description of Asynchronous Transfer Mode (ATM), and will cover several current research efforts in the area of data partitioning. A discussion of the limitations inherent with IP multicast groups is included as well.

### **2.1 Asynchronous Transfer Mode (ATM)**

#### **2.1.1 History**

Communications networks have evolved tremendously since the first telephone exchange was installed in 1878. The advent of such services as telegraphs, cable television, and packet-switched networks (such as the Internet) have resulted in a plethora of dedicated networks, with each of these being able to transfer a single type of information (voice, data, or video). Attempts have been made to combine services and integrate emerging technologies into the existing networks, but in most cases the best option was to simply install a new network [2, 10, 22].

The invention and proliferation of digital transmission and digital switching technology introduced the possibility of combined networks, in which data of any type could be carried efficiently by a single network. In 1972 the Comite Consultatif Internationale de Telegraphique et Telephonique (CCITT), also known as the International Telegraph and Telephone Consultative Committee, issued recommendation G.705, which defined the first vision of an Integrated Systems Digital Network (ISDN) that would combine voice, video, data, and other services onto a single network. Two interface rates were defined: The basic rate interface at 144 kilobits per second (kbps), and the primary rate interface at 1.5 or 2.09 megabits per second (Mbps). While these two rates can support a wide range of services such as voice and data transfer, high-bit-rate services such as image and video services required much greater bit rates. Accordingly, ISDN was divided into narrowband ISDN (N-ISDN) operating at the 2.09 Mbps maximum rate, and broadband ISDN (B-ISDN) at higher rates. To better categorize the types of traffic B-ISDN was envisioned to service, four classes of

service were defined, as shown in Table 2-1. In 1990, CCITT selected ATM as the transport technique for B-ISDN [10, 22].

Table 2-1: B-ISDN service classes [22].

Criteria	Class A	Class B	Class C	Class D
Bit Rate	Constant	Variable		
Connection Mode	Connection-oriented			Connectionless
Timing Synchronization	Required		Not Required	
Example	Telephony	Video	Data Transfer	LAN Interconnection

### 2.1.2 ATM

In broad terms, ATM describes a data encapsulation and switching method in which fixed-size data units, or *cells*, are rapidly switched from the source via a sequence of virtual paths and channels through any interconnected switches to the destination. An ATM cell is 53 bytes long, with 5 bytes of header information and 48 bytes of data; the fixed cell size allows switching to be done in hardware, resulting in dramatically increased throughput.

#### 2.1.2.1 ATM Adaptation Layers (AALs)

An ATM Adaptation Layer (AAL) serves as the interface between higher layer services and the ATM layer (see Figure 2-1). The AAL consists of two internal layers:

- 1) The Convergence Sublayer (CS) provides required encapsulation of data, flow control and error recovery where necessary, and segmenting into Service Data Units (SDUs).
- 2) The Segmentation and Reassembly (SAR) segments the SDUs from the CS into 48-byte payloads for the ATM cells.

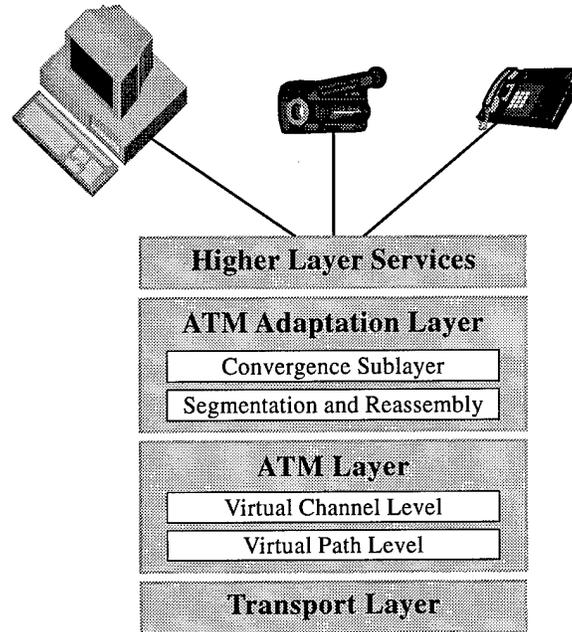


Figure 2-1: ATM Layer Hierarchy [2].

Initially, 4 distinct AALs were defined, to correspond to the 4 service classes. It was later determined that there was little functional difference between class C and D services, so AAL3 and AAL4 were merged [6]. An additional adaptation layer (AAL5) was later added to address the relatively large overhead incurred with AAL3/4 [2, 10].

#### 2.1.2.1.1 AAL1

AAL1 is an unreliable service for connection-oriented constant bit rate (CBR) traffic such as telephony. The AAL1 CS treats each transmitted bit individually, and when either 368 or 376 bits have been received (depending upon the connection), they are blocked into a 46- or 47-byte PDU and passed to the SAR, along with sequencing and control information. The SAR adds error and parity checking information to fill out 48 bytes, then delivered to the ATM layer to be placed into a cell and transmitted [2, 10].

#### **2.1.2.1.2 AAL2**

AAL2 has not yet been fully defined, since there are few data devices that require a synchronous connection with a variable bit rate. An example of this would be a video transmission system which transmits only that information that has changed [2].

#### **2.1.2.1.3 AAL3/4**

AAL3/4 deals with asynchronous variable bit rate (VBR) traffic such as that found on a LAN. Variable length data blocks up to 65,535 can be handled by AAL3/4; it is assumed that larger blocks will be segmented by higher layers before being passed to AAL3/4. AAL3/4 has stronger error checking than any of the other AALs, and also includes information to assist the destination system in reassembling the data. The AAL3/4 CS adds 8 bytes of information to the data block, and then delivers 44-byte segments to the SAR where additional control and error-checking bits are added to fill out the 48-byte cell [2].

#### **2.1.2.1.4 AAL5**

AAL5, also known as Simple and Efficient Adaptation Layer (SEAL), was developed to address the minimum 16.9% overhead associated with AAL 3/4. No information is added to assist data reassembly, and the error checking is over the entire data block as opposed to each cell (as with AAL3/4). The AAL5 CS adds 8 bytes of control information, but the SAR adds nothing, so 48-bytes of the CS data is sent per cell [2].

### **2.1.2.2 Virtual Paths and Channels**

Section 2 of ITU-T Recommendation I.150 defines ATM as "a connection-oriented technique, in which connection identifiers are assigned to each link of a connection when required, and released when no longer needed" [5]. To facilitate routing through an ATM network, each ATM cell header includes a connection identifier which contains a Virtual Path Identifier (VPI) and Virtual Channel Identifier (VCI). The VPI/VCI has local significance (on the current link) only; at each ATM switch either or both of these fields may be changed

depending on the routing of the cell. Virtual Channel Connections (VCCs) detail end-to-end paths, and are made up of Virtual Channel Links (VCLs) [2, 24]. VCLs are made up of segments where switching is done based solely on the VPI, as shown in Figure 2-2. In this example, VCC 1 consists of two VCLs (VCL1 and VCL2).

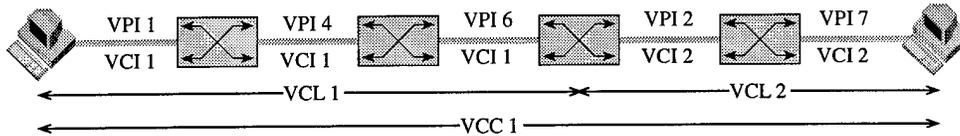


Figure 2-2: Virtual Circuit and Virtual Path Switching

## 2.2 Data Partitioning

Initial DIS exercises centered around single-site networks, consisting of small numbers of simulators (< 10) with relatively few entities (< 1000) [20]. However, recent exercises such as the Synthetic Theater of War-Europe (STOW-E) have shown this expectation to be severely inadequate, and future exercises are being planned with over 100,000 entities spread across 50 sites [4].

In DIS exercises of such magnitude, the main constraint is the limited number of times per second a computer can service network interrupts and still be able to simulate its entities [8, 27]. To allow DIS exercises to scale as envisioned, some mechanism must be used to reduce the amount of unwanted data a simulation host receives during the exercise.

The goal of data partitioning (also called relevance filtering) is to present to each host on the network the smallest subset of all PDUs such that the simulation running on that host receives all information pertaining to those entities that it can detect. This may mean that some small amount of extraneous information is presented to the host, but this is acceptable; what is not acceptable is the host not receiving information it needs concerning an entity.

Several research projects are currently underway seeking ways to partition the data in DIS exercises. In the sections to follow I will cover the three most prominent: RITN, JPSSD, and NPSNET.

### 2.2.1 RITN

The Real-Time Information Transfer and Networking (RITN) is a joint Advanced Research Projects Agency (ARPA)-Defense Modeling and Simulation Office (DMSO) sponsored program that is part of ARPA's Synthetic Theater of War (STOW) program. Located at the Massachusetts Institute of Technology, the RITN program is providing basic technology development in high speed wide area networks, network security, and Application Control Techniques (ACT). The RITN program has been underway since late 1994.

The primary goal of the RITN research group with respect to DIS is to provide scalability for DIS exercises by developing protocols and algorithms to permit construction of simulations which can operate successfully in exercises of any scale. To achieve this, they have developed a specialized system architecture required at each exercise site (Figure 2-3) [4]:

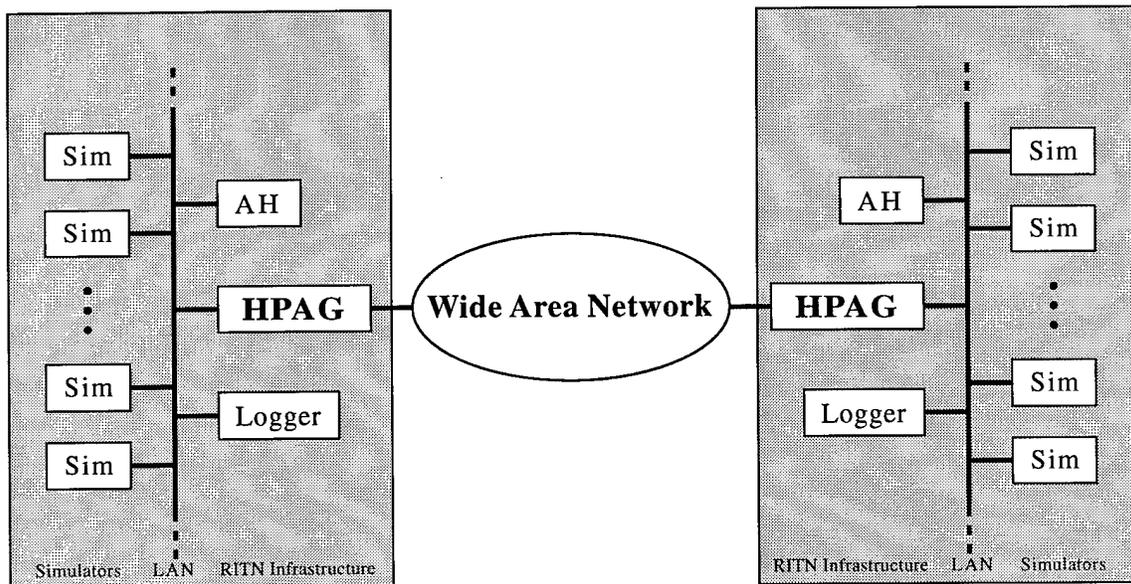


Figure 2-3: RITN System Architecture

The High-Performance Application Gateway (HPAG) serves as a "gateway" to the Wide Area Network (WAN), and handles Internet Protocol (IP) multicast group traffic routing [26]. The Agent Host (AH) is responsible for establishing IP multicast groups based on the simulations' data requests (matching subscribers for data with publishers of that specific data). A data logger is also employed (but not necessary) for traffic analysis [29].

In the Fall of 1995, the RITN program conducted a series of tests referred to as Engineering Demonstration 1A (ED-1A). The battlefield used for these tests was partitioned *a priori* into a uniform array of square, two-dimensional grid cells, with an IP multicast group uniquely associated with each grid cell. Several different grid sizes, ranging from 1.25 km to 5 km, were tested, and resulted in an average PDU reduction at each workstation of approximately 53% [29].

While demonstrating the efficacy of IP multicast groups in reducing irrelevant network traffic, the actual improvement at each workstation was significantly less due to the inherent limitations of current commercial network interfaces. In a typical Ethernet shared-media environment, most multicast packets are actually received and processed by a workstation's kernel; those packets of interest are passed on to the application, while the rest are rejected. This is obviously an improvement over a broadcast paradigm (in which all packets are passed on to the application), but CPU cycles are still being wasted while the kernel examines each packet. It is estimated that better filtering of multicast packets in hardware (either at the interface, or within the network) could reduce the number of packets handled at each workstation by a factor of two or more [29].

### **2.2.2 JPSD**

The Army's Joint Precision Strike Demonstration (JPSD) program incorporates the concept of "interest management", wherein client simulators are supported by a Run Time Gateway (RTGW), which manages the client interest expressions and communicates with other RTGWs to pass PDUs to those simulators that have an interest in them. Similar to the RITN

structure, the JPSD RTGW acts as a “gateway” to the rest of the virtual world, as shown in Figure 2-4 [20].

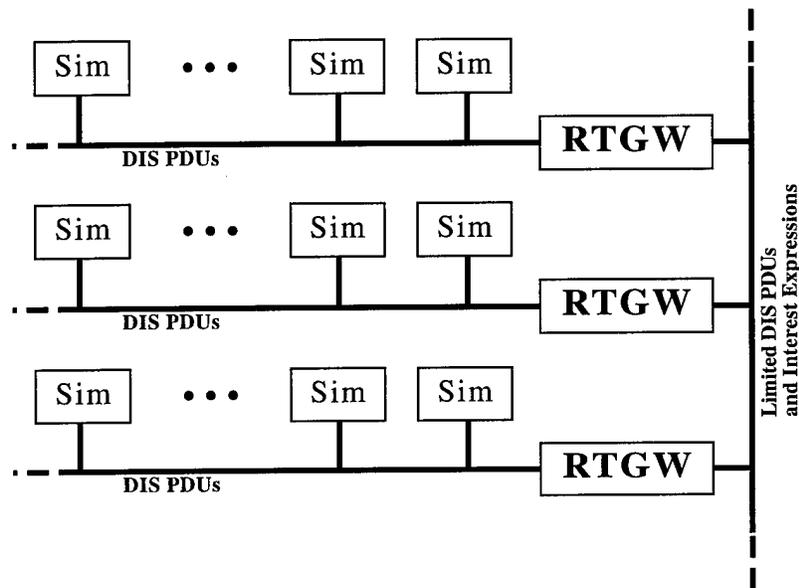


Figure 2-4: JPSD System Architecture

Each segment in the JPSD architecture is associated with a unique IP multicast group, and the RTGWs communicate with each other via multiple IP multicast groups (one for interest expressions, and a number for PDU transmissions). In this manner, the majority of the network burden is shifted from the simulator hosts to the RTGW; each host need only subscribe to one IP multicast group, and conceptually each simulator views the RTGW as a single, giant host capable of simulating all the external entities in which the simulator is interested.

To take maximum advantage of the RTGW infrastructure, JPSD requires *a priori* knowledge about the exercise in order to properly segment the simulators, associating those with similar interests behind a common RTGW. Ad hoc joining of simulators to the exercise degrades the advantages of the RTGW topology.

The JPSD RTGW also employs a Ground Truth Database (GTD) which tracks each entity and allows calculation of  $\Delta$ -values, such as when an entity has moved, or has changed

appearance. The GTD enables the RTGW to perform a limited culling of PDUs, allowing simulators to request updates either more or less frequently than they would typically receive.

The Interest Expression (IE) schema of JPSPD allows not only a geographic partitioning of the data (as with RITN), but virtually any other type of partitioning as well with suitable IEs. For example, a simulator could request to receive only information about a certain type of entity, or only entities from a certain simulator [20].

### **2.2.3 NPSNET**

The Naval Postgraduate School is developing a grid-based partitioning schema for use with its NPSNET 3D vehicle simulator. Similar to the RITN structure, the NPSNET system associates a unique IP multicast address with each grid; however, the NPSNET system uses 4-km radius hexes instead of squares. As with RITN, *a priori* knowledge of multicast addresses is necessary for a simulator to know which addresses to subscribe to. NPSNET does not employ a dedicated server (such as RITN's Agent Host) to handle multicast group subscription and management; instead, the oldest entity within a multicast group is considered the "leader" of that group and responds to all join requests sent by entities wishing to receive information concerning that group. While this reduces the overhead and complexity of the NPSNET exercises, it causes additional overhead on the simulator that is responsible for the current "leader" of a multicast group [15].

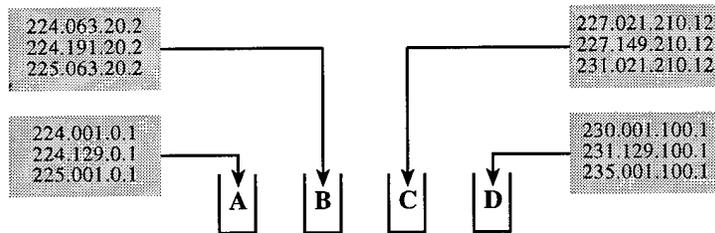
### **2.3 IP Multicast**

In each research effort discussed so far, the method used for selectively broadcasting PDUs is IP multicast. There are several reasons for this:

- 1) IP Multicast is an IETF standard (RFC 1112) and is supported by a variety of operating systems including SGI's Irix and Sun's Solaris [9].
- 2) Most modern computer systems employ some level of multicast group filtering relatively low on the network protocol stack, resulting in early discard of irrelevant packets [29].

- 3) From a DIS perspective, IP multicast network addressing closely resembles the broadcast medium for which DIS was developed; minimal modifications to DIS applications are necessary [20].
- 4) Protocols such as Internet Group Management Protocol (IGMP) and the Multicast Backbone (MBONE) allow limited multicast-group transmissions across WANs including the Defense Simulation Internet (DSI) and the commercial Internet [9].

