Calhoun: The NPS Institutional Archive

Theses and Dissertations | Thesis Collection

1995-06

A network software architecture for large scale virtual environments

Macedonia, Michael R.

Monterey, California. Naval Postgraduate School

http://hdl.handle.net/10945/31461

DUDLEY KNOX LIBRARY

Calhoun is a project of the Dudley Knox Library at NPS, furthering the precepts and goals of open government and government transparency. All information contained herein has been approved for release by the NPS Public Affairs Officer.

Dudley Knox Library / Naval Postgraduate School
411 Dyer Road / 1 University Circle
Monterey, California USA 93943

http://www.nps.edu/library

NAVAL POSTGRADUATE SCHOOL
Monterey, California

DISSERTATION

A NETWORK SOFTWARE ARCHITECTURE FOR LARGE SCALE VIRTUAL ENVIRONMENTS

by

Michael R. Macedonia

June 1995

Dissertation Supervisor: Michael J. Zyda

Approved for public release; distribution is unlimited.

DTIC QUALITY INSPECTED 3

19961024 049

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 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
June 1995

3. REPORT TYPE AND DATES COVERED
Dissertation

4. TITLE AND SUBTITLE
A NETWORK SOFTWARE ARCHITECTURE FOR LARGE-SCALE VIRTUAL ENVIRONMENTS (U)

5. FUNDING NUMBERS

6. AUTHOR(S)
Macedonia, Michael R

7. PERFORMING ORGANIZATION NAME(S) AND ADDRESS(ES)
Naval Postgraduate School
Monterey, CA 93943-5000

8. PERFORMING ORGANIZATION REPORT NUMBER

9. SPONSORING/ MONITORING AGENCY NAME(S) AND ADDRESS(ES)

10. SPONSORING/ MONITORING AGENCY REPORT NUMBER

11. SUPPLEMENTARY NOTES
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.

12a. DISTRIBUTION / AVAILABILITY STATEMENT
Approved for public release; distribution is unlimited.

12b. DISTRIBUTION CODE

13. ABSTRACT *(Maximum 200 words)*

We present a network software architecture for solving the problem of scaling large distributed simulations. The motivation for our effort is to expand the capability of virtual environments to serve large numbers (more than 1,000) of simultaneous users. The fundamental idea of our approach is to logically partition virtual environments by associating spatial, temporal, and functionally related entity classes with network multicast groups. Furthermore, we exploit the actual characteristics of the real-world large-scale environments that are simulated by focusing or restricting an entity's processing and network resources to its area of interest via a local Area of Interest Manager (AOIM) and a persistent object protocol.We first discuss related work in the area of networked virtual environments and the problems of developing scalable VEs. The dissertation also provides a taxonomy for discussing VEs in terms of communication methods, data, processes, and views. Moreover, we describe the Distributed Interactive Simulation (DIS) efforts and the limits of DIS today. Finally, we present our theory and the results of simulations using the AOIM. We used data from the U.S. Army National Training Center and the Janus combat model to show how the movement rates and densities of thousands of combat systems allows the use of the AOIM by an military entity to limit network traffic and simulation computation, maintain acceptable reliability, and minimize the effects of latency.

14. SUBJECT TERMS
Computer Graphics, Simulator, Simulation, Networks, Virtual worlds, Artificial Reality, Synthetic Environments, NPSNET, Distributed, Interactive, Multicast

15. NUMBER OF PAGES
256

16. PRICE CODE

17. SECURITY CLASSIFICATION OF REPORT	18. SECURITY CLASSIFICATION OF THIS PAGE	19. SECURITY CLASSIFICATION OF ABSTRACT	20. LIMITATION OF ABSTRACT
Unclassified	Unclassified	Unclassified	Unlimited

NSN 7540-01-280-5500

Standard Form 298 (Rev. 2-89)
Prescribed by ANSI Std. 239-18

Approved for public release; distribution is unlimited

A NETWORK SOFTWARE ARCHITECTURE FOR LARGE SCALE VIRTUAL ENVIRONMENTS

Michael R. Macedonia
B.S., United States Military Academy, 1979
M.S., University of Pittsburgh, 1989

Submitted in partial fulfillment of the
requirements for the degree of

DOCTOR OF PHILOSOPHY IN COMPUTER SCIENCE

from the

NAVAL POSTGRADUATE SCHOOL
June 1995

Author: [redacted]
Michael R. Macedonia

Approved by: [redacted]

David R. Pratt
Assistant Professor of Computer Science

Ted Lewis
Chairman and Professor of Computer Science

[redacted]

Dan C. Boger
Professor of Systems Management

Michael G. Sovereign
Professor of Operations Research

[redacted]

Michael J. Zyda
Professor of Computer Science
Dissertation Supervisor

Approved by: [redacted]
Ted Lewis, Chairman, Department of Computer Science

Approved by: [redacted]
Richard S. Elster, Dean of Instruction

ABSTRACT

We present a network software architecture for solving the problem of scaling large distributed simulations. The motivation for our effort is to expand the capability of virtual environments to serve large numbers (more than 1,000) of simultaneous users. The fundamental idea of our approach is to logically partition virtual environments by associating spatial, temporal, and functionally related entity classes with network multicast groups. Furthermore, we exploit the actual characteristics of the real-world large-scale environments that are simulated by focusing or restricting an entity's processing and network resources to its area of interest via a local Area of Interest Manager (AOIM) and a persistent object protocol.

We first discuss related work in the area of networked virtual environments and the problems of developing scalable VEs. The dissertation also provides a taxonomy for discussing VEs in terms of communication methods, data, processes, and views. Moreover, we describe the Distributed Interactive Simulation (DIS) efforts and the limits of DIS today. Finally, we present our theory and the results of simulations using the AOIM. We used data from the U.S. Army National Training Center and the Janus combat model to show how the movement rates and densities of thousands of combat systems allow the use of the AOIM by an military entity to limit network traffic and simulation computation, maintain acceptable reliability, and minimize the effects of latency.

TABLE OF CONTENTS

I. INTRODUCTION 1
A. THESIS STATEMENT 1
B. SCALABILITY 1
C. EXPLOITING REALITY 5
1. Goals 5
2. Method 5
D. DISSERTATION OVERVIEW 8
II. RELATED WORK 10
A. OVERVIEW 10
B. TAXONOMY 10
1. Communication 11
a. Bandwidth 11
b. Distribution 14
c. Latency 16
d. Reliability 18
e. Summary 20
2. Views 20
3. Data 22
a. Replicated homogeneous world 23
b. Shared, centralized databases 23
c. Shared, distributed databases with peer-to-peer updates 25
d. Shared, distributed, client-server databases 26
e. Summary 27
4. Processes 27
C. SUMMARY 30
III. SIMNET and DIS 31
A. OVERVIEW 31
B. SIMNET 31
C. DIS PROTOCOL 36
1. Protocol Data Units 37
2. Simulation Philosophy 37
3. DIS Performance Values 42
4. Problems with the DIS Protocol 45
a. Enormous bandwidth and computational requirements 45
b. Multiplexing of different media at the application layer 46
c. Lack of an efficient method of handling static objects 46
d. Models and world databases must be replicated at each simulator 47
e. Abstractions 47
f. Security 48
g. Summary 49
5. Reasons for Problems 49
a. Event and State message paradigm 49
b. Real-time system trade-off's 50
c. No "middleware" layer 50
d. Origins as small unit training systems 51

D. APPLICATION GATEWAYS 53
1. Techniques 54
a. PICA 54
b. QES 54
c. Grid filtering 55
d. Culling 55
e. Summary 55
2. Application Gateway Problems 55
E. SUMMARY 57
IV. NPSNET OVERVIEW 58
A. INTRODUCTION 58
B. EVOLUTION 59
C. NETWORK ENTITY/PDU PROCESSING 61
D. HETEROGENEOUS PARALLELISM 69
1. Advantages 71
2. Disadvantages 73
E. DEMONSTRATED RESULTS USING IP BROADCAST AND DIS 73
F. SUMMARY 77
V. NETWORK INFRASTRUCTURE 78
A. INFRASTRUCTURE 78
B. WIDE AREA NETWORKS 78
C. LOCAL AREA NETWORKS 81
1. ATM 83
D. IMPLICATIONS 84
VI. IP MULTICAST 87
A. IP MULTICAST AND THE MBONE 87
1. Bandwidth Constraints 89
2. Internet Group Management Protocol 91
3. Topology and Event Scheduling 94
4. Protocols 94
5. Data Compression 95
6. Application Tools 96
B. VIRTUAL ENVIRONMENT RESEARCH WITH MULTICAST 97
C. NPSNET AND MBONE 101
D. SYMPOSIUM ON 3D INTERACTIVE GRAPHICS 103
E. SUMMARY 106
VII. THEORY 107
A. GOALS 107
1. Computational and Bandwidth Requirements 107
2. Maintain the Current DIS Semantics 107
3. Reduced Latency for New Entrant Learning 108
4. Fully Distributed 108
5. Elimination of the Static and Dead Entity Problem 108
6. Localization of Reliability Problems 108
B. HEURISTICS 108
1. Domain Specificity 108
2. Behavior Characterization 109
3. Partitioning 110
4. Real-time Efficiency 111
5. Communications Model 111
a. Light-weight interactions 112

b. Network pointers 112
c. Heavy-weight objects 113
d. Real-time streams 113
e. Summary 113
C. THE BATTLEFIELD ENVIRONMENT 114
1. Limited Entity Area of Interest 114
2. Aggregate Behavior 117
D. DIS AREA OF INTEREST MANAGER 121
1. Spatial Associations 122
2. Hexagonal Tessellation 123
3. Persistent Object Protocol 127
4. Rationale 130
5. Entity Interactions 131
E. SUMMARY 131
VIII. SIMULATION 132
A. OBJECTIVES 132
B. MEASURES OF EFFECTIVENESS 132
1. Communication Traffic Rate 132
2. Multicast Groups 132
3. Entities in Multicast AOI 133
4. Group Transition Rate 133
C. HYPOTHESES 133
D. EXPERIMENTAL DESIGN 134
1. Step One: Combat Scenario 134
2. Step Two: Janus Modeling 137
3. Step Three: Janus Post-processing 139
4. Step 4: HexSim 139
5. Hypothesis Testing 142
E. EXPERIMENTAL ASSUMPTIONS 145
F. DATA AND ANALYSIS 147
a. Communication traffic among entities 147
b. Active multicast groups 149
c. Maximum Entities in AOI 152
d. Entity transitions 153
e. Bandwidth scalability 156
f. Entities in AOI 158
G. SUMMARY 159
IX. CONCLUSIONS 161
A. IMPLICATIONS OF WORK 161
1. Bandwidth 161
2. Computation 162
3. New Entrant Learning 162
4. Distributed Processing 162
5. Static Entity Problem 163
6. Localization of Reliability Problems 163
7. DIS Semantics 163
B. LIMITATIONS OF THE WORK 163
C. CONTRIBUTIONS 165
D. FUTURE WORK 166
E. REVIEW 166
LIST OF REFERENCES 167

APPENDIX A 177
APPENDIX B 196
APPENDIX C 226
INITIAL DISTRIBUTION LIST 232

LIST OF TABLES

1: Vehicle Appearance PDU. 35
2: Entity State PDU 38
3: LAN technologies. 81
4: Military intervisibility values in meters. 115
5: Peak multicast communications traffic. 147
6: Cell occupancy 150
7: Peak number of entities in AOI for multicast. 152
8: Entity transitions and density 156
9: Bandwidth vs. number of entities 158
10: AOI vs. number of entities 158

LIST OF FIGURES

1. Scalability 4
2. Relationship between entity and multicast groups. 7
3. Communication issues. 11
4. Examples of broadcast, multicast, and unicast. 14
5. Distributed model. 16
6. Two views of the simulation. 21
7. Data models for VEs. 22
8. Centralized model. 24
9. Server vs. client communication in Netrek. 25
10. Close-Combat Tactical Trainer (CCTT). 32
11. SIMNET simulator. 33
12. Dead reckoning. 39
13. First player in. 40
14. Next player(s) in. 41
15. Ship position on different screens. 41
16. Ship position after update. 41
17. Time traces of PDU events. 42
18. PDU rates (10 second sample rate). 43
19. Bandwidth vs. number of players. 44
20. Problems with DIS. 45
21. An armored brigade in the attack at the National Training Center. 52
22. Application Gateway (AG) setup. 54
23. AG processing. 56
24. Medic conducting first-aid in NPSNET. 58
25. Evolution of NPSNET networking. 61
26. Incoming PDU handling in NPSNET-IV. 63
27. Entity State PDU processing in NPSNET-IV. 64
28. Entity update operation in NPSNET-IV. 66
29. Control of local entity movement in NPSNET-IV. 67
30. Processing of incoming Detonation PDUs by NPSNET-IV. 68
31. Process models. 71
32. Multi-process flow. 72
33. Implementation of IP Broadcast and IP Multicast in NPSNET. 74
34. SIGGRAPH'93 setup. 76
35. Rich VE network topology. 85
36. Asymmetric VE network topology. 85
37. Simple illustration of multicast communications. 88
38. Map of the MBONE [31]. 89
39. Source-based multicast trees. 92
40. Core-based multicast trees. 93
41. Session directory (sd) tool. 97
42. Visual audio tool (vat). 98
43. Object-based filtering. 100
44. MBONE routing between NPS and SRI. 102
45. IP Multicast between SRI and NPS. 103

46. Location of participants. 105
47. Military communication networks. 116
48. US division in the defense. 118
49. Battlefield at NTC. 119
50. Area of Interest Manager. 121
51. Mapping Cartesian to Hex coordinate system. 125
52. Area of Interest for vehicle mapped to a subset of multicast groups. 126
53. Hexagon geometry. 126
54. Cell transition from current active to northeast cell in MODSIM II. 128
55. Entity transition protocol. 130
56. Simulation development. 135
57. Brigade task force organization. 136
58. Entity object and module definitions. 140
59. One and two km hexes. 143
60. Three and four km hexes. 144
61. Regression analysis for peak multicast traffic. 148
62. Mean multicast (4 km hex) vs. broadcast traffic. 148
63. Traffic using four km radius cells. 149
64. Distribution of entity peak bandwidth. 150
65. Regression analysis for active multicast groups. 151
66. Entity distribution using hex coordinates. 151
67. Regression analysis for peak entities in AOI. 152
68. Peak entities in AOI vs. hex size. 153
69. Max entities in AOI by hex size. 154
70. Regression analysis for total entity transitions. 154
71. Peak entities in hex (top) and total transitions vs. hex size 155
72. Distribution of peak traffic. 157
73. Comparison of max entities in AOI. 159
74. Distribution of entities per hex. 160

PREFACE

A. MOTIVATION FOR BUILDING VIRTUAL ENVIRONMENTS

This thesis explores potential methods for constructing large-scale virtual worlds (VW). Potential is a critical modifier because these worlds still do not exist. However, we are on the cusp of a technological wave. Like the advertisement from AT&T says: "You will". The second reason for what follows is prescriptive and it provides an idea of how to build some of the next generation of dreams. These dreams are the subject of a new field within computer science -- the study of virtual environments-- whose form and potential we are just beginning to learn.

The importance of our work stems from the development of two *recent* technologies meant for building systems of systems -- networks and computer graphics. In particular, it is multicast internetworks as embodied in the experimental Internet Multicast Backbone (MBONE) and real-time interactive 3D graphics that makes possible the idea of large-scale virtual environments (VEs). That the technology rapidly advances is demonstrated by the fact that the MBONE and many of the protocols that support it did not exist before 1991 [29].

Data networks transform virtual reality (VR) into a shared environment, allowing real-time interaction among people and processes without regard to their location. It is this illusion that allows the use of virtual environments for broad areas of research and application including distance learning, Computer-Supported Cooperative Work (CSCW), MUltiuser Dungeons (MUDs), distributed simulation, and group entertainment. Furthermore, the synthesis of VR and networks has led to a number of new terms and fields which contain the "tele" root: telerobotics, telemedicine, teleoperators, telepresence, teletravel.

For example, Randy Pausch at the University of Virginia has suggested the most promising use of virtual environments and networks will be for applications where people

at different locations need to jointly discuss a three-dimensional object, such as radiologists using a VR representation of a CAT scan [104]. Several researchers are already examining the use of telepresence for surgical applications. Philip Green at SRI International is developing for ARPA's virtual medicine project a teleoperated laproscopic device that uses force reflection to provide haptic feedback to a distant surgeon [141].

Another exciting concept is that of virtual libraries being developed at the Microelectronics Center for North Carolina (MCNC) Center for Communications. Their project, Cyberlib, will allow patrons to venture into "the information space independently" or go to a "virtual reference desk" from anywhere across the United States via the Internet [78]. We already have virtual newspapers. The San Jose Mercury News publishes its entire text (including classifieds) via the American Online service using graphically-based software for the Macintosh and IBM personal computers.

The comic strip Doonesbury was not far off the mark when a character received a visit from a representative of the Home Shopping Network (HSN) who brought a "new virtual reality shopping helmet" that "once installed, you can explore our computer generated shopping environment with over 275,000 exciting new products to choose from!" The irony is that it not really a joke. HSN is actually pursuing the dream of virtual Walmarts through new cable television and distributed computing technologies.

The technology of Dick Tracy is becoming reality. We can recall the Videophone which, for forty years, would soon be showing up in our neighborhoods. Now you can buy the device from AT&T and MCI for less than $1000.

An even less serious but, perhaps a more lucrative, combination of virtual environments and networking is already in use. Both Genie and the Imagination Network services provide networked interactive, multi-user VR games albeit with slow telephone lines and

Definition. The terms virtual reality, virtual world, virtual environment, and synthetic environment, are used interchangeably here. Their differences are mostly minor nuances and rooted in the politics of a new technology. We define them in general to mean "the technology for moving through and interacting with a three dimensional computer generated environment" such that the experience is perceived to be real [163].

A major component of the real world is the ability to communicate and interact with other people or entities, whether one-on-one or in a crowd. Perhaps the most appropriate word that defines the goal of this work is Metaverse, Neil Stephenson's networked virtual world in his novel "Snow Crash"[137]. In this world, VR is used as a social medium and its existence is based on a common communications protocol for defining both the world and its participants.

limited graphics. Sega and Nintendo have already announced their intent to provide interactive multi-player games over cable and satellite services.

VR and high speed networks are going to be the tools that allow us to explore Mars and our own oceans. NASA Ames is examining the use of VR to control robotic explorers, while scientists at the Monterey Bay Aquarium Research Institute are integrating such diverse technologies as computer simulations, robotics, and VR with a sophisticated undersea local area network to explore the nation's newest marine sanctuary.

High speed networks also allow VR systems to take advantage of distributed resources including shared databases, multimedia sources, and processors, while providing the required computational power for building the most demanding VR applications. They will provide VR applications with access to huge data sets generated by space probes, dynamic

climatic information from weather models, and real-time imaging systems such as ultrasound.

B. Large Virtual Environments

The applications mentioned above exist or are in development today. Yet, these VEs are limited to small numbers of users. Until recently, very little has been done to develop the fundamental research, ideas and technologies for large-scale VEs. Our effort exploits this revolution in networking so that we can expand the capabilities of simulations and virtual environments to serve medium to *large numbers* (greater than 1000) of simultaneous users. This has been a core but as yet unrealized idea of virtual reality. As Jaron Lanier wrote:

> Virtual reality (VR) was intended to emphasize the social nature of shared (networked) virtual environments, and to emphasize that one's own body was in the world as well as in the external environment [82].

Moreover, interest by the government, military, and telecommunications industry in large distributed virtual environments has been rapidly growing. In Figure 1 we see some of the military uses and programs for distributed virtual environments. Defense organizations like ARPA and the US Army have recognized their importance through programs such as the Combined Arms Tactical Trainer (CATT), a distributed 3D simulator for training tank crews and units, and the Louisiana Maneuver (LAM) exercises which will involve major elements of the Army in large simulated battles at the end of the decade to test new operational concepts.

The US Army has plans for the LAM initiative later in this decade which envisions 10,000 to 100,000 autonomous and human players participating in a simulation over a global wide-area network [147].

MILITARY USES FOR LARGE-SCALE VIRTUAL ENVIRONMENTS

- C3 experimentation.
  ARPA Warbreaker.
- Unit training.
  CATT.
- Concept development.
  Louisiana Maneuvers.
- Joint warfare training.
  Synthetic Theater of War 97.
- Concurrent engineering for large systems.
  Navy ship design.

Figure 1. Military uses for VEs

In a military context, a virtual environment of this magnitude (described as a Synthetic Theater of War [STOW]) enables a sense of collocation - sharing the same virtual battle space - among many geographically dispersed players such as an aviation unit in Texas flying in support of an armored cavalry squadron in Georgia. The Army will begin examining the effect of its next generation RAH-66 Comanche light armed scout helicopter on the joint battlefield at a simulator at Fort Rucker, Ala. The Comanche simulator will be hooked up via a distributed interactive simulation network, allowing the Army to run models looking at what Comanche brings to the Army's 21st century fighting force. According to the developers: "The guy can sit in the cockpit [and] can tie in with war games that are being run, these advanced warfighting experiments, and actually show the value of Comanche on the digital battlefield [8]."

The Federal Aviation Administration is also interested in large scale virtual environments in order to train the complete system of air controllers, aviators, and equipment. Boe-

ing is building a large scale virtual environment for its B777 airliner, the first modern aircraft built without a mock-up, to speed the process of concurrent engineering. Though Boeing's effort is on a massive scale with respect to the complexity and size of the data involved, the number of concurrent users may be equally immense when considering the huge number of engineering and subcontractor teams involved. Similarly, the Navy is exploring the use of simulation-based design for ships [129].

Commercial interest is also keen. Companies such AT&T, TCI, and Nintendo are pursuing distributed VR for multi-player games. Robert Kavner, former CEO for Multimedia Products and Services, AT&T, in a speech to the 1994 Winter Consumer Electronics Show stated:

> The only dedicated gaming network in today's narrowband world is ImagiNation Network, in which we are a part-owner -- and, more importantly, with whom we are working to develop new services. It has advanced graphics and lots of interactive flexibility. As people use this communications-intensive service, they're seeing its potential and adapting it to their life-styles. We are often asked why are we working with this small, online network? We are working with ImagiNation Network to find ways that people can use the network to strengthen their sense of community.

C. SUMMARY

We want to build software that allows the above capability and more. However, much remains to be done in the development of a network architecture to support large-scale virtual environments. The next chapters will present the scalability problem, past and current work in developing VEs, our solution and research supporting its validity.

ACKNOWLEDGMENTS

A grateful thanks to those who helped me pursue this path. In particular, Paul Barham, for his aid on the NPSNET chapter, Don Brutzman, for his persistence, Rich Gossweiller for the use of his figures, Steve Casner for the MBONE map, and my committee -- Dan Boger, Ted Lewis, Dave Pratt, Mike Sovereign, and Mike Zyda -- for their wisdom and guidance. Finally, I thank my father and mother, Ray and Madonna, my wife and closest friend, Terri, and my children, Rebecca and Bobby for their love and support.

I. INTRODUCTION

A. THESIS STATEMENT

Virtual environment software architectures can exploit wide area multicast communications and entity relationships to partition the virtual world and enable the development of ***scalable*** distributed interactive simulations for military applications.

B. SCALABILITY

The good news for those interested in developing distributed virtual environment is that advances in computer architectures and graphics, as well as standards such as the Distributed Interactive Simulation (DIS) and SIMNET (Simulator Networking) protocols have made small-scale (less than 300 players) realistic simulations possible [96].

The bad news is that, until recently, the network software components of virtual environments has often been overshadowed by issues such as human interface design or graphics realism. This is reflected in the paucity of academic research as will be shown in the chapter on related work. The network component has often been an afterthought for many research efforts. Though the Department of Defense (DOD) has invested billions in the development of distributed simulation, very little of that money has been spent on nonproprietary research. Furthermore, only in the past five years has serious interest been taken by the academic community in virtual environments. In turn, this has led to the fact that there have not been any good solutions to the scalability problem.

Network bandwidth is part of the problem. In a hypothetical case developed in a study by Loral, in schemes such as SIMNET, 100,000 players could require 375 Mbit per second (Mbps) of network bandwidth to each computer participating in the simulation. This is an unrealistic requirement for an affordable system in this decade [84].

We define a scalable architecture as a general framework that supports a virtual environment with increasingly larger number of concurrent dynamic entities or players without fundamental modifications to that architecture. **The scalability problem is: how do we construct virtual world architectures that efficiently scale from a handful of participating entities to thousands of players without substantially modifying the architecture?** Efficiency is defined as linear or constant growth in the resources (e.g., processors) required for the virtual environment as the VE increases in the number of participating entities.

Computational loads are another major issue. The Army faced the problem of scalability directly in 1994 with the Synthetic Theater of War -Europe (STOW-E) demonstration. The goal of the exercise was to demonstrate the capability to simulate 10,000 entities over a wide area network connecting a number of sites in Europe and the United States. Earlier studies had predicted that 10,000 entities was a feasible target by 1994 and that 100,000 was obtainable by 2000 [68]. STOW-E was composed of manned simulators, field-instrumented systems, Semi-Autonomous Forces (SAFOR or SAF) and traditional constructive war-game simulations (e.g. Brigade/Battalion Simulation or BBS). However, only 1,800 entities could be represented because of computational bottlenecks in both the simulators and in support equipment [133].

Furthermore, the systems participating did not have real world aspects such as dynamics or collision detection nor would it be possible with any technology conceivable in the near future. Considering Pentland's example in the text box, if we had 1000 objects, each intersecting the bounding box of 10 others, and each object had 1000 vertices, then contact

Faster computers and networks alone will not satisfy the requirements for scalable VEs. First, faster networks require faster processors merely to copy packets from the network into user space even before the application touches the protocol data unit (PDU). Second, the ever-present demand for more realism will introduce a rapid rise in computational and space complexity with even modest size VEs. Alex Pentland states that "before such worlds can be built we must first invent algorithms for dynamics, collision detection, and constraint satisfaction that scale linearly (or better!) with increasing problem size" [105]. He noted that for collision detection between two non-convex bodies with n polygons each requires comparing all polygons against all polygons, at a cost of $O(n^2)$ expensive operations, where n is the number of polygons. Using the finite element method (FEM) for physically-correct modelling requires $O(n^3)$ calculations for n vertices on an object.

We conjecture that on the order of one thousand entities are the upper bound limit to which a single host can realistically manage in real-time despite future advances in computer and graphics architectures because of computational complexity.

detection would require 10^{10} operations or roughly 1,000,000 MFLOPS for interactive performance. Rigid body dynamics require 10^9 operations for the same size object [105].

Even commercial systems are finding the limits to current technology in creating networked environments for large numbers of users. American Online (AOL) provides low-bandwidth multiuser games, messaging, and chat services for over 600,000 subscribers, yet has been forced to block new users at times because customer network demand has overwhelmed AOL's host computer. As one observer put it - "America Online's problems are

not trivial" [94]. AOL will likely delay introduction of new services to dampen growth - not a good sign for proponents of the information superhighway.

Another key challenge is that the appropriate systems involving human operators must deliver packets with minimal latency (less than 100 ms) and generate textured 3D graphics at 30-60 Hz to guarantee the illusion of reality [158]. On top of this is the need to provide real-time audio, video, and imagery services for the simulation of player communication services.