As was shown in ED-1A, however, most interface hardware only supports schemes for hashing IP multicast addresses into a relatively small number of bins, accepting or rejecting packets based on bin number, as shown in Figure 2-5. While the exact number of bins varies between manufacturers (and also between machines from the same manufacturer), this hashing scheme means that in order to receive packets from a particular multicast group, all packets whose address hashes into the same bin must be processed by the kernel as well. The large number of multicast groups necessary for relevance filtering (ED-1A used over 400 [29]) lessen the effectiveness of this approach.



*Figure 2-5: Notional IP multicast group address hashing. Example based on IP-to-Ethernet multicast hashing, in which the low-order 23 bits of the IP multicast group address determine which bin a packet is placed into [9].*

## 2.4 Conclusion

The preceding summary of current DIS research efforts in the area of data partitioning indicates that all three efforts have selected geographic grid-based partitioning as the model

that will be used. Primary focus has been in the area of IP multicast group allocation, with relatively little notice paid to emerging technologies such as the Xpress Transfer Protocol (XTP) or Asynchronous Transfer Mode (ATM) backbones (beyond serving as a lower protocol for IP). Further, efforts are aimed primarily toward a static partitioning scheme, which requires significant pre-simulation setup and coordination. The next chapter will attempt to address some of these problems, both from the perspective of harnessing emerging network technology and in reducing (or eliminating) pre-simulation preparation.

### ***3. Approach***

Previous research efforts in the area of relevance filtering have primarily centered around a fixed geographic partitioning scheme, using IP multicast as the underlying network protocol. While attempting to capitalize on the success of these efforts, I seek to improve on them by exploring other network transport options as well as examining a dynamic battlespace partitioning method to optimize relevance filtering.

#### **3.1 Network Protocols**

As was shown in the previous chapter, all research efforts to date have used IP multicast as the underlying network protocol, with each geographic "cell" being assigned a unique multicast address. This approach has the advantage of being easily transportable from network to network; however, as was shown in Chapter Two, it is not always the optimal solution. Given my initial research goal to develop a solution for an ATM-to-the-desktop system, I have three different options: Local Area Network Emulation (LANE), IP-over-ATM, and native ATM. I will briefly cover the advantages and disadvantages of each, and explain my rationale for choosing to use native ATM.

##### ***3.1.1 Local Area Network Emulation (LANE)***

Fore Systems' ATM adapters support the ATM Forum's LANE 1.0 protocol. By using LANE, one or more "virtual LANs", known as emulated LANs (ELANs), can be set up on a single network, with traffic on each LAN being distinct from the others. ELANs transparently support multicast and broadcast. To applications, the ELAN appears as another network interface, and no modifications are required to software beyond specifying this new interface. Using ELANs would allow a heterogeneous simulation environment containing both bridges to other LANs, legacy Ethernet or FDDI/CDDI systems (depending on the ATM switch configuration), and ATM-based systems, and could be bridged using standard equipment for use across WANs. In

addition, ELANs support a limited level of security, in that the systems which are permitted to join the ELAN can be limited to a predefined list.

LAN emulation is expensive, however. The LAN emulation software is run on each host, incurring CPU overhead for the network operations. In addition, there is a significant overhead associated with LAN emulation, as Figure 3-1 shows; a typical 144-byte entity state PDU (ESPDU) requires 265 bytes to be transmitted. Table 3-1 shows the overhead for the three most prevalent PDUs in a DIS exercise.

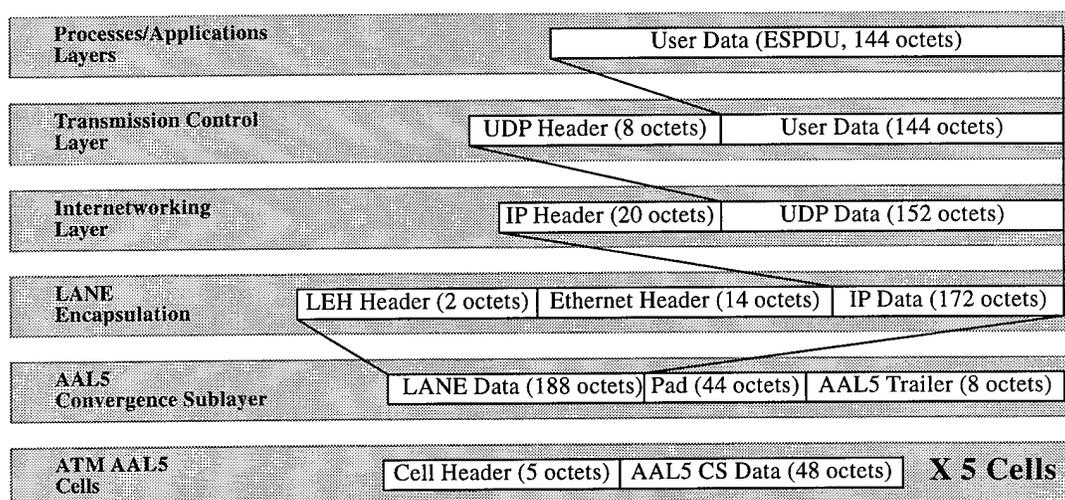


Figure 3-1: Encapsulation of a 144-byte ESPDU using LANE. The pad in the AAL5 Convergence Sublayer is required to ensure 48-byte segmentation. [2, 13, 14]

Table 3-1: Overhead associated with LANE.

DIS PDU Type	Length (bytes)	UDP	IP	LAN Emulation	ATM	Total Length	% Overhead
Entity State	144	8	20	16	77	265	45.6
Fire	88	8	20	16	27	159	44.6
Detonation	104	8	20	16	64	212	50.9

In addition to CPU and network overhead, either an ATM host on the network or the ATM switch itself must run software to handle ELAN administration and maintenance. This additional requirement, in conjunction with the packet overhead shown above, results in significantly lessened network throughput, as Table 3-2

illustrates. For example, LANE throughput on a WebForce Indy (65.2 Mbps) is over 45% less than that achieved using IP-over-ATM on the same architecture (120.3 Mbps).

*Table 3-2: Performance Figures for ATM adapters available at AFIT. MTU is the Maximum Transmission Unit. SPANS is Fore Systems' proprietary signaling protocol used for establishing ATM connections between host adapters, and indicates IP-over-ATM throughput [13].*

ATM Adapter		MTU Size (bytes)	Window Size (bytes)	Buffer Size (bytes)	Demonstrated Throughput (Mbps)
<b>GIA-200E</b> (WebForce Indy)	LANE	1516	64	32768	65.2
	SPANS	9188	64	8192	120.3
<b>ESA-200E</b> (Indigo 2 Impact)	LANE	1516	48	24576	58.5
	SPANS	9188	64	12288	84.8
<b>VME-200E</b> (Onyx)	LANE	1516	48	24576	59.4
	SPANS	9188	64	49152	117.6

### 3.1.2 IP-Over-ATM

Another option is IP-over-ATM. Fore Systems' ForeThought 4.0 software implements both RFC 1577 Classical IP and their proprietary implementation of TCP/IP, known as FORE IP. The conceptual view of IP-over-ATM showing the successive encapsulation layers is shown in Figure 3-2 [11]. Because the existing TCP/IP interface is used, the underlying ATM infrastructure is transparent to the application; this allows the use of generic TCP/IP structures with little or no modifications.

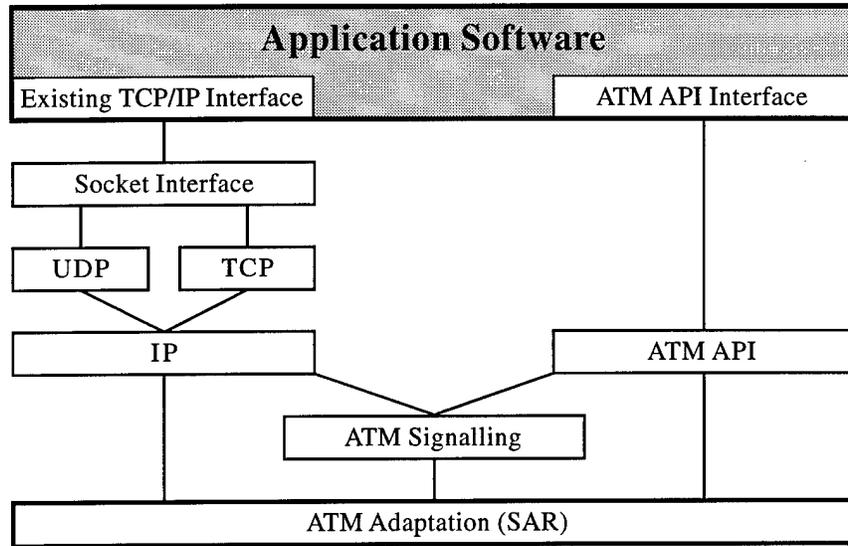


Figure 3-2: ForeThought implementation of TCP/IP and ATM API protocols.

By using IP-over-ATM, any of the existing partitioning schemes could be used as a base for this research. IP is the most ubiquitous of transport protocols, forming the basis of the DSI network and the commercial Internet, so any partitioning scheme developed could be used on a number of different networks. JPSD, for example, was developed using IP-over-ATM, and has been successfully bridged to other networks. FORE IP supports the IETF standard multicast, albeit with the same multicast limitations that were discussed in the previous chapter. IP multicast packets can be transmitted over WANs through the use of multicast routers, which will “tunnel” the multicast group information as necessary for transport across the Internet or other WAN.

IP-over-ATM is computationally less expensive than LANE, since the LANE encapsulation operations are not required. However, each PDU sent over an IP network (ATM or otherwise) incurs a fixed packet overhead due to the requirements for encapsulation. Figure 3-3 shows the ESPDU from Figure 3-1 as it is successively encapsulated for IP-over-ATM down to the ATM AAL5 cell level, and Table 3-3 lists the overhead associated with each of the three PDUs in Table 3-1 when transmitted via an IP-over-ATM network. Note that the detonation PDU requires 47 bytes of pad in

the AAL5 Convergence Sublayer, resulting in over half the total transmitted information being overhead.

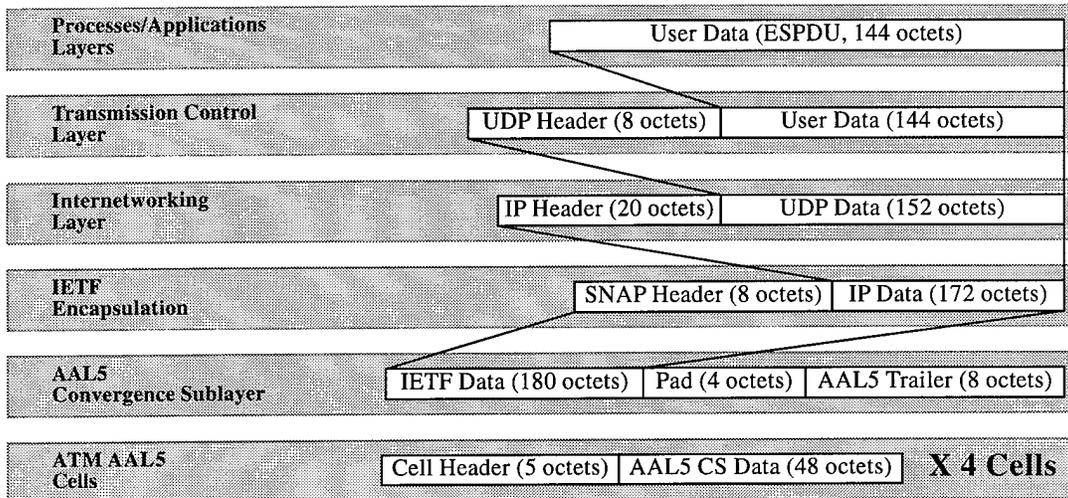


Figure 3-3: Encapsulation of a 144-byte entity state PDU using IP-over-ATM. The pad in the AAL5 Convergence Sublayer is required to ensure 48-byte segmentation. [2, 14]

Table 3-3: Overhead associated with IP-over-ATM.

DIS PDU Type	Length (bytes)	UDP	IP	ATM	Total Length	% Overhead
Entity State	144	8	20	40	212	32.1
Fire	88	8	20	43	159	44.6
Detonation	104	8	20	80	212	50.9

### 3.1.3 Native ATM

The final option available is to use native ATM as the network transport mechanism. At first glance, native ATM appears to be the obvious choice: By removing the IP encapsulation and going straight to the ATM adaptation layer (as shown in Figure 3-4) the per-PDU overhead drops. In those cases where the PDU is aligned on a 48-byte boundary within the convergence sublayer, the decrease is fairly significant, such as with the fire PDU (Table 3-4).

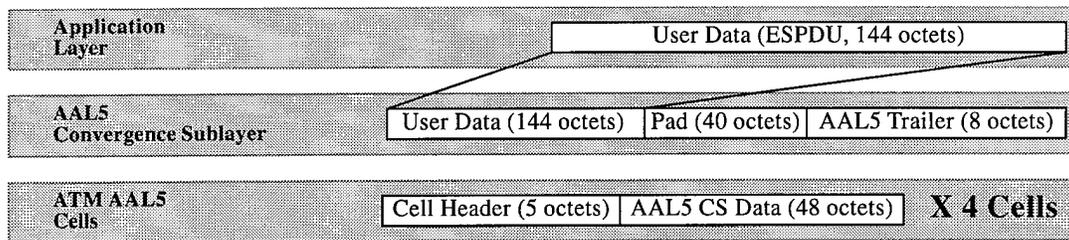


Figure 3-4: Encapsulation of a 144-byte entity state PDU using native ATM. The pad in the AAL5 Convergence Sublayer is required to ensure 48-byte segmentation. [2, 14]

Table 3-4: Overhead associated with native ATM.

DIS PDU Type	Length (bytes)	ATM (AAL5)	Total Length	% Overhead
Entity State	144	68	212	32.1
Fire	88	18	106	16.9
Detonation	104	55	159	34.6

Complementing the decreased overhead (more PDU data per ATM cell), the host also has less decapsulation to do, resulting in increased data rates within the host itself. In his testing for the JPSD project, Dr. Edward Powell determined that on average, a host running with a Fore Systems ATM adapter could process approximately 2.5 times as many native ATM packets as it could IP-over-ATM packets before experiencing packet loss due to interface congestion [19].

Another advantage to using native ATM is that an ATM network, by nature, is already WAN-adapted; no further encapsulation or “tunneling” of information is required for transport across ATM WANs, since no logical distinction is made between a VPI/VCI that ends at a host and a VPI/VCI that connects ATM switches. Each host on an ATM network is identified by a unique 20-byte sequence, which identifies the switch and port for that host. Fore Systems has implemented the International Code Designator (ICD) ATM addressing scheme, as shown in Figure 3-5. For a local (on the same switch) ATM connection, however, only a Network Service Access Point (NSAP) and an Application Service Access Point (ASAP) are needed. The NSAP is an 8-byte

address which is analogous to an IP address, and refers to the specific switch and port; the ASAP is numeric, similar to an IP port number, and indicates the connection over which a specific application will accept connections.

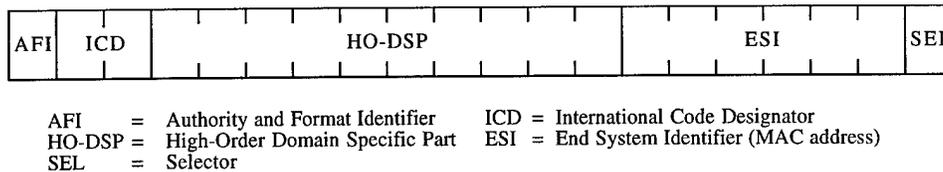


Figure 3-5: ICD ATM addressing scheme

Native ATM supports a form of multicasting as well. On AFIT's switch, up to 4096 multicast groups per switching fabric can be established (depending on the memory model selected), with up to four fabrics installed, for a total of 16,384 multicast groups. This is a bit misleading, though, in that ATM multicast groups are not many-to-many, but are one-to-many, with sender-initiated joins. This is required because ATM is inherently a connection-oriented (point-to-point) system. To support the broadcast paradigm of DIS, each simulation host could open a multicast connection with every other host in the exercise, which would require N-1 inbound connections at each host for an exercise with N hosts. A better option is to use one or more rebroadcast hosts, which will serve as the root of a multicast tree and will rebroadcast all PDUs to all hosts participating in the exercise, as shown in Figure 3-6. In this manner, each host requires only one outbound connection and one inbound connection for each rebroadcast host. While this scheme may seem inefficient, since each PDU must be transmitted twice, the actual cell replication is handled in hardware by the switch, resulting in negligible additional latency when compared with IP-over-ATM or other broadcast methods.

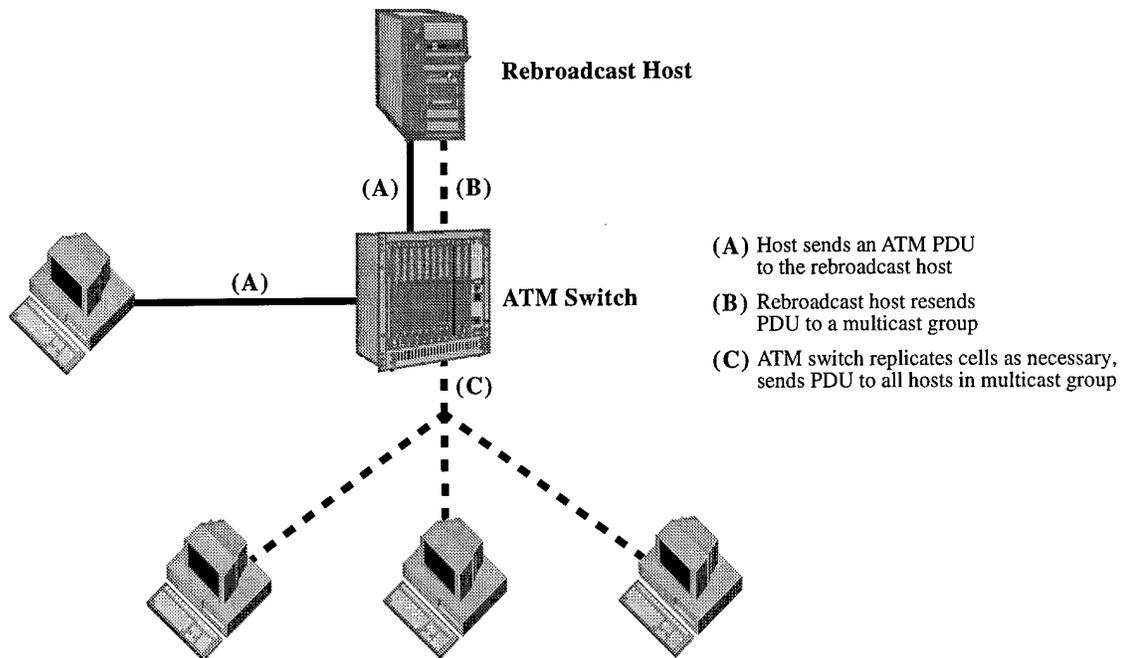


Figure 3-6: Multicast rebroadcasting using an agent.

In the end, I decided to use native ATM as my transport protocol. While some form of rebroadcast agent is necessary to emulate higher-level broadcast abilities, my primary goal is to reduce the CPU overhead on each simulation host as much as possible. By removing the IP or LANE encapsulation overhead, I feel that the best utilization of resources is realized.

### 3.2 Dynamic Partitioning

Early in the thesis formulation I considered several static partitioning schemes. I examined the fixed geographic grid of RITN and NPSNET, and the adaptive scheme implemented with JPSD. While all three methods reduce irrelevant traffic delivered to a host, they all require a large investment in pre-exercise setup and coordination. What has not been implemented is a dynamic, run-time-adaptive partitioning, wherein intelligent simulation management agents ("managers") monitoring the exercise would determine the ideal partitioning of the network traffic based upon the state of the exercise at any given time. With our decision to use native ATM as the network

protocol, it becomes clear that the rebroadcast agents can also serve as these managers, with each handling a portion of the network traffic based on adaptive segmentation criteria. Further consideration of this problem led us to adopt a "supermanager", which is running at a predetermined location on the network. The supermanager maintains state information on the current partitioning scheme as well as address information for each of the rebroadcast managers. In this manner, a simulation can join the exercise at any time with no *a priori* information needed beyond the address of this supermanager. The supermanager determines which rebroadcast agents are responsible for the joining simulation's entities and communicates the necessary address information to the joining simulation, which then directs all future PDUs from those entities to the specified rebroadcast agent. An additional agent is necessary on the simulation host to track entity-address mappings, and to change those mappings as an entity moves around the virtual battlespace.

To reduce the computational overhead on the individual components, to avoid unnecessary network traffic, and to provide a clear demarcation of simulation management responsibilities, all components are as self-contained as possible. The rebroadcast agents themselves determine when their partition should be divided, based on such factors as their CPU load, network interface load, or entity count. When a partition divide is indicated, the agent communicates this to the supermanager. Included in this information are "hints", if possible, indicating how the partition should be split; for example, if the partition criteria is geographic, then the agent could include information about where the highest concentration of entities is, thus aiding the supermanager in determining an optimal split.

### **3.3 Component Hierarchy**

Having each component be self-contained implies that an actual hierarchy of these agents is necessary, with each component having specialized knowledge about the virtual world. The superagent becomes the Battlespace Manager (BSM), responsible for overall battlespace partitioning and tracking of address information for the rebroadcast

agents. The rebroadcast agents become Partition Managers (PMs), rebroadcasting PDUs to interested simulations. Finally, the simulation agent becomes the Daemon Manager (DM), which tracks the address each entity sends to. These three components interact as shown in Figure 3-7.

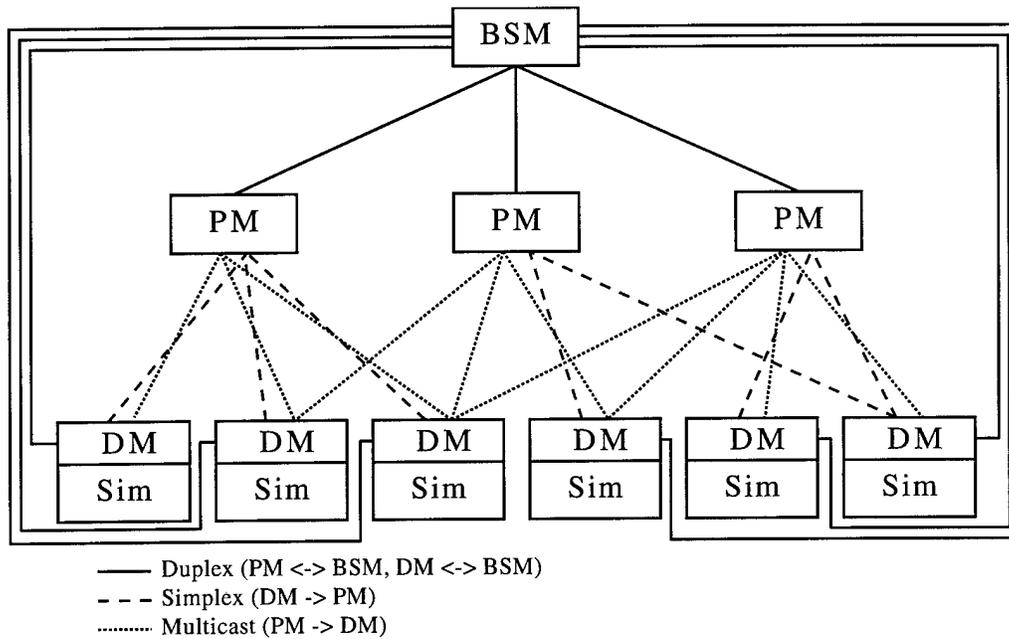


Figure 3-7: Component interaction

### 3.3.1 Daemon Manager (DM)

The DM is responsible for addressing and transmitting PDUs received from the simulation. The conceptual view of the DM depicting network interaction is shown in Figure 3-8.

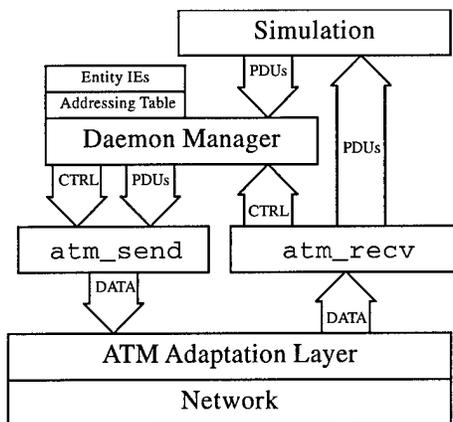


Figure 3-8: *DaemonManager* conceptual view

Each simulation will have its own DM. The DM maintains an entity lookup table, which contains information indicating which PM is managing each entity that is being controlled by the simulation. If no address information exists for a specific entity, all PDU's from that entity are sent to the BSM.

The DM generates an Interest Expression (IE) for each entity it maintains address information on. The DM transmits this IE to the PM for that entity three times during an exercise:

- 1) When the entity is first assigned to a PM;
- 2) When the IE for that entity changes (due to sensor degradation, environment, or damage);
- 3) When the PM controlling the entity changes.

### 3.3.2 *Partition Manager (PM)*

The PM acts as the root of a multicast tree, rebroadcasting all PDU's that it receives. When a PM is notified (by the BSM) that a DM wishes to join the multicast tree, it opens a branch connection on its multicast tree for that DM. When a PM is notified that a DM is no longer interested in receiving its PDU's (for example, when an

entity is destroyed, or when the entity moves beyond sensor range of the partition), it prunes that specific branch only. The PM conceptual view is shown in Figure 3-9.

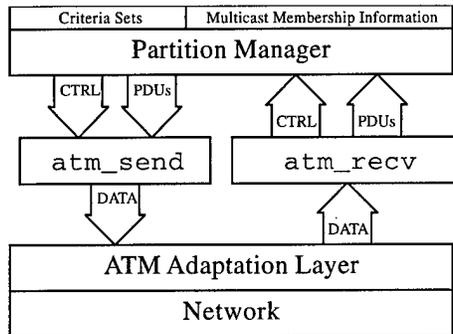


Figure 3-9: PM conceptual view

Each PM is assigned a criteria set by the BSM. This criteria set determines the parameters for which that PM has responsibility; for example, a PM might be responsible for all entities within a certain geographic region. All PDUs received by the PM are tested against this criteria. If an entity no longer meets this criteria, the PM forwards the PDU to the BSM in addition to sending the PDU to its multicast group.

When a PM receives an IE from a DM, it calculates any modifications to the IE (for example, expanding the range of a geographic IE to encompass the entire range of the PM) and sends the (possibly) modified IE to the BSM.

If the number of entities a PM services exceeds a given threshold (either in terms of total count, total network traffic, or other predefined discriminator) the PM requests to be split by the BSM. If possible, information suggesting an optimal partition division is included in the request to the BSM.

### 3.3.3 Battlespace Manager (BSM)

The BSM creates the initial PM for an exercise, and creates new PMs as necessary to divide an existing partition. The BSM calculates each PM's criteria set and passes this to the PM as necessary. Figure 3-10 shows the BSM conceptual view.

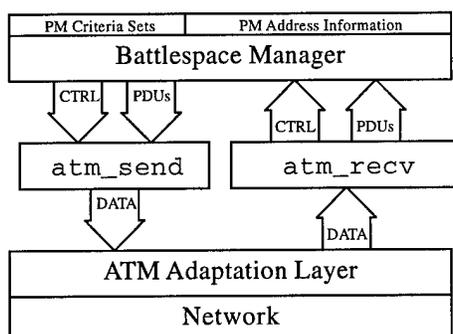


Figure 3-10: BSM conceptual view

When the BSM receives an entity state PDU (from either a DM or a PM) it determines which PM has responsibility for this entity and notifies the DM controlling that entity of the new address information.

Upon receiving an entity's IE from a PM the BSM determines which PMs have information in which the entity is interested and notifies those PMs to add the receiving address of the entity's DM to their multicast trees. All other PMs are notified to remove the entity from their multicast tree if they were sending to it previously.

### 3.4 ATM-Specific Issues

With the choice to use native ATM, two other issues present themselves: Quality of Service (QoS) to request, and the Maximum Transmission Unit (MTU) to allocate for each connection.

### 3.4.1 Quality of Service (QoS)

QoS requests are specific to ATM, and indicate the expected bandwidth requirements for each ATM connection. When opening an ATM connection using the AAL5 API, these requirements are specified in terms of peak, mean, and mean burst, as shown in Table 3-5. Each parameter is further refined into a “target” and “minimum” value. When the connection request is received by the switch, it attempts to allocate resources (such as cell buffers) to meet the target bandwidth request. If insufficient resources are unavailable to provide the target bandwidth, the minimum request is considered. If the system is unable to provide even the minimum requested bandwidth, the connection request is refused. A user can request zero bandwidth, indicating that the connection will carry low-priority, loss-tolerant traffic; such connection requests are never refused, but are accepted with the understanding that *all* data carried on that connection is subject to loss during periods of network congestion.

Table 3-5: Bandwidth reservation parameters for an ATM connection request.

Parameter	Meaning
Peak Bandwidth	Maximum burst rate at which a source produces data (kilobits/second)
Mean Bandwidth	Average bandwidth expected over the lifetime of the connection (kilobits/second)
Mean Burst	Average size of packets sent during periods of peak bandwidth utilization (kilobits)

While requesting zero bandwidth would guarantee a connection, I feel that some bandwidth reservation should be made based on the expected PDU generation rates. Since the ESPDU accounts for up to 98% of the traffic in a DIS exercise [21], I used 144 bytes (1152 bits) as the average packet size. To determine the rate of generation, I used the results of research performed at the Naval Postgraduate School, which postulates a maximum of 1000 entities per simulation host [15, 30]; therefore, I chose 500 entities to be a high-end average count per host. ESPDUs are required to be sent ever 5 seconds at a minimum, so this results in at least 500 ESPDUs every 5 seconds, or

an average of 100 ESPDUs/second. This gives a mean bandwidth value of 112.5 kilobits/second, with a mean burst value of 1.125 kilobits/second:

$$\text{Mean Bandwidth} = \frac{100 \text{ ESPDUs}}{\text{second}} \times \frac{1152 \text{ bits}}{1 \text{ ESPDU}} \times \frac{1 \text{ kilobit}}{1024 \text{ bits}} = 112.5 \frac{\text{kilobits}}{\text{second}}$$

$$\text{Mean Burst} = \frac{1152 \text{ bits}}{1 \text{ ESPDU}} \times \frac{1 \text{ kilobit}}{1024 \text{ bits}} = 1.125 \frac{\text{kilobits}}{\text{second}}$$

Peak Bandwidth is set at zero. If a simulation desires to send more data than the bandwidth reserved for it, it is accepted on a resource-available basis.

### **3.4.2 Maximum Transmission Unit (MTU)**

The Maximum Transmission Unit (MTU) defines how large the block of information to be sent can be when received at the AAL layer. Data that is larger than this requires segmenting at a higher layer. The Fore Systems interfaces used in this thesis have adjustable MTUs, up to the maximum AAL MTU of 65,535 bytes, but the default is the Classical IP configuration of 9,180 bytes [11]. While it may prove beneficial to adjust this size for DIS PDUs, for this research I chose not to.

## **3.5 Conclusion**

In this chapter I have described the development of a dynamically-partitioned, native-ATM based DIS exercise system. This system uses a hierarchy of simulation managers to handle the battlespace partitioning as well as the requisite translations between ATM's point-to-point model and the broadcast paradigm currently expected by DIS simulators. In the next chapter, I discuss implementation details concerning this hierarchy, and describe some of the problems encountered during implementation.

## **4. Design**

### **4.1 Overview**

All components are implemented in C on Silicon Graphics Workstations. My original goal was to use C++, but incompatibilities in the API libraries provided by Fore Systems required me to use C. I reused code to the maximum extent possible by using a shared library (`libDIS_ATM`). While in some instances this may have resulted in code that was less optimized, I feel that the high degree of modularity allows for easier maintenance and extensions. In the sections to follow I will first discuss features that are common to all components, and then I will discuss each component in detail.

In regard to this thesis, when I refer to a “program” I mean one of the three components (Battlespace Manager, Daemon Manager, or Partition Manager). A “process” is an individual UNIX process running under the umbrella of a program; for example, the Battlespace Manager starts three processes at runtime.

Function prototypes are provided for reference in Appendix A, and Appendix B contains the structure of the 10 new PDU types I created to support this structure.

### **4.2 Common Features**

#### **4.2.1 Shared Memory Arenas**

Each program consists of three or more processes running in parallel, called a *process group*. This is necessary to avoid any one section of the program from blocking out the rest; for example, if a large number of PDUs are being read from the network, there would be no servicing of PDUs waiting to be sent. However, each process needs to be able to access common memory structures, and updates made to these structures by one process must be visible to the others.

To implement this “shared-construct” scheme, I chose to use a *shared memory arena*, which is initialized at startup. Arenas are defined as “a specially allocated area of shared

memory that resembles a memory-mapped file" [23]. Each arena is mapped to the same location in the local address space of each process that shares the arena, and data structures allocated out of this arena are visible to all processes that have joined the arena. Arena information is stored in a file on disk, allowing processes that are created later to be able to map previously-created data structures into their address space (and thus have local visibility into them).

To create a shared-memory arena, each program calls the system command `usinit("filename")`, where *filename* is the full path to a local file. The first process to call `usinit()` will create the file, returning a pointer; subsequent processes calling `usinit()` will join the arena and map the arena into its local address space. The operating system keeps track of which processes are active within an arena, and will close the arena file after all processes have terminated.

Each process allocates data structures out of a shared-memory arena via calls to either `usmalloc()` or `uscalloc()`, adjusted via `usrealloc()` or `usrealloc()`, and removed using `usfree()`.

Memory collision is prevented by the use of locks for all critical areas. When a process wishes to enter a critical area, it attempts to set the lock for that area by calling `ussetlock()`. If the lock is already set by another member of the process group, the process blocks until the lock is free; otherwise, the lock is set and the process continues. When out of the critical area the process calls `usunsetlock()` to free the lock. Multiple processes attempting to set a lock queue in FIFO order [23]. Data structures and locks within an arena are initialized in the `InitMemory()` library call (Appendix A).

#### **4.2.1.1 Shared Data Structures**

There are three data structures that are common to all programs: `buffers`, `ATMinfo`, and `fdlist`. I describe the composition of each and their purpose in this section.

#### 4.2.1.1.1 The buffers Data Structure

The buffers data structure contains data buffers, semaphores, and control registers used when sending or receiving PDUs. The elements of buffers are shown in Table 4-1.

Table 4-1: buffers data structure.

Element	Type	Use
mode	int	Indicates which program (BSM, DM, or PM) initialized this data structure.
pids	int [3]	The process IDs of the three main processes (main, SendDaemon, ListenDaemon).
num_bufs	int	How many data buffers are contained in this structure.
ctrl_semaphore	int	The semaphores used with the main process and the SendDaemon.
control	int *	A pointer to an array of two integers which serve as control registers for passing messages between the processes; one is allocated for the main process, and one for the SendDaemon.
data	data_t *	A pointer to a block of num_bufs records, each of which is a structure itself (see below).
lock	unlock_t [2]	The process locks. Two locks are defined: one is set when modifying the data buffers, the other when modifying the control registers.

The control registers are bit-masked flags that contain a code indicating the operation to be performed (for example, send a specified buffer), the buffer number, and the fdlist index. The processes communicate control information about a specific buffer using these registers.

The variable num\_bufs indicates how many send/receive data buffers are to be allocated. The structure of each send/receive data buffer is shown in Table 4-2.