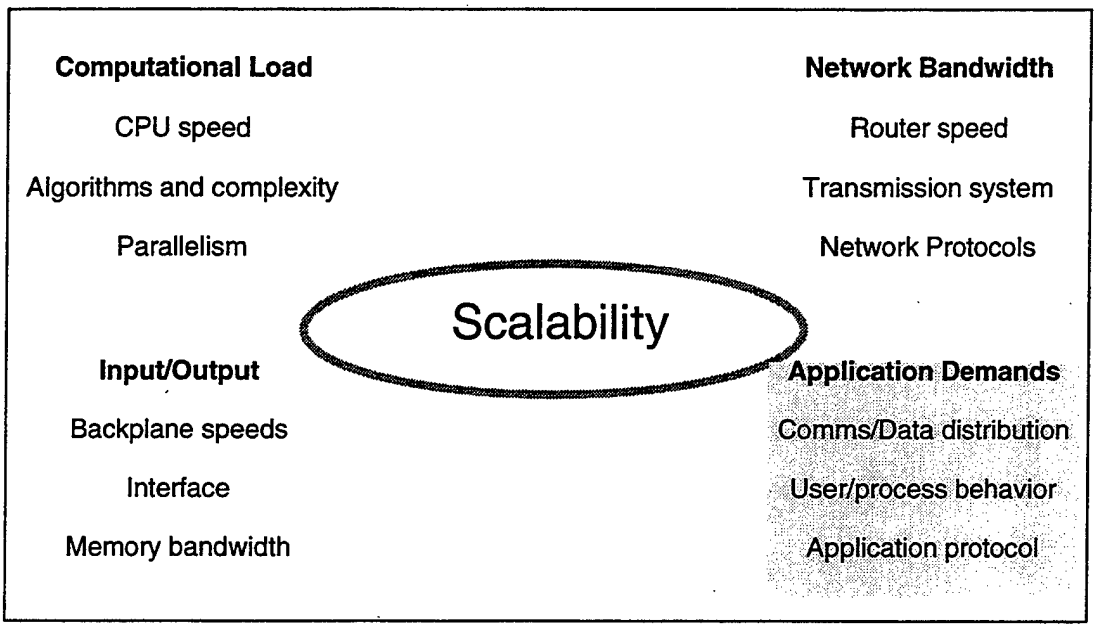


Figure 1. Scalability

Designing a scalable architecture requires that we address the "technology formula" for scalability, as James Chung has referred to in a study of enabling technologies for large scale distributed simulation [36]. The formula is a list of issues that involve those discussed above as well as the host processor processing and bandwidth, local area network speeds, and wide area network switching and bandwidth. Figure 1 show the many parameters for developing a scalable architecture.

Perhaps the most difficult requirement for design is the need to examine the complex interaction between the applications in the VE, the user's behavior in that environment that influences network traffic, and the algorithms and protocols that bind the applications over the network. *We will discuss them in more depth in the chapters on related work and DIS.* However, it is the behavior of the entities and the dominant characteristics of military VEs that we wish to exploit.

C. EXPLOITING REALITY

We propose a set of heuristics that provide a network software architecture for solving the problem of scaling very large distributed simulations.

1. Goals

Our goals are to reduce the computational and bandwidth requirements of VE entity hosts and limit the growth of those requirements as the VE scales. Second, we desired to maintain the current Distributed Interactive Simulation semantics. Third, the architecture should minimize the time for new VE entrants to learn the state of the world and eliminate the use of "heartbeat" state messages. Finally, we desire an architecture that is distributed.

2. Method

Our method is analogous to the concept of locality of reference exploited by cache systems but applied to the distributed VE architecture.

The fundamental ideas behind our approach are to:

- exploit the actual characteristics of the real-world large scale environments that are simulated, e.g., the characteristics and behavior of the entities involved,
- logically partition virtual environments by associating spatial, temporal, and functionally related entity classes with network multicast groups,
- localize the communication of entities to the classes to which they belong,

- use transitions among classes to maintain weak consistency within these VEs with a persistent object protocol,
- restrict an entity's processing and network resources to its area of interest via a local Area of Interest Manager (AOIM)
- change the current communications model for DIS.

Our approach is also *domain specific,* because *we are making assumptions about the temporal and geometric coherence of a VE.* We will show that an entity in the military domain only needs to be aware of, communicate with, and compute state at most with a few hundred other entities. Though the techniques may generalize to other types of virtual environment applications (e.g. entertainment, emergency training), the primarily interest of our effort is to emulate real-world military domains. In particular, our focus is on DIS applications. The approach is also *efficient* because as the number of participants increases in the VE, aggregate bandwidth demand increases linearly with the total number of entities while tail-link bandwidth and host processor utilization increase with respect to the areas of interest of the entities represented on the individual host and subnet.

In the real world, which virtual environments emulate, entities have a limited area of interest. For example, a modern tank on a battlefield can effect and observe other entities typically out to a range of less than five km. On the other hand, a person on foot typically has an area of interest of only several hundred meters. This would be the case for a dismounted infantryman or a human simulated for a typical role-playing adventure game. The entities whose areas of interest overlap are members of a *spatial class* or group in the VE.

Entities also may belong to a *functional class* in which an entity may communicate with a subset of entities. Therefore, simulated radio traffic should be restricted only to the interested parties of the group. Other types of functional classes could be related to system management or services such as time.

Another example of a functional class in the military domain would be a VE "air control" group. The group would include entities that are primarily concerned with entities or events occurring in the air. Therefore, air defense and aircraft entities would comprise the majority of the group. Aircraft and air defense systems are relatively sparse compared to other combat systems such as tanks. Air defense systems would also belong to a small subset of the spatial class.

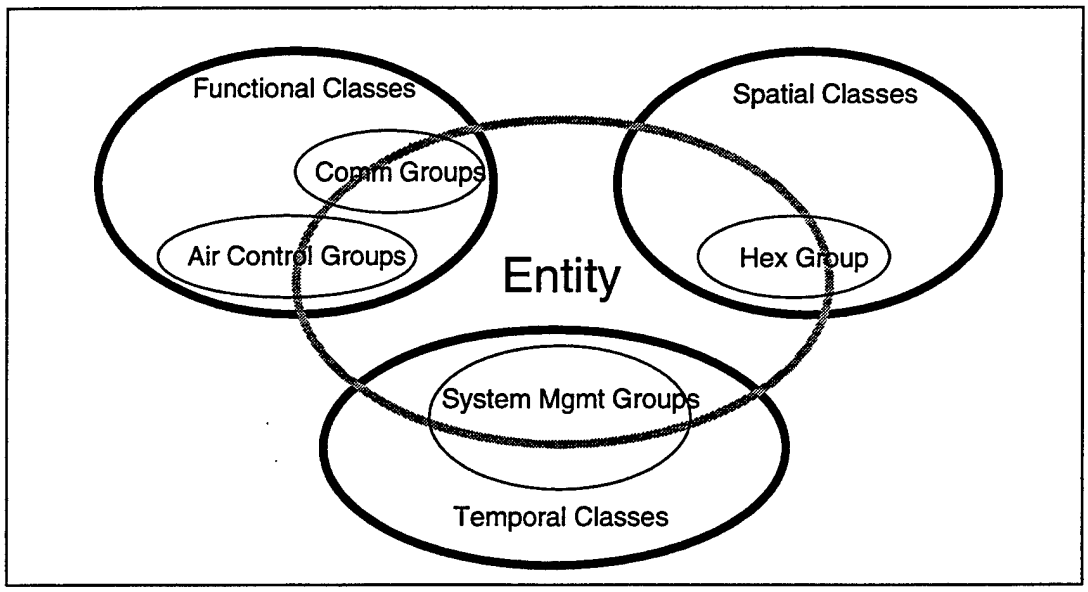


Figure 2. Relationship between entity and multicast groups.

Finally, entities can belong to a *temporal class*. For example, some entities do not require real-time updates of state changes. A system management entity might only need updates every several minutes. A simulator of a space-borne sensor only needs a general awareness of ground vehicle entities and therefore can accept low-resolution updates. When there is a need for more resolution, the simulator, like aircraft entities, can focus and become part of a spatial group.

In Figure 2 we illustrate the relationship between a ground-based air defense entity and the variety of classes described above. It is associated spatially with other entities on the ground, and it has a functional relationship with aircraft (its primary interest) and other en-

tities with which it communicates. It has some non-real-time communications requirements for system administration purposes and therefore belongs to temporal class.

D. Dissertation Overview

The next three chapters survey the related work in the area of networked virtual environments. Chapter III presents related work and the problems of developing scalable VEs. The chapter also gives a taxonomy for discussing VEs in terms of communication methods, data, processes, and views.

Chapter IV is an continuation of the discussion on related work but specifically describes the SIMNET and Distributed Interactive Simulation effort, the problems and limits of DIS today, and some current research efforts into scaling DIS to support 100,000 entities through the use of application gateways. SIMNET and DIS provide the base technology for most systems used today and their discussion permits insight into the problem of scalability.

Chapter V describes the evolution of one of the most important research VEs in common use today -- NPSNET-IV. We also relate our implementation of the DIS protocol, how NPSNET-IV exploits the use of parallelism, and past experience with the DIS protocol over WANs using IP Broadcast.

Chapters VI and VII are presentations of the network infrastructure required for large-scale VEs and our architecture: high-speed LAN and WAN architectures, multicast network protocols, and the Internet Multicast Backbone. We also discuss in Chapter VII current research into using multicast for VEs and some of the implications for this work.

The following chapters describe efforts at exploiting multicast communication for VEs. Chapter VIII discusses our research goals, heuristics and theory. The chapter defines the Area of Interest Manager (AOIM) concept and service model in the context of multicast

internetworks. It provides our approach to the DIS scalability problem. We detail the protocol used for partitioning virtual environments using spatial classes.

Chapter IX presents the results of our simulation using the AOIM and our architecture. We used data from the U.S. Army National Training Center and the Janus combat model to show how movement rates and vehicle densities allow the use of the AOIM by an entity to limit network traffic and simulation computation, maintain acceptable reliability, and minimize the effects of latency. Finally, the concluding chapter discusses the implications of this work, areas of research not addressed in this work, and future work required for building large scale VEs.

II. RELATED WORK

A. OVERVIEW

In this chapter we discuss related work within a context of what needs to be considered when building large-scale VEs. This framework is necessary to have a coherent understanding of the many components of distributed VE systems.

Currently, there are relatively few examples of academic VE systems in which to apply the framework. Research into large-scale distributed virtual worlds has been limited because of a number of practical factors besides those noted in the previous chapter. Immature network technology has relegated most distributed VEs to Ethernet local area networks (LAN). Large-scale VEs will need to use WANs to expand both their physical geographic scope and number of participating hosts. Furthermore, until recently, real-time graphics performance had been confined to specialized and very expensive computer image generators. Software development and graphics databases has also progressed slowly and the interfaces for immersing the human into the environment have been primitive at best [51].

These problems are being overcome by the rapid growth of high-speed internetworks, the availability of low-cost, off-the-shelf graphics workstations, and the development of standard graphics tools and libraries. However, few VEs have attempted to involve all these components so there is limited experience with them and published results are rare.

B. TAXONOMY

Virtual environments mimic many aspects of operating systems. For example, the now defunct system developed by the Human Interface Technology Laboratory, Virtual Environment Operating Shell (VEOS), provided much of the functionality of a distributed operating system in the way of programming language services [79,21]. However, we are primarily concerned with a single application and not a general purpose computing envi-

ronment. Therefore, the most important questions about virtual environment software architectures we address here are:

- What is distributed?
- What are the modalities of the distribution?
- Why is it distributed?

1. Communication

Several aspects of communication are largely responsible for answering the three questions above. The primary dimensions as shown for VEs are bandwidth, latency, distribution schemes, and reliability (Figure 3).

Communication Issues For VEs

- Distribution
  - Broadcast, multicast or unicast
- Latency
  - Lag, jitter
- Reliability
  - Acknowledgments, negative acknowledgments
- Bandwidth

Figure 3. Communication issues.

a. Bandwidth

We pay particular attention to the effect of bandwidth in this chapter because the available network bandwidth determines the size and richness of a virtual environment. As the number of participants increases so do the bandwidth requirements. On local area networks (LANs), this has been not a major issue because technologies such as Ethernet (10 Mbps) are relatively inexpensive and the number of users for VEs has been limited. In

contrast, for wide area networks (WANs) bandwidths have been generally limited to T1 (1.5 Mbps) but the potential user base is much larger. Practically all VEs have used Ethernet for local communication and T1 for WANs.

However, networks are now becoming fast enough to be true extensions to the computer's backplane and for the development of distributed VR applications. Distributed VR can require enormous bandwidth to support multiple users, video, audio and the exchange of 3D graphic primitives and models in real-time. Moreover, the mix of data requires new protocols and techniques to handle data appropriately over a network link. The technologies providing these gains in performance blur the traditional distinction between local area and wide area networks (LANs and WANs). There is also a convergence between networks that traditionally carried only voice and video over point-to-point links (circuit-switching) and those that handle packet-switched data.

The actual number of VEs to take advantage of these high speed networks have been small and have been associated with Grand Challenge (high performance computing) problems. The Multidimensional Applications and Gigabit Internetwork Consortium (MAGIC) network is a gigabit-per-second ATM-based network that connects Minneapolis, Sioux Falls, Lawerence, Kansas, Kansas City and Ft. Leavenworth, Kansas. MAGIC is designed to allow a commander to see three-dimensional photo-realistic computer generated images of a very large area of interest in real time, both from ground level and from the air, using data stored in a remote database. These images will be generated from elevation data (Digital Elevation Maps), aerial photographs, models of buildings, and models of vehicles whose positions will be updated in real-time via the Global Positioning System. For example, a terrain database of Germany viewed in Kansas on a workstation receives images from California which are texture mapped onto the terrain in real-time [136]. The network provides trunk speeds of 2.4 Gbps and access speeds of 622 Mbps,

allowing an application to use a supercomputer (CM-5) to process data from a database at a second location, and display the results on a workstation at a third location.

The NASA Computational Aerosciences Project is planning to use high speed networks to support visualization of large Computational Fluid Dynamics (CFD) data set by distributing processing onto several supercomputers across the United States. Gigabit networks will be required to move actual geometries generated by the supercomputers to be rendered on a remote graphics workstation [92].

The Electronic Visualization Laboratory at the University of Illinois has used a combination of Ethernet, Fiber Distributed Data Interface (FDDI) and High-Performance Parallel Interface (HPPI) networks to develop a distributed VE application. The operator navigated through the VE using a CAVE (a system that projects images on three walls or hemi-cube for simulating "walkthroughs"), which was connected to an SGI Onyx workstation used for rendering and control which in turn was connected to a CM-5 used for the actual simulation [115].

The Human Interface Technology Lab (HITL) and Fujitsu Research Institute have jointly formed the GreenSpace Project which has as a goal to develop a virtual common among 100 or more participants using SONET and ATM. However, they are currently using Basic Rate Interface (BRI) ISDN lines for demonstration [43]. Lockheed has proposed (but not implemented) the use of ATM for moving large terrain data sets among different VE systems for DIS [95].

In summary, researchers are beginning to take advantage of high speed networks which are critical for building large, realistic VEs. In later chapters we will discuss how the network infrastructure is changing to deliver more and ubiquitous bandwidth to support large VEs. However, the VEs discussed here have been primarily small with respect to the number of interactive users and have used point-to-point links.

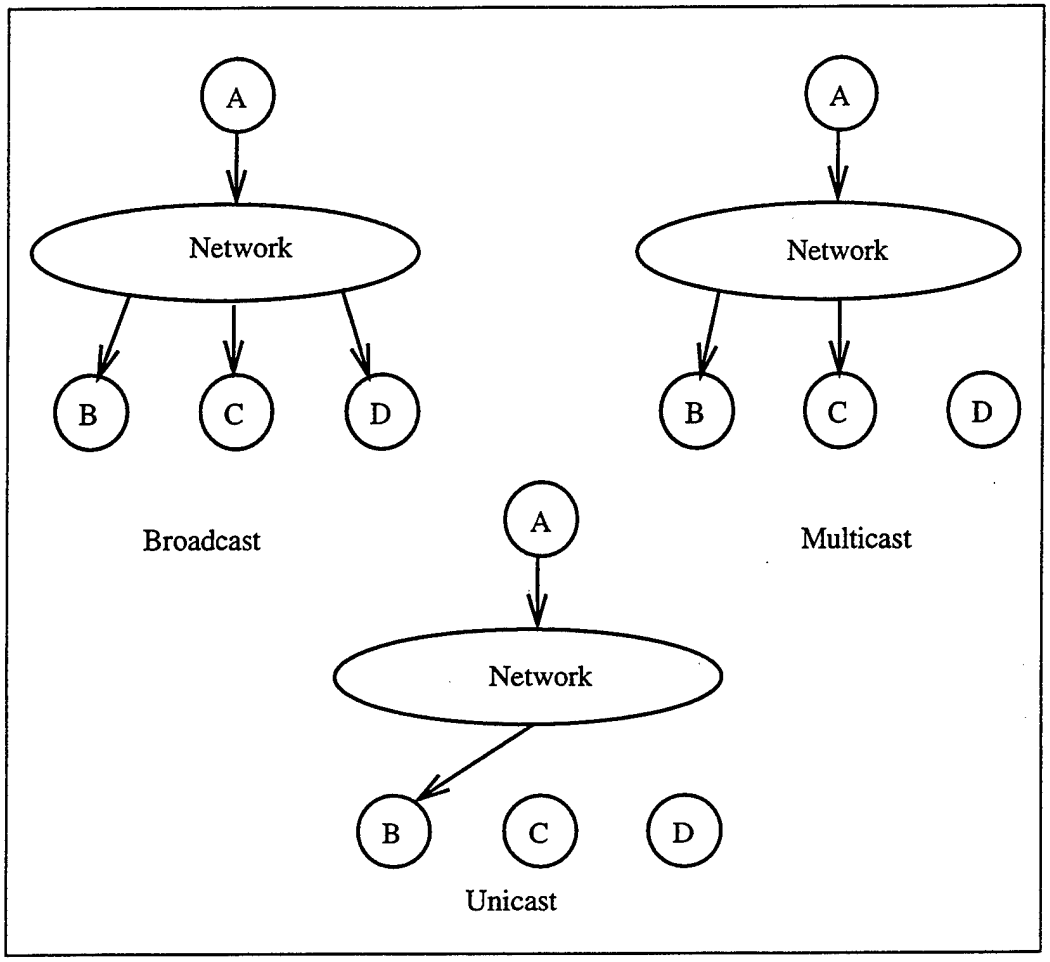


Figure 4. Examples of broadcast, multicast, and unicast.

b. Distribution

Some distribution schemes scale better than others. Three methods are shown in Figure 4. Multicast services allow arbitrarily-sized groups to communicate on a network via a single transmission by the source [106]. Multicast provides one-to-many and many-to-many delivery services for applications such as teleconferencing and distributed simulation in which there is a need to communicate with several other hosts simultaneously. For example, a multicast teleconference allows a host to send voice and video simultaneously to a set of (but not necessarily all) locations. With broadcast, data is

sent to all hosts while unicast or point-to-point establishes communication between two hosts.

Most distributed VEs have employed some form of broadcast (hardware-based or IP) or point-to-point communications. For example, the MR Toolkit Peer Package, which is used for creating distributed virtual reality applications over the Internet, uses unicast for communications among the applications though the developers have considered using IP Multicast [122]. The Mitsubishi Electric Research Laboratories (MERL) VE uses unicast to transmit state changes over an Ethernet LAN though the developers intend to use broadcast to go beyond a four user limit. A new version in development uses IP Multicast [114].

Unicast is also the general approach for Grand Challenge applications like MAGIC. Another example is the Virtual Windtunnel in which the network is a logical part of the visualization system much in a manner analogous to traditional image generators [92].

However, these schemes are bandwidth inefficient for large groups. Furthermore, broadcast, which is used in SIMNET and most DIS implementations, is not suitable for internetworks because the network becomes flooded with unwanted traffic and it is difficult to avoid routing loops. Moreover, IP broadcast requires that all hosts examine a packet even if the information is not intended for that host, incurring a major performance penalty for that host because it must interrupt operations in order to perform this task at the operating system level. (SIMNET uses the hardware multicast capability of Ethernet but only to create a single multicast group for the entire distributed simulation.) Point-to-point requires the establishment of a connection or path from each node to every other node in the network for a total of N*(N-1) virtual connections in a group (see Figure 5). For example, with a

1000 member group, each of the 1000 individual hosts would have to separately address and send 999 identical packets.

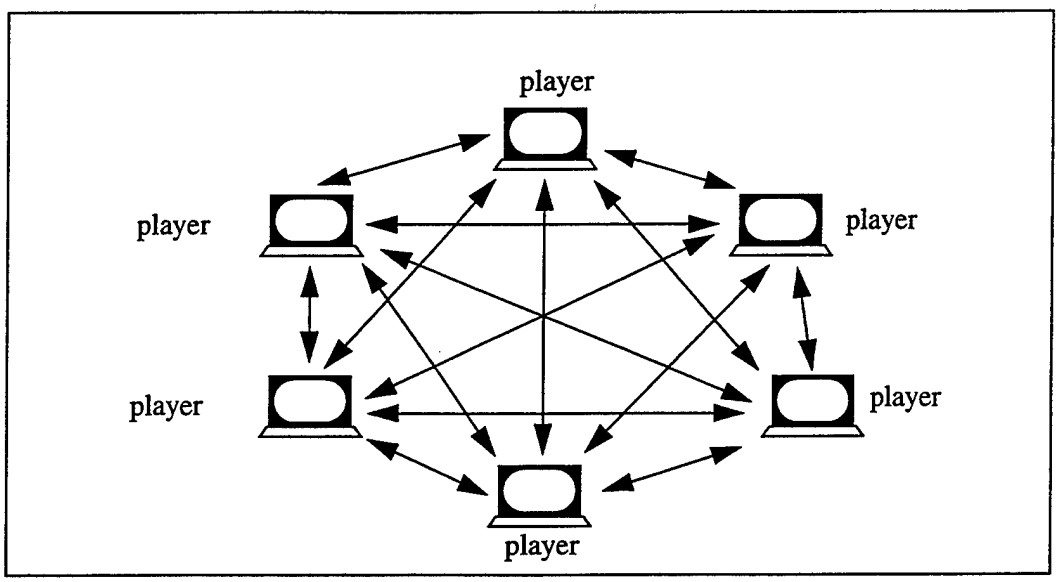


Figure 5. Distributed model.

c. *Latency*

Another dimension of communication is latency which controls the interactive and dynamic nature of the virtual environment -- how well the players mesh in behavior. If a distributed environment is to emulate the real world, it must operate in real-time in terms of human perception. A key challenge is that the appropriate systems involving human operators must deliver packets with minimal latency (less than 100 ms) and generate textured 3D graphics at 30-60 Hz to guarantee the illusion of reality [158]. On top of this is the need to provide real-time audio, video, and imagery services for the simulation of player communication services.

Latency is a problem for network cue correlation. Sawler notes that both the delay of an individual cue (e.g., seeing an object move) and the variation in the length of the delay are important, particularly in closely coupled tasks which require a high degree of correlation (e.g., flying in formation) [120]. This becomes a major challenge in systems that

use wide area networks because of delays induced by long paths, switches and routers. Network latency can be reduced to a certain extent by using dedicated links (or virtual ones using protocols like the Reservation Protocol [45]), improvements in router and switching technologies, faster interfaces and computers.

However, more bandwidth is not necessarily a complete solution. Operating at gigabit speeds presents a new set of problems. New methods of handling congestion are required because of the high ratio of propagation time to cell transmission time [61]. By the time that a computer in New York sends a message telling a host in San Francisco to stop sending data, it is too late to have stopped a gigabit worth of information from being transmitted.

The bottlenecks will most likely be in the network interfaces, memory architectures and operating systems of the computers on either end. For example, early Fore Systems ATM interfaces for the SGI Indigo could only handle 20 Mbit/s of data even though the media can deliver 140 Mbps [164]. The slow progress in increasing the interface performance of FDDI is an example of the lag in technologies we will probably see as high speed networks are fully deployed. Nor have memory speeds kept up with the leaps made in CPU and network performance.

At the operating system level, most VR applications are built on commercial versions of UNIX which are not designed for real-time performance. There is also debate over the adequacy of current transport protocols like TCP, which interface the operating system with the network, and more recent protocols like Versatile Message Transaction Protocol (VMTP) and Xpress Transfer Protocol (XTP - which was designed to be implemented in silicon) that claim to be more efficient [117].

Other methods are available for ameliorating the effects of latency. BBN developed dead-reckoning techniques to abstract data from simulators. This technique,

discussed later, not only reduces communications loads on the network but also reduces perceived delays because of predictive modeling by the local host [96]. Singhal has proposed a dead-reckoning method that exploits position history [131]. However, lag can never be totally eliminated and for environments where the VE is widely distributed (e.g. Earth to Mars). Therefore, techniques such as synthetic fixtures are used which provide force and visual clues to operators in limited domains about that environment [121].

d. Reliability

Finally, communications reliability often forces a compromise between bandwidth and latency. Reliability means that systems can logically assume that data sent is always received correctly, thus obviating the need to periodically re-send the information. Unfortunately, to guarantee delivery, the underlying network architecture must use acknowledgment and error recovery schemes that can introduce large amounts of delay - a common case on WANs and with large distributed systems. Additionally, some transport protocols such as TCP use congestion control mechanisms that are unsuitable for real-time traffic because they throttle back the packet rate if congestion is detected.

Reliable multicast protocols are currently not practical for large groups because in order to guarantee that a packet is properly received at every host in the group, an acknowledgment and retransmission scheme is required [103, p. 230]. With a large distributed simulation, reliability, e.g., as provided in TCP, would penalize real-time performance merely by having to maintain timers for each host's acknowledgment and by holding up flow when a packet is lost for retransmission. Flow control introduces delay to the network to reduce congestion. Therefore it is also not appropriate for DIS which can recover from a lost packet more gracefully than from late arrivals -- it is impossible for real-time simulations to go backward in time. For example, when a packet is lost the receiving host notifies the sender, possibly invalidating a number of packets already sent because of

propagation and network processing delay. The sender must retrieve a copy of the packet lost and retransmit it. This also affects the windowing behavior which in turn slows throughput.

This does not mean that researchers are not trying to develop both a reliable and *scalable* multicast service. The communication for The Swedish Institute of Computer Science's Distributed Interactive Virtual Environment (DIVE) is provided by the ISIS system, developed by Ken Birman, which uses reliable multicast to guarantee that the virtual environment databases are accurately and synchronously replicated [24]. (A recent version of ISIS implements a reliable transport layer on top of IP Multicast). However, a peer group with more than twenty or thirty members is about as large as can be efficiently supported by ISIS [142]. Brian Whetten and Simon Kaplan have recently developed the Reliable Multicast Protocol (RMP) which is based on a token ring protocol that sits atop IP Multicast. This method uses sequencing and negative acknowledgments (NACKs). They claim that RMP should scale to hundreds of users across the Internet [157].

The problem with this method is the potential for NACK implosions, in which a group of receivers simultaneous send NACKs, adding to congestion and consequently causing the loss of more packets which introduce more NACKs. Again, reliable systems are not likely to operate in real-time. As Partridge in [103] states, "the problem of reliable multicasting over internets has not been solved".