Table 4-2: Send/receive buffer data structure.

Element	Type	Use
size	int	The number of bytes received, or the number of bytes to send.
status	int	Control flag indicating whether a buffer is in use or free
data	char[MAX_PDU]	The data buffer. MAX_PDU is defined as 512 bytes.

#### 4.2.1.1.2 The *ATMinfo* Data Structure

*ATMinfo* contains the ATM NSAP address of the current host and the interface being used by the program. Its composition is shown in Table 4-3.

Table 4-3: *ATMinfo* data structure.

Element	Type	Use
interface	char *	A string pointer to the interface being used (i.e. "/dev/fa0")
address	Atm_address	The 8-byte ATM NSAP.

#### 4.2.1.1.3 The *fdlist* Data Structure

The *fdlist* structure maintains information on all open file descriptors (fds). For the Daemon Manager and the Partition Manager, the *fdlist* tracks the number of entities that are addressed to a particular fd. The Partition Manager also uses the *fdlist* to track the entities that are using that fd. Table 4-4 lists the elements of *fdlist*.

Table 4-4: *fdlist* data structure.

Element	Type	Use
fd	int	The file descriptor returned by the system.
using	int	The number of entities either transmitting on this fd (DM) or receiving on this fd (PM). This field is unused by the BSM.
eids	eid_t *	A pointer to an array of entity IDs. Used by the PM only.
ATM	Atm_endpoint	The ATM address (NSAP and ASAP) to which this fd is connected.

#### 4.2.2 Process Flow

Each program utilizes the same basic process flow, as shown in Figure 4-1; a discussion of each process in this flow follows.

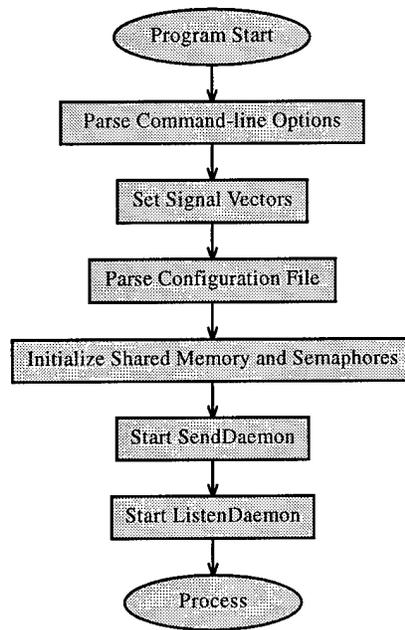


Figure 4-1: Common process flow.

#### 4.2.2.1 Parse Command-line Options

Each program recognizes three command-line options:

- c Specifies a configuration file different than the system default (BSM.configure, DM.configure, or PM.configure, depending on the program).
- h Display a usage message and exit.
- q Terminate a running program. This is necessary since in most cases the processes will be run in the background, and in some cases will spawn off multiple subprocesses during their existence. Terminating the main program (via a Control-C if it is running in the foreground, or by sending the process a SIGKILL signal) will not necessarily terminate all running subprocesses, and will not clean up the semaphores. This option provides a graceful way to clean up and terminate all processes.

Additional command-line options are ignored, except with PMs, which require additional information to be specified on the command line. These options are covered in the PM-specific section below.

#### 4.2.2.2 Set Signal Vectors

The UNIX operating system allows processes and users to interact via the software signal system (for example, by executing the `kill` command on the command line, or the `kill()` function within a program). A process responds to a signal based upon the signal vector associated with that signal. By changing the value of that signal vector, the action associated with that signal can be changed. Table 4-5 lists the signals that are intercepted, along with the default actions associated with them. In each case, the vector is set to execute the `ShutDown()` function, which will ensure all processes are terminated and all memory structures and semaphores are removed.

Table 4-5: Signals vectors changed.

Signal Name	Signal Number	Default Signal Action	Signal Event
SIGHUP	1	Exit	Hangup
SIGINT	2	Exit	Interrupt
SIGTERM	15	Exit	Terminated

#### 4.2.2.3 Parse Configuration File

Most program options are contained in a configuration file, which each program attempts to parse at run-time. The configuration file is optional for the PM; if no file is specified, or the program cannot open the file, the program gives a warning and continues. Both the BSM and the DM require a configuration file. The default configuration files are `BSM.configure` for the Battlespace Manager, `DM.configure` for the Daemon Manager, and `PM.configure` for the Partition Manager, and are expected to be in the same directory as the program executable (unless an alternate configuration file is specified on the command line, in which case a full path is required).

The configuration file entries common to all programs are listed in Table 4-6, along with the default parameters for those entries. If a specific entry is not contained in the configuration file, the default value is used.

Table 4-6: Common configuration file entries.

Option	Action	Default Value
BSM_SAP	Assign the ASAP to connect with on the host running the BSM.	5001
BUFFERS	Specify the number of memory buffers to allocate for sending and receiving PDUs	50 (BSM) or 100 (DM/PM)
INTERFACE	Specify the interface to use for sending and receiving PDUs	/dev/fa0

Each program also has configuration options unique to it, which are listed in program-specific sections below.

#### 4.2.2.4 Initialize Shared Memory and Semaphores

The `InitMemory()` function handles the memory allocation and initialization for the `buffers` data structure and returns a pointer. The first process calling `InitMemory()` passes a `CREATE` flag, which initializes the arena file for the process group and allocates memory for `buffers`. After the data structure has been allocated and initialized, the `usputinfo()` function stores the address of `buffers` into the arena file. Subsequent processes pass the `JOIN` flag to `InitMemory()`, which registers the new process as a member of the arena and returns the pointer to the previously-allocated `buffers` via a call to `usgetinfo()`. In this manner, programs can dynamically join and leave the arena without having a predetermined memory location for `buffers`.

#### 4.2.2.5 Start SendDaemon

The `SendDaemon` runs as a separate process via the `sproc()` command, so that it may run asynchronously as necessary to transmit PDUs. At startup, the `SendDaemon` enters an infinite loop, blocking on the send semaphore until another processes signals it by setting the semaphore to unblock the `SendDaemon`.

The `SendDaemon` determines which buffer to send, and where to send it to, by examining the bit fields in its control register. Data is sent by calling `atm_send()`.

Once the data has been sent, the send semaphore is reset and the SendDaemon will block again unless another process is waiting to send. Figure 4-2 shows the SendDaemon's process flow. The SendDaemon will run until it is terminated by the Shutdown() procedure.

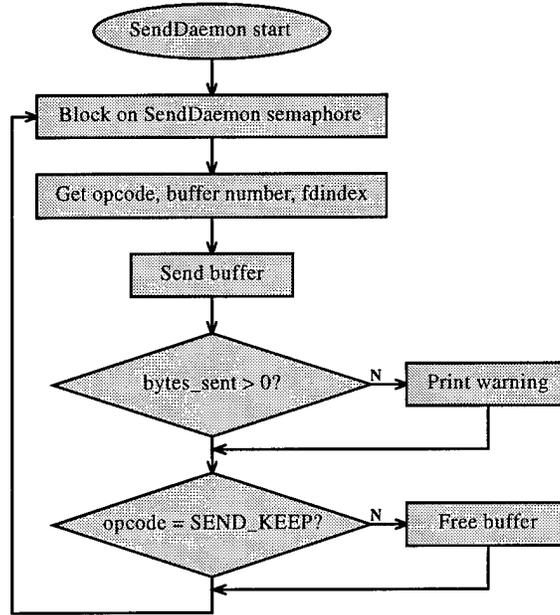


Figure 4-2: SendDaemon process flow.

#### 4.2.2.6 Start ListenDaemon

The ListenDaemon also runs as a separate process via the `sproc()` command, so that it may run asynchronously to handle incoming connection requests. The ListenDaemon assigns each connection request a new `fd` (via a call to `GetFD()`) and starts a ReceiveDaemon to service that connection. The ListenDaemon then returns to listening for incoming connection requests.

ATM requires applications to agree that connections will be simplex (one-way), duplex (two-way), or multicast. Since each of the three programs expects to receive a different type of connection request (see Figure 4-3), ListenDaemon examines the mode field of `buffers` to determine what sort of connection request to expect. Figure 4-3 shows the ListenDaemon process flow.

The ListenDaemon runs until it is terminated by the Shutdown() procedure. ListenDaemon terminates all ReceiveDaemons prior to terminating.

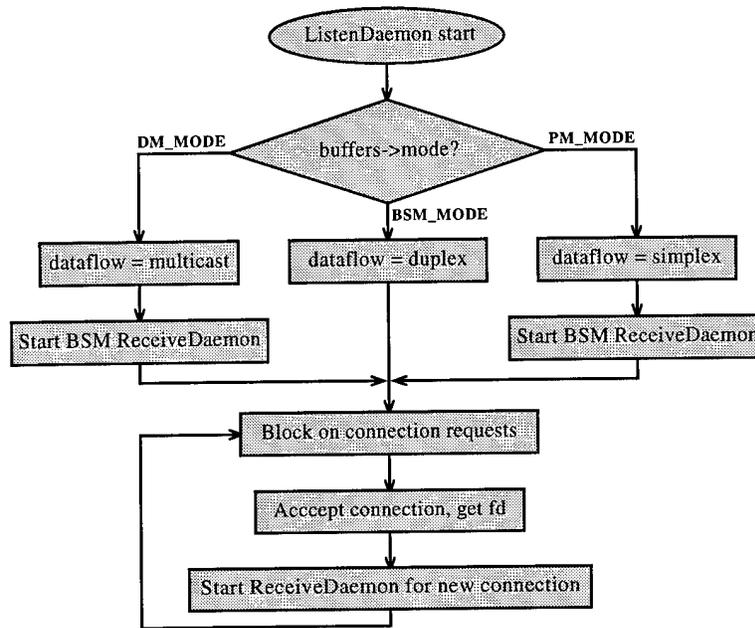


Figure 4-3: ListenDaemon process flow.

#### 4.2.2.6.1 ReceiveDaemons

Since ATM is a connection-oriented protocol, a separate fd is required for each incoming connection. A separate ReceiveDaemon is started for each connection. The ATM API passes PDUs to the ReceiveDaemon when all cells have been received from the network by the ATM adaptation layer.

A special situation occurs when a PM closes a branch of its multicast tree. The PM cannot close a branch remotely, so instead it sends an MCAST\_CLOSE message to the branch (see section 4.4 below). When the ReceiveDaemon receives an MCAST\_CLOSE message, it closes the fd it is listening on and terminates. Otherwise, it stores the data it has received in a send/receive buffer and marks it for the appropriate recipient (BSM/DM/PM or the simulation). In the case of control messages, the semaphore for the BSM/DM/PM process is set to cause that process to unblock. Figure 4-4 illustrates the ReceiveDaemon process flow.

ReceiveDaemons run until terminated by an MCAST\_CLOSE message, until the remote end of the connection is closed, or until terminated by ListenDaemon.

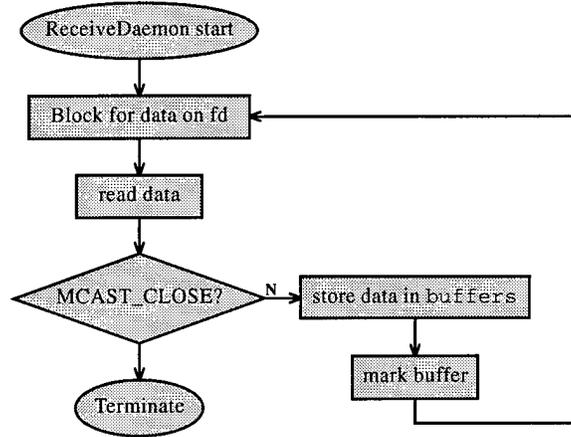


Figure 4-4: ReceiveDaemon process flow.

#### 4.2.2.7 Process

Once all external processes are running, each program enters its main processing loop. These process loops are described in the sections to follow.

### 4.3 Battlespace Manager

#### 4.3.1 Overview

The Battlespace Manager (BM) maintains information about the state of the simulation partition scheme at any given time. When a partition is to be divided, the BSM determines the new criteria for each partition, initializes and starts the new PM (on a remote machine if indicated), and updates the criteria structures on the respective PMs.

#### 4.3.2 Configuration File

Additional configuration file entries for the BSM are listed in Table 4-7; all are required.

Table 4-7: BSM configuration file entries.

Option	Action
CRITERIA_TYPE	The type of criteria being used: Currently, geographic and entity_type are allowed, and geographic is implemented.
PM_COMMAND	The command string to start a PM on a remote machine; it should be a complete pathname.
PM_HOSTFILE	A file containing the list of hosts to run PMs on.

### 4.3.3 Process

The process flow specific to the BSM is shown in Figure 4-5.

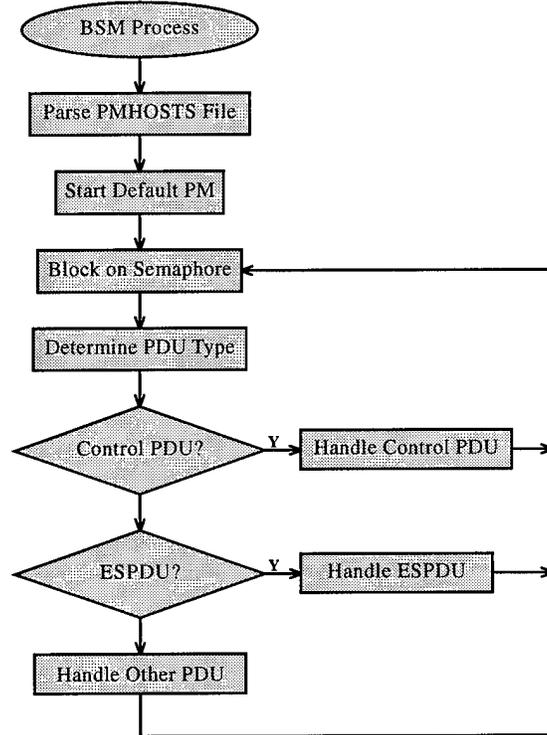


Figure 4-5: Battlespace Manager process flow.

The BSM enters its main processing loop by reading the `PM_HOSTFILE` and storing the file entries into a `PMhosts` array. Each entry into this array is the name of a host where one or more PMs may be run, as well as a priority associated with this host.

Next, the default PM is started on the host with the highest priority in the `PMhosts` array. The criteria set for this PM is initialized to the default for the type of partitioning being used; for geographic partitioning, the initial partition consists of all possible XYZ coordinates. All PMs are started via the `rexec()` command.

Information concerning the default PM (and any other PMs started during the course of the exercise) is stored in a dynamically-allocated array `PMarray`. For each active PM this array stores its criteria set, its ATM address information (NSAP/ASAP), and its `PMid`. `PMids` are sequential and assigned by the BSM when the PM is started.

The BSM then blocks on a semaphore, waiting for PDUs to be received from the network.

#### ***4.3.4 BSM Handling of ESPDUs***

ESPDUs are sent to the BSM in two circumstances:

- 1) By a DM, when the entity first enters an exercise (sends its first ESPDU);
- 2) By a PM, when the entity has transitioned beyond the scope of responsibility of that PM (for example, when an entity has moved beyond the geographic boundaries of that PM).

In either case, the BSM examines the PDU, and based on the criteria scheme being used, determines which PM now has responsibility for that entity by comparing the relevant fields in the ESPDU to the criteria set of each PM. Since the criteria sets of each PM are disjoint, only one PM will match. The BSM encodes the address information for the new in an `ADDRESS_ASSIGN` PDU and sends the PDU to the DM responsible for that entity, either directly if the ESPDU came from the DM, or indirectly (via a PM) if the ESPDU was forwarded from a PM.

#### **4.3.5 BSM Handling of Control PDUs**

The BSM recognizes three types of control PDUs: PM\_REGISTER, PM\_IE, and PM\_SPLIT.

The PM\_REGISTER PDU is sent by a PM that has just started and is ready to receive its criteria set. Included in the PM\_REGISTER PDU is the address information for that PM (both NSAP and ASAP) and the PMid. The BSM extracts the address information from the PDU and stores it in the PMarray for that PMid. The BSM generates a CRITERIA\_ASSIGN PDU containing that PM's criteria set and sends this back to the PM.

The PM\_IE PDU is sent by a PM and contains an entity's interest expression (IE)—what information that entity desires to receive—expanded if necessary by the PM (see the PM section below). The BSM extracts the information concerning the entity from the PDU (entity ID and ATM address information), then compares the IE to the criteria sets for *each* PM. If a PM has data in which the entity is interested, then the BSM sends an MCAST\_ADD PDU to that PM containing the address information for the entity; if the entity is not interested in that PM's data, the BSM sends an MCAST\_DELETE PDU to that PM. It is important that one or the other is sent to each PM, since a PM may have been sending to that entity previously; if the entity's IE has changed such that it is no longer interested in that PM's information, the PM must initiate the closing of the connection.

A PM\_SPLIT PDU indicates that a PM has reached its threshold for servicing PDUs and that its criteria should be divided, with a new PM taking part of the load. If possible, the PM will provide information to assist the BSM in determining an optimal split. For geographic partitioning, the PM provides the XYZ coordinates of the average entity location, and the BSM uses this to determine the boundaries of the two new partitions. The BSM starts a new PM on the host in the PMhosts array with the highest priority, and gives each PM its revised criteria set via a CRITERIA\_ASSIGN PDU.

### 4.3.6 BSM Handling of Other PDUs

Some PDUs sent by an entity do not fit into any PM's scope of responsibility. For example, in a geographic partitioning scheme, those PDUs which do not have a geographic location (such as fire PDUs) cannot be assigned to any PM. To handle this, each DM forwards all PDUs of this type to the BSM, which sends it to all active PMs for rebroadcast. While this may seem inefficient and even contrary to the partitioning goals, evidence has shown that these PDUs account for less than 5% of the overall network traffic during an exercise. In my opinion, the network overhead and latency incurred is less than the processing that would be required to determine which partition these PDUs should be sent to (if an assignment is even possible).