Netrek, a popular Internet multiplayer game that uses X Window system graphics, took the approach using different degrees of reliability to gain better real time performance [99]. (Mark Pullen of GMU has suggested a similar concept for DIS called the Selective Reliability Transport Protocol [110].) Previous versions of the game used TCP. New versions have a protocol that:

- guarantees the reliability of certain packets with TCP such as error conditions and session setup and for information that is sent infrequently (server message

of the day)

- does not guarantee reliability for frequent and noncritical data such as player state (speed, direction)
- allows switching on demand from TCP to UDP/TCP and back
- won't hang or cause abnormal termination if a UDP packet is lost

e. Summary

In summary, large-scale VEs require:

- multicast communications for efficient distribution of data over wide-area networks,
- low latency networks which can be aided by dead-reckoning,
- a mix of reliable and unreliable transport mechanisms,
- and high aggregate bandwidth.

2. Views

Views are the windows into the virtual environment from the perspective of the people or processes who use it. We define two kinds of useful views for distributed environments. The first one is the synchronous view. An example of this is in a distributed flight simulator where one machine controls the forward image, and two other hosts each process the left and right cockpit window perspectives. The images are coordinated to give the illusion that they are all part of single cockpit view. Synchronism requires both high reliability and low latency. Therefore, virtual environments that require synchronous views are for practical reasons restricted to local area networks. An example of such an environment is the RAVEN simulator developed by Southwest Research Institute for NASA synchronizes shuttle astronaut viewpoints which are rendered on different machines to improve rendering performance [32]. The CAVE uses a similar approach to synchronize each image frame projected on the screen of a hemi-cube with the added component of synchronizing the sim-

ulation run separately on a CM-5. Originally, this was done using a SCRAMNET (proprietary fiber optic, shared memory LAN). Later, this was accomplished using multiple raster managers on an SGI Reality Engine Onyx and shared memory.

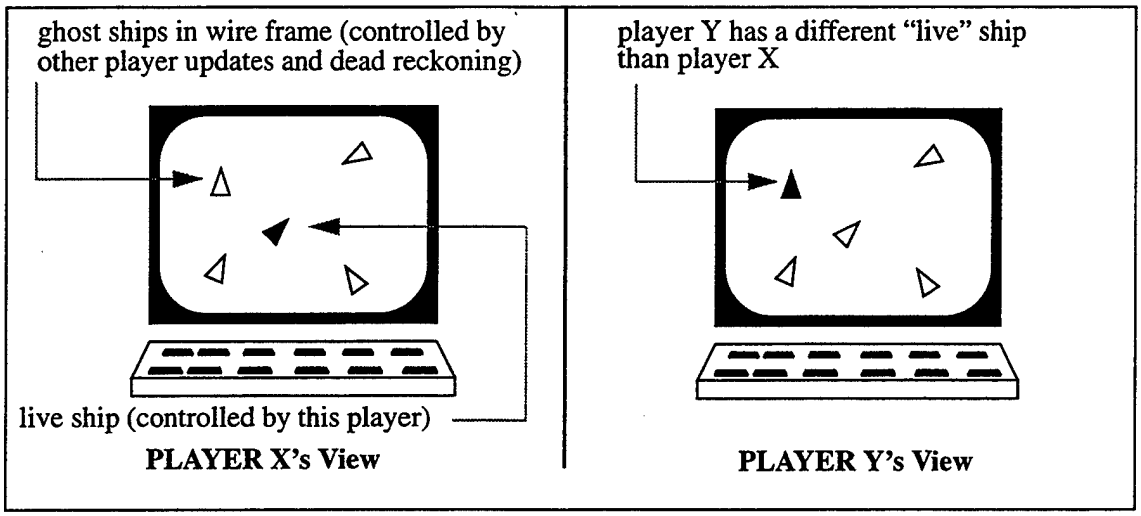


Figure 6. Two views of the simulation.

Synchronous views are also important for computer-aided design and systems used for concurrent engineering. Mark Gisi and Cristiano Sacchi developed CoCad which allows users to have shared, synchronous views of CAD designs in order to allow collaboration among geographically dispersed team members [58].

The second and most general concept is the asynchronous view. In this paradigm, multiple users have individual control over when and what they can see in the virtual environment concurrently (Figure 6). Participants can be physically separated over a local area network or a wide area network. Their awareness of each other's presence, if they are represented by an object, is brought about *inside* the virtual environment. NPSNET uses the asynchronous model where each view is typically that of the simulated entity. Views not associated with an entity are often referred to as "magic carpets" or "stealth" vehicles. Stealth entities only "listen" to the distributed world traffic because there is no need for the world to have knowledge of the viewer.

Large-scale VEs will use asynchronous views because of the cost of synchronization over wide-area networks. Synchronous views will be important for small VEs in which precise cooperative manipulation of objects is required and for applications or device communication distributed over LANs.

3. Data

Perhaps the most difficult decision of building a distributed environment is determining where to put the data relevant to the state of the virtual world and its objects. These decisions affect the scale, communication requirements, and reliability of the VE data. For example, a real-time system requiring strong consistency will be inherently difficult to scale because of the need for causality and automaticity [140]. For now, at least, large VEs only allow weak consistency among group members. We will present a concept later that modifies this requirement.

Data Models for VEs

- Replicated homogeneous world database
  SIMNET
- Shared, centralized
  VISTEL
- Shared, distributed, peer-to-peer
  DIVE
- Shared, distributed, client-server
  BrickNet

Figure 7. Data models for VEs.

There are many conceivable ways of distributing persistent or semi-persistent data (see Figure 7). We present some of the most prevalent methods in current VEs:

a. Replicated homogeneous world

A common method for large VEs is to initialize the state of every system participating in the distributed environment with a homogeneous world database containing information about the terrain, model geometry, textures, and behavior of all that is represented in the virtual environment. Communicated among all the users of the environment are object state changes such as vehicle location or events such as the detonation of a simulated missile or collisions between two objects. The advantage of this approach is that messages are relatively small. The disadvantages are that it is relatively inflexible and that as virtual environment content increases so must everyone's database. Moreover, over time, the world becomes inconsistent among the participants through the loss of state and event messages. This is the model for SIMNET, a distributed military simulation system for use on Ethernet LANs developed by BBN for ARPA. However, once a simulation begins, each host maintains its own database without making any effort at guaranteeing consistency except through the use of "heartbeat" messages and event updates [108].

b. Shared, centralized databases

On the other hand, the Virtual Space Teleconferencing System (VISTEL) uses a shared world database. As its name implies, VISTEL is a teleconferencing system that displays 3D models of each conference participant. Changes in a model's shape, reflecting changes in a person's facial expression, are sent via messages to a central server and redistributed. Only one user at a time can modify the database (see Figure 8)[100].

This is the model used by MUDs except that they typical employ relatively primitive clients. Text-based MUDs use TCP connections to a central server that does almost all the computation and maintains the state of the VW. For text communication, this typically scales to about fifty concurrent users who "move" about among rooms, create and

delete new objects or actions, and communicate with each other. LamdaMOO from Xerox Palo Alto Research Center is probably the most advanced MUD and we discuss it further in Chapter VI [42]. Using a centralized server for 3D virtual worlds is obviously limited to a few participants because of input/output (I/O) contention, and the complexity of the database. Distributed MUDs have been tried in which users could move through a portal from one MUD to another [22].

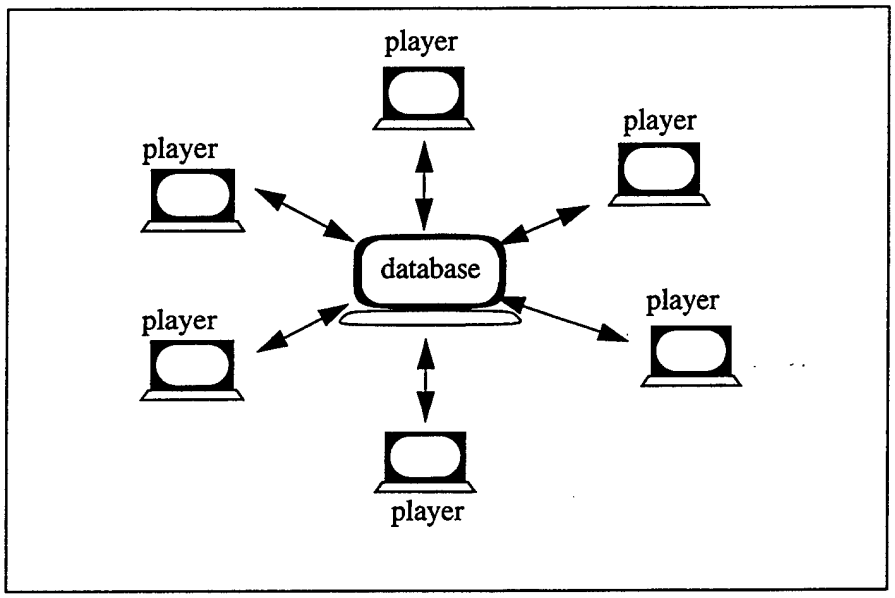


Figure 8. Centralized model.

This is demonstrated in Netrek which scales to about 18 players with UDP and uses an asymmetric communications model [99]. The data in Figure 9 from J. Mark Noworolski shows how the server becomes the bottleneck because it must retransmit all other players's state to each client. In this case communication from individual clients are only 168 bytes per second. The server, on the other hand, must take every client message and redistribute to it to all the other clients with an order of magnitude increase in bandwidth required. The concept of portals is also used by other VEs which we will discuss below.

Server -> Client network usage:
Maximum CPS during normal play: 3588 bytes per second
Standard deviation: 918
Total bytes received 1795888, average CPS: 803.0

Client -> Server network usage:
Maximum CPS out during normal play: 168 bytes per second
Standard deviation out: 21
Total bytes sent 20580, average CPS: 18.0

Figure 9. Server vs. client communication in Netrek.

c. Shared, distributed databases with peer-to-peer updates

Many distributed systems strive to simulate shared memory architectures. For example, DIVE has a homogeneous fully-replicated distributed database. However, unlike SIMNET the entire database is dynamic and uses reliable multicast protocols to actively replicate new objects. A disadvantage with this approach is that it is difficult to scale up because of the communications costs associated with maintaining reliability and consistent data across wide area networks. However, modeling terrain interactions, such as building a berm, still would be very expensive (though highly desirable) in terms of the number of polygons that would need to be created, changed, and communicated in DIVE [24]. DIVE is limited to a handful of active participants though the developers plan to make changes to improve scalability.

Virtual environments that use Linda, the parallel programming language, also trade performance for a relatively simple blackboard programming model. For example, Denis Amselem of SRI developed a VE using Linda with an unusual hand-held interface -- a portable LCD television with a space tracker for VE navigation. Performance of the multiuser system limited it to three participants [6]. The simplicity and illusion of shared

memory presented by Linda is also the reason why this system suffered from poor performance. Data must reside *somewhere.* In this case, it was on a central server.

d. Shared, distributed, client-server databases

Another technique is to use a variant of the client-server model in which the database is partitioned among clients and communication is mediated by a central server. For example, in BrickNet, as an entity moves through the virtual environment, its database is updated by an object-request broker on a server that has knowledge of which client maintains that part of the world [130]. BrickNet may be most appropriate for large CAD environments because it attempts to tackle the walkthrough problem of a virtual environment that has huge numbers of component models and provides multiple views simultaneously to a group of users. However, in a dynamic large scale world, the servers can quickly become I/O bottlenecks (as was the case with AOL), increasing the inherent latency of the virtual environment. The developers of BrickNet have suggested some possible solutions including providing multiple distributed servers.

In a similar approach to BrickNet and DIVE, the Model, Architecture and System for Spatial Interaction in Virtual Environments (MASSIVE) system uses a spatial model for data partitioning among clients. In this case, an entity declares its world to a local "aura" manager which in turn informs other aura collision managers. These managers broker between objects by detecting proximal collisions and informing each of the peer entities of mutual interface references[60].

Pure client-server systems that strictly use classic remote procedure calls (RPC) do not scale well for a number of reasons. Partridge points out that the RPC is poorly suited for high speed networks because communication is achieved by sending a message and waiting for a reply. As relative delay of gigabit networks increases, RPCs become expensive. This is also true from the standpoint of VE interaction [103,123].

e. Summary

Replicated world databases are more communication efficient than centralized or distributed shared database schemes. However, they generally lack a way for maintaining world consistency -- a problem with unreliable transport mechanisms like UDP. They also lack the ability to update the VE with new objects or behaviors. However, large VEs could use a mixed model -- client initialization with small replicated data sets and a distributed client-server model. This would allow more data consistency and persistence if a mechanism or heuristic is used to reduce transfer latency.

4. Processes

Distributing processes to multiple hosts increases the aggregate computing power associated with a simulation. We can use this not only to provide the capability to distribute views but also handle a variety of input devices. SIMNET and its descendants, such as DIS-compliant systems, also make use of the aggregate computing power by taking advantage of a technique called dead-reckoning, discussed in detail later, to reduce the need for network communication.

With the exception of the DIS model, practically all distributed environments assume that the same kind of processes are running on each host that has the same function (architectures may differ). The advantage of this approach is consistency. The disadvantage is that it is very inflexible. DIS is a protocol designed so that different developers can create different simulations on different machines that theoretically can share in the same virtual environment because they can communicate at some common level. The problem with this is that no protocol is complete and DIS is not an exception. For example, new objects cannot be introduced without a change in the standard.

The AVIARY system has homogenous processes but contains Object Servers which permits migration of lightweight objects to enable load balancing. These objects represent

the entities and the processes which control them. DIVE uses the concept of process groups from ISIS to partition the VE into rooms or spatial regions. The MR Toolkit distributes processes that support different components of the VE such as the input devices [122]. It provides an interpreted language, the Object Modeling Language (OML) that allows platform independence for developing virtual environments. OML specifies the behavior and geometry of VE entities. Similarly, BrickNet uses a language called Starship. BrickNet objects can share or transfer behaviors that are specified in Starship. These behaviors are either environmentally-dependent, reactive, or capability based.

Gavin Bell and Tony Parisi of SGI have developed the Virtual Reality Modeling Language (VRML) which "is a language for describing multi-participant interactive simulations -- virtual worlds networked via the global Internet and hyperlinked with the World Wide Web" [107]. The current version is similar to the SGI Open Inventor ASCII 3D graphics file format combined with Multipurpose Internet Mail Extensions (MIME).

Other more general-purpose scripting languages may provide the capability to migrate processes and objects across diverse platforms by using active messaging. The proprietary AT&T Telescript language is an innovative communication technology that considers the network as a platform, on top of which you can run applications that are not bound to a specific node of the network, but, to the contrary, are intended to move around the network during their execution. Telescript itself is an interpreted language that is specifically designed for communication. It provides primitives that allow the script to suspend, migrate to another node of the network, and resume execution from the same point. The key idea is procedural messaging. With Telescript, write agents are sent around the network to accomplish the tasks you want. Instead of having a client dealing with a server by means of a set of messages sent back and forth, you build an agent and send it where the server is. The agent is smart enough to interact with the server, and returns to the sender with the required

information. In this way, bandwidth consumption is reduced and we can build agents that seek information on our behalf [113].

ScriptX from Kaleida Labs X is a multimedia-oriented development environment in which classes, objects, and their relationships can be reconfigured during execution. Methods can be redefined and new objects added at run time. ScriptX code is semi-compiled into a bytecode representation, similar to that in Smalltalk, that is then interpreted by a platform-specific virtual-machine interpreter [149].

Safe-Tcl is a language for active or agent-based mail in which the data delivered through the mail constitute a program in a well- specified language, allowing the program to be automatically evaluated on behalf of the recipient when the mail is "read." The syntax of Safe-Tcl is identical to the syntax of Tcl [101]. No syntactic constructs are changed. The only difference, therefore, between Tcl and Safe-Tcl is the set of available primitive functions and procedures. Safe-Tcl may be described as an "extended subset" of Tcl, in that the "dangerous" primitives in Tcl have been removed, while certain new primitives have been added [19]. An example of a dangerous primitive is an exec call in Tcl which can start up a new process. Eliminating this from Safe-Tcl helps avoid the possibility that a script could generate unwanted behavior by the receiving host. (We used Tcl/Tk for developing and testing grid algorithms. See Appendix C.)

In a distributed VE, clients can be homogeneous as is the case for DIVE. Clients can also be dissimilar except for the communications protocols among them, providing interoperability (e.g., DIS and SIMNET systems, which exchange standard state and event data). Furthermore, processes can be designed to migrate across homogeneous architectures like AVIARY. New scripting languages like Safe-Tcl offer the opportunity for migrating processes across heterogeneous systems, therefore permitting efficient exchange of object behaviors as well as entity state within large, heterogenous VEs.

C. SUMMARY

We have discussed in this chapter some of the VEs related to this work in the context of how communications, views, data, and processes are distributed. We have not exhausted all the considerations for developing VEs but have emphasized those aspects critical to scaling environments. Most of the systems described here scale to accommodate a handful of users. We also know that systems that demand strong data consistency, causality, and reliable communications at the same time need to support real-time interaction are not likely to scale very well. Furthermore, if the system is to be geographically dispersed, then high-speed, multicast communication is required.

In the next chapter we examine the most successful distributed VEs, SIMNET and those that use the DIS protocol. Though DIS and SIMNET have been useful for groups into the hundreds they too have found their limits. We identify some of those problems and why they exist.

III. SIMNET and DIS

The Consortium for Slow Commotion Research (CSCR) is pleased to respond to your research program announcement (RFC 1216) on Ultra Low-Speed Networking (ULSNET). CSCR proposes to carry out a major research and development program on low-speed, low-efficiency networks over a period of several eons. Several designs are suggested below for your consideration.
(d) It is proposed to use SIMNET to simulate this system [35].

A. OVERVIEW

The Simulator Networking (SIMNET) and Distributed Interactive Simulation (DIS) protocols require special consideration because they represent the largest and oldest overall effort to develop distributed 3D interactive virtual worlds. We describe in this chapter the origins of SIMNET, its relationship with DIS, the DIS protocol, and the problems and limitations of both. The chapter concludes with a discussion on the use of Application Gateways as a method for overcoming these limitations.

B. SIMNET

SIMNET is a distributed military training system originally developed for ARPA and the US Army by Bolt Beranek and Newman (BBN), Perceptronics, and Delta Graphics. SIMNET evolved out of the requirement to provide realistic combat training to armored vehicle and aircraft crews. Instead of emphasizing operator skills such as driving a tank, SIMNET provides an environment for small units -- crews, platoons, and companies -- to learn how to organize and fight as a team [68].

Collective training is important in the military but very expensive. However, SIMNET has demonstrated that collective training in network simulators is effective while being less costly than field exercises. For example, in 1987, a U.S. Army tank platoon trained together using SIMNET and won the prestigious Canadian Army Trophy competition. This was a significant milestone for the U.S. Army and simulation. The victory was a first for a U.S. team in international competition and persuaded the U.S. Army to proceed with develop-

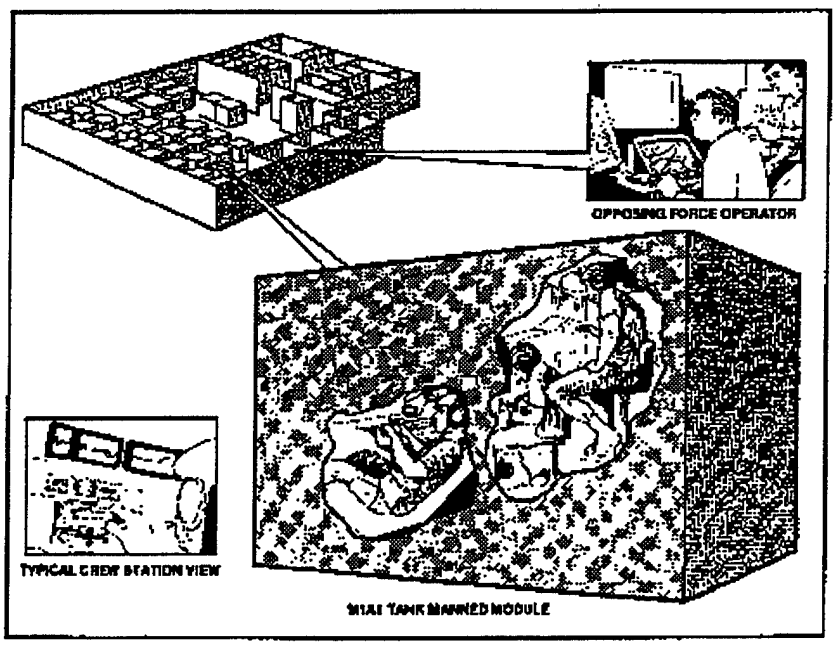


Figure 10. Close-Combat Tactical Trainer (CCTT).

ment of more advanced distributed simulation systems. The US Army has over 200 SIMNET simulators which will be replaced in the near-future by the Close-Combat Tactical Trainer (CCTT) being developed by Loral and depicted in Figure 10. CCTT will use the DIS protocol [148].

The major components of SIMNET, as shown in Figure 11,are the vehicle mock-up with individual crew stations, a low-resolution (by 1995 standards) Computer Image Generator (CIG), a static scene database (e.g., terrain), a host computer for display-list generation and processing the simulation application, and an Ethernet network [109]. These elements have been adopted by a number of VEs and continue to be standard architecture in the DIS world. Commercial VR-arcade systems, like Battletech, have adopted the LAN networked architecture [80].

The principles of SIMNET are:

- No central computer is used for event scheduling.
- Each simulation host is autonomous, maintaining its own state and world database.

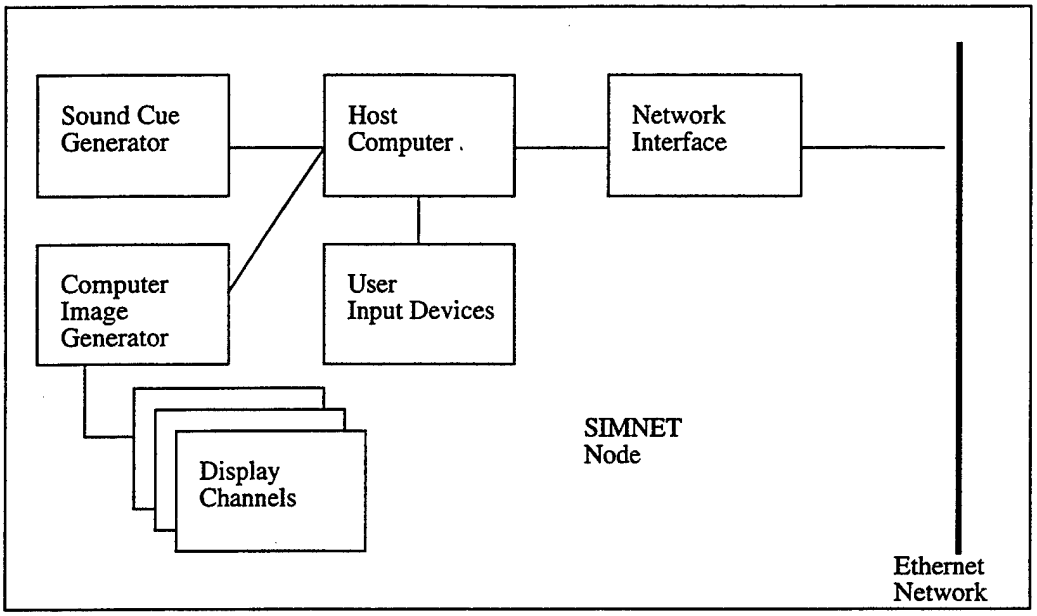


Figure 11. SIMNET simulator.

- As simulation expands each new simulator brings its own resources.
- Simulators communicate only changes in state.
- Dead reckoning is used to reduce communications processing [26].

The rationale for these principles was to provide scalability with regards to computation, I/O, and network bandwidth. Furthermore, it enhanced reliability over the entire simulation by avoiding a single point of failure. They also aided the goal to use commercially available hardware for the component subsystems [68].

The ARPA SIMNET program also develop automated simulations to enrich the VE with virtual enemies and provide common battlefield functions like logistics and artillery. The simulations, known as Semi-Automated Forces (SAF), can generate less than two hundred entities with simple behaviors. They are controlled by a person at workstation who makes tactical decision on SAF placement, movement, and missions.

The SIMNET architecture was pioneering work though it was partly inspired by the Atari arcade game Battlezone [62, p. 93]. While simple VEs (e.g., MUDs and network

games) have existed since at least the 1970's, these early environments have not had the same ambitious goals as SIMNET. For example, SIMNET took the approach in 1984 of having a fully distributed environment when this was considered both a major risk and expensive [68]. The developers pursued this path even though networking was in its infancy at the time. However, the design of the system's protocols did much to simplify the requirements.

SIMNET has three classes of protocols: a simulation protocol to convey information between entities, a data collection protocol for simulation management, and an association protocol to provide transport and session level services over Ethernet. The simulation protocol consists of a number of PDUs which convey state or event information. For example, the Vehicle Appearance PDU convey a vehicle's location, orientation, and status. Table 1 shows the structure of this PDU. Fire, Indirect Fire, Collision and Impact PDUs identify common battle field events.

SIMNET uses several clever network techniques. The first is dead-reckoning which is discussed later in the section on DIS. This substantially reduces bandwidth requirements of the network at the cost of extra computation by every host. Another is the use of a flag in the Vehicle Appearance PDU to indicate whether the vehicle is stationary. This allows a host to stop performing dead-reckoning on an entity, therefore saving processing cycles. (Regrettably, the DIS standard did not directly retain this. It is indicated by the dead-reckoning parameters.) Finally, the association protocol exploits the multicast capabilities of Ethernet by mapping exercise groups to Ethernet multicast addressees. Several different exercise groups could coexist on the same LAN without having the application itself demultiplex the stream -- the Ethernet interface could distinguish between Ethernet frames containing the SIMNET PDUs that its client host needed.

Field Size (Bytes)	Vehicle Appearance PDU Fields	
6	Vehicle ID	Site Host Vehicle
1	Vehicle Class	Tank, Simple, Static, Irrelevant
1	Force ID	
8	Guises	Object Type - Distinguished Other
24	World Coords	Location - x,y,z
36	Rotation Matrix	
4	Appearance	
12	Markings	Text field
4	Timestamp	
32	Capabilities	
2	Engine Speed	
2		Stationary bit and padding
24	Vehicle Appearance Variant	Velocity vector Turret azimuth Gun Elevation

Table 1: Vehicle Appearance PDU.

SIMNET has several problems that are associated with its success as an engineering demonstration. First, as Dale Henderson noted in 1990, there exists little research or documentation of the network performance of SIMNET, though some studies gave some insight into the characterization of SIMNET traffic [26, p. A-13]. The use of Ethernet multicast also identifies a weakness in SIMNET -- it is tied to Ethernet technology. SIMNET is more than an application protocol, it also incorporates different layers of the OSI stack which make it difficult to internetwork using different LAN topologies such as FDDI [26].

Efforts to make SIMNET work over WANs have used bridged Ethernet LANs over the Defense Simulations Internet (DSI) or dedicated T1 links. PDUs are often aggregated or bundled in large packets to avoid the greedy terminal problem of Carrier Sense Multiple

Access with Collision Detection (CSMA/CD) LANs. With the greedy terminal problem, a particular host interface will constantly offer a new frame to the net. Because of the contention mediation algorithms used by Ethernet, this interface receives priority over others for the network. Most SIMNET traffic for a particular subnet on the bridged network is generated by the combination of other traffic from the other subnets. Therefore, the bridge interface "hogs" the net. Bundling eases this problem by reducing the burstiness of simulation traffic. However, this is at the cost of network latency [55].