## 4.4 Daemon Manager

### 4.4.1 Overview

The Daemon Manager (DM) acts as an interface between the simulation and the rest of the network. It maintains partition membership information for each active entity within its simulation and addresses PDUs from those entities to the appropriate PM.

### 4.4.2 Configuration File

The DM has one additional configuration file entry, which is required:

Table 4-8: DM configuration file entry.

Option	Action
BSM_HOST	The name of the host where the BSM is running.

### 4.4.3 Process

The process flow specific to the DM is shown in Figure 4-6.

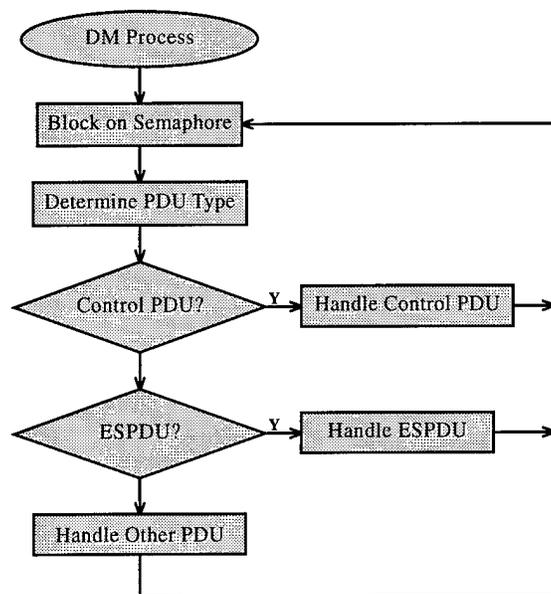


Figure 4-6: DM process flow.

As with the BSM, the DM operates within a loop, blocking on a semaphore until unblocked to handle a PDU.

#### 4.4.4 DM Handling of ESPDUs

The DM is responsible for directing outbound ESPDUs from the simulation to the appropriate PM, or the BSM if no PM has yet been assigned for that entity. This relieves the simulation of knowledge concerning the underlying network architecture. The DM maintains a lookup table for the entity ID to PM address mapping. When an ESPDU is received from the simulation, the DM checks this lookup table to see if this entity has been assigned to a PM; if so, the SendDaemon is notified of the destination address to send the ESPDU to. If the entity has no entry in the lookup table, or has not been assigned to a PM, the ESPDU is directed to the BSM and the entity is added to the lookup table if necessary.

Handling of inbound ESPDUs (and other PDU types as well, except for the control PDUs) is the responsibility of the simulation. The DM is not unblocked for non-control PDUs received from the network.

#### ***4.4.5 DM Handling of Control PDUs***

The DM recognizes one type of control PDU, the ADDRESS\_ASSIGN. The ADDRESS\_ASSIGN PDU is sent by the BSM to indicate which PM is responsible for a given entity based on the criteria being used. Each DM examines the PDU to determine if it is meant for one of the DM's entities (by comparing the site and app fields in the PDU to the DM's site and app). If the fields match, the DM extracts the address information from the PDU and opens a connection to that address (if one does not already exist), and will increment the count of entities using that connection, and will update the entity's address information in the lookup table to reflect the new address. All further ESPDUs from that entity are directed to the indicated PM. If the entity was previously sending to a different PM, the DM decrements the number of entities using the previous connection, and closes the connection if no other entities are using it.

After updating its lookup table, the DM generates a DM\_IE for this entity. This PDU contains the interest expression for the entity whose address assignment was updated, and is sent to the new PM. In this implementation, an entity's interest expression consists of a radius around the entity from which all PDUs should be received, and represents the range of the entity's most sensitive sensors (electronic, infrared, or visual).

#### ***4.4.6 DM Handling of Other PDUs***

The DM reverts to a broadcast mode when dealing with PDUs that do not lend themselves to PM assignment. All such PDUs are directed to the BSM.

### **4.5 Partition Manager**

#### ***4.5.1 Overview***

The Partition Manager (PM) acts as a rebroadcast agent for its partition, sending PDUs to those simulations which have expressed an interest in the PM's criteria set. It maintains information on which entities are interested in its information, and opens and closes connections as directed by the BSM.

### 4.5.2 Command-line Options

Under normal circumstances, a PM will not be started in a standalone mode; usually the BSM starts PMs as they are needed to handle partition divisions. When starting a PM, the BSM passes two additional parameters with the PM\_COMMAND: The identification number of the new PM (PMid), and the name of the host running the BSM (BSM\_host).

It is possible to start a PM manually, but the two parameters listed above must be specified, or the PM will exit with a warning.

### 4.5.3 Configuration File

The PM has one additional configuration file entry, which is required:

Table 4-9: PM configuration file entry.

Option	Action
SPLIT_COUNT	The number of entities being serviced at which a split will be initiated.

### 4.5.4 Process

The process flow specific to the PM is shown in Figure 4-7.

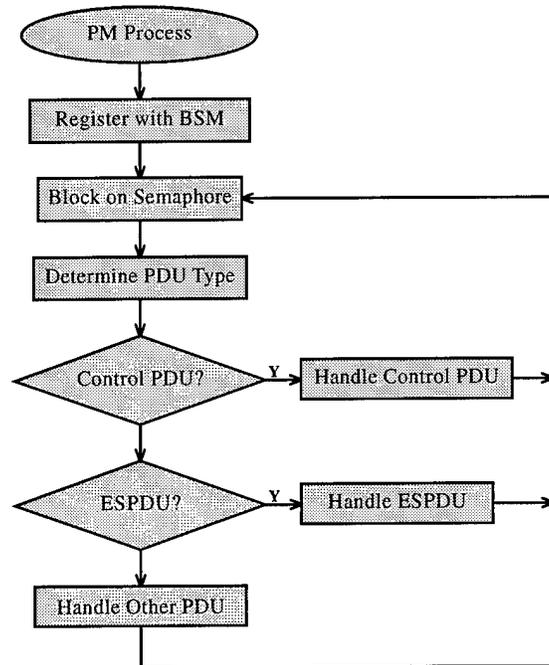


Figure 4-7: PM process flow.

The PM enters its main processing loop by registering with the BSM via a PM\_REGISTER PDU, which contains the PMid and address information to be used for DM connections.

The PM then blocks on a semaphore, waiting for PDUs to be received from the network.

#### 4.5.5 PM Handling of ESPDUs

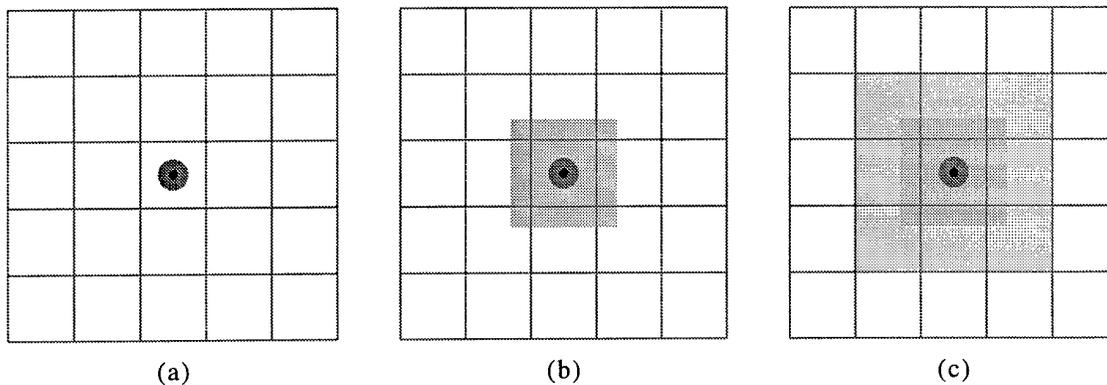
All ESPDUs received by a PM are immediately sent to the PM's multicast group. In addition, the ESPDU is compared against the PM's criteria set to determine if the entity has transitioned beyond the scope of the PM's responsibility. If so, the ESPDU is also sent to the BSM to be assigned to a new PM.

#### ***4.5.6 PM Handling of Control PDUs***

The PM recognizes six types of control PDUs: ADDRESS\_ASSIGN, CRITERIA\_ASSIGN, DM\_IE, MCAST\_ADD, MCAST\_DELETE, and PM\_DELETE. The ADDRESS\_ASSIGN PDU is generated by the BSM and indicates which PM an entity's PDUs should be directed to, as discussed in the BSM section above. Since the BSM keeps no information about the DMs in an exercise, it sends all ADDRESS\_ASSIGN PDUs back to the sender—in this case, the PM. The PM in turn sends the ADDRESS\_ASSIGN PDU to its multicast group.

CRITERIA\_ASSIGN PDUs are sent by the BSM to give a PM its initial criteria set, or to update a PM after a partition division. The criteria set contained in the CRITERIA\_ASSIGN PDU will replace the PM's existing criteria set.

DM\_IE PDUs contain the interest expression of an entity that is in the PM's scope of responsibility. In the case of geographic partitioning, the DM\_IE contains a radius value; since interest expressions are sent only when an entity joins a partition, it is necessary to expand this radius value to encompass the entire geographic range of the partition. In this manner, the entity can move anywhere within the bounds of the partition and still be assured of receiving PDUs from all entities within its range of interest. The PM expands the entity's interest range as shown in Figure 4-8, then generates a PM\_IE PDU containing the expanded IE and sends that to the BSM.



*Figure 4-8: PM geographic IE expansion. (a) The entity's IE is expressed as a radius centered on the entity. (b) The PM expands this radius to encompass the entire geographic range of the partition, and (c) sends this range to the BSM. The BSM sends an MCAST\_ADD to all PMs whose partition is in the expanded area, and an MCAST\_CLOSE to all others.*

MCAST\_ADD and MCAST\_DELETE PDUs are generated by the BSM, and indicate that an entity has interest in a PM's scope of responsibility (or does not, in the case of MCAST\_DELETE). With MCAST\_ADD, the PM opens a connection to the indicated address (if it is not already sending to that address) and adds the entity ID to the list of those entities using that connection (if it is not already on the list). For MCAST\_DELETE, the PM determines if it is sending to the indicated address, and if so, if the entity is on the list of those using that connection. If the connection exists and the entity is on the list, the entity is removed from the list. If no other entities at that address are using the connection, the PM generates an MCAST\_CLOSE message containing the address and sends it to the multicast group, where the ReceiveDaemon at that address will intercept it and close the branch connection.

The PM\_DELETE message is sent by the BSM at exercise end. When a PM receives a PM\_DELETE message it will exit its processing loop and execute the Shutdown() procedure to terminate all processes.

#### **4.5.7 PM Handling of Other PDUs**

As with the BSM and DM, no special handling is done with other PDUs received; they are sent to the multicast group unmodified upon receipt.

#### **4.6 Program Communication**

In order to implement the partitioning scheme discussed here, a specialized communication structure is required. Ten new PDU types are used. Appendix B has detailed descriptions of each PDU.

#### **4.7 Exercise Overview**

In this section I illustrate some of the communication processes that take place to register an entity and its interest expression, split an existing partition, and transition an entity to a new PM.

Figure 4-9 illustrates the communication that takes place when an entity first enters an exercise.

- (a) The DM managing that entity sends its initial ESPDU to the BSM. The DM will also send any further ESPDUs from that entity to the BSM until the entity is assigned to a PM. This may result in multiple ESPDUs from the entity are received at the BSM (with an address lookup for each ESPDU), but it ensures that address information will not get lost due to packet discards within the network.
- (b) The BSM determines the controlling PM by comparing the entity's attributes to the criteria sets for each active PM. The BSM then sends the ESPDU to the controlling PM for rebroadcast, and notifies the DM of the PM's address.
- (c) The DM opens a connection to the PM (if a connection does not already exist for another entity) and sends all future ESPDUs from that entity to the PM.

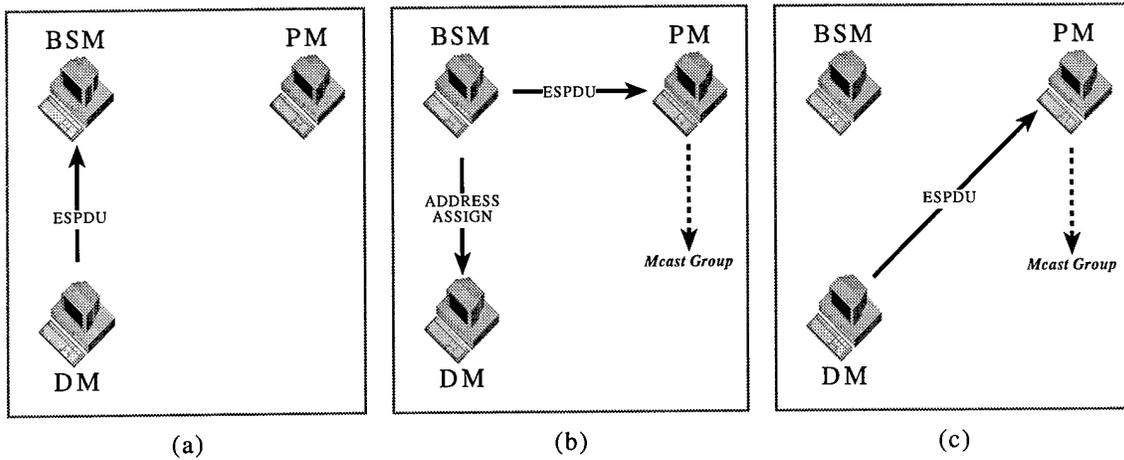


Figure 4-9: Entity Registration

Figure 4-10 shows the communication flow that occurs when a DM registers an entity's interest expression.

- (a) The DM send a DM\_IE expression for the entity to the PM. This DM\_IE is specific to the criteria type in use; for example, in a geographic partitioning scheme this interest expression is a radius centered on the entity.
- (b) The PM changes the interest expression (if necessary) and forwards it to the BSM. The BSM tests the entity's interest expression against the criteria set of each active PM.
- (c) If the entity is interested in receiving PDUs from a PM, the BSM sends that PM an MCAST\_ADD PDU for the entity. The PM opens an Mcast connection to the entity's DM (if a connection does not already exist).
- (d) The BSM sends an MCAST\_DELETE PDU to those PMs that do not meet the entity's interest expression. If the PM is sending to that entity's DM, the entity ID is removed from the "using" list.

- (e) If The PM is sending to that entity's DM and no other entities managed by that DM have expressed an interest in the PM's PDUs, the PM sends an MCAST\_CLOSE message to the DM to close the connection to that PM.

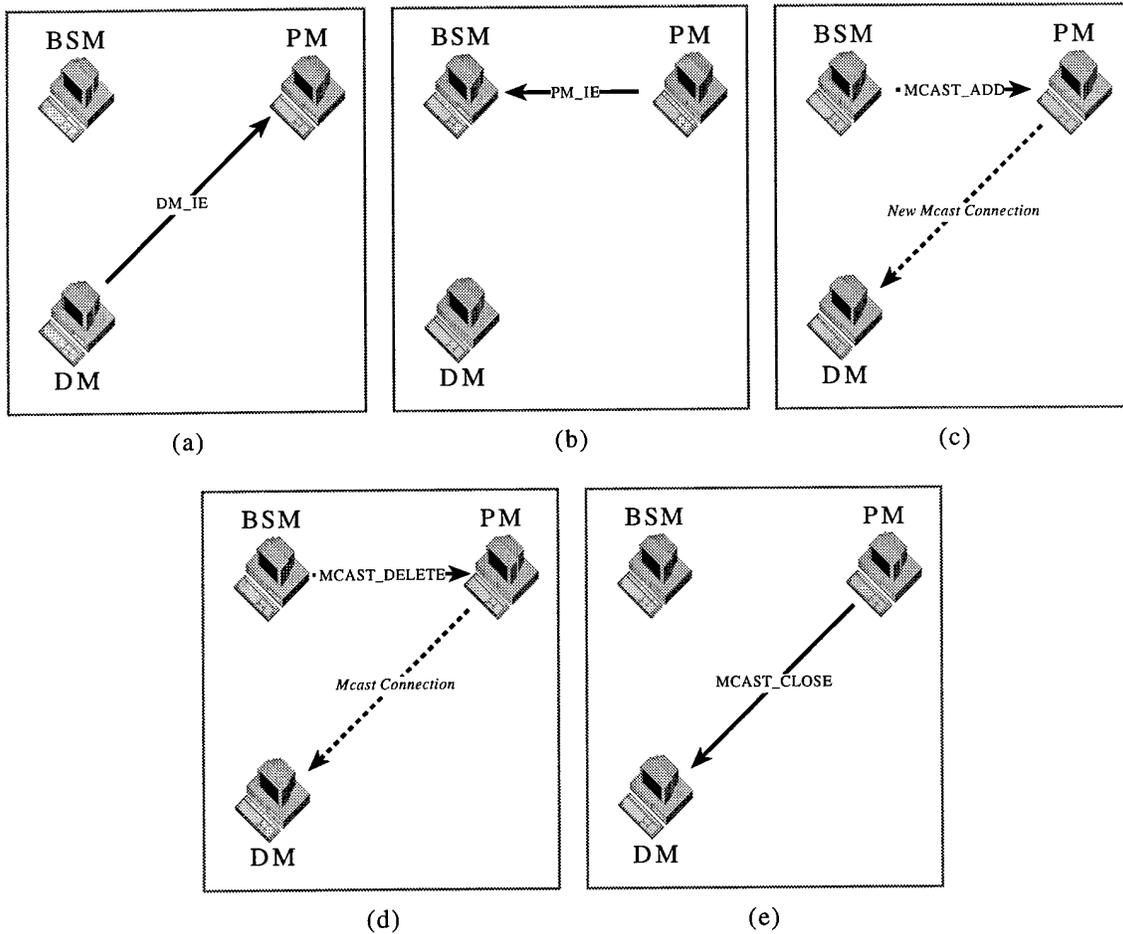


Figure 4-10: Interest Expression and MCAST\_ADD Process.