Finally, the protocol was the product of BBN and had been defined primarily for Army training needs. SIMNET was a de facto standard but incomplete. If a different contractor developed its own Appearance PDUs to satisfy a Navy requirement for ships then the "standard" could diverge. Therefore, there was a need to have a standard that could more easily accommodate new entity types while providing interoperablity among different simulation applications at the same time. This concerns led to the development of the Distributed Interactive Simulations protocol.

C. DIS PROTOCOL

Large scale virtual environments will likely be composed of many different and unique hardware and software platforms developed independently. The DIS protocol attempts to provide a basis for which to tie those systems together. DIS is a group of standards being developed by the Department of Defense and industry that address communications architecture, format and content of data, entity information and interaction, simulation management, performance measures, radio communications, emissions, field instrumentation, security, database formats, fidelity, exercise control and feedback. A second purpose is to provide specifications to be used by government agencies and engineers that build simulation systems. The effort is also meant to define the terminology of DIS [69].

The DIS standard has adopted many aspects of the SIMNET protocol, including its general principles, terminology, and PDU formats. (Note the similarity between the Vehicle Appearance and Entity State PDUs in Table 1 and Table 2.) Perhaps the largest change was that DIS does not retain the association protocol.

1. Protocol Data Units

Simulation state and event information is conveyed, in a similar manner to SIMNET, by twenty-seven PDUs defined by the IEEE 1278 DIS standard, but only four of these are for entity interaction. The remainder of the PDUs are for transmitting information on supporting actions, electronic emanations, and for simulation control [71]. The Entity State PDU (ESPDU) is used to communicate information about a vehicle's current state, including position, orientation, velocity, and appearance (see Table 2). The Fire PDU contains data on any weapons or ordnance that are fired or dropped. The Detonation PDU is sent when a munition detonates or an entity crashes. The actual structure of a PDU is very regimented and is explained in full detail in [71].

2. Simulation Philosophy

The networking technique used in NPSNET-IV, evolved from SIMNET, and embodied in DIS follows the *players and ghosts* paradigm presented in [17]. In this paradigm, each object is controlled on its own host workstation by a software object called a Player. On every other workstation in the network, a version of the Player is dynamically modeled as an object called a Ghost.

The Ghost objects on each workstation update their own position on every iteration through the simulation loop, using a dead-reckoning algorithm. The Player tracks both its actual position and the predicted position calculated with dead-reckoning. An updated Entity State PDU is sent out on the network when the two postures differ by a predetermined

Field Size (Bytes)	Entity State PDU Fields	
12	PDU Header	Protocol Version. Exercise ID. PDU Type. Padding. Time Stamp. Length in bytes.
6	Entity ID	Site. Application. Entity.
1	Force ID	
1	Number of Articulation Parameters	
8	Entity Type	Entity Kind. Domain. Country. Category. Subcategory. Specific. Extra.
8	Alternative Entity Type	Same type of information as above.
12	Linear Velocity	X,Y, and Z (32 bit components).
24	Location	X,Y, and Z (64 bit components).
12	Orientation	Psi, Theta, Phi (32 bit components)
4	Appearance	
40	Dead Reckoning Parameters	Algorithm. Other Parameters. Entity Linear Acceleration. Entity Angular Velocity.
12	Entity Markings	
4	Capabilities	32 Boolean Fields.
N * 16	Articulation Parameters	Change. ID. Parameter Type. Parameter Value.

Table 2: Entity State PDU.

error threshold, or when a fixed amount or time has passed since the last update (nominally five seconds). When the updated posture (location and orientation) and velocity vectors are received by the Ghost object, the Ghost's is corrected to the updated values and resumes dead-reckoning from this new posture. Figures 12 to 17 demonstrate how dead-reckoning works [109] [59].

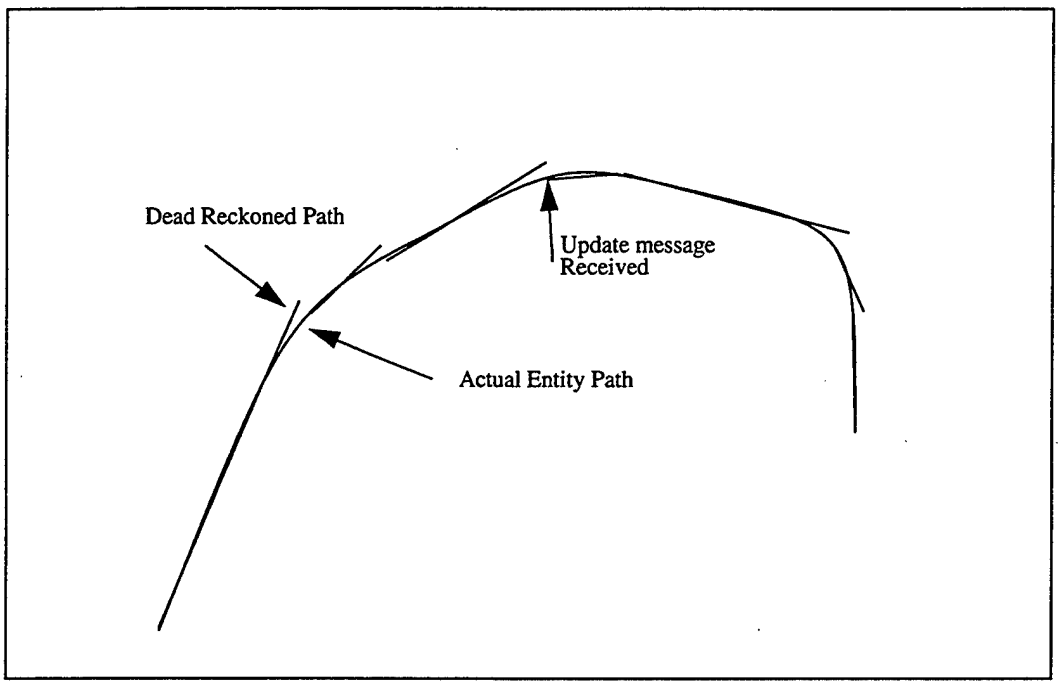


Figure 12. Dead reckoning.

The primary purpose of the Ghost concept is to reduce network traffic by minimizing updates at the cost of extra computation by the host system. Analysis of this method has confirmed that it works well with a wide variety of simulated vehicles with different performance characteristics, including high-speed aircraft. Harvey and Shaffer showed that even with a three meter position error threshold, first order dead-reckoning algorithms were sufficient to model an F16 with a 66% decrease in network traffic -- from fifty Hz to seventeen Hz for Entity PDU updates [64]. Alternatively, an updated PDU could be sent after every frame. However, even at a 10 Hz frame rate this is wasteful - 11520 bits per second

(bps) for one entity - when considering that in a large simulation many entities are stationary.

Figure 17 shows how dead-reckoning works when modeling munitions. In this case, when five bombs are dropped in NPSNET-IV, five Fire PDUs are generated followed by the instantiating of five Entity State PDUs representing each bomb. Initially, the bombs are horizontal and they rotate to point downward. As the bombs change orientation, the dead-reckoning thresholds are exceeded. Less time is required for updates as the bombs maintain their orientation. They hit the ground and a Detonate PDU is issued for each one and they stop sending Entity PDUs.

Figure 18 demonstrates the advantages of the Ghost technique. It traces the number of PDUs per second generated by nine simulated aircraft in a highly dynamic situation (everyone was trying to shoot each other down). Even the spike of approximately seventy six PDUs per second is half the rate expected of a system not using dead-reckoning (nine entities x fifteen Hz = 135 PDUs/sec if updates are issued on every frame).

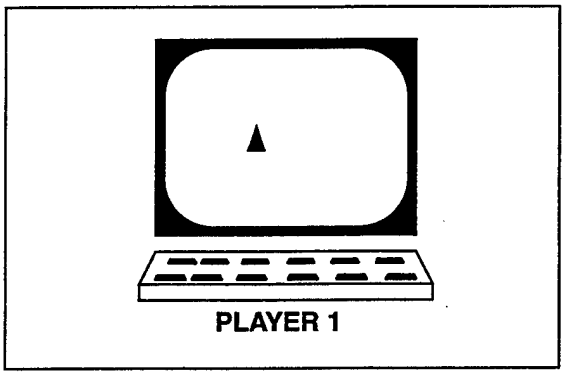


Figure 13. First player in.

Typical bandwidth used by one entity in this simulation is approximately 3200 bps. The graph also shows another characteristic of DIS traffic induced by dead-reckoning -- it is very bursty and not easily modeled because it depends on the behavior of the participants, the general scenario, the type of systems simulated, the number of players involved, and the duration of the exercise.

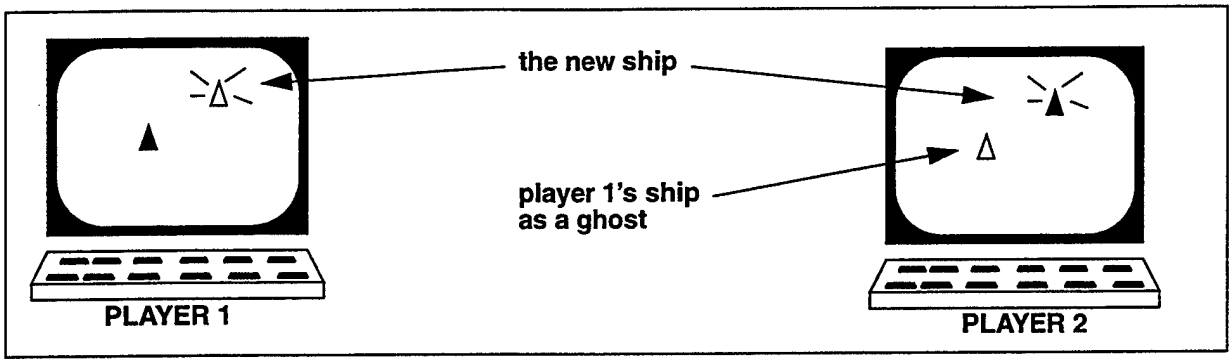


Figure 14. Next player(s) in.

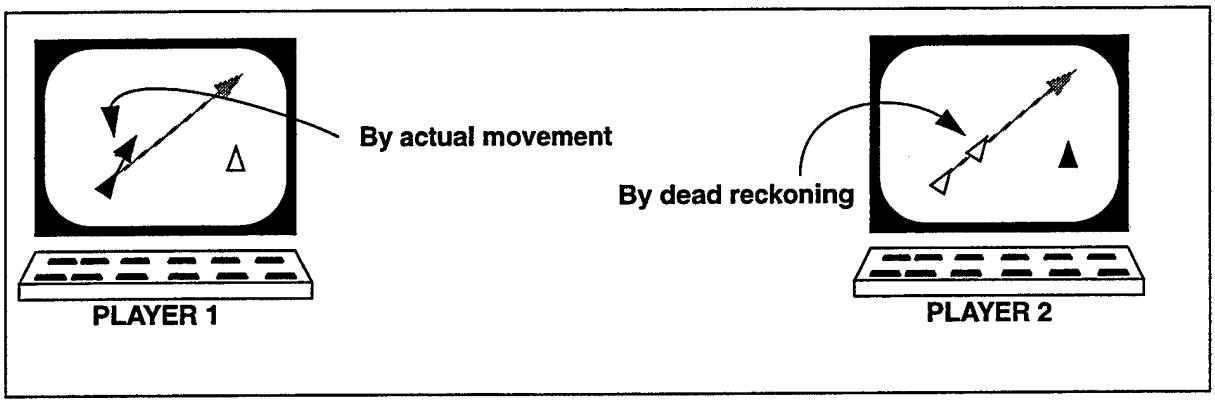


Figure 15. Ship position on different screens.

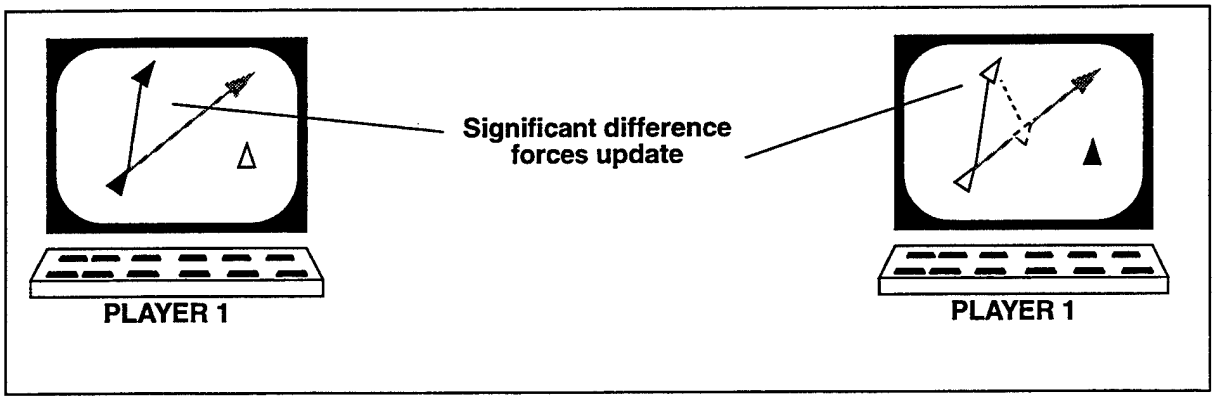


Figure 16. Ship position after update.

DIS also inherited two other concepts from SIMNET that influence bandwidth and reliability. First, with the exception of resupply operations, there are no transactions among entities about their state. Every entity is presumed honest. For example, when a vehicle

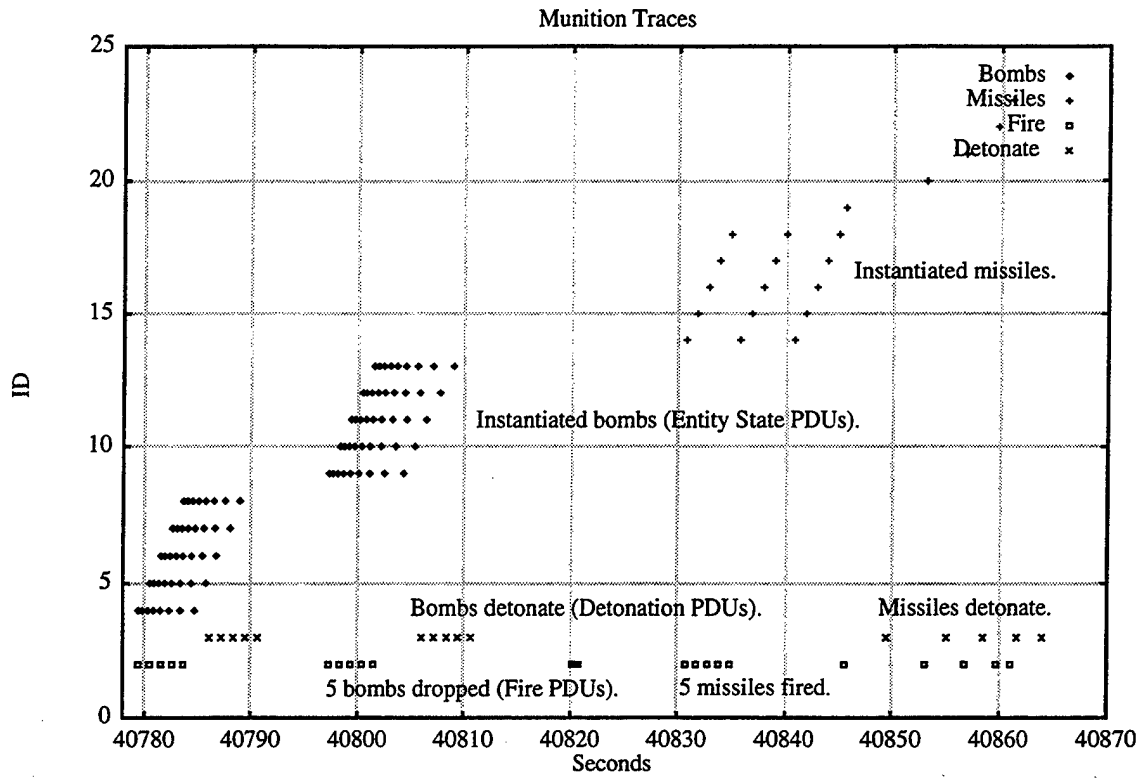


Figure 17. Time traces of PDU events.

fires a weapon, it is the determiner of who was hit and communicates that to all other entities. The targeted entities inform everyone including the shooter what damage occurred. In this manner, only one entity must make a ballistic computation and only the targets must compute the effects; this minimizes overall simulation computation and avoids the need to consume bandwidth and simulation time by negotiating 'who shot who'.

Secondly, there is no central database or servers, therefore an entity must 'learn' about the world by state updates from other entities. Entities that are not doing anything (e.g., parked vehicles) still must periodically send an update packet, as mentioned above, even though their state does not change.

3. DIS Performance Values

NPSNET-IV is capable of generating one ESPDU at a peak rate of one per frame per entity. For example, at thirty frames per second it can generate thirty packets per second.

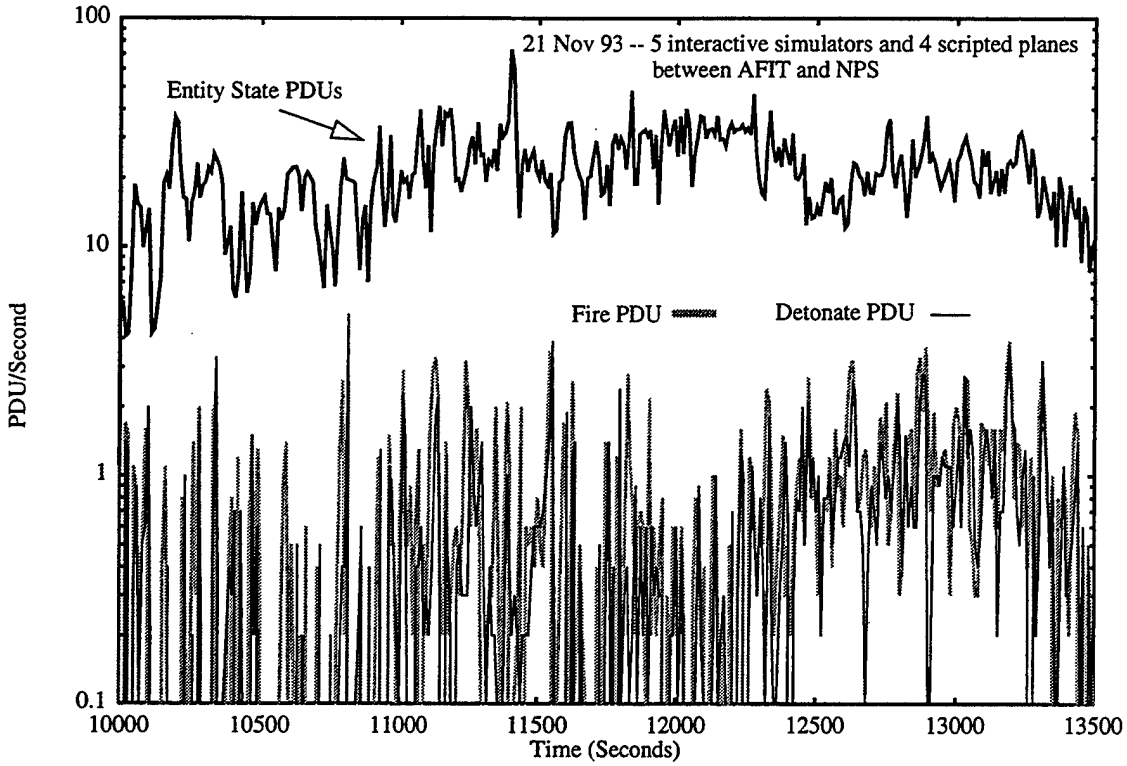


Figure 18. PDU rates (10 second sample rate).

Weapons are restricted to one firing per second. Upon a firing, the simulation generates two additional packets: one Fire PDU to establish the firing and one Entity State PDU to model the munition (depending on the munition type) (see Figure 17). Upon munition impact, one Detonation PDU is transmitted. During past experimentation, we have observed a peak rate of twelve Entity State PDUs per second for the aircraft. The packet rate was constrained by dead reckoning thresholds. In addition to the twelve packets per second, the aircraft simulator generated at most three packets for a weapon firing sequence totaling fifteen packets per second. The peak packet transmission rate for this type of simulator is fifteen packets per second [160].

As can be seen in Figure 18, Entity State PDUs dominate DIS network traffic. Figure 19 shows how these rates, along with bandwidth, affect the number of simultaneous users that can participate in the distributed virtual environment using NPSNET-IV. The band-

widths are representative of current communications technology -- 28 Kbps for high speed modems, 1.5 Mbps for T-1 lines, and 10 Mbps for Ethernet (though Ethernet becomes thoroughly saturated at lower data rates depending on the number of active senders. For example, with 100 active senders with a message size of 2000 bits, the maximum effective throughput is 4 Mbps [134, p. 360]). Note that this is a simple model of performance. It assumes (falsely) that traffic would be perfectly distributed and that hosts and routers could actually process the PDUs without delay at the higher rates. We discuss these issues and provide some experimental data later. However, the graph is provided to give an approximate feel for the entity-bandwidth-PDU relationship. With NPSNET-IV, we approach a maximum of 300 players on an Ethernet LAN without modification to the DIS protocol.

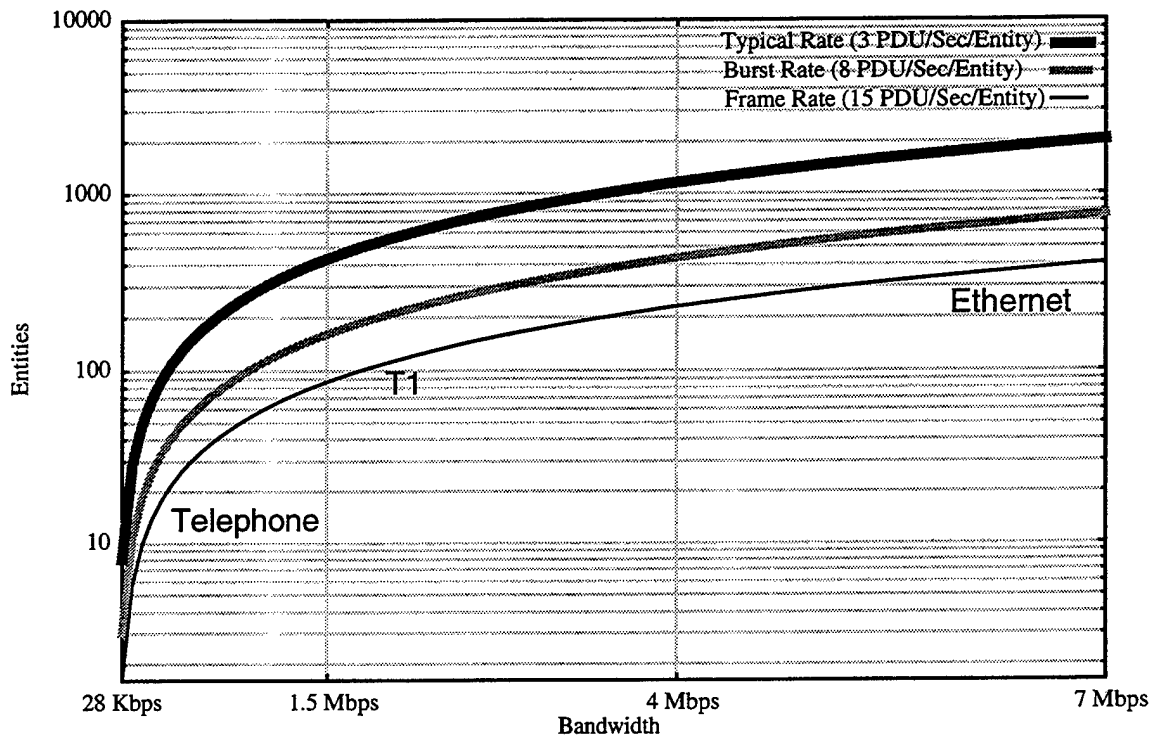


Figure 19. Bandwidth vs. number of players.

4. Problems with the DIS Protocol

Problems with DIS

- Scalability.

Bandwidth, latency, and **computational** requirements.

- Security.

End-to-end encryption.

- Handling stationary objects.
- Model and database replication.

Terrain, behaviors.

- Fidelity.

V&V, semantics, complexity, and negative training.

Figure 20. Problems with DIS.

Unfortunately, SIMNET, which was developed for small unit training, and its descendant, DIS, are currently not suitable for large scale multiplayer VEs. We list several major problems (see Figure 19) associated with scaling the current suite of DIS protocols in order to illustrate the difficulties of building large scale VEs:

a. Enormous bandwidth and computational requirements

In schemes such as SIMNET and DIS, a simulation with 100,000 players could require, under worse case assumptions according to a study done by Loral, up to 375s Mbit per second (Mbps) of network bandwidth to each computer participating in the simulation, an unrealistic requirement for an affordable system in this decade [84]. Maintaining the state of all other entities, particularly with dead-reckoning algorithms (which use second-order kinematics equations), will be a major bottleneck for large-scale simulation. Recent

experiences with the U.S. Army's Simulated Theater of War (STOW) have shown this to be the case [133]. This was also the experience reported by the earlier WAREX exercises [36].

b. Multiplexing of different media at the application layer

The current DIS protocol requires the application to multiplex and demultiplex different types of real-time data (e.g., simulation packets, audio, and video) at the application layer rather than at the network or transport layers. Therefore, the virtual environment must treat continuous video streams identically to bursty simulation traffic, i.e., allocation of buffers and timing at the application layer [53]. The consequence is that it is difficult to build DIS applications that can adequately accommodate all types of data. For example, audio data must be read at synchronized periods that do not necessarily coincide with the simulation loop and involve longer messages that delay processing of entity state or event data. The DIS community would probably be better off adopting well-tested applications or protocols developed for the Internet such as Van Jacobson's and Steven McCanne's Visual Audio Tool (vat) or the Real Time Protocol (RTP) [28].

c. Lack of an efficient method of handling static objects

Large numbers of static entities such as bridges and buildings may change with respect to an event (e.g., an explosion). These and other stationary objects must send update messages at regular intervals to inform the participants of their current state. For example, a tank that has been destroyed must constantly inform the world that it is dead in order to inform new entrants or other entities that may have missed the original state change message.