Figure 4-11 shows the communication flow that occurs when an entity is transitioned to another PM.

- (a) The PM examines each ESPDU to determine if that entity still meets the PM's criteria. If an entity does not meet the criteria, the PM flags the entity in its `fdlist` using field and sends a copy of the ESPDU to the

BSM. If more PDUs from that entity are received by the PM they will not be forwarded to the BSM. The PM also sends the ESPDU to its Mcast group, regardless of whether it meets the criteria or not.

- (b) The BSM determines which PM the entity should now be sending to as shown in Figure 4-9, except that the ADDRESS\_ASSIGN PDU is sent back to the PM that forwarded the ESPDU. The PM in turn forwards the ADDRESS\_ASSIGN PDU to its Mcast group.
- (c) The DM managing the entity receives the ADDRESS\_ASSIGN PDU; all other DMs ignore it. The DM removes the entity from the list of entities sending to the previous PM, and closes the previous PM's connection if no other entities are using it. The DM then changes the entity's address to the newly-assigned PM and opens a connection to the PM (if one does not already exist). Finally, a DM\_IE PDU is sent to the new PM to register that entity's interests, which then proceeds as shown in Figure 4-10.

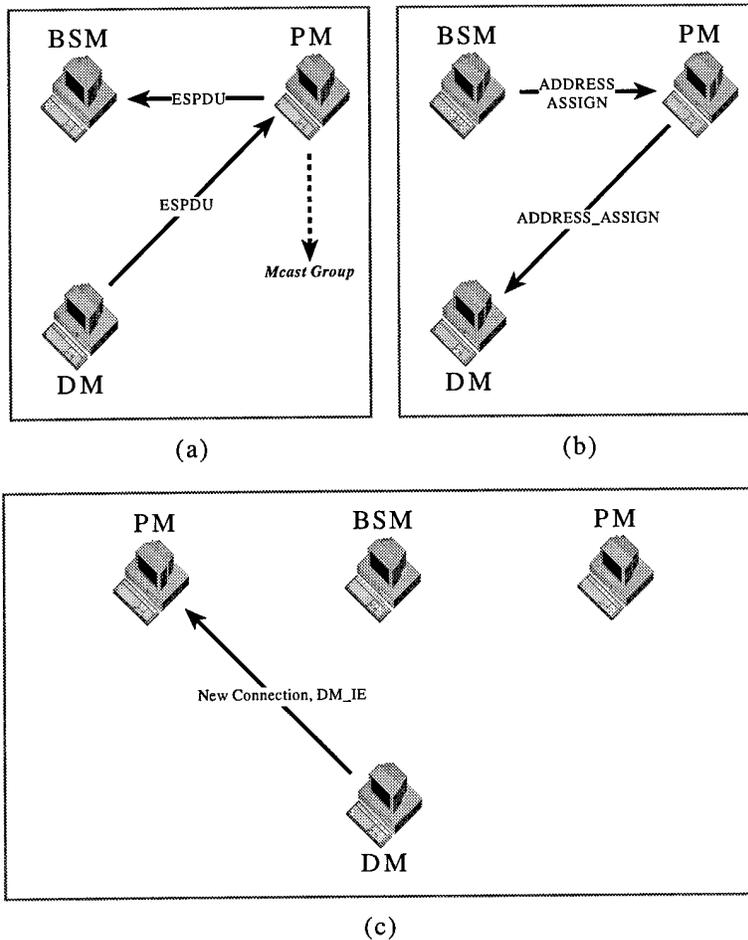


Figure 4-11: Entity Transition.

Figure 4-12 shows the communication flow that occurs when PM has exceeded its threshold to be split.

- (a) The PM sends a PM\_SPLIT PDU to the BSM, to include information concerning how the partition should be split. For geographic criteria, the PM sends the XYZ coordinates of the highest entity activity.
- (b) The BSM splits the partition of the old PM, initializes a new PM, then gives each PM its new criteria set via a CRITERIA\_ASSIGN PDU.

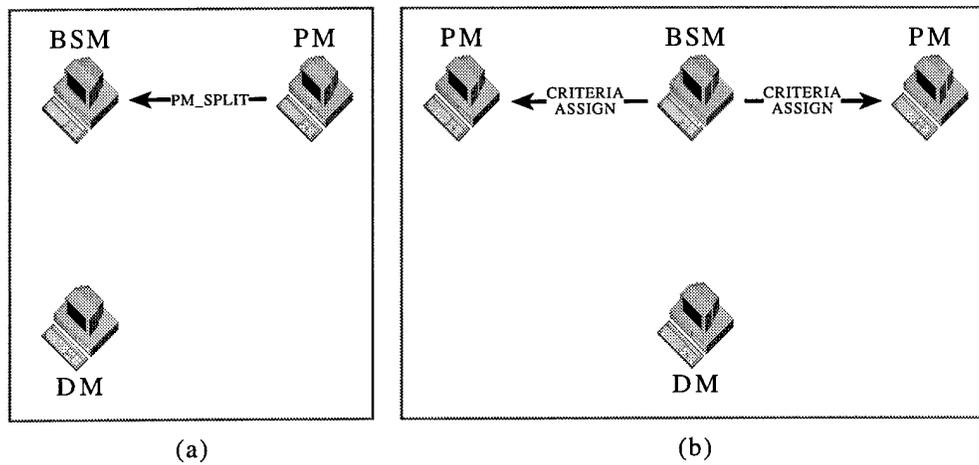


Figure 4-12: PM Split.

## 4.8 Conclusion

This chapter presents the design details for the three programs I developed to implement dynamic partitioning using ATM. The two main aspects of this design are the use of native ATM as a transport protocol, and the adaptive partitioning that changes based on the state of the exercise. The next chapter will discuss the results of my implementation.

## **5. Results**

### **5.1 Overview**

Two goals were identified in Chapter One: An ATM-aware DIS exercise system compatible with existing AFIT simulations, and a method of dynamic partitioning that allowed adaptive relevance filtering. The ATM network installed at AFIT was originally designed to have 9 systems; however, due to procurement delays, only six systems were available for testing. This number was further reduced to four due to compatibility problems between the Fore Systems libraries and installed hardware at AFIT. Hence, exhaustive testing was not possible.

Nevertheless, the first goal has been achieved; the ATM exercise system was successfully tested with AFIT's DIS Manager and the Synthetic Battle Bridge (SBB). Comparisons with the previous Ethernet communication method have shown that PDU transmissions over ATM are much more reliable (no collisions) and immediate. Additionally, with Ethernet there was the problem of receiving PDUs out of order, causing the DIS Manager to have to continually compare PDU timestamps and discard the out-of-order PDU. While this was not a problem with ESPDUs, other applications such the SBB's shared viewpoint rely on a fairly constant data stream, and rejected PDUs appear as "jumps" in the viewpoint. Because ATM is connection-oriented, all PDUs transmitted are received in order.

For other testing, a minimal test setup was used consisting of two DM hosts, one BSM host (also running a PM), and one host running a second PM. Three separate tests were run and are described below:

- 1) Output from 84 entities (from AFIT's Gaggle Generator) was simulated on one DM host, with 7,952 simulation PDUs generated over approximately 5 minutes. Partition splitting was not tested in this run.
- 2) Output from 25 entities (from ModSAF) was simulated on the two DM hosts, with 16,143 simulation PDUs generated per system over 7 minutes.

The PM's SPLIT\_COUNT variable was set to an arbitrary value of 40, requiring the default PM to be split when the second simulation was started.

- 3) Output from 70 entities (from AFIT's Gaggle Generator) was simulated on the two DM hosts, with 113,190 simulation PDUs generated per system over approximately 5 minutes. SPLIT\_COUNT was set at 80 to force an almost immediate split when the second simulation began transmitting.

For all tests, geographic criteria was used, and each PM had a single criterion (a geographic range) in its criteria set.

## 5.2 ATM Results

The vendor-supplied program `atmstat` was used to monitor all ATM statistics. For transition and latency times, a timestamp from `gettimeofday` was included in the PDUs when transmitted and then compared against the current system time when the PDU was received back at the host. This timestamp includes seconds and microseconds from the system clock, and was added to the value by the `SendDaemon` just before PDU transmission and compared within the `ReceiveDaemon` immediately after the PDU was received from the network.

The results from the three test runs are summarized in Table 5-1, and discussed in the sections below. Regardless of the number of PDUs being transmitted or the rate of transmittal, no cells were lost at either the AAL level (on the simulation host) or in the switching fabric (at the ATM switch), indicating that network congestion was not a factor in any of the times recorded.

Table 5-1: ATM test results.

	Program	PDUs Sent		Latency (ms)			PM Start (sec)
		Simulation	Control	PDU's	ADDRESS	MCAST	
Run 1	BSM	—	168	min 2.850	min 19.311	min 76.196	—
	DM	7,952	84	avg 7.156	avg 32.244	avg 95.527	
	PM	—	84	max 9.618	max 57.167	max 123.412	
Run 2	BSM	—	72+90	min 2.702	min 19.194	min 81.728	1.03213
	DM	16,143	36	avg 6.993	avg 47.215	avg 124.520	
	PM	—	54+18	max 10.540	max 121.042	max 142.821	
Run 3	BSM	—	204+254	min 3.035	min 20.003	min 88.679	1.46220
	DM	113,190	102	avg 7.404	avg 51.128	avg 137.184	
	PM	—	154+50	max 10.681	max 125.693	max 166.104	

### 5.2.1 Run 1

For the first run, my primary goal was to determine the latency between initial PDU transmission and the completion of the address assignment and receipt of multicast PDUs, as well as measuring the ratio of control PDUs to simulation PDUs. The entities in this simulation were all aircraft, and were relatively static (straight and level) with little entity interaction.

A total of 7,952 simulation PDUs and 84 control PDUs were sent by the DM; this is consistent with expected results, since a DM\_IE was generated for each entity. Similarly, the PM sent 84 PM\_IE PDUs, and the BSM generated 84 ADDRESS\_ASSIGN PDUs, 84 MCAST\_ADD PDUs, and 0 MCAST\_CLOSE PDUs. Total PDU count for the exercise was 8,290, counting the PM\_REGISTER and CRITERIA\_ASSIGN PDUs at the start of the exercise, for a total control PDU overhead of 4.07%.

The average time between an ESPDU being sent to the BSM and the receipt of an ADDRESS\_ASSIGN PDU for that entity was 32.244 milliseconds, with a maximum time of 57.167 milliseconds. The average time between a DM\_IE PDU being received at the PM and an MCAST\_ADD being received for that entity was 97.527 milliseconds, with a maximum time of 123.412 milliseconds.

### **5.2.2 Run 2**

The second test run was designed as a low-level simulation to demonstrate partition splitting. The startup time for the new PM was measured, along with the transition times for entities moving to the new PM. The entities represented in this test run were a mix of aircraft and ground forces, with moderate entity interaction to include fire and detonation PDUs.

A total of 16,143 simulation PDUs and 36 DM\_IE PDUs were sent by each DM; the higher number of DM\_IE PDUs was due to entities switching partitions, or being switched when the second PM was initialized. The default PM sent 54 PM\_IE PDUs, the second PM sent 18. The BSM generated 72 ADDRESS\_ASSIGN PDUs, 90 MCAST\_ADD PDUs, and 0 MCAST\_CLOSE PDUs. Total PDU count for the exercise was 32,583, counting the PM\_REGISTER and CRITERIA\_ASSIGN PDUs at the start of each PM, for a total control PDU overhead of 0.912%.

The average time between an ESPDU being sent to the BSM and the receipt of an ADDRESS\_ASSIGN PDU for that entity was 47.215 milliseconds, with a maximum time of 121.042 milliseconds (when the second PM was being initialized). The average time between a DM\_IE PDU being received at the PM and an MCAST\_ADD being received for that entity was 124.520 milliseconds, with a maximum time of 142.821 milliseconds.

The second PM initialization required 1.03213 seconds, with most of this delay encountered in the `rexec` call.

### **5.2.3 Run 3**

Run 3 was designed to determine the stability of this architecture when presented with a large number of PDUs in a relatively short time. All entities were aircraft, with the same level of interaction as in the first run; however, to simulate a high PDU transmission rate, I removed the timing constraints, forcing the simulation hosts to transmit the PDUs as fast as they could. The PDUs transmitted were from a simulation that actually ran for over 30 minutes in real time.

A total of 113,190 simulation PDUs and 102 DM\_IE PDUs were sent by each DM; the higher number of DM\_IE PDUs was due to entities switching partitions, or being switched when the second PM was initialized. The default PM sent 154 PM\_IE PDUs, the second PM sent 50. The BSM generated 204 ADDRESS\_ASSIGN PDUs, 254 MCAST\_ADD PDUs, and 0 MCAST\_CLOSE PDUs. Total PDU count for the exercise was 227,251, counting the PM\_REGISTER and CRITERIA\_ASSIGN PDUs at the start of each PM, for a total control PDU overhead of 0.383%.

The average time between an ESPDU being sent to the BSM and the receipt of an ADDRESS\_ASSIGN PDU for that entity was 51.128 milliseconds, with a maximum time of 125.693 milliseconds (when the second PM was being initialized). The average time between a DM\_IE PDU being received at the PM and an MCAST\_ADD being received for that entity was 137.184 milliseconds, with a maximum time of 166.104 milliseconds.

The second PM initialization required 1.46220 seconds, again with most of this delay encountered in the `rexec` call.

#### ***5.2.4 Analysis of ATM results***

The statistics I gathered concerning latency support the use of ATM for DIS exercises. The maximum application-to-application PDU latency I observed was 10.681 milliseconds (in run 3), well within the 100-millisecond window identified both by Calvin [4] and Pullen [21] as a requirement for a DIS network architecture. I wanted to determine ATM's minimal latency, so I ran several tests in which only two ESPDUs were sent. The first ESPDU initiated the ADDRESS\_ASSIGN/MCAST\_ADD cycle, and when all control PDUs had been sent, I sent the second ESPDU to determine the no-load delay between transmission and receipt of the PDU back at the system. The average value for these tests was measured at 2.35 milliseconds.

The overhead varied inversely with the number of PDUs in the exercise, from 4.07% (8,290 PDUs) to 0.383% (227,251 PDUs). Since the number of control PDUs is highest at simulation start (when all entities are registering with their controlling PMs) and drops as the

simulation progresses, it is reasonable to expect that the overhead factor will continue to drop as the exercise continues. Partition splitting and entity transitions will continue to account for some network traffic, but the test results indicated that in most cases each entity remained in a single partition.

The MCAST\_ADD latency represents the time between an entity requesting to receive a PM's rebroadcasts and the PM starting to send PDUs to that entity's DM. During this time, the entity is theoretically not receiving PDUs in which it is interested, so this delay is obviously critical. In practice, though, the system does not track which *entity* a PDU is destined for, only which DM. As a result, in most cases the DM controlling that specific entity is already receiving PDUs from the specified PM. During exercise startup, this delay may represent an unacceptable "blindness" for an entity. However, since all other entities will be experiencing similar delays, I do not consider this to be a factor.

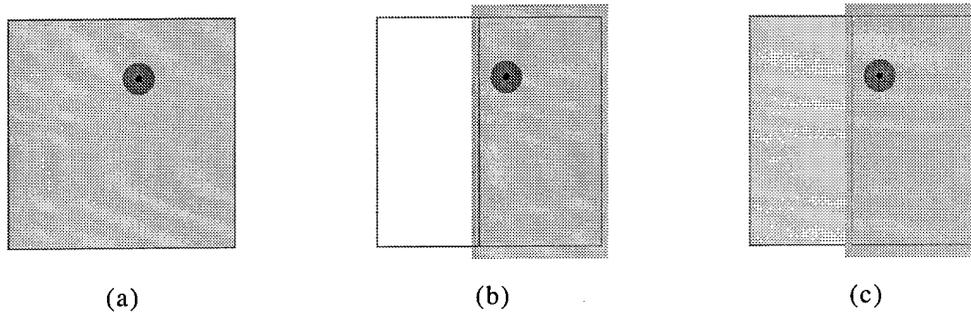
The largest delays occurred during the period immediately following the second PM initialization, when a large number of entities were being transitioned to the new PM. The actual startup delay for the new PM is not relevant, though, since the existing PM continues to rebroadcast PDUs, resulting in no loss of simulation data while the new PM initializes.

### **5.3 Dynamic Partitioning Results**

Results obtained from the dynamic partitioning aspect of my research are harder to quantify. The results shown above indicate that the dynamic partitioning mechanism works as expected; PMs are started as necessary and open multicast connections to DMs that have expressed interest in their PDUs; DMs address PDUs to the correct PMs; and entities are transitioned to a new PM as necessary.

However, with geographic criteria as the partitioning scheme in my testing, no advantage was shown in respect to reducing PDUs delivered to a specific host. The main problem was a limited number of test systems combined with the current "overlap" factor required by geographic partitioning. Because each entity's IE was expanded to encompass the boundaries of the partition, the simulation was still receiving data from the entire exercise,

as shown in Figure 5-1. While this does not immediately disprove the basic premise of dynamic partitioning, it does demonstrate that with geographic criteria no advantage will be gained until the number and size of partitions reaches a point where an entity's IE does not overlap all partitions.



*Figure 5-1: IE overlap. (a) At exercise start, an entity's IE covers the entire partition. (b) After splitting, the IE is expanded to "overlap" other adjacent partitions. (c) resulting in the simulation still receiving all PDUs in the exercise.*

Given the necessary expansion of interest expressions (as discussed in Chapter Four), it appears that the dynamic partitioning scheme is ill-suited for geographic criteria. Other criteria, such as those which use entity type or entity country, would yield better results under this architecture, since no overlap would be introduced; the data would be perfectly partitioned. An entity could express interest in receiving PDUs from all F-15s, for instance, without receiving PDUs from F-16s, even if they were in the same flight; if the number of F-15s in a simulation proved excessive for a PM to handle, further division could occur on such factors as sending site, or even entity country (allowing separation of Israeli F-15s from United States F-15s). It is important to note, however, that whatever partitioning scheme is used, it must be complete (covering all possibilities within that scheme) and disjoint (since an entity can belong to only one partition). For each criteria in a PM's criteria set, there must exist one or more other PMs that include the entities excluded by that criteria.

An additional factor that we discovered was entity "thrashing" between partitions. This was caused mainly by the "hint" method used by the PMs in requesting a split; by sending the average XYZ location of all entities in the exercise, the split was made at a point where

some entities were geographically situated close to a partition boundary. As these entities maneuvered, the partition in which they were located switched rapidly between PMs, with each switch requiring the overhead concomitant with re-addressing, IE registration, and Multicast add/delete messages. In the most extreme case, one entity in run #3 alternated between the two PMs nine times in the space of two minutes. While no PDUs were lost due to transitions, it still resulted in unanticipated overhead that could perhaps be avoided if a better division method were used.

#### **5.4 Conclusion**

In this chapter I have shown that the PDU latency incurred in an ATM DIS exercise system are well within acceptable levels, even accounting for the retransmissions necessary with ATM. I have further shown that although a dynamic partitioning scheme is in theory viable, the expected advantages will likely not be realized with geographic criteria until a significant partitioning of the battlespace has occurred.

## **6. Conclusions**

### **6.1 Summary of Problem**

The problem I set out to solve in this research was twofold: to develop a DIS exercise system which takes advantage of ATM's greater speed and bandwidth, and to determine if a dynamic relevance filtering system would provide a reduction in excess PDUs delivered to a specific host.

### **6.2 Contributions**

My research has provided an exercise system that will allow AFIT to conduct DIS exercises over the ATM network, using native ATM as the interface between the simulation and the network. This system has been tested with the current DIS Manager software.

I have also shown that dynamic relevance filtering is feasible, and entity transitions occur without data loss to the exercise as a whole. While dynamic relevance filtering does not provide adequate PDU reduction when geographic partitioning is used, other criteria should provide much better compartmentization of PDUs. For geographic, however, it appears that a static partitioning scheme provides better improvements.

### **6.3 Future work**

While my research was successful in meeting the stated research goals, it also presented some options for enhancement of this system. The most important of these is addressing the shortcomings of dynamic partitioning; other areas such as consolidation of the BSM/PM system, signal-driven ReceiveDaemons, using AAL\_NULL as the ATM adaptation layer, and the establishment of permanent virtual circuits should also be explored.

#### **6.3.1 Hybrid Partitioning**

My testing demonstrated that when geographic criteria is used, the main disadvantage with dynamic partitioning is limited PDU reduction at a given host until the granularity of

partitions was such that an entity's IE did not overlap all partitions. We have theorized a possible method to resolve this. A static partitioning scheme, in which a predetermined grid size is overlaid onto the virtual battlefield, could be calculated by the BSM at simulation start. Each PM would be responsible for a region of grids. Initially, groups of grids would be combined into a single multicast group; as the number of entities in the exercise increased, additional multicast groups would be created, with each group encompassing fewer of these predefined grids. In this manner, simulations would still be able to join or leave the exercise on an ad hoc basis, with no *a priori* knowledge required beyond the name of the BSM host, yet the granularity of the partitions would already be in place, possibly avoiding the pitfalls encountered in the current scheme. Work on this method is now in progress.

### **6.3.2 Merged BSM and PM**

For most of an exercise the BSM is idle, responding only when an entity needs to be assigned to a new PM or a PM has exceeded its SPLIT\_COUNT. It should be possible to incorporate the functionality of the BSM into the PMs, with the PMs communicating between themselves for entity transitions and IE parsing. Since this inter-PM communication would require relatively low bandwidth, an optimal solution may be found in a hybrid IP/ATM system. The PMs would communicate with each other via an IP multicast group (many-to-many), while the actual simulation traffic would still be carried by native ATM transmissions.

### **6.3.3 Signal-Driven ReceiveDaemons**

The current scheme uses a separate ReceiveDaemon process to service each incoming connection. I attempted to use polled file descriptors, wherein a data stream would notify a single process when it had data to be delivered, but this feature is apparently unsupported by the Fore Systems interface. Later versions of the interface may allow this. There are also methods to do signal-driven IO, in which the data streams would generate a SIGIO message when data is ready. This presents the problem of not knowing *which* data stream is ready; all incoming fds would need to be scanned.

#### **6.3.4 AAL\_NULL**

By using AAL5 as the ATM adaptation layer, the effective payload in each cell increases, but there is still overhead incurred at the convergence sublayer. Fore Systems allows the use of a "null" adaptation layer (AAL\_NULL) which adds no overhead at this layer; the user is responsible for segmenting the data into 48-byte cell payloads. By using AAL\_NULL an advantage may be gained in throughput.

#### **6.3.5 Permanent Virtual Circuits**

The current ATM interface uses Switched Virtual Circuits (SVCs) for each connection, with the ATM switch building and removing these circuits as necessary. This "on-the-fly" setup and teardown incurs delay, which is advertised as being less than 11 milliseconds [13]; this delay is encountered each time a connection is established. If the topology of an exercise is known beforehand, permanent virtual circuits (PVCs) may be allocated before the exercise, reducing the connection setup time. While this will require additional pre-exercise preparation, the PVCs, once established, would remain so until they were manually removed, so they could be used for multiple exercises. Further, since the PM host machines are specified at runtime, it may be possible to create the PVCs at runtime as well. Note that this would not affect the dynamic partitioning scheme implemented here, since the ATM connections treat PVCs and SVCs identically. An added advantage with this scheme is that a single fd can be associated with multiple incoming and outgoing PVCs, perhaps providing a better method for receiving than using a dedicated ReceiveDaemon for each connection.

#### **6.3.6 Modified DIS PDUs**

A final suggestion is to re-examine the DIS PDU structure in an attempt to align more of the PDUs to fit the 48-byte ATM cells. For example, it was shown in Chapter Three that a typical 144-byte ESPDU, when transmitted over an AAL5 connection, required 32.1% overhead; if the size of the ESPDU could be reduced to 136 bytes, the overhead drops to 14.5%. Since the ESPDU can account for up to 98% of the network traffic in a DIS exercise, the reduction in overhead by modifying just this PDU would be significant.

## 6.4 Conclusions and Recommendations

ATM promises tremendous bandwidth increases over current networking technologies. Future enhancements planned for ATM (such as receiver-initiated joins and many-to-one multicast groups) will improve the adaptability of ATM to DIS exercises and could possibly do away with the rebroadcast agent hierarchy I implemented.

My research has indicated that dynamic relevance filtering (at least in my implementation of it) provides minimal gain when applied to geographic partitioning. Other partitioning criteria hold greater promise with regard to dynamic partitioning, but the question then becomes one of utility: in an actual DIS exercise, how many entities will be interested in receiving PDUs from only a certain type of entity? A better approach would most likely combine the advantages of dynamic relevance filtering with that of a static overlay, as suggested in the "future work" section above.

## Appendix A: Function Prototypes

This appendix contains the prototypes for all functions contained in the DIS ATM library (`libDIS_ATM.a`), along with a brief description of what each function does.

**int AddUsing(const int fdindex, const eid\_t eid)**

**AddUsing** adds the entity `eid` to the list of entities that are using the fd `fdindex`, either for receiving (PM) or sending (DM). 0 is returned on success, or -1 if the add was unsuccessful (unable to increase the size of the list, or the entity is already listed as using the fd).

**int BufDone(int bufnum, int mode, int PDUtype, int fdindex)**

**BufDone** handles the marking, clearing, and addressing of buffers.

*mode == FILL\_MODE:*

If `PDUtype` is `CTRL_DATA` (a control PDU) or `PDU_DATA` (a PDU from the simulation — DM mode only), the buffer `bufnum` will be marked for the BSM/DM/PM. In addition, the semaphore that the BSM/DM/PM is blocked on will be set to wake the process. 0 is returned on success, or -1 if a semaphore error occurs.

If `PDUtype` is `APP_DATA` (a PDU from the network) the buffer will be marked for the simulation and `bufnum` will be queued for the simulation. 0 is returned (always successful).

*mode == RECV\_MODE:*

The buffer `bufnum` is marked as available. 0 is returned (always successful).

**void CloseFD(int fdindex, int really\_close)**

**CloseFD** closes the file descriptor referenced by `fdindex` if `really_close` is true. The function `atm_close` closes the actual connection, and the structure `fdlist[fdindex]` is marked idle. If `really_close` is false, the structure `fdlist[fdindex]` is marked idle, but no call to `atm_close` is made (used by the PM to track multicast connections). 0 is returned (always successful).

**int ConnectFD(int fdindex, Atm\_dataflow df)**

**ConnectFD** attempts to open a connection to the address stored in the `fdlist[fdindex]` structure. The `df` field specifies whether the connection should be simplex, duplex, or multicast. The `atm_connect` function is called to make the actual connection. 0 is returned on success. -1 is returned on error (connection refused, or invalid address).

**void ControlPDU(const int bufnum, const int PDUtype,  
void \*PDUdata)**

**ControlPDU** generates a control PDU in the buffer `bufnum`. `PDUtype` indicates what type of PDU to be generated (for example, `ADDRESS_ASSIGN`). `*PDUdata` is a void pointer that is typecast to the appropriate data type, and contains the information necessary (such as ATM address information) to generate the PDU.

**int CriteriaMatch(int bufnum, PM\_criteria\_t PMcriteria)**

**CriteriaMatch** is used by the BSM and PM. The ESPDU contained in buffer `bufnum` is compared against the criteria set `PMcriteria`. If the ESPDU meets the requirements of the criteria set, 0 is returned, -1 otherwise.

**int DeleteUsing(const int fdindex, const eid\_t eid)**

**DeleteUsing** removes the entity `eid` to the list of entities that are using the `fdindex`. 0 is returned on success, or -1 if the delete was unsuccessful (the entity is not listed as using the `fd`).

**int FindBuf(int mode)**

**FindBuf** locates a free buffer (if `mode` is `FILL_MODE`), or returns the buffer number containing the oldest PDU received from the network destined for the simulation(`mode` is `RECV_MODE`). If `mode` is `FILL_MODE`, the buffer number to use is returned. If `mode` is `RECV_MODE`, the buffer number containing the PDU is returned, or -1 if no PDUs are waiting.

**int GetFD(int qlen, Atm\_sap sap, int real\_fd)**

**GetFD** allocates `fdlist` structures and connects to the ATM interface if requested.

An unused index in `fdlist` is located and stored in a local variable (`index`). If `real_fd` is true, the function `atm_open` is called to connect to the interface in READ/WRITE mode, and the file descriptor returned by the system is stored in in the `fdlist[index]` structure. The function `atm_bind` is then called to bind to the interface. If `qlen` is non-zero, a server connection is requested (for ListenDaemon); otherwise, a normal connection is requested. If `sap` is non-zero, **GetFD** attempts to use that value as the ASAP to use on the local interface; otherwise, an ASAP is assigned by the system. `index` is returned on success, -1 on error (no indexes available in `fdlist`, failure to connect to the interface, or failure to bind to the fd).

If `real_fd` is false, no connection is attempted (used by the PM to track multicast addresses). `index` is returned on success, -1 if no `fdlist` index is available.

**int InitATM(info\_t \*ATMinfo)**

**InitATM** initializes the ATM interface on the host and stores the device name and local ATM address in `ATMinfo`. 0 is returned on success, -1 on error (invalid interface specified, or unable to get local ATM address).

**buffer\_t \*InitMemory(int mode, char \*arena\_file)**

**InitMemory** initializes the shared-memory data structures within `buffers` (if `mode` is CREATE) or returns the address of the existing `buffers` structure (if `mode` is JOIN). `*arena_file` is the filename of the arena to create/join. The address of `buffers` is returned on success, NULL on error (failure to allocate any of the data structures or the arena).

**void ListenDaemon()**

**ListenDaemon** listens for incoming connection requests, allocates a new structure within `fdlist` for the new connection, and assigns a ReceiveDaemon to handle the connection.

**void ReceiveDaemon(int fdindex)**

**ReceiveDaemon** receives data on an inbound connection and stores this information into a buffer for use by the BSM/DM/PM or the simulation. **ReceiveDaemon** monitors the connection pointed to by `fdlist[fdindex]`.

**void Shutdown(int signal)**

**Shutdown** closes all connections, kills all running processes, closes all open files, and deallocates all memory structures, then calls `exit(signal)`.

**int SendBuf(int bufnum, int fdindex, int mode)**

**SendBuf** sends the data in `bufnum` out the `fd` in the `fdlist[fdindex]` structure, if the `fd` is in a connected state by unblocking the **SendDaemon**. If the `fd` is not connected, no data is sent (for example, if a PDU is received by PM when no multicast recipients have been connected to). If `mode` is `SEND_FREE`, the buffer is then marked as idle. 0 is returned on success, -1 on error (invalid `fd` connection, connection closed, or unable to unblock **SendDaemon**).

**void SendDaemon()**

**SendDaemon** sends data out to the network. The buffer to send from and where to send to is passed to **SendDaemon** via its control register in `buffers`.

## Appendix B: Control PDUs

This appendix describes the control PDUs created to manage a DIS exercise using the BSM/DM/PM structure. All PDUs are based on the V2.0.4 action request PDU.

### ADDRESS ASSIGN

Field	Size (bits)	Value
<b>PDU Header</b>	96	
Protocol Version	8	0x04
Exercise ID	8	Set at runtime
PDU Type	8	0x10
Protocol Family	8	0x05
Time Stamp	32	Set at PDU send
Length	16	64 (bytes)
Padding	16	unused
<b>Originating Entity</b>	48	
Site	16	0xFFFFE (don't care)
Application	16	0x0000 (BSM_APP)
Entity	16	0xFFFFE (don't care)
<b>Receiving Entity</b>	48	
Site	16	Receiving site
Application	16	Receiving application
Entity	16	Receiving entity
Request ID	32	Set at PDU send
Action ID	32	0x0000F000
Fixed Datum Records	32	Number of fixed datum records (3)
Variable Datum Records	32	Number of variable datum records (0)
<b>Fixed Datum #0</b>	64	
Datum ID	32	0x00
Value	32	ASAP of the PM
<b>Fixed Datum #1</b>	64	
Datum ID	32	0x01
Value	32	First 4 NSAP bytes of the PM
<b>Fixed Datum #2</b>	64	
Datum ID	32	0x02
Value	32	Last 4 NSAP bytes of the PM

**USAGE:** Generated by the BSM, directed to a DM, and will indicate the address of a PM to which a specific entity's PDUs should be directed.

## PM REGISTER

Field	Size (bits)	Value
<b>PDU Header</b>	96	
Protocol Version	8	0x04
Exercise ID	8	Set at runtime
PDU Type	8	0x10
Protocol Family	8	0x05
Time Stamp	32	Set at PDU send
Length	16	72 (bytes)
Padding	16	unused
<b>Originating Entity</b>	48	
Site	16	0xFFFFE (don't care)
Application	16	0x0002 (PM_APP)
Entity	16	0xFFFFE (don't care)
<b>Receiving Entity</b>	48	
Site	16	0xFFFFE (don't care)
Application	16	0x0000 (BSM_APP)
Entity	16	0xFFFFE (don't care)
Request ID	32	Set at PDU send
Action ID	32	0x0000F001
Fixed Datum Records	32	Number of fixed datum records (4)
Variable Datum Records	32	Number of variable datum records (0)
<b>Fixed Datum #0</b>	64	
Datum ID	32	0x00
Value	32	PMid of the PM
<b>Fixed Datum #1</b>	64	
Datum ID	32	0x01
Value	32	ASAP of the PM
<b>Fixed Datum #2</b>	64	
Datum ID	32	0x02
Value	32	First 4 NSAP bytes of the PM
<b>Fixed Datum #3</b>	64	
Datum ID	32	0x03
Value	32	Last 4 NSAP bytes of the PM

**USAGE:** Generated by the PM, directed to the BSM. The PM will send one of these PDUs to the BSM after it has initialized itself and is ready to receive its criteria.

## CRITERIA ASSIGN

Field	Size (bits)	Value
<b>PDU Header</b>	96	
Protocol Version	8	0x04
Exercise ID	8	Set at runtime
PDU Type	8	0x10
Protocol Family	8	0x05
Time Stamp	32	Set at PDU send
Length	16	Variable
Padding	16	unused
<b>Originating Entity</b>	48	
Site	16	0xFFFFE (don't care)
Application	16	0x0000 (BSM_APP)
Entity	16	0xFFFFE (don't care)
<b>Receiving Entity</b>	48	
Site	16	0xFFFFE (don't care)
Application	16	0x0002 (PM_APP)
Entity	16	0xFFFFE (don't care)
Request ID	32	Set at PDU send
Action ID	32	0x0000F002
Fixed Datum Records	32	Number of fixed datum records (0)
Variable Datum Records	32	Number of variable datum records (1)
<b>Variable Datum #0</b>	VV	
Datum ID	32	0x00
Datum Length	32	Variable
Value	VV	Criteria set
Padding	VV	Padding to a multiple of 64 bits

**USAGE:** Generated by the BSM, directed to the PM. Used to give the PM the criteria set it will use to determine entity membership. The existing criteria set on the PM will be replaced.