Moreover, a major influence on the PDU rate is that late arriving players have no idea of what has transpired so the only way they can learn is to listen for a while and

accumulate knowledge from the Entity PDUs being issued. This is called the *new entrant learning problem.* Therefore, the DIS protocol requires that Entity PDUs be sent after a predetermined time even though their state has not changed. How much time? If the time is too long then new entrants will operate for that period without complete knowledge of the world and realism is lost (e.g., being shot by vehicles they did not know existed but should have known). Update too often and a major burden is added to the network in the form of additional PDUs per second. Moreover, it is presumed that in large simulations many of the entities will be stationary (e.g., missile batteries) or static. Unfortunately, the answer to the PDU update rate question is not known at this time.

d. Models and world databases must be replicated at each simulator

No mechanism in DIS exists to distribute objects on demand. For large-scale simulation, this is a necessity, particularly when the simulators are heterogenous, controlled by different organizations, and little effective coordination is expected prior to an exercise. Furthermore, it is not feasible nor efficient for each simulator to store every model and database for a 100,000 entity simulation. For example, a human simulation (e.g., a dismounted infantryman) on land normally does not need to concern itself with satellites.

e. Abstractions

DIS cannot communicate all the abstractions required to present a consistent and complete view of reality. For example, there is an unresolved issue of communicating dynamic changes to the environment. Clouds or static entities like or the ground may change with respect to an event (e.g., a bulldozer carving a ditch). Representing realistic views of the weather consistently and efficiently on a large number of heterogenous systems is a major and possibly intractable challenge. Other phenomena, like electromagnetic radiation (e.g. radar) are problematic because it is difficult to communicate efficiently a sender's state in a continuous medium.

Another insidious problem with fidelity is caused by the fact that DIS assumes every simulator to be truthful. An unintended side effect, in a large-scale heterogenous simulation, is that the quality of realism expressed by simulators may vary by wide margins, reducing the overall realism of the simulation. For example, a highly realistic and accurately portrayed F15 simulator would not compete favorably against a simulator that is only functionally realistic but without the difficult flight dynamics of a high performance aircraft [63]. Morrison also notes that simulators that use munitions must have a model for the effects of each munition on each entity type where each interaction is very complex [97].

There is also no requirement in DIS for any model to follow any physical laws though some attempt is being made with the Newtonian protocol [97]. As we stated in our discussion of complexity earlier, this is of limited benefit because of the computational burden of Finite Element Analysis and collision detection.

More fundamental is the problem that DIS is good for all uses of modeling--training. testing, and analysis. As Wayne Hughes says in his critique of DIS:

> Now you have a recipe for confusion rather than clarification of results. I concede the promise of testing in otherwise inaccessible computer jungles, virtual mountains and synthetic snowstorms, but I wonder whether a pound of common sense larded with some military experience isn't as useful as an objective simulation that misses the fog, confusion, exhaustion and fright of real war [67].

f. Security

Military simulations may need to operate at various classified levels. However, security for large-scale DIS is problematic when using broadcast communications. This is because of the differing security requirements and access authorizations of participants who need to share some information with all the participants and all data with some. Moreover, the current encryption technology for end-to-end communication does not

support high speed networks (greater than T-1 speeds) (The ARPA program manager in charge of STOW has stated that encryption technology is at risk [139, p. C-51].)

g. Summary

As show above, the current DIS architecture has substantial limitations both on scalability and realism that are not necessarily solved by faster computers. In the next section we discuss the origins of the problems to better understand how we can overcome these limitations.

5. Reasons for Problems

a. Event and State message paradigm

A basic requirement for DIS has been that the simulation of the VE must be, as a whole, stateless - data is fully distributed among the participating hosts and entities are semi-persistent. Therefore, every entity must be made aware of every event (e.g., a missile detonation communicated by a Detonation Protocol Data Unit or DPDU) just on the chance it may need to know it. According to the protocol, an entity must, on a regular basis, communicate all of its state information (an Entity State Protocol Data Unit or ESPDU) to every member of the group - even though the data contained in the ESPDU is often redundant and unnecessary (e.g., aircraft markings). More importantly, these "keep alive" messages can comprise 70% of the traffic for large-scale simulations [150]. One third to a half of the updates come from dead entities according to Van Hook [20].

This paradigm as applied in DIS does not take into consideration that different simulated systems have different real-world sensing capabilities that translate into each entity's VE data requirements. In a large VE, it is unlikely that two entities representing ground vehicles separated by 200 km need to be aware of each other. Yet, under the current architecture they must inform each other of state changes and updates.

The rationale for this is to avoid the reliability problems of a central server, to simplify communication protocols, and minimize latency while guaranteeing that hosts entering a simulation would eventually build their entity database through entity state and event messages. Furthermore, the use of broadcast ESPDU updates is part of the effort to maintain consistent views among the simulators within a particular tolerance.

b. Real-time system trade-off's

Reliability (a guarantee that data sent is received) normally is compromised for real-time performance in large distributed groups. This is because in order to be truly reliable the system requires the use of acknowledgment schemes such as the one used in Transport Control Protocol (TCP) which defeats the notion of real-time, particularly if a player host must establish a virtual connection with every other entity host to ensure that each received data correctly. Therefore, large-scale environments must rely on connectionless (and therefore unreliable) network protocols such as the User Datagram Protocol (UDP) for wide-area communications.

The corollary is that a real-time environment should avoid transactions between entities since this requires reliable communications. Furthermore, schemes that use a central database do not work well in a large VE due to I/O contention. For example, AT&T's Imagination network limits the number of concurrent players in a game to four because they are centrally served and bandwidth is limited to the speed of modems (less than 28 Kbps) [51].

c. No "middleware" layer

There does not exist a DIS protocol component that mediates between distributed VE applications and the network. The current DIS paradigm implies the use of a bridged network because every message is broadcast to every entity. However, internetworking

(routing over the network layer) is necessary for large-scale simulations because it provides the capability to use commercial services as opposed to private networks to bring together diverse, geographically dispersed sites; use different local network topologies and technologies (e.g., Ethernet and FDDI); and take advantage of "rich" topologies for partitioning bandwidth, providing robustness and optimization of routes for minimizing latency. Confining DIS to the data link layer requires the use of bridges which are an order of magnitude slower to reconfigure after a topological change than routers while the number of stations is limited to the tens of thousands. A network with routers is limited to the numbers accommodated by the address space [106].

d. Origins as small unit training systems

Many of these problems devolve from the fact that until recently DIS and SIMNET were used exclusively for small scale training simulations. In this mode it has been relatively easy to insure that the VE components have homogenous sets of models and terrain databases by replicating them at each host. The lack of middleware stems from the monolithic nature of these small scale environments which could be distributed using a single LAN. Hence, *broadcast* communication was sufficient for these limited environments.

These origins have also influenced the current assumptions about the density and rates of activity of entities in large-scale simulations that do not necessarily match the real world. Players in SIMNET participated for short periods (several hours) and were highly active because the purpose of the simulation was to train crews in coordinated drills. Furthermore, the density of entities with respect to the simulated area of play was high because that best represented a small unit engaged in close combat and because of the difficulty in using large terrain data bases.

The SIMNET experience contrasts sharply with real large scale exercises. Figure 21 depicts the location of some of the 784 entities from an actual combat training scenario at the U.S. Army National Training Center (NTC), Ft. Irwin, CA and replayed in the Janus Combat Model. The area is 60 x 50 km. Friendly vehicles are blue; enemy vehicles are red. Other symbols represent obstacles and artillery registration points. In this case a US Army brigade-level task force, composed of armor, mechanized and light infantry are attacking two Soviet-style mechanized battalions. Our analysis from this exercise showed that in the ten hours of total maneuver, one third of the vehicles did not move.

Figure 21. An armored brigade in the attack at the National Training Center.

D. APPLICATION GATEWAYS

The scalability issue for DIS has been a topic for a number of research efforts previously under the ARPA Project 10,000 and now under the Real Time Information Transfer and Networking (RITN) program. Dan Van Hook and Duncan Miller of MIT-Lincoln Labs along with Danny Cohen of Perceptronics have suggested and explored a number of techniques to reduce DIS bandwidth requirements on network tail-links and minimize the number of packets offered to the WAN.

The primary focus of their effort has been the design of Application Gateways (AG) that would mediate between the subnets over a WAN and the simulations running on each subnet. This approach has been driven by a number of practical factors. First, encryption devices, known as the Network Encryption System (NES), from Motorola have been used to secure demonstrations like Warbreaker and STOW, which are often classified. The NES is limited to T-1 speeds (1.5 Mbps) and, more importantly, to a fixed Ethernet frame per second limit (approximately 200 Ethernet frames per second) [20]. Packets are not only lost if the limit is exceeded, but the NES also crashes. To reduce the packet count and help with the greedy terminal problem, the AG can "bundle" or aggregate PDUs from the subnet together into larger PDUs that fit within the Ethernet MTU. DSI also prefers large packets [18].

Another imperative for the AG has been then the fact that dozens of contractors now provide different simulators for large exercises like STOW. Experimenting with the protocols used by hundred of simulators is impractical in such situations even though these exercises are supposed to demonstrate scalability schemes. Even simulators that use SIMNET protocols still participate with DIS applications via protocol translators (e.g., Cell Adapter Unit or CAU). Finally, the AG has been seen as way for accomplishing scaling exercises now while perhaps applying the successful gateway algorithms to DIS clients in the future.

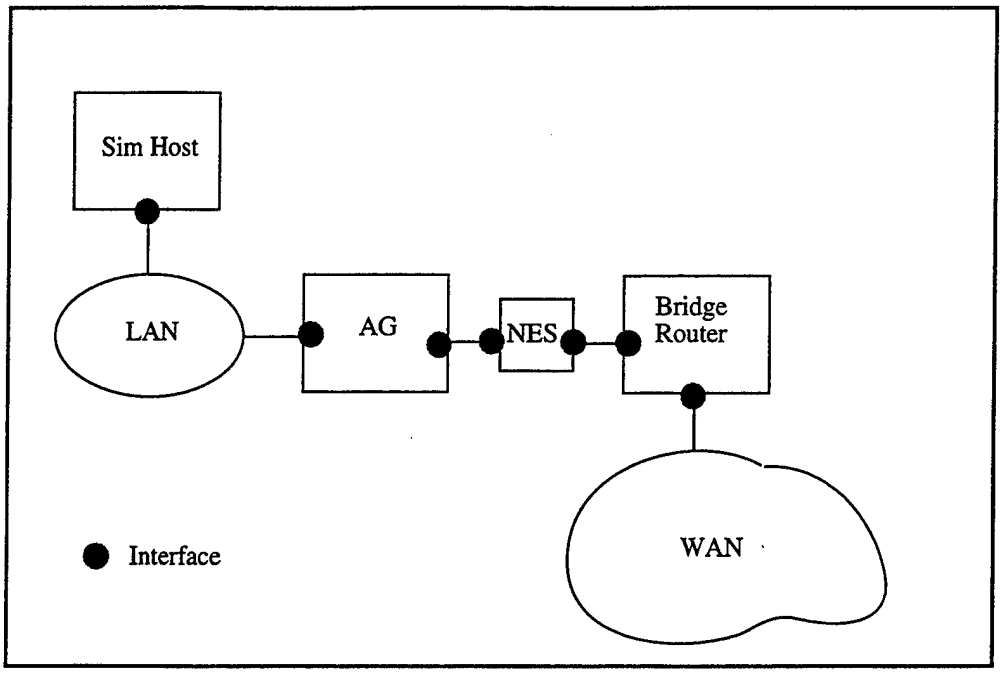


Figure 22. Application Gateway (AG) setup.

1. Techniques

Besides bundling several techniques have been either proposed or demonstrated with an AG. For STOW, these techniques included [151]:

a. PICA

The delta compression technique, Protocol Independent Compression Algorithm (PICA), takes advantage of the fact that most information in DIS entity state PDUs is redundant from one issuance to the next. The primary changes occur in the field for orientation and location. Therefore, the AGs exchange only the changes in the ESPDU, occasionally sending the entire PDU.

b. QES

With the quiescent entity service (QES) the AG determines and keeps track of the stationary entities on the subnet. The heartbeat ESPDUs of the entities are not forwarded.

Rather, the other AGs are notified which entities are "quiet" and they recreate the heartbeats on each net.

c. *Grid filtering*

The AG filters out PDUs which are outside the geographic playbox of the entities on the subnet. This is done at the application layer by the receiving gateway.

d. *Culling*

The AG discards irrelevant ESPDU data by entity type or non-DIS traffic. For example ships are removed for a subnet with ground vehicles.

e. *Summary*

The AG essentially attempts to scale DIS without changing the underlying DIS model. Client applications on the subnets remained unchanged. Instead the AG acts as an intermediary between the LAN and the WAN. The specific techniques cleverly deal with the bandwidth issues.

2. Application Gateway Problems

There are number of problems with using the AG and the techniques described above. First, the AG can be a significant network bottleneck. Unlike bridges and routers, the AG does significant processing of PDUs at the application layer. Figures 22 and 23 show WAN to LAN processing. For example, grid filtering requires it to read every packet directed to the AG from the WAN, do a system interrupt, run a device driver, copy the packet from the interface to kernel space and then to user space, where the PDU coordinates are compared to the grid filter coordinates. At that point the PDU is discarded or forwarded for more processing. (An assumption also must be made about where the entities will go in order to determine the playbox).

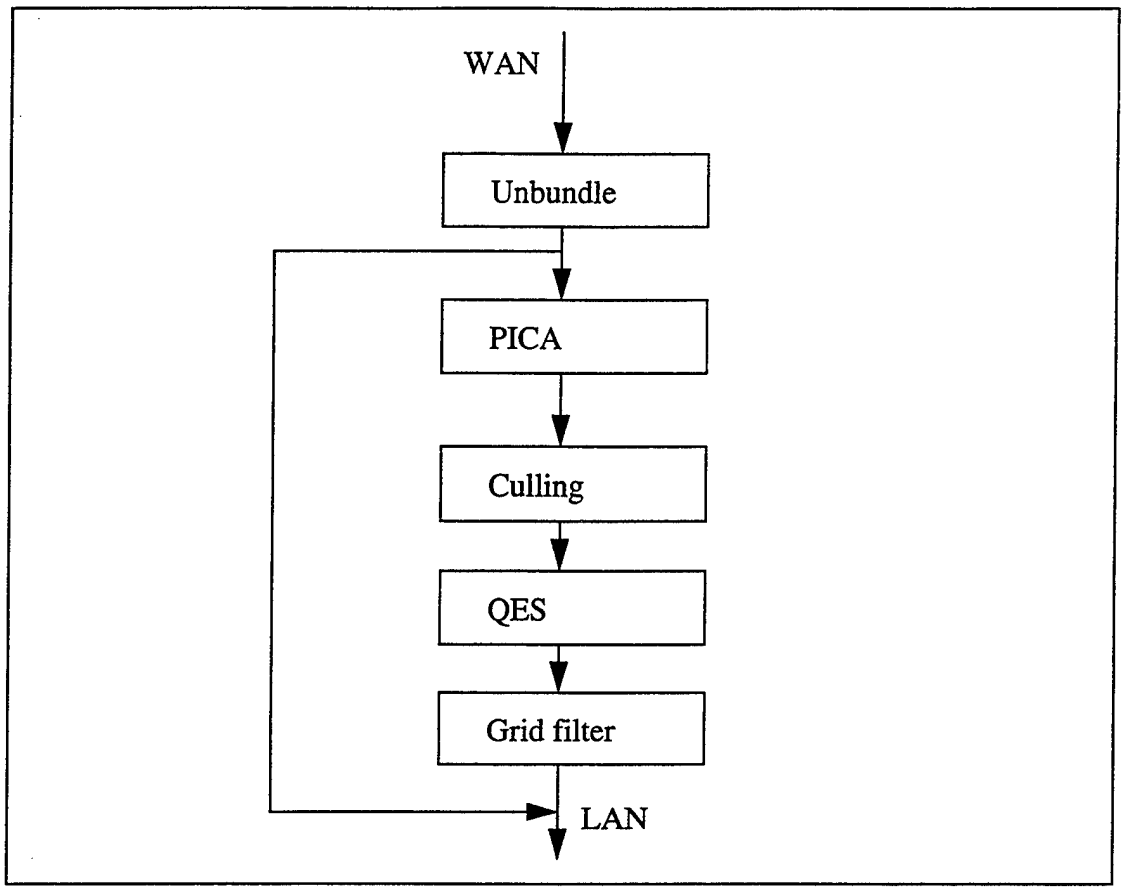


Figure 23. AG processing.

If this was the only processing involved the performance penalty might be acceptable. However, the AG must maintain and compute state for all relevant entities for both the PICA and quiescent entity service. Moreover, the aggregation of PDUs introduces latency.

The AGs used for STOW were Indigo workstations. These same types of workstation, Indigo 2 R4400s, were used for logging PDUs for later analysis. The maximum rate for merely logging PDUs was less than 3000 PDUs per second. *The implication is that a simulator would not likely keep up with that rate. Assuming that 60% of the entities are communicating in any second and that each entity sends eight PDUs per second then approximately 625 entities can be supported by the simulator.* Furthermore, the AG is an-

other device between peer applications. A PDU must go through two host interfaces, four for the AG, four for encryption, and at least four for the router (Figure 22).

PICA has some potential pitfalls. First, because DIS uses unreliable transport mechanisms (and network in the case of ATM), if a cell or packet is lost, the loss of a PDU delta introduces error for the next several PDUs. If the PDUs are bundled the impact may be much more adverse.

Therefore, some mechanism for reliability is required. The use of sequencing and negative acknowledgments (NAKs) has been suggested for PICA but this may have other unintended consequences such as NAK implosions. In this case a packet is lost. This causes other AGs to issue NAKs which in turn cause congestion and the loss of more packets which result in more NAKs. Responding to NAKs also introduces more latency. PICA also does not affect non-ESPDU traffic. This will be a problem as non-ESPDU traffic such as simulation voice and data communication increase.

E. SUMMARY

These potential problems of the AG and have finally led many researchers to believe that fundamental changes in the DIS protocol are required. For example, the RITN effort will likely eliminate the AG and exploit emerging network technology and algorithms to move session management and bandwidth reduction techniques back to the application host. In this context, Van Hook has suggested some interesting techniques for using multicast communication that we address in later chapters as we discuss an architecture that overcomes the current limitations of DIS.

IV. NPSNET OVERVIEW

A. INTRODUCTION

The NPSNET-IV networked virtual environment, developed at the Computer Science Department of the Naval Postgraduate School (NPS) in Monterey, California [162] is our primary experimental testbed for developing VEs. Furthermore, we have been exploring new technologies such as multicast networks, constructing methods for substantially reducing bandwidth requirements with distributed simulation protocols, and designing our software to exploit parallel computing architectures. (Figure 24 shows an image from NPSNET.)

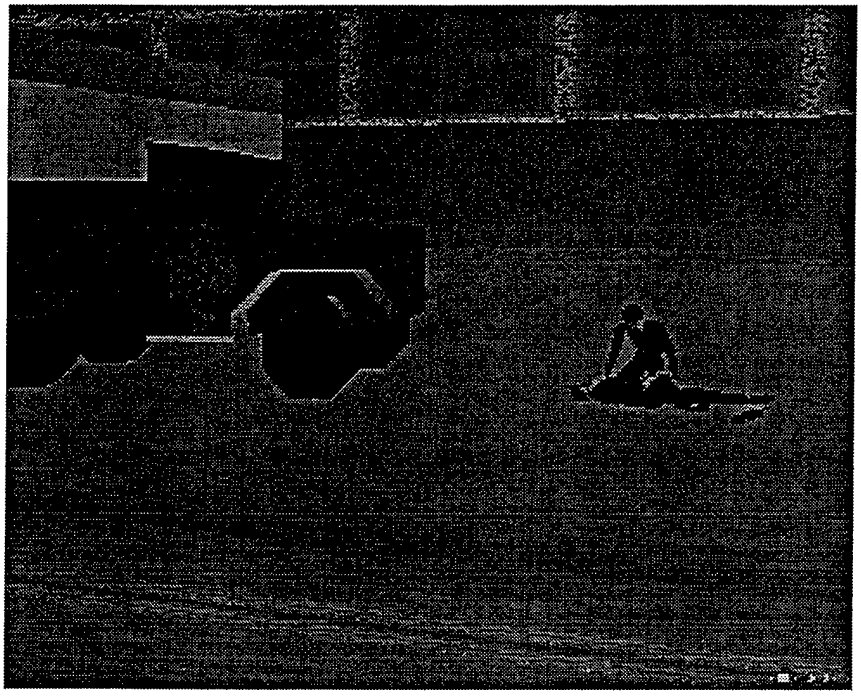


Figure 24. Medic conducting first-aid in NPSNET.

The NPSNET Research Group has devoted itself to exploring several functional areas of interactive simulation including:

- application and network level communication protocols.

- object oriented techniques for virtual environment construction.
- real-time physically-based modeling (e.g., smoke, dynamic terrain, and weather).
- multimedia (audio, video and imagery).
- artificial intelligence for autonomous agents or entities.
- integrating robots into virtual worlds.
- human interface design (stereo vision, system controls).

NPSNET-IV is unique in distributed simulation because it incorporates all of the following in an operational visual simulator to study the above areas:

- **Distributed Interactive Simulation (DIS 2.0.3) protocol** for application level communication among independently developed simulators (e.g., legacy aircraft simulators, constructive models, and real field instrumented vehicles).
- **IP Multicast**, the Internet standard for network group communication, to support large scale distributed simulation over internetworks.
- **Heterogeneous parallelism** for system level pipelines (e.g., draw, cull, application, and network) and for the development of a high performance network software interface.

B. EVOLUTION

NPSNET has evolved significantly since its early origins as a distributed environment (see Figure 25). Previous incarnations of NPSNET (I and II) used a locally designed network scheme that required Ethernet as the local area network protocol and incorporated an ASCII-encoded application level protocol to convey the simulation state. The packet, or (in ISO terminology) application protocol data unit (PDU), lengths were disproportionately long for the amount of information they contained and they did not comply with any particular standard. Moreover, the application protocol did not use any internetworking proto-

col, therefore it restricted use of NPSNET to the local LAN segment and a single network technology (Ethernet).

Another early developmental effort of NPSNET was the NPSStealth. NPSStealth was a version of NPSNET that integrated a translator for the Bolt Baranek and Newman (BBN) developed SIMNET protocol for interaction over local and long-haul networks. The inclusion of the SIMNET protocol enabled NPSStealth to participate in distributed simulations with other simulators that used the SIMNET protocol.

These efforts provided our group with a substantial amount of experience in designing a distributed virtual environment. However, these early versions were not intended for large scale simulations. In order to develop such a system, we needed to adopt the DIS protocol for interoperability with other simulators and create a scalable software and network architecture [70].

NPSNET-IV can be configured by the user as a simulator for an air, ground, nautical (surface or submersible), virtual vehicle, or human. (A virtual vehicle is a non-invasive entity that maneuvers in the simulated world but is not represented by a model - a stealth vehicle.) The user controls the vehicle by selecting one of several interface devices which include a flight control system (FCS) (throttle and stick), a six degrees of freedom Spaceball, and/or a keyboard. The system models vehicle movement on the surface of the earth (land or sea), below the surface of the sea, and in the atmosphere. Other vehicles in the simulation are controlled by users on other workstations. These users can either be human participants (e.g. the medic in Figure 24), rule-based autonomous entities (e.g., Modsaf), or entities with scripted behavior.

The system is automatically configured for distributed network operations at start-up. In this mode, the operator interacts with other human participants who are "piloting" other vehicles. The virtual environment is populated with vehicles operated by players at NPS

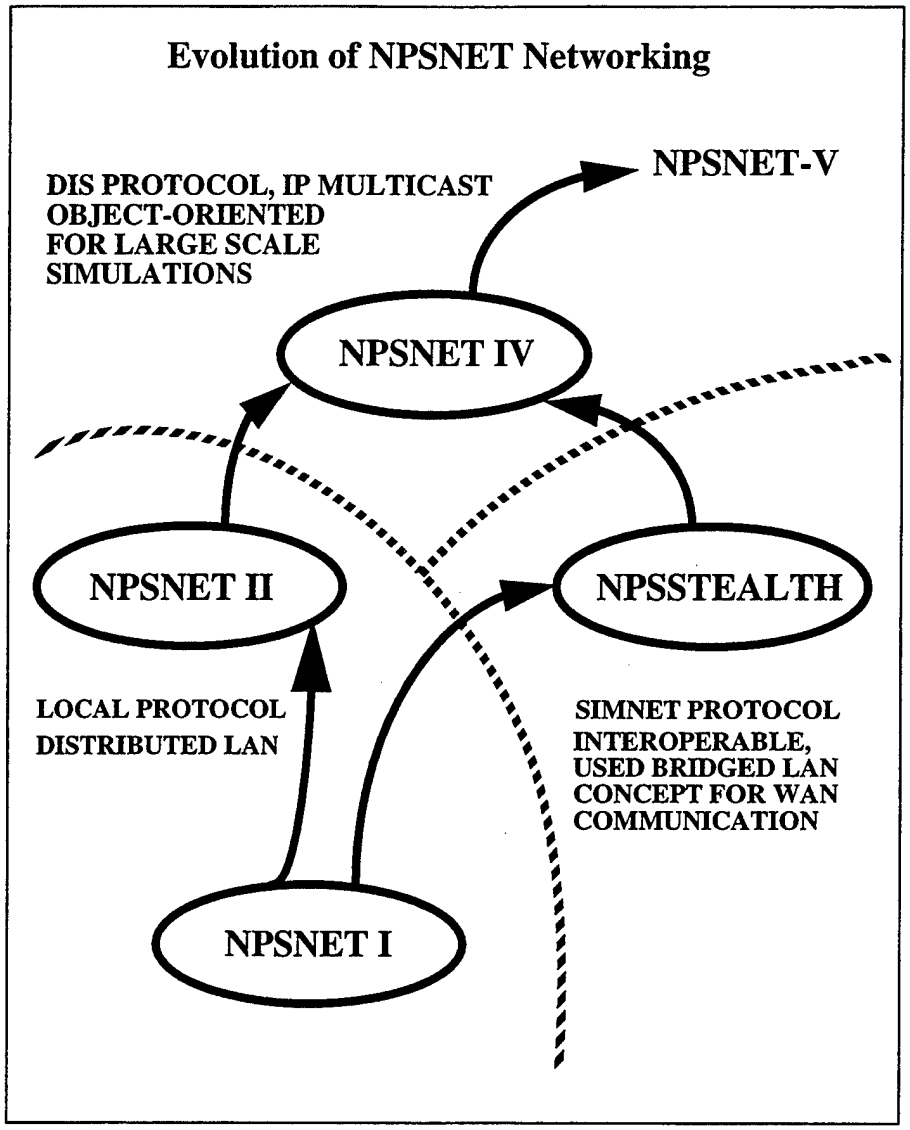


Figure 25. Evolution of NPSNET networking.

and other participants from around the country; and a variety of static and dynamic objects which exhibit numerous visual and multimedia behaviors including sound. For example, flying by a farm evokes a chorus of animal noises.

C. NETWORK ENTITY/PDU PROCESSING

In order the keep track of remote (or networked) entities, the local simulator must have some type of entity list containing entity identification, postures, geometry model data and

other pertinent information. To reduce network traffic in a DIS simulation, the local simulation system must dead reckon each remote entity's position between receiving actual Entity State PDUs. This dead reckoning implies that the data concerning remote entities maintained in the entity list is lower resolution compared to the higher resolution knowledge of the locally controlled entity.

An overview of the processing of incoming PDUs by NPSNET-IV is diagrammed in Figure 26. This diagram is a mixture of a flow chart and data flow. Red ovals and lines represent data and data flow while black boxes and lines represent events and control flow [11].

Figure 26 illustrates the separate, asynchronous network reader process that continually reads the network for incoming PDU information. NPSNET can run either in the old style SIMNET flat world or the new style DIS round world coordinates by specifying a runtime command line argument. DIS exercise filtering takes place at the network reader level so only the incoming PDUs for the current exercise are passed on to the application via shared memory.

As part of the application's main simulation loop, a PDU is read from the shared memory queue and is processed based on its type. Once a PDU is processed, its associated memory is freed.

Incoming Entity State PDUs notify the local simulator of changes in a remote entity's state. Even if a remote entity is not moving, a "heartbeat" PDU message is sent at regular intervals. A de facto standard is that a PDU must be sent at least every five seconds. Figure 27 shows the major steps of handling incoming Entity State PDUs by NPSNET-IV. As illustrated, three main data structures are used in conjunction with maintaining entity information: the Entity Type List, the Entity Hash Table and the Entity Object List. The Entity Type List contains information concerning the types of entities that can be accurately rep-

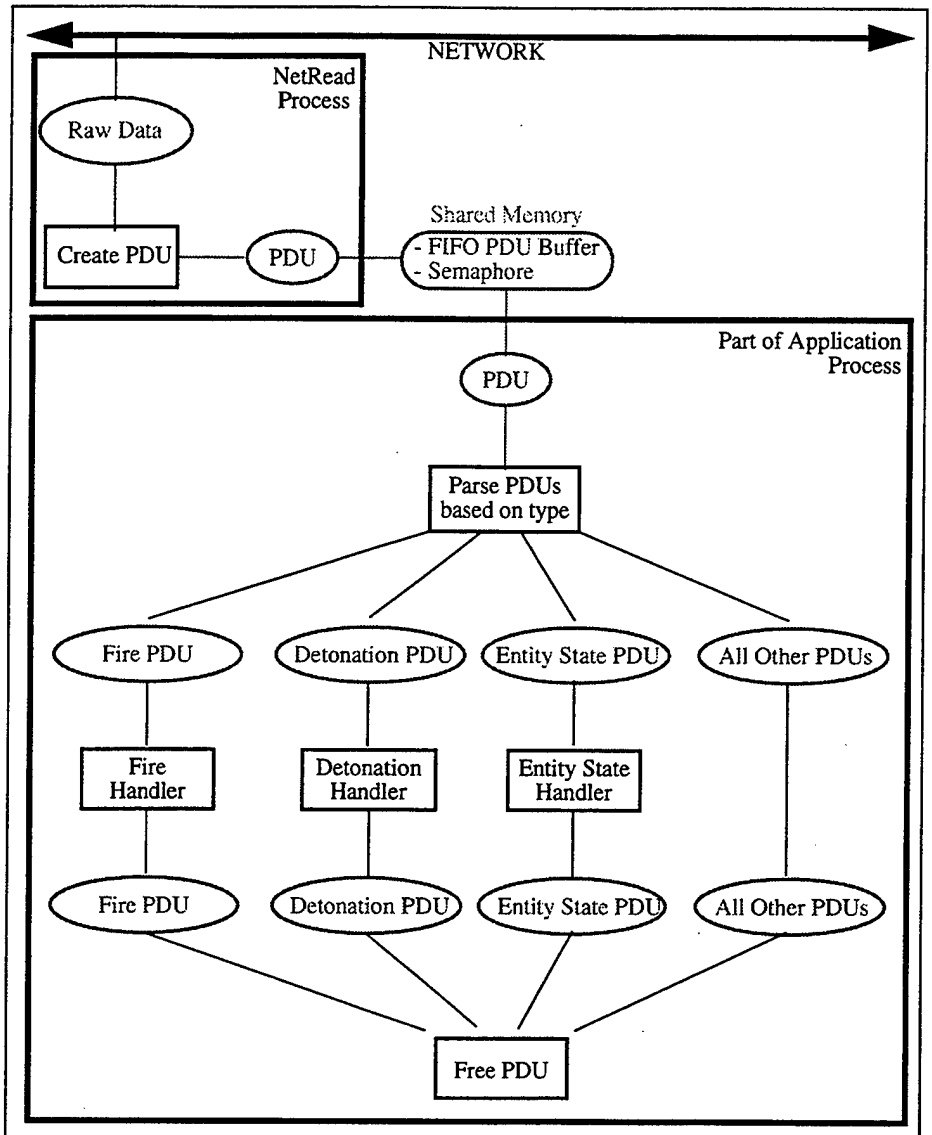


Figure 26. Incoming PDU handling in NPSNET-IV.

resented for the simulation. If an unknown or unsupported entity type comes across the network during a simulation, then the closest existing supporting match is used from the Entity Type List.

The first time an Entity State PDU is seen for a new entity, a hash table entry is made and a new entity object is created according to the type of entity. Subsequent Entity States for an entity are hashed to find the corresponding entity in the Entity Object List. The entity

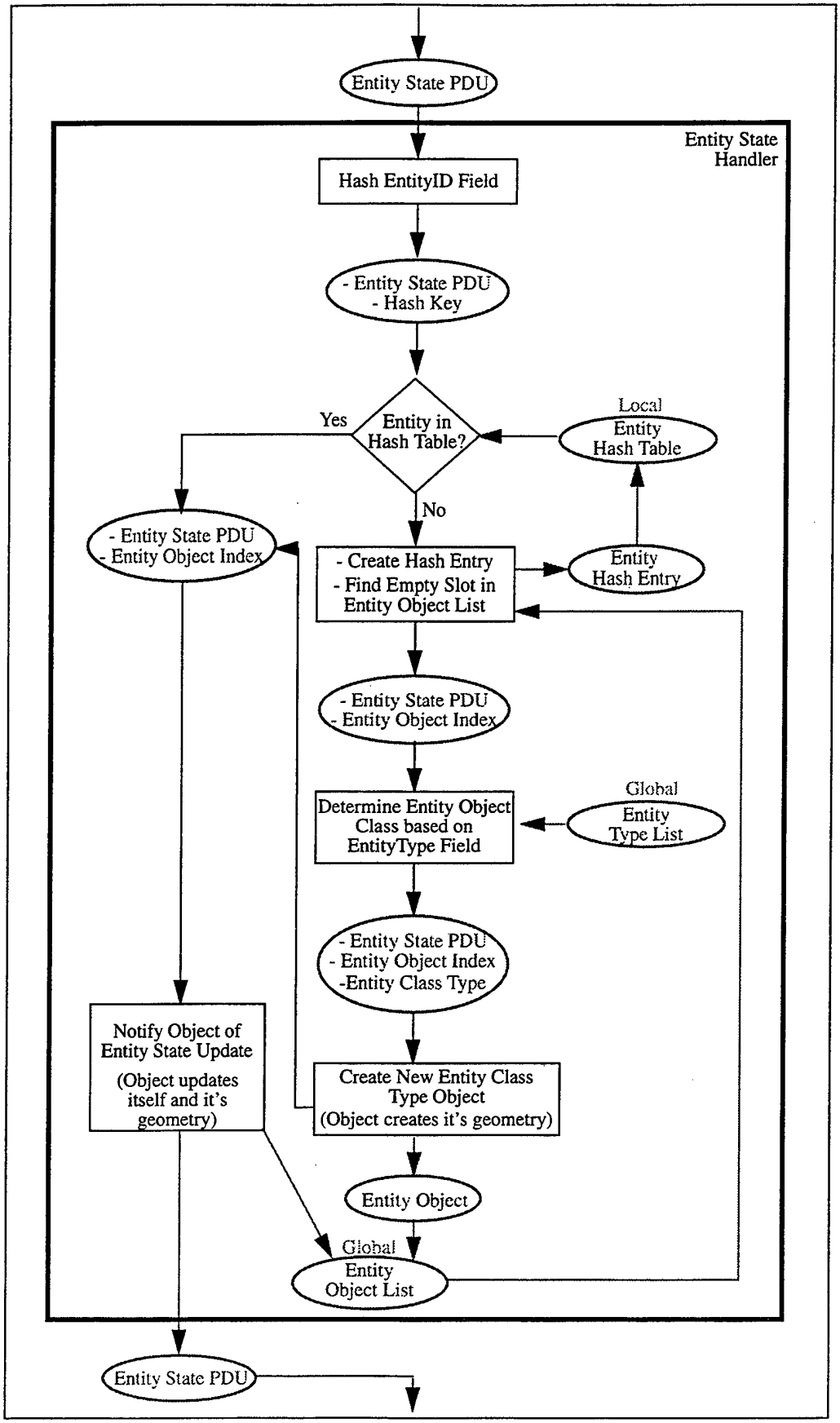


Figure 27. Entity State PDU processing in NPSNET-IV.

object is then notified to update its data from the information contained in the PDU. In this manner, the Entity Object List contains the most current information from all remote entities.

This technique reveals another problem with DIS because of redundant addressing. Hashing is done on the site, host, and entity id in the PDU. These ids are not well-managed by a central authority as is the case of Internet addresses. We have encountered on occasion several sites using the same site id. This can cause problems because of name-space collision for large-group and is reflected by aliasing of entities in NPSNET. In the future, we will likely change to hashing on the Internet address and entity id.

Each time through the main simulation loop, each active entity object (both remote and local) is notified to perform a moveDR (dead-reckoned move) operation. Upon receiving this request, the entity dead reckons its new posture based on its last posture, the amount of time since the last move/update and the dead reckoning algorithm and data defined by the remote entity. Figure 28 illustrates the major events for a remote entity update. If an Entity State PDU has not been received for a remote entity in two time periods, then the local object corresponding to the remote entity is deactivated. This time-out period is equal to twelve seconds in NPSNET-IV which corresponds to two, five second periods with two seconds grace time for network delays.

Figure 29 shows how the local entity's PDU is processed, formed, and issued. Note that the local entity maintains a dead reckoned model of itself to determine whether it has reached a predetermined error threshold with respect to its actual position. If so, or a collision has occurred, then an ESPDU is issued.

When a Fire PDU is received, the type of munition is determined based on information from the PDU and effects are identified based on this type. If a visual effect is matched to

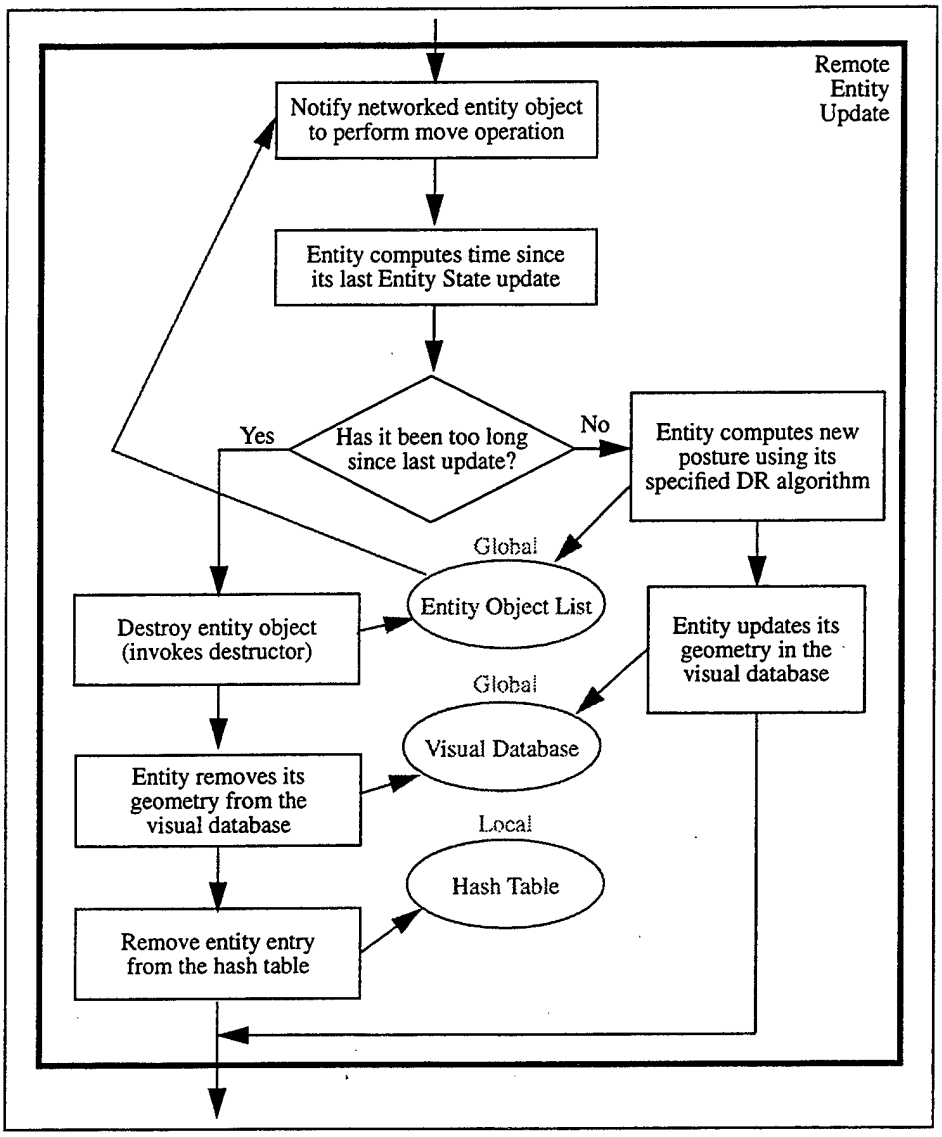


Figure 28. Entity update operation in NPSNET-IV.

the type of munition being fired, then the visual database is updated to reflect the visual effect.

When a Detonation PDU is received, several possibilities exist for processing the event. Figure 30 demonstrates the processing of incoming Detonation PDUs by NPSNET-IV. DIS specifies several different detonation types depending on the result of the event.

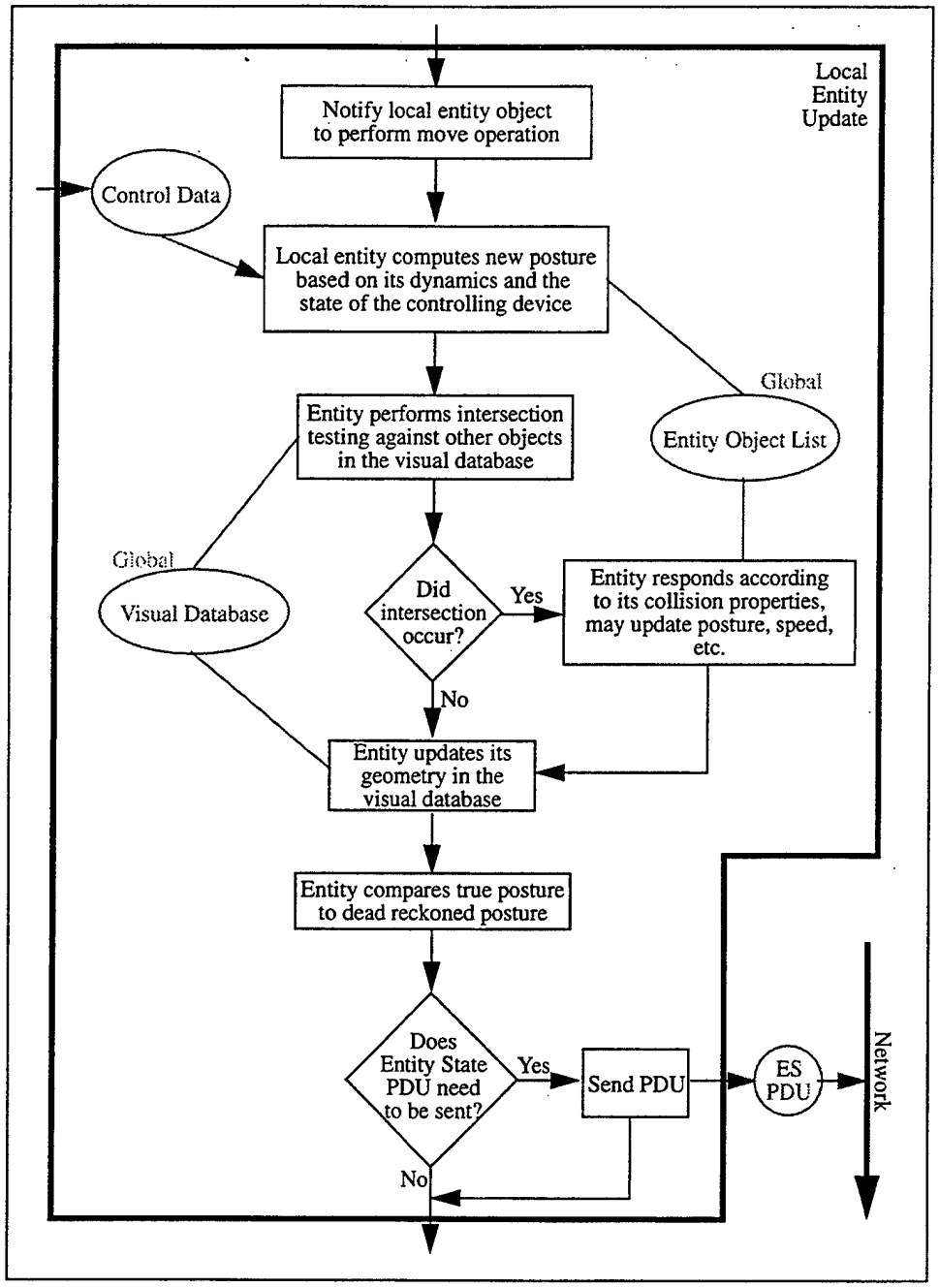


Figure 29. Control of local entity movement in NPSNET-IV.

For munitions that hit the terrain, NPSNET-IV conditionally craters the ground and starts a smoke plume at the point of impact depending on the size of the munition. For detonations that hit on or near an entity, the local entity's position is compared to the impact point and

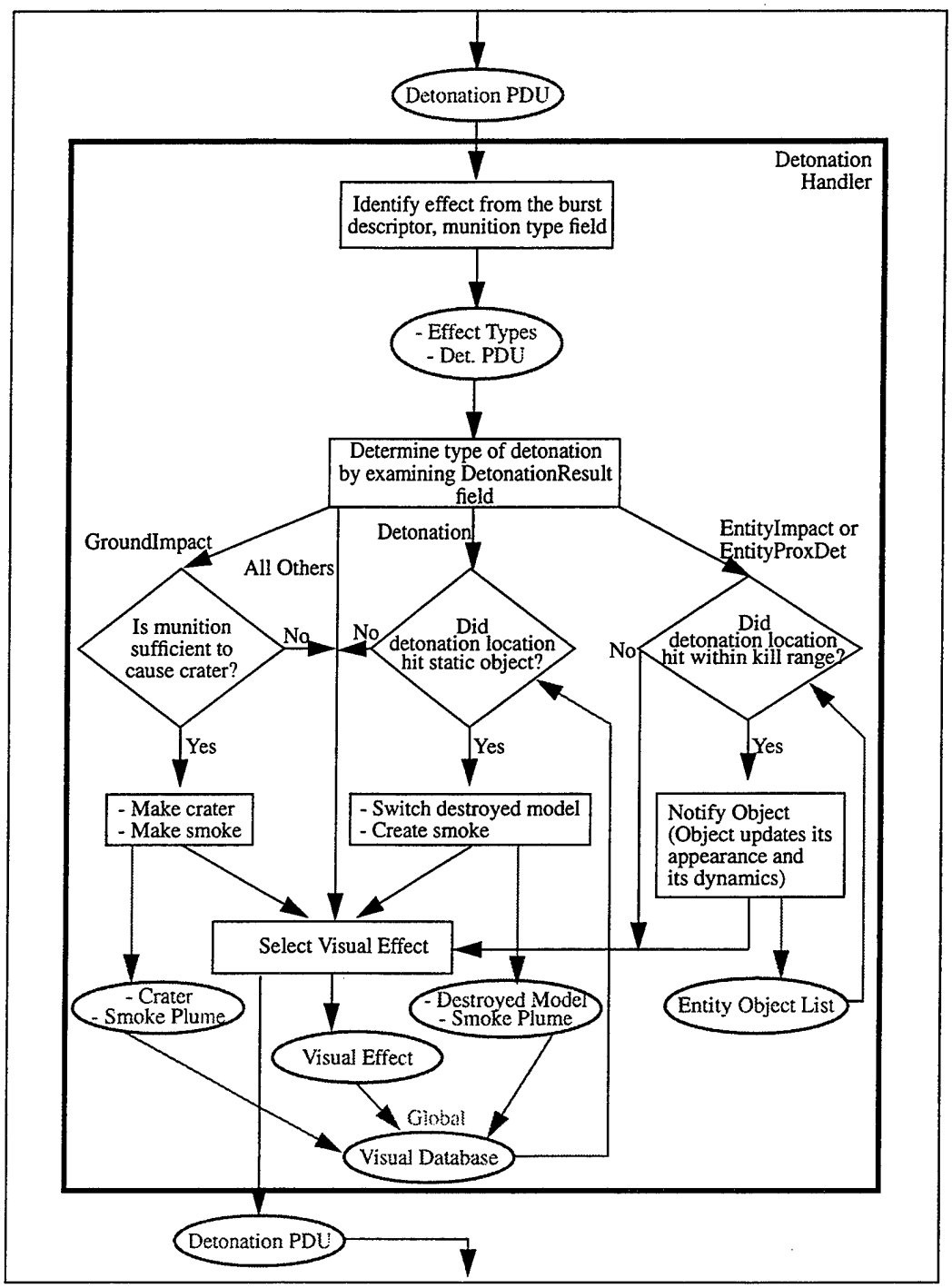


Figure 30. Processing of incoming Detonation PDUs by NPSNET-IV.

if the impact is within the kill range, then the local entity object is instructed to die. If a detonation occurs on or within a non-networked, static object, then the static object is lo-

cally destroyed and a smoke plume is started if the munition is of sufficient size. All types of detonations, except for explicit duds, are identified by the munition type to see if a visual effect is appropriate. If so, then the effect is added to the visual database.

All other incoming PDUs are ignored by NPSNET-IV. This decision was made due to the fact that NPSNET-IV is designed more for reasons of virtual world research in general than to be a fully supported DIS application. Also, most of the other PDUs are still under development and are a source of debate regarding their applicability in the DIS community.

D. HETEROGENEOUS PARALLELISM

Man-in-the-loop simulators function in real (wall-clock) time and are measured in the human perception time frame, which is approximately 100 milliseconds. Update information must be received by all other participating simulation hosts in sufficient time for reading the information from the network, updating the database, and rendering the display.

NPSNET-IV is designed to minimize system latency and maintain a frame rate (throughput) of at least 10 frames per second. To accomplish this we have taken advantage of the multiprocessor architecture and heterogenous parallelism offered by the SGI Onyx computers with Reality Engine2 graphics and the Performer graphics library.

However, as noted by Paul Barham, NPSNET-IV uses the idea of run to completion frame rates in that no matter how much time is needed for each algorithm step per frame, the step is allowed to complete before continuing [11]. So, if there are 100 entities coming across the network, all 100 are updated each frame regardless of how long it takes. This implies that a frame rate cannot be guaranteed and thus the visuals may drop below the perceivable smooth motion rate of 10 Hz. The interface used relies primarily on the keyboard for input. This is because the FCS, KAFlight and Spaceball systems have very few buttons for input of options. This system was used since Motif/X was not well integrated with Performer 1.2. Only the built-in Level of Detail (LOD) processing of Performer is used to con-

trol the complexity of the graphics scene. This implies that when the view of the world contains many polygons that the entire system and frame rate becomes dependent on rendering all of the polygons [11, 125, 126].

Exploiting parallelism to achieve high performance graphics is not a new concept and has in fact been incorporated into the Performer library. As an example, Akeley showed in 1989 that a sixfold increase in frame rates could be achieved using a four processor architecture instead of a uniprocessor system [3]. This is because out-of-view regions of a geometric database can be culled using spatial partitioning by one of the processors.

The effect is to reduce the number of polygons needed to be rendered by the drawing processor and eventually sent to the graphics hardware. In NPSNET-IV, the cull thread traverses the database, selects the geometry that is in the view volume, does Level of Detail selection, and constructs the display list for the draw function. In turn, draw renders the display list data structure.

We have expanded the use of parallelism to improve the *network throughput* as well as graphics throughput. Network processing makes significant demands for system resources, particularly within the context of large distributed environments. When a simulator receives data, a network frame is examined and copied from the interface to a system buffer by means of a direct memory access (DMA) block write. The operating system checks the IP and UDP headers to see if the packets are correctly addressed. For IP multicast, this may require demultiplexing among several addresses and destination application ports. The datagram is again copied by the CPU to the application memory space where it must be decoded by the network thread. The type of PDU and length are determined to allocate the appropriately sized structure in the shared memory.

With a four-processor machine, NPSNET-IV divides the draw, cull, application, and network threads into a system level pipeline using shared memory (IRIX arenas) for com-

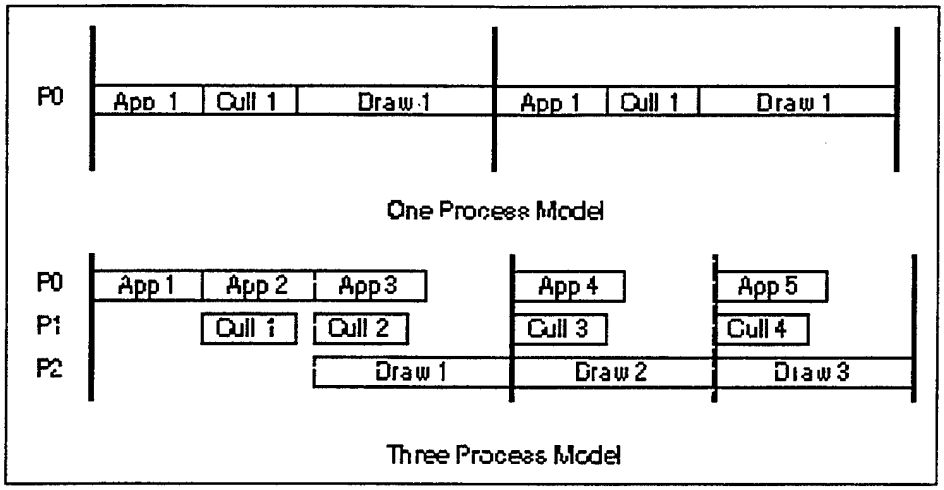


Figure 31. Process models.

munication among the tasks (Figure 31). At the first stage, the network thread uses blocking reads on the IP receive socket and immediately writes new PDUs to an arena (Figure 32).

1. Advantages

The advantages of placing the network thread on its own processor are twofold: it off-loads the work that would otherwise compete with the application, cull, or draw processes; and it immediately receives and buffers PDUs when offered by the network interface (rather than waiting for the other threads to complete), thus reducing the probability that packets are dropped at the lower system network levels. For example, with a simulation frame rate of 10 Hz and an arrival rate of 1000 PDUs per second, approximately 15K bytes must be buffered by the network thread while the simulation loop performs the following:

- updates the terrain database and the visual database maintained by Performer.
- computes dead-reckoning and detonation effects.
- takes user commands to the vehicle via flight control sticks or a spaceball.
- updates the posture of the vehicle and computes the firing solutions for weapons.