## DM INTEREST EXPRESSION

Field	Size (bits)	Value
<b>PDU Header</b>	96	
Protocol Version	8	0x04
Exercise ID	8	Set at runtime
PDU Type	8	0x10
Protocol Family	8	0x05
Time Stamp	32	Set at PDU send
Length	16	Variable
Padding	16	unused
<b>Originating Entity</b>	48	
Site	16	0xFFFFE (don't care)
Application	16	0x0001 (DM_APP)
Entity	16	0xFFFFE (don't care)
<b>Receiving Entity</b>	48	
Site	16	0xFFFFE (don't care)
Application	16	0x0002 (PM_APP)
Entity	16	0xFFFFE (don't care)
Request ID	32	Set at PDU send
Action ID	32	0x0000F003
Fixed Datum Records	32	Number of fixed datum records (3)
Variable Datum Records	32	Number of variable datum records (1)
<b>Fixed Datum #0</b>	64	
Datum ID	32	0x00
Value	32	ASAP of the DM
<b>Fixed Datum #1</b>	64	
Datum ID	32	0x01
Value	32	First 4 NSAP bytes of the DM
<b>Fixed Datum #2</b>	64	
Datum ID	32	0x02
Value	32	Last 4 NSAP bytes of the DM
<b>Variable Datum #0</b>	VV	
Datum ID	32	0x00
Datum Length	32	Variable
Value	VV	Interest Expression
Padding	VV	Padding to a multiple of 64 bits

**USAGE:** Generated by the DM, directed to the PM. The variable datum will contain the IE data (i.e. geographic range, or other interest registration).

## PM INTEREST EXPRESSION

Field	Size (bits)	Value
<b>PDU Header</b>	96	
Protocol Version	8	0x04
Exercise ID	8	Set at runtime
PDU Type	8	0x10
Protocol Family	8	0x05
Time Stamp	32	Set at PDU send
Length	16	Variable
Padding	16	unused
<b>Originating Entity</b>	48	
Site	16	0xFFFFE (don't care)
Application	16	0x0002 (PM_APP)
Entity	16	0xFFFFE (don't care)
<b>Receiving Entity</b>	48	
Site	16	0xFFFFE (don't care)
Application	16	0x0000 (BSM_APP)
Entity	16	0xFFFFE (don't care)
Request ID	32	Set at PDU send
Action ID	32	0x0000F004
Fixed Datum Records	32	Number of fixed datum records (3)
Variable Datum Records	32	Number of variable datum records (1)
<b>Fixed Datum #0</b>	64	
Datum ID	32	0x00
Value	32	ASAP of the DM
<b>Fixed Datum #1</b>	64	
Datum ID	32	0x01
Value	32	First 4 NSAP bytes of the DM
<b>Fixed Datum #2</b>	64	
Datum ID	32	0x02
Value	32	Last 4 NSAP bytes of the DM
<b>Variable Datum #0</b>	VV	
Datum ID	32	0x00
Datum Length	32	Variable
Value	VV	Interest Expression
Padding	VV	Padding to a multiple of 64 bits

**USAGE:** Generated by the DM, modified by the PM, directed to the PM. The variable datum will contain the IE data from the DM, as expanded by the PM.

## MCAST ADD

Field	Size (bits)	Value
<b>PDU Header</b>	96	
Protocol Version	8	0x04
Exercise ID	8	Set at runtime
PDU Type	8	0x10
Protocol Family	8	0x05
Time Stamp	32	Set at PDU send
Length	16	64 (bytes)
Padding	16	unused
<b>Originating Entity</b>	48	
Site	16	From the entity expressing interest
Application	16	From the entity expressing interest
Entity	16	The entity expressing interest
<b>Receiving Entity</b>	48	
Site	16	0xFFFFE (don't care)
Application	16	0x0002 (PM_APP)
Entity	16	0xFFFFE (don't care)
Request ID	32	Set at PDU send
Action ID	32	0x0000F005
Fixed Datum Records	32	Number of fixed datum records (3)
Variable Datum Records	32	Number of variable datum records (0)
<b>Fixed Datum #0</b>	64	
Datum ID	32	0x00
Value	32	ASAP of the DM
<b>Fixed Datum #1</b>	64	
Datum ID	32	0x01
Value	32	First 4 NSAP bytes of the DM
<b>Fixed Datum #2</b>	64	
Datum ID	32	0x02
Value	32	Last 4 NSAP bytes of the DM

**USAGE:** Generated by the BSM, directed to a PM; indicates that the DM at the address contained in the fixed datum records should be added to the PM's multicast group on behalf of the entity specified in the originating entity ID.

## MCAST DELETE

Field	Size (bits)	Value
<b>PDU Header</b>	96	
Protocol Version	8	0x04
Exercise ID	8	Set at runtime
PDU Type	8	0x10
Protocol Family	8	0x05
Time Stamp	32	Set at PDU send
Length	16	64 (bytes)
Padding	16	unused
<b>Originating Entity</b>	48	
Site	16	From the entity expressing interest
Application	16	From the entity expressing interest
Entity	16	The entity expressing interest
<b>Receiving Entity</b>	48	
Site	16	0xFFFFE (don't care)
Application	16	0x0002 (PM_APP)
Entity	16	0xFFFFE (don't care)
Request ID	32	Set at PDU send
Action ID	32	0x0000F006
Fixed Datum Records	32	Number of fixed datum records (3)
Variable Datum Records	32	Number of variable datum records (0)
<b>Fixed Datum #0</b>	64	
Datum ID	32	0x00
Value	32	ASAP of the DM
<b>Fixed Datum #1</b>	64	
Datum ID	32	0x01
Value	32	First 4 NSAP bytes of the DM
<b>Fixed Datum #2</b>	64	
Datum ID	32	0x02
Value	32	Last 4 NSAP bytes of the DM

**USAGE:** Generated by the BSM, directed to a PM; indicates that the entity specified in the originating entity ID should be removed from the list of interested entities at the address contained in the fixed datum records.

## MCAST CLOSE

Field	Size (bits)	Value
<b>PDU Header</b>	96	
Protocol Version	8	0x04
Exercise ID	8	Set at runtime
PDU Type	8	0x10
Protocol Family	8	0x05
Time Stamp	32	Set at PDU send
Length	16	64 (bytes)
Padding	16	unused
<b>Originating Entity</b>	48	
Site	16	0xFFFE (don't care)
Application	16	0x0002 (PM_APP)
Entity	16	0xFFFE (don't care)
<b>Receiving Entity</b>	48	
Site	16	0xFFFE (don't care)
Application	16	0x0001 (DM_APP)
Entity	16	0xFFFE (don't care)
Request ID	32	Set at PDU send
Action ID	32	0x0000F007
Fixed Datum Records	32	Number of fixed datum records (3)
Variable Datum Records	32	Number of variable datum records (0)
<b>Fixed Datum #0</b>	64	
Datum ID	32	0x00
Value	32	ASAP of the specified DM
<b>Fixed Datum #1</b>	64	
Datum ID	32	0x01
Value	32	First 4 NSAP bytes of the specified DM
<b>Fixed Datum #2</b>	64	
Datum ID	32	0x02
Value	32	Last 4 NSAP bytes of the specified DM

**USAGE:** Generated by the PM, directed to a DM; indicates that DM whose address matches that contained in the fixed datum records should close the connection that this PDU arrived on.

## PM SPLIT

Field	Size (bits)	Value
<b>PDU Header</b>	96	
Protocol Version	8	0x04
Exercise ID	8	Set at runtime
PDU Type	8	0x10
Protocol Family	8	0x05
Time Stamp	32	Set at PDU send
Length	16	Variable
Padding	16	unused
<b>Originating Entity</b>	48	
Site	16	0xFFFFE (don't care)
Application	16	0x0002 (PM_APP)
Entity	16	0xFFFFE (don't care)
<b>Receiving Entity</b>	48	
Site	16	0xFFFFE (don't care)
Application	16	0x0000 (BSM_APP)
Entity	16	0xFFFFE (don't care)
Request ID	32	Set at PDU send
Action ID	32	0x0000F008
Fixed Datum Records	32	Number of fixed datum records (1)
Variable Datum Records	32	Number of variable datum records (1)
<b>Fixed Datum #0</b>	64	
Datum ID	32	0x00
Value	32	PMid of the PM requesting to be split
<b>Variable Datum #0</b>	VV	
Datum ID	32	0x00
Datum Length	32	Variable
Value	VV	Split suggestion
Padding	VV	Padding to a multiple of 64 bits

**USAGE:** Generated by the PM, directed to the PM. The variable datum will contain a suggestion of how the PM's criteria set should be split, if possible (for example, the XYZ coordinates of the highest activity in a geographic partitioning).

## PM DELETE

Field	Size (bits)	Value
<b>PDU Header</b>	96	
Protocol Version	8	0x04
Exercise ID	8	Set at runtime
PDU Type	8	0x10
Protocol Family	8	0x05
Time Stamp	32	Set at PDU send
Length	16	40 (bytes)
Padding	16	unused
<b>Originating Entity</b>	48	
Site	16	0xFFFFE (don't care)
Application	16	0x0000 (BSM_APP)
Entity	16	0xFFFFE (don't care)
<b>Receiving Entity</b>	48	
Site	16	0xFFFFE (don't care)
Application	16	0x0002 (PM_APP)
Entity	16	0xFFFFE (don't care)
Request ID	32	Set at PDU send
Action ID	32	0x0000F009
Fixed Datum Records	32	Number of fixed datum records (0)
Variable Datum Records	32	Number of variable datum records (0)

**USAGE:** Generated by the BSM, directed to the PM. Since each PM has a separate connection with the BSM, no PMid is necessary. The PM receiving a PM DELETE PDU will immediately terminate all processes.

- [1] -----, "The SIMNET Architecture for Distributed Interactive Simulation." Prepared for the 1991 Summer Computer Simulation Conference. IDA Simulation Center Library Files Serial Number 50514. July 1991.
- [2] Armitage, Grenville John. "The Application of Asynchronous Transfer Mode to Multimedia and Local Area Networks." Thesis submitted for the Degree of Doctor of Philosophy to the University of Melbourne, Australia. January 1994.
- [3] Boner, K.E., et al. "Battle Force Inport Training/Simulator Networking: SIMNET Protocol Suitability Considerations." Technical Note 1614, Naval Ocean Systems Center, San Diego CA. June 1990.
- [4] Calvin, James O., et al. "Application Control Techniques System Architecture." *Proceedings of the 12th DIS Workshop*. February 1995.
- [5] CCITT SGXVIII Draft recommendation I.362, "B-ISDN ATM Adaptation Layer (AAL) Functional Description", June 1992.
- [6] CCITT SGXVIII Draft recommendation I.363, "B-ISDN ATM Adaptation Layer (AAL) Specification", June 1992.
- [7] CCITT SGXVIII Report R109, Draft recommendation I.150, "B-ISDN Asynchronous Transfer Mode Functional Characteristics", July 1992.
- [8] Chung, J.W. "An Assessment and Forecast of Commercial Enabling Technologies for Advanced Distributed Simulation." Technical report, Institute for Defense Analysis. October 1992.
- [9] Comer, Douglas E. *Internetworking with TCP/IP, Vol. I*. Prentice-Hall, New Jersey. 1991.
- [10] Flanagan, William A. *Asynchronous Transfer Mode User's Guide*. Flatiron Publishing, Inc., New York. 1994.
- [11] Fore Systems ForeRunner™ ESA-200 ATM ESIA Bus Adapter for Silicon Graphics User's Manual. March 1996.
- [12] IEEE Standards Department. "Standard for Distributed Interactive Simulation—Communication Architecture Requirements." Draft Standard P1278.2. 1994.
- [13] Johnson, A. Steven, Fore Systems' Network Engineer. Personal communication. July 1996.
- [14] Krivda, Cheryl D. "Analyzing ATM Adapter Performance: The Real-World Meaning of Benchmarks". Efficient Networks, Inc. white paper. 1996.
- [15] Macedonia, Michael R., et al. "Exploiting Reality with Multicast Groups: A Network Architecture for Large-scale Virtual Environments." *Proceedings of the Virtual Reality Annual International Symposium '95*, March 1995.
- [16] McDonald, L. Bruce, et al. "Standard Protocol Data Units for Entity Information and Interaction in a Distributed Interactive Simulation." *Proceedings of the 13th Interservice/Industry Training Systems Conference*. 1991.

- [17] Meliza, Larry L. and Seng Chong Tan. "Application of the SIMNET Unit Performance Assessment System to After Action Reviews." *Proceedings of the 13th Interservice/Industry Training Systems Conference*. 1991.
- [18] Pope, Arthur R. "The SIMNET Network and Protocols." Bolt Beranek and Newman Inc Report Number 7627 prepared for the Defense Advanced Research Projects Agency. June 1991.
- [19] Powell, Edward T. Personal communication. May 1996.
- [20] Powell, Edward T., et al. "Joint Precision Strike Demonstration Simulation Architecture." *Proceedings of the 14th DIS Workshop*. March 1996.
- [21] Pullen, J. Mark, and David C. Wood. "Networking Technology and DIS." *Proceedings of the IEEE*, Vol 83, No 8, 1156-1167. August 1995.
- [22] Sadawi, Tarek N., Ammar, Mostafa H., and Hakeem, Ahmed El. *Fundamentals of Telecommunication Networks*. John Wiley & Sons, Inc. New York. 1994.
- [23] Sloane, Gary, and Kevin Walsh. "Parallel Programming on Silicon Graphics Computer Systems". Silicon Graphics, Inc., Document Number 007-0770-020. 1991.
- [24] Stallings, William. *Data and Computer Communications*, fifth edition. Prentice-Hall, New Jersey. 1996.
- [25] Thorpe, Lt Col Jack A. "The New Technology of Large Scale Simulator Networking: Implications for Mastering the Art of Warfighting." *Proceedings of the Ninth Interservice/Industry Training Systems Conference*. 1987.
- [26] Troxel, Gregory D. "High Performance Application Gateway Bilevel Multicasting." *Proceedings of the 12th DIS Workshop*. February 1995.
- [27] Van Hook, Daniel J., et al. "An Approach to DIS Scaleability." *Proceedings of the 11th DIS Workshop*, September 1994.
- [28] Van Hook, Daniel J., et al. "Approaches to Relevance Filtering." *Proceedings of the 11th Workshop on Standards for the Interoperability of Distributed Simulations*. September 1994.
- [29] Van Hook, Daniel J., et al. "Performance of STOW RITN Application Control Techniques." *Proceedings of the 14th DIS Workshop*, March 1996.
- [30] Zeswitz, Steven Randall. "NPSNET: Integration of Distributed Interactive Simulation (DIS) Protocol for Communication Architecture and Information Interchange." Thesis submitted for the Degree of Master of Computer Science to the Naval Postgraduate School, Monterrey, California. September 1993.

## Vita

Captain David T. Hightower was commissioned in the USAF on 4 May 1991 after graduating from the University of Florida with a Bachelor of Science in Computer Science. His first assignment was temporary duty at Basic Computer/Communications Officer Training at Keesler Air Force Base, Mississippi. Upon graduation, he was assigned to the Air Force Wargaming Institute at Maxwell AFB, where he served as a system and network administrator for 3 years. He received a scholarship to attend the Air Force Institute of Technology and entered the school of Engineering in May 1995. Captain Hightower served four years in the United States Army as an air traffic controller prior to pursuing his degree at UF.

Permanent Address: 1136 Swamp Mill Circle  
Sumter, SC 29154

REPORT DOCUMENTATION PAGE			Form Approved OMB No. 0704-0188
Public reporting burden for this collection of information is estimated to average 1 hour per response, including the time for reviewing instructions, searching existing data sources, gathering and maintaining the data needed, and completing and reviewing the collection of information. Send comments regarding this burden estimate or any other aspect of this collection of information, including suggestions for reducing this burden, to Washington Headquarters Services, Directorate for Information Operations and Reports, 1215 Jefferson Davis Highway, Suite 1204, Arlington, VA 22202-4302, and to the Office of Management and Budget, Paperwork Reduction Project (0704-0188), Washington, DC 20503.			
1. AGENCY USE ONLY (Leave blank)	2. REPORT DATE December 1996	3. REPORT TYPE AND DATES COVERED Master's Thesis	
4. TITLE AND SUBTITLE Dynamic Relevance Filtering in Asynchronous Transfer Mode-Based Distributed Interactive Simulation Exercises		5. FUNDING NUMBERS	
6. AUTHOR(S) David T. Hightower, Captain, USAF			
7. PERFORMING ORGANIZATION NAME(S) AND ADDRESS(ES) Air Force Institute of Technology 2750 P Street WPAFB OH 45433-6583		8. PERFORMING ORGANIZATION REPORT NUMBER  AFIT/GCS/ENG/96D-09	
9. SPONSORING/MONITORING AGENCY NAME(S) AND ADDRESS(ES)		10. SPONSORING/MONITORING AGENCY REPORT NUMBER	
11. SUPPLEMENTARY NOTES			
12a. DISTRIBUTION AVAILABILITY STATEMENT Approved for public release; distribution unlimited		12b. DISTRIBUTION CODE	
13. ABSTRACT (Maximum 200 words) As Distributed Interactive Simulation (DIS) exercises continue to grow in scale, the need to support a large number of players has become apparent. The demands on the network and the simulation hosts in large exercises, though, have proved to be prohibitive, requiring significant computational overhead to filter through the information and extract what is relevant to a particular simulation. Some mechanism is needed to reduce irrelevant network traffic received by a system, while increasing the bandwidth available for the DIS exercise. Previous research efforts in this area have centered primarily on fixed geographic partitions of the battlespace to reduce the traffic at a given host. This geographic partitioning cannot adapt to the changing battlespace, and requires relatively significant pre-exercise setup and coordination. Our research has been to implement a DIS exercise system using native ATM interfaces, and to determine if a dynamic partitioning system is feasible and will provide a sufficient reduction in network traffic to allow DIS exercises to scale to the target 100,000 entities. A support infrastructure for DIS over ATM was developed and tested with current AFIT DIS applications, and a prototype dynamic partitioning system using geographic criteria was implemented.			
14. SUBJECT TERMS DIS, distributed, interactive, simulation, networking, UDP/IP, ATM, asynchronous, transfer, mode, communications protocols		15. NUMBER OF PAGES 95	
		16. PRICE CODE	
17. SECURITY CLASSIFICATION OF REPORT Unclassified	18. SECURITY CLASSIFICATION OF THIS PAGE Unclassified	19. SECURITY CLASSIFICATION OF ABSTRACT Unclassified	20. LIMITATION OF ABSTRACT UL