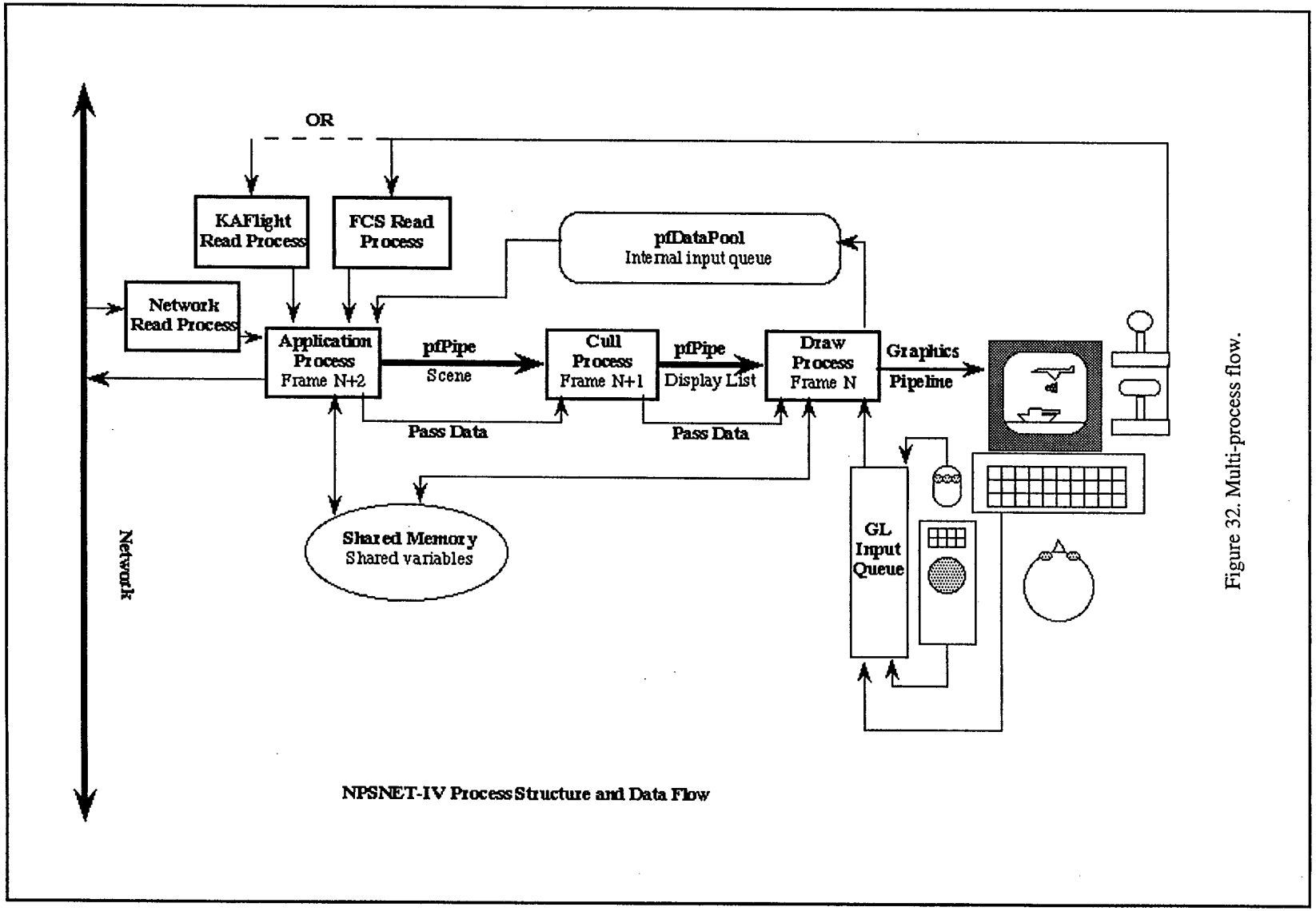


Figure 32. Multi-process flow.

- writes new PDUs to the network process.
- hands off to the cull process.

Note that the size required for this buffer (arena) is variable and dependent on the network load and the complexity of the scene rendered which influences the cycle time for frame updates.

2. Disadvantages

There are several disadvantages to the NPSNET-IV approach. First, the implementation is specific to SGI systems because we use special IRIX system-calls and Performer. Second, there is a penalty incurred by the extra data copying which causes more contention for memory, bus bandwidth, and the CPU which must do the copying. Another cost is that latency may increase in a particular stage with respect to the CPU buffering data. However, these problems are more than offset by the overall gains in system throughput and by the fact that a CPU is dedicated to the network data movement in a multiprocessor machine [37]. Furthermore, the handling of DIS traffic at the application level will likely become more complicated as the protocol matures, demanding more processing resources. An example of this is that future DIS PDUs are expected to be aggregated within a UDP datagram to reduce data link and network level processing while further increasing the overhead of decoding by the receiving application.

E. Demonstrated Results using IP Broadcast and DIS

We were able to demonstrate NPSNET-IV to a large audience at the SIGGRAPH'93 Tomorrow's Reality Group albeit using IP broadcast over a bridged wide area network for compatibility with other sites. We have also successfully conducted demonstrations and test of our multicast version over the MBONE that we discuss in the next chapter.

Figure 33 shows some of the difference in implementation between the multicast and broadcast code. The broadcast code merely requires that the socket be designated as SO_BROADCAST. Multicast sockets require several additional parameters including time-to-live (ttl) which provides network scoping. We also prevent listening to our own PDUs by setting the no loopback option.

```
if (network_mode == BCAST_MODE ) {

   /* Mark send socket to allow broadcasting */

   if (setsockopt(sock_send, SOL_SOCKET, SO_BROADCAST, &on,
         sizeof(on)) < 0) {
         perror("ERROR: DIS_net_manager unable to mark send
               socket for BCAST -\n    ");
         return (FALSE);
         }
   }
   else if ( network_mode == MCAST_MODE ){

         /* Configure send socket time-to-live for multicast */

         if (setsockopt(sock_send,IPPROTO_IP,IP_MULTICAST_TTL,
                  &mc_ttl,sizeof(mc_ttl)) < 0) {

         perror("ERROR: DIS_net_manager unable to mark send
               socket for MCAST -\n    ");
         return(FALSE);
         }

         /* Configure local packet loopback for multicast -- */

         if (setsockopt(sock_send,IPPROTO_IP,IP_MULTICAST_LOOP,
                  &loopback,sizeof(loopback)) < 0) {

               perror("ERROR: DIS_net_manager unable to config
                     ure loopback -\n    ");
               return(FALSE);
         }
```

Figure 33. Implementation of IP Broadcast and IP Multicast in NPSNET.

At SIGGRAPH, participants were able to operate in a three-dimensional virtual environment via local and wide area communication networks with other players using independently developed simulations at geographically dispersed sites. Our SGI hosts in Anaheim, California communicated locally through Ethernet and to other sites via a T-1 based private network. Other participating sites were the Air Force Institute of Technology (AFIT), Dayton, Ohio, the Simulation Center, Advanced Research Projects Agency (ARPA), Arlington, Virginia, the Naval Postgraduate School (NPS) in Monterey, California, and Naval Research and Development (NRaD) in San Diego, California (see Figure 34). The connection with ARPA was unique in that it included simultaneous high-quality two-way video and audio over the network. The video from Arlington was displayed at the convention center on a Sony high-definition television using the Advanced Television (ATV) format -- while images from the simulation were shown at ARPA Warbreaker on a 15 meter "video wall".

Five sites were interconnected using the Defense Simulation Internet (DSI) and other leased T-1 facilities. NPS, AFIT, and the Naval Research and Development (NRaD) facility in San Diego, California were connected by DSI. ARPA was connected to NRaD using their own facilities. NRaD bridged the two networks to a leased T-1 link that terminated in the Anaheim Convention Center. Because the T-1 was multiplexed with video, simulation bandwidth was restricted to 704 Kbps.

A network load analysis was conducted during SIGGRAPH to approximate an upper bound of the number of simulation hosts and entities that can reliably participate on our Ethernet segment. The bandwidth required for the distributed simulation experiments was predictable based on the previous discussion. There was a maximum of eleven hosts simulating 50 entities during the free-play scenario. Seven high-performance aircraft were simulated while four of the simulators modeled multiple slow moving vehicles.

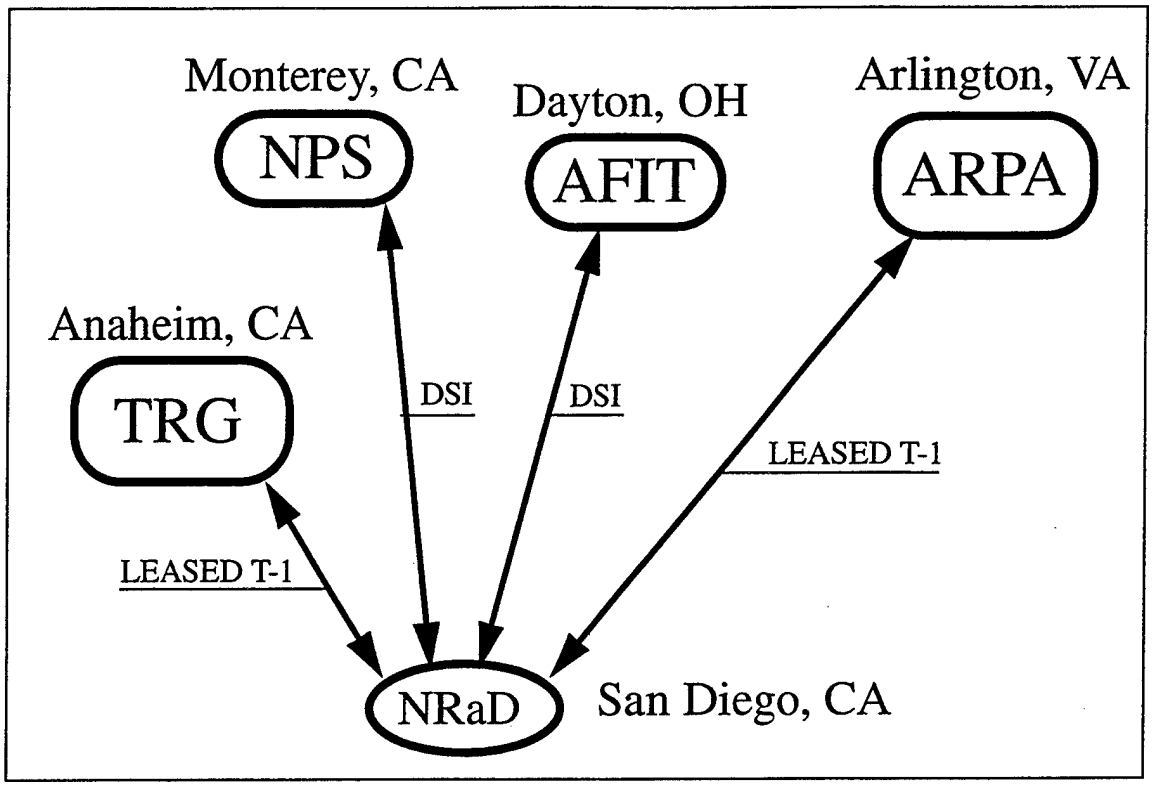


Figure 34. SIGGRAPH'93 setup.

The average broadcast packet length was 190 bytes, including network overhead. Over 90% of the traffic was represented by Entity State PDUs. Simulation traffic peaked at 168 packets per second accounting for 2.5% of Ethernet and 16% of T-1 bandwidth. Extrapolating from these numbers Ethernet could handle at most roughly 600 players at 70% utilization while each T-1 link could accommodate only 130. Note, again, that this extrapolation only gives us only upper bounds using the current DIS protocol.

We met our goals of minimizing latency and demonstrating compatibility with DIS 2.0.3. Comments from observers and players indicated that there were no noticeable delays in interacting with other players over the wide-area and local nets. Moreover, NPSNET-IV operated smoothly with AFIT's Virtual Cockpit which also uses the DIS standard. With the configuration used at SIGGRAPH, we could expect no more than several hundred simultaneous interactive players over the network. Overall, NPSNET-IV has demonstrated robust network capabilities.

F. SUMMARY

In this chapter we discussed our implementation of a VE using DIS, NPSNET, the internal architecture of NPSNET and some of the methods we have used to meet the performance requirements of a real-time simulation. We also described its performance using IP broadcast and dedicated network links. In the next chapter we will present the infrastructure that will permit us to build large-scale VEs without the use of dedicated or proprietary networks.

V. NETWORK INFRASTRUCTURE

A. INFRASTRUCTURE

The ability to construct large-scale virtual worlds is dependent on the network infrastructure to support them. The infrastructure includes the physical links and the network, routing, and transport protocols. This chapter discusses the US national infrastructure and how it can support large-scale VEs.

B. WIDE AREA NETWORKS

The fabric of our national telecommunications infrastructure is being radically altered by the rapid installation of fiber optic cabling capable of operating at gigabit speeds for long-haul traffic. The change has come so fast that AT&T wrote off $3 billion worth of equipment in a single year to replace its analog plant with digital systems. Long distance carriers have been installing Synchronous Optical Network (SONET) switches to support those speeds. SONET is a US and international standard for optical signals, the synchronous frame structure for multiplexed digital traffic, and operations procedures for fiber optic switching and transmission systems. SONET allows lower speed channels such as OC-3 (155 Mbps) to be inserted and extracted from the main rate. SONET defines data transmission speeds to 2.4 Gbps and the major carriers believe that this can be extended to 10 Gbps for a single fiber link [10] (Data rates will likely go much higher in the next century. Japan's telephone company, NTT, has announced transmission of a 20 Gbps data stream over 600 miles of fiber and is working to increase throughput to 100 Gbps using soliton technology.) Both MCI and Sprint have announced that they will have SONET completely deployed by the end of 1995.

The new switches will also incorporate asynchronous mode technology (ATM). ATM provides fast variable rate packet switching using fixed-length 53-byte cells. This permits

ATM networks to carry isochronous (video and voice) data at SONET speeds. ATM is likely to support multicasting (though this capability will more likely be provided by higher layer services) and the International Telecommunication Union (ITU) has written a standard for interfacing ATM with SONET. AT&T and Sprint began providing WAN ATM services in 1994.

ATM may offer another significant advantage because it is a virtual network service offered by the major carriers. Multicasting will not necessarily reduce bandwidth requirements on the backbone. However, the aggregate bandwidth of a large scale simulation could be absorbed by the carrier's backbone network. A 50,000 player distributed environment would roughly require, for simulation traffic alone, a 150 Mbps backbone, approximately the bandwidth of the standard ATM OC-3 rate.

Two other high speed services are being offered today: switched multimegabit data service (SMDS) and frame relay. SMDS, based on the IEEE 802.6 Metropolitan Area Network (MAN) standard, is connectionless, uses frames and fixed length cells, and offers speeds up to 34 Mbps with plans to upgrade to 155 Mbps. It is currently only being offered in local metropolitan areas. Frame relay is connection-oriented (dial-up) and offers speeds up to 1.544 Mbps. Neither of the services are considered well suited for voice or video applications though they are likely to reduce the cost of wide area network services.

The major carriers are not alone in this effort to push wide area networking to faster speeds. The backbone of the Internet, NFSnet, has been completely upgraded to T-3 (45 Mbps) and will transition to OC-3 by 1995. The backbone rate will likely go to OC-12 (622 Mbps) by 1996. The National Science Foundation (NSF) is in charge of this effort as part of the overall National Research and Education Network (NREN) project, which is one of the four components to the US High Performance Computing and Communications program, originally sponsored by Clinton Administration Vice President Al Gore. Part of the project is the installation of OC-12 networks at several regional test beds: Aurora, Casa,

Blanca, Nectar, and Vistanet. They are going after the "Grand Challenge" applications ranging from medical imaging to interactive visualization using ATM, SMDS, and SONET technologies [78].

At the local loop, the telephone line between the central office and customers, intense competitive pressures in the cable and telephone industries are spurring the development of new technologies to allow the currently installed copper lines to operate at megabit speeds without expensive repeaters. The high-bit-rate digital subscriber loop (HDSL) is an encoding scheme being used now to deliver duplex T-1 service. Another scheme that is in the trial stage, asymmetric digital subscriber loop (ADSL), provides 1.5 Mbps in one direction and 16 Kbps in the other. With the use of new compression standards such as CCITT's H.261 (Px64) and MPEG, ADSL-II, a follow-on technology with 3 to 4 Mbps transport capability, could carry real-time video, audio and VR data [66].

The rewiring of the local loop has also begun. Tele-Communications Inc. (TCI), the nation's largest cable company, has announced that it intends to upgrade by 1996 the broadband lines to over 90 percent of its customers with fiber. TCI is doing this in order to support increased channel capacity, High Definition Television (HDTV - which when uncompressed requires 1.2 Gbps bandwidth), and VR services such as games from Sega/Genesis. TCI also will try to counter the threat of the telephone companies entering this lucrative market.

AT&T has had a test bed for developing the fiber optic local loop for several years near Pittsburgh, Pennsylvania. Bell Atlantic, a regional phone carrier, is conducting tests in its employees' homes of a system that delivers movies over the telephone line. The Regional Bell Operating Companies (RBOCs) recently proposed to the Clinton Administration a plan to rewire the local loop with fiber optic cable within 10 years in exchange for permission to enter the information services market and manufacture telecommunications equip-

ment. A bill that failed to pass last Congress year would have permitted the RBOCs to enter the cable business.

C. Local Area Networks

The connection from a 3D graphics workstation to high speed WANs will most likely come from a local area network (see Table 3). Current 1995 workstation architectures (particularly with respect to the memory and system interfaces) such as SGI's can accommodate at least T-3 bandwidths. This was demonstrated by the XUNET testbed which developed an ATM host interface for the SGI Indigo VME bus. Their software and interface was able to simultaneously send and receive at 40 Mbps TCP packets encapsulated in ATM cells through SGI Power series hosts acting as routers [15].

Most LANs use Ethernet (10 Mbps) which is inadequate for the high performance demands of VR and multimedia. Several companies have endorsed the proposal of standards for 100 Mbps Ethernet. Additionally, through the use of switching hubs -- often referred to as collapsed backbones and with gigabit speed backplanes -- a workstation can use all the bandwidth available on a Ethernet segment.

LAN Technology	Capacity Mbit/s	Year of Final Standard	Status
Ethernet	10	1985	In use.
FDDI	100	1989	In use
HPPI	800/1600	1992	In use.
Fibre Channel	132.8 - 1064.2	1993	Some products available.
ATM	45-622	1993	Some products available.
FDDI-FO	1250	Aprox. 1995	Not available

Table 3: LAN technologies.

FDDI (100 Mbps) is used extensively in supercomputer centers. However, most host interfaces operate in the 20-50 Mbps range though the SGI interface can operate at least to

90 Mbps. A new standard for FDDI over unshielded twisted pair (UTP) wiring may make FDDI more affordable for general computing. Unfortunately, both FDDI and Ethernet technologies are not ideal for isochronous data because there is no guaranteed data rate or prioritizing in the protocols. The American National Standards Institute (ANSI) has developed FDDI-II to address this problem by dynamically allocating bandwidth to isochronous applications. ANSI is working on FDDI Follow-On (FDDI-FO) to be finished in the middle of the decade. FDDI-FO will likely be designed for speeds up to 1.25 Gbps. IEEE has also established a working group 802.9 that has issued a final draft of a standard, Integrated Services LAN Interface, which defines a LAN that carries voice, data, and video traffic over UTP.

The high performance parallel interface (HPPI) is an ANSI standard that supports 32 and 64 bit interfaces that run at rates of 800 Mbps and 1600 Mbps respectively. It is a switched architecture and operates over a distance of 25 meters on copper cables connecting supercomputers and their peripheral devices. A serial version of HPPI for fiber optic cables has been proposed to extend the range to 10 km. At the NREN Casa test bed researchers from Los Alamos, Caltech, Jet Propulsion Lab, San Diego Supercomputing Center and UCLA are developing HPPI-SONET interfaces to connect supercompters over multiple OC-3 circuits, providing 1.2 Gbps to 2.5 Gbps bandwidth [33].

Fibre Channel is a proposed ANSI standard for very high speed lans. It is designed to connect over 4000 computers and peripheral over several kilometers at data rates up to 1062.4 Mbps. Fibre Channel will provide a number of upper layer network services that HPPI does not, and it has the backing of IBM and Sun Microsystems. Another proposed standard, Scalable Coherent Interface (SCI), has a potential speed of 8 Gbps [33].

1. ATM

ATM has also been deployed for local area networks. The allure of ATM is that it might eliminate the distinction between wide and local area networks, providing high speed connectivity from desktops across the United States. The Aurora and Nectar test beds are investigating the use of ATM host interfaces for supercomputers [33]. The NRL-sponsored RITN effort is also investigating the use of ATM with DIS.

Several vendors, including Fore Systems Inc. and Adaptive Corp. are selling ATM switches for LANs. Fore Systems sells interface cards for SGI, DEC, and Sun workstations. Each workstations is linked via fiber optic cable or UTP to a switch at 140 Mbps.

The Internet Engineering Task Force (IETF) has also developed a standard for ATM and IP, though an interface to ATM's multicasting service has yet to be defined. Fore supports IP Multicast with its ASX-100 ATM switch. Multicasting is performed by the switch hardware. Multicasting can be performed over Switched Virtual Channels (SVCs) as well as Permanent Virtual Channels (PVCs). Multicast SVCs are created using the Fore proprietary SPANS protocols. Moreover, several researchers are investigating other methods for providing IP Multicast over ATM [156]. ATM is also expected to provide on-demand quality of service guarantees in terms of latency and bandwidth that are not now available with most networks. This will be necessary for real-time applications like distributed virtual environments.

Finally, we believe that the DIS protocol could be optimized for ATM by making the PDUs conform to the 48 byte data payload of the ATM cell. For example, Danny Cohen, of Perceptronics, has suggested that much of the Entity State PDU could be reduced in size by eliminating redundant information such as the host and site identification which is already contained in the IP address of the sender. Some other ways to reduce the size include:

- use canonical versus hierarchial representation of the entity types, and appearances.

- reducing precision. For example, DIS using 64 bit floating point numbers for coordinates which give an accuracy of 0.2nm in a 500 KM box -- as Cohen points out, a bit of overkill [38].

- eliminate the use of the Entity State PDU to communicate markings (use some other means to do this).

- use a single dead-reckoning algorithm.

- eliminate guises (alternate identities).

- eliminate the capabilities field. Only the first 4 bits are defined and they should be part of the entity type.

The overall benefit of Cohen's suggestions would be to reduce the processing and bandwidth requirements for DIS.

D. IMPLICATIONS

The changes in the wide and local area networks will have several implications on the construction of large-scale VEs. There will likely be two general network bandwidth models for large-scale VEs in the future. The first, which we call a "rich" topology will have a high-speed (better than 100 Mbps) local area network connected into a national SONET Gbps backbone via a DS-3 or OC-3 tail-link (see Figure 35).

The other model for the near future is likely to be the asymmetric "home" approach. In this case large amounts of bandwidth (> 10 Mbps) will be available from some senders (e.g. on-demand video services) while most homes will have less than T-1 outbound capability (see Figure 35).

The rich model is preferable but the asymmetric approach is also valid for a large-scale VEs if there are only a few devices per end node. This is because communication is asymmetric within a group. (The ratio of communications is likely to be on the order of N to 1.) Essentially, the bottleneck will become the ability of the devices to process the data in real-time. Another implication is that these network infrastructures will not only support more

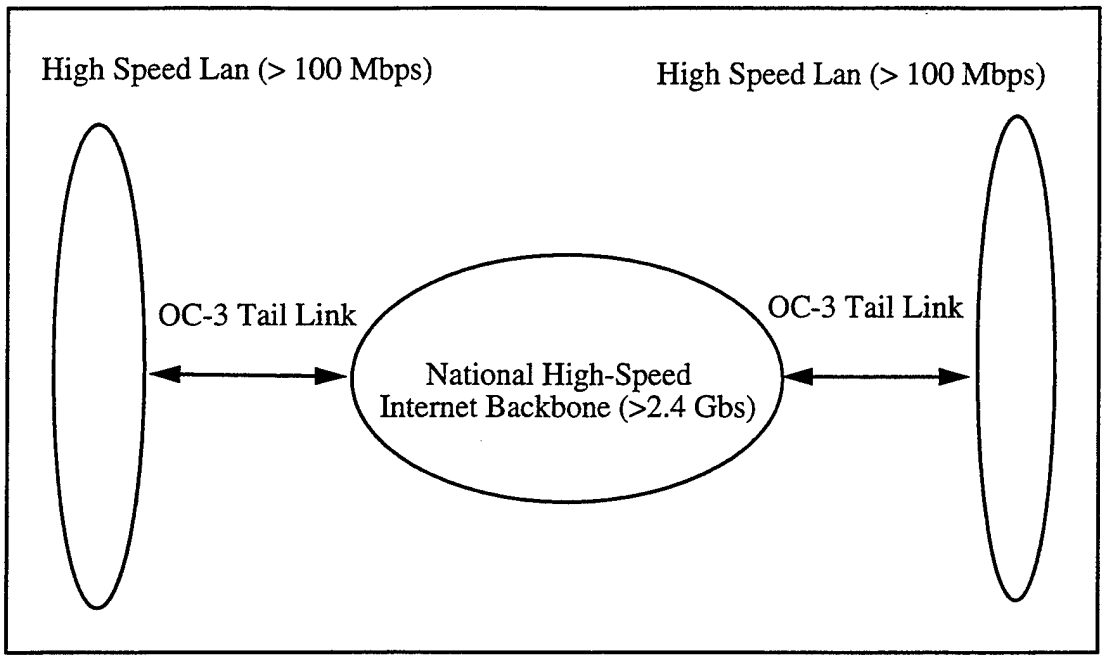


Figure 35. Rich VE network topology.

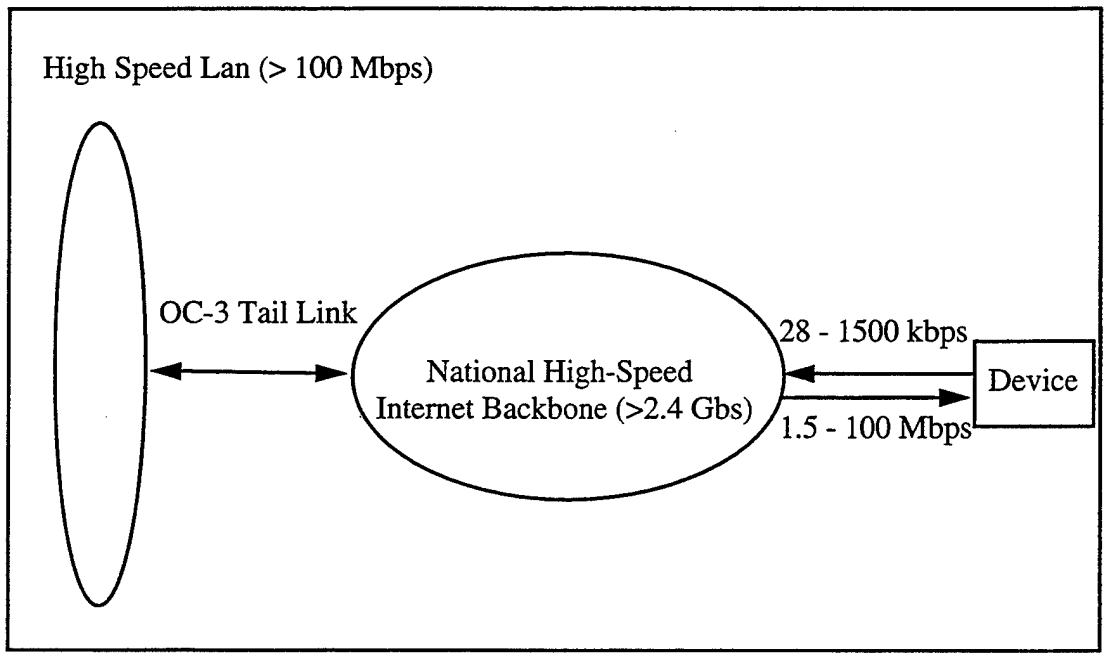


Figure 36. Asymmetric VE network topology.

entities but also more types of data (voice, video) and services such as multicast. In the next chapter, we discuss the current Internet multicast protocols and an experimental implementation -- the MBONE.

VI. IP MULTICAST

A. IP MULTICAST AND THE MBONE

MBONE is a virtual network that has been in existence since early 1992. It was named by Steve Casner [27] of the University of Southern California Information Sciences Institute and originated from an effort to multicast audio and video from meetings of the Internet Engineering Task Force. Today, hundreds of researchers use MBONE to develop protocols and applications for group communication.

Multicast provides one-to-many and many-to-many network delivery services for applications such as videoconferencing and audio where several hosts need to communicate simultaneously. Figure 37 shows this. Players X, Y, and Z send data to the IP Multicast group address 224.11.22.56 rather than explicitly forwarding packets to each and every player. The network takes over this requirement. Players A and B send and receive traffic relevant only to their group, 224.11.22.33, while C is a member of both and participates in each session.

Multicasting has existed for several years on local area networks such as Ethernet and Fiber Distributed Data Interface. However, with Internet Protocol multicast addressing at the network layer, group communication can be established across the Internet. IP multicast addressing [44] is an Internet standard (Request For Comment 1112) developed by Steve Deering [45] of the Xerox Palo Alto Research Center and is supported by numerous workstation vendors, including Sun, Silicon Graphics, Digital Equipment Corporation, and Hewlett-Packard. Categorized officially as an IP Class D address, an IP multicast address

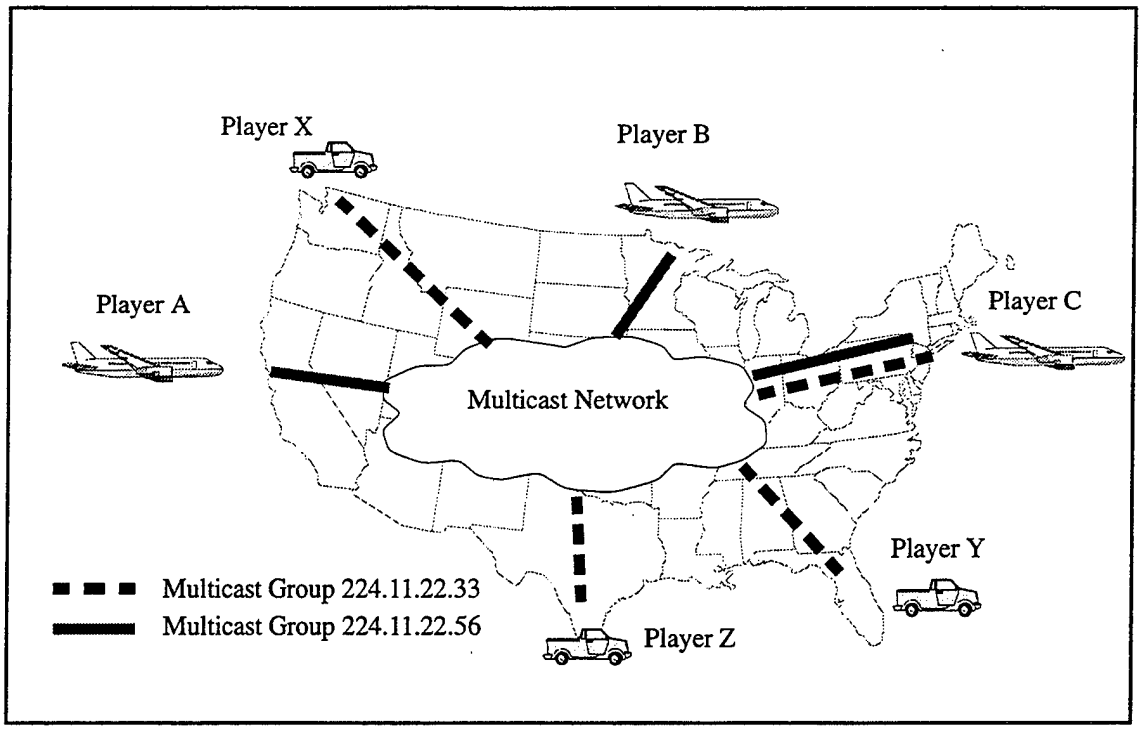


Figure 37. Simple illustration of multicast communications.

is mapped to the underlying hardware multicast services of a LAN. Two things make multicasting feasible on a worldwide scale:

- installation of high bandwidth Internet backbone connections, and
- widespread availability of workstations with adequate processing power and built-in audio capability.

The reason MBONE became a virtual network is that it shares the same physical media as the Internet (see Figure 38). It uses a network of routers (mrouters) that can support multicast. These mrouters are either upgraded commercial routers, or dedicated workstations running with modified kernels in parallel with standard routers.

MBONE is augmented by "tunneling," a scheme to forward multicast packets among the islands of MBONE subnets through Internet IP routers that (typically) do not support IP multicast. This is done by encapsulating the multicast packets inside regular IP packets. As installed commercial hardware is upgraded to support multicast traffic, this mixed sys-

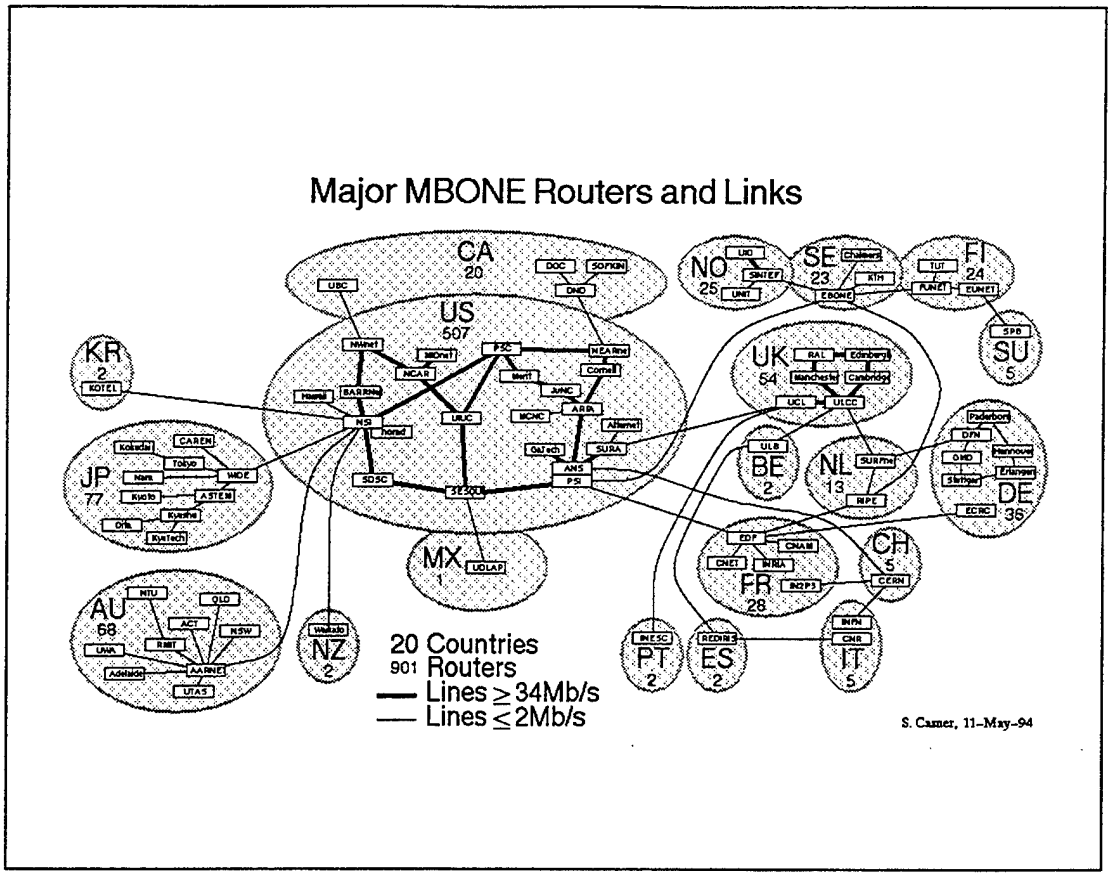


Figure 38. Map of the MBONE [31].

tem of specially dedicated mrouters and tunnels will no longer be necessary. We expect that most commercial routers will support multicast in the near future, eliminating the inefficiencies and management headaches of MBONE, duplicate routers and tunnels. Cisco systems started supporting IP Multicast in its routers in 1994.

1. Bandwidth Constraints

Bandwidth is a key constraint to MBONE. The reason a multicast stream is bandwidth-efficient is that one packet can touch all workstations on a network. Thus, a 128-kilobit per second video stream (typically 1-4 frames per second) uses the same bandwidth whether it is received by one workstation or 20. That is good. However, that same multicast packet is ordinarily prevented from crossing network boundaries such as routers. The rea-

sons for this current restriction are religious and obvious from a networking standpoint. If a multicast stream that can touch every workstation could jump from network to network without controls, the entire Internet would quickly become saturated by such streams. Therefore, controls are necessary.

MBONE can control multicast packet distribution across the Internet in two ways:

- It can limit the lifetime of multicast packets, and
- It can use sophisticated pruning algorithms to adaptively restrict multicast transmission.

Responsible daily use of the MBONE network consists merely of making sure you do not overload your local or regional bandwidth capacity. MBONE protocol developers are experimenting with automatically pruning and grafting subtrees, but for the most part MBONE uses thresholds to truncate broadcasts to the leaf routers. The truncation is based on the setting for the time-to-live (ttl) field in a packet that is decremented each time the packet passes though an mrouter. A ttl value of 16 would limit multicast to a campus, as opposed to values of 127 or 255, which might send a multicast stream to every subnet on the MBONE (currently about 13 countries). A ttl field is sometimes decremented by large values under a global thresholding scheme provided to limit multicasts to sites and regions if desired.

These issues can have a major impact on network performance. For example, a default video stream consumes about 128 Kbps of bandwidth, or nearly 10 percent of a T1 line (a common site-to-site link on the Internet). Several simultaneous high-bandwidth sessions might easily saturate network links and routers. This problem is compounded by the fact that general-purpose workstation routers that MBONE typically uses are normally not as fast or robust as the dedicated hardware routers used in most of the Internet.

2. Internet Group Management Protocol

When a host on an MBONE-equipped subnet establishes or joins a common shared session, it announces that event via the Internet Group Management Protocol (IGMP). The mrouter on the subnet forwards that announcement to the other mrouters in the network. Groups are disbanded when everyone leaves, freeing up the IP multicast address for reuse. The routers occasionally poll hosts on the subnets to determine if any are still group members. If there is no reply by a host, the router stops advertising that host's group membership to the other multicast routers. MBONE routing protocols are still immature and their ongoing design is a central part of the MBONE experiment.

Most MBONE routers use the Distance Vector Multicast Routing Protocol (DVMRP), which some network researchers commonly consider inadequate for rapidly changing network topologies because routing information propagates too slowly [106]. DVMRP computes routes from every source to every receiver, building a set of shortest-path source trees for every sender in a group (see Figure 39). When group membership changes, the routes must be recomputed. MBONE is small enough that DVMRP routing is not a problem. However, some researchers speculate that, for a larger network with frequently changing group memberships, these routing techniques will be computationally inefficient.

The Open Shortest Path Working Group has proposed a Multicast extension to the Open Shortest Path link-state protocol that addresses this issue using an algorithm developed by Deering [98]. OSPF also must dynamically compute a source tree for each participant in a multicast group. Cisco Systems is supporting a protocol called Protocol Independent Multicast (PIM) which uses the concept of rendezvous points (RP) for "sparse" or shared trees [52].

A presumed advantage of using RPs is the ability to balance the costs of the communication among the senders in a multicast group through sharing a common multicast tree.

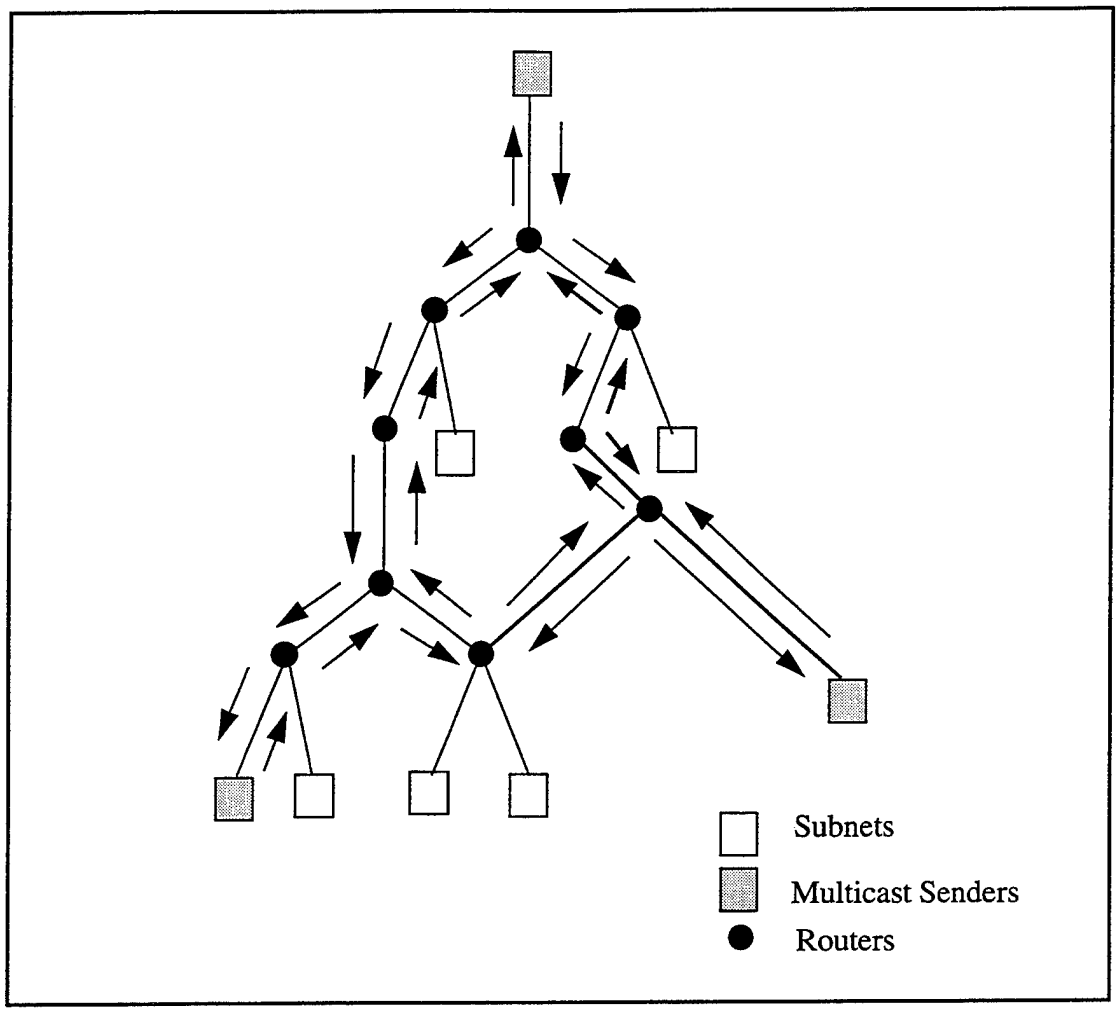


Figure 39. Source-based multicast trees.

The RP, for example, might be selected as the shortest-path center of the tree. For example, Figure 40 shows how a single multicast tree is formed from the "core". However, computing the best single-tree (called a Steiner tree) is an NP-Complete problem. Robert Voigt at NPS is conducting research in determining the best efficient method for selecting the RP or core router [154].

A critical issue is the stability of the multicast group. If members of the group leave and join often from diverse locations on the network, the multicast tree core will likely need to change as well. This raises concerns about the stability of the RP and the implication of

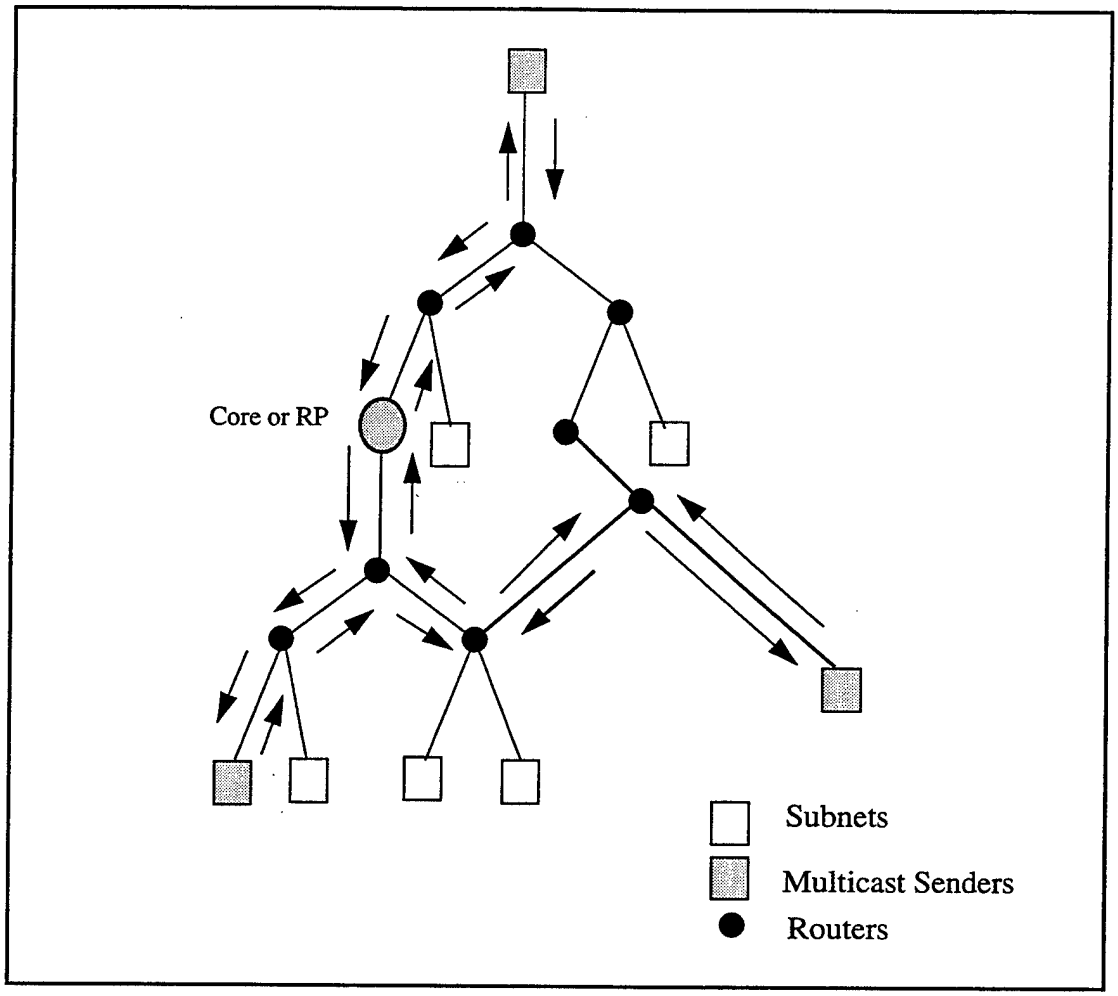


Figure 40. Core-based multicast trees.

this are currently unknown.*The implication for large-scale VEs is that we want to minimize group changes and if they occur, distribute those changes over time.*

However, Deering notes that scaling problems of the MBONE are implementation problems, not problems of the IP Multicast service model as is the case with the ST protocol used by the DSI [144]. (ST is sender-initiated and therefore does not scale well.) Deering believes that most of the solutions are known and that they will be invisible to hosts and multicast applications. For example, IGMP has had a low join latency but leave latency was

on the order of 5 minutes. Both join and leaves are accomplished in the same time that it takes for the receiving host to notify the upstream router (milliseconds) [46].

3. Topology and Event Scheduling

The MBONE community must manage the MBONE topology and the scheduling of multicast sessions to minimize congestion. By the beginning of 1995, some 1500 subnets and 25 countries were already connected worldwide, about the size of the entire Internet in 1990. Topology changes for new nodes are added by consensus: A new site announces itself to the MBONE mail list, and the nearest potential providers decide who can establish the most logical connection path to minimize regional Internet loading [44].

Scheduling MBONE events is handled similarly. Special programs are announced in advance on an MBONE event electronic mailing list. Advance announcements usually prevent overloaded scheduling of Internet-wide events and alert potential participants. Cooperation is key. Many people are surprised to learn that no single person or entity is "in charge" of either local topology changes or event scheduling.

4. Protocols

The magic of MBONE is that teleconferencing can be done in the hostile world of the Internet where variable packet delivery delays and limited bandwidth play havoc with applications that require some real-time guarantees. Limited experiments demonstrated the feasibility of audio over the ARPAnet as early as 1973. However, only a few years ago, transmitting video across the Internet was considered impossible. Development of effective multicast protocols disproved that widespread opinion.

The key network concepts that make MBONE possible are IP multicast and real-time stream delivery via adaptive receivers. For example, in addition to the multicast protocols, many MBONE applications are using the draft Real-Time Protocol on top of the User Da-

tagram Protocol and Internet Protocol. RTP [28], being developed by the Audio-Video Transport Working Group of the Internet Engineering Task Force, provides timing and sequencing services, permitting the application to adapt and smooth out network-induced latencies and errors.

Related real-time delivery schemes are also being evaluated. The end result is that even with a time-critical application such as an audio tool, participants normally perceive conversations as if they are in real time. This is because there is actually a small buffering delay to synchronize and resequence the arriving voice packets.

5. Data Compression

Other aspects of this research include the related needs to compress a variety of media and optionally provide privacy through encryption. Several techniques to reduce bandwidth include Joint Photographic Experts Group compression, wavelet-based encoding, and the ISO standard H.261 for video.

Encodings for audio include Pulse Coded Modulation (PCM) and Group Speciale Mobile (GSM) (the name of the standardization group for the European digital cellular telephony standard). Besides concerns for real-time delivery, audio is a difficult media for both MBONE and teleconferencing in general. This is because of the need to balance signal levels for all parties, who may have different audio processing hardware (for example, different microphones and amplifiers). Audio also generates lots of relatively small packets, which are the bane of network routers.

Compression is not generally an appropriate technique for DIS PDUs. Compression relies on extracting redundant symbols from a stream and encoding the symbols. However, DIS PDUs are generally small and discrete, causing the compression to be too much of an

overhead to be worth the effort and is penalized by latency if the information is extracted from multiple PDUs [135].

6. Application Tools

Besides basic networking technology, MBONE researchers are developing new applications that typify many of the goals associated with developing large-scale VEs. Session availability is dynamically announced using a tool called sd (session directory), which displays active multicast groups. Sd answers the question: How do I know what groups do I need to join and with what applications? The sd tool also launches multicast applications and automatically selects unused addresses and IP ports for any new group. Steve McCanne and Van Jacobson of the University of California Lawrence Berkeley Laboratory developed sd (Figure 41).

Video, audio, and a shared drawing whiteboard are the principal MBONE applications, provided by software packages called nv (net video), vat (visual audio tool), and wb (whiteboard). The principal authors of these tools are Ron Frederick of Xerox Palo Alto Research Center for nv, and McCanne and Jacobson for vat and wb (Figure 42).

Additional tools are also available or under development. Winston Dang of the University of Hawaii has created imm (Image Multicaster Client), a low-bandwidth image server. It typically provides live images of Earth from various geostationary satellites at half-hour intervals in either visible or infrared spectra. Henning Schulzrinne of AT&T/Bell Laboratories developed nevot, a network voice terminal providing multiple party conferences with a choice of transport protocols. Eve Schooler of the Information Sciences Institute is part of a team developing mmcc, a session orchestration tool and multimedia conference control program. Stephen Lau of SRI International is experimenting with using graphics workstation windows as image drivers. Kurt Lidl of UUnet Technologies, Falls Church, Virginia, is working on a network news distribution application that uses multicast

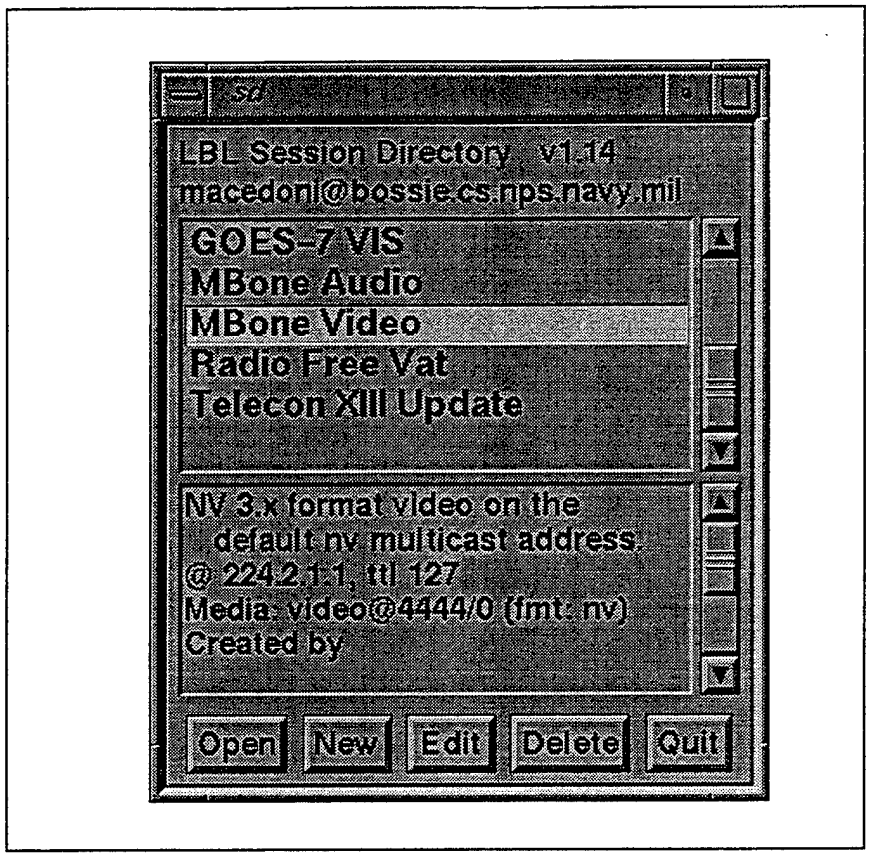


Figure 41. Session directory (sd) tool.

to reduce overall Internet loading and expedite news delivery. (Their goal is 120 ms total propagation coast to coast -- which is amazing since light takes about 16 ms to make that trip.)

B. Virtual Environment Research with Multicast

Some researchers have proposed different ideas about using multicast to support virtual environments.The partitioning of virtual worlds into spaces is a common metaphor for VEs. Multi-User Dungeons (MUDs) have used this idea and projects like *Jupiter* from Xerox PARC have extended this to associating "rooms" with multicast video and audio teleconferences [42]. Also at Xerox, Schilit and Theimer have developed an active map service (AMS) that publishes the location of objects in a region using dynamic multicast groups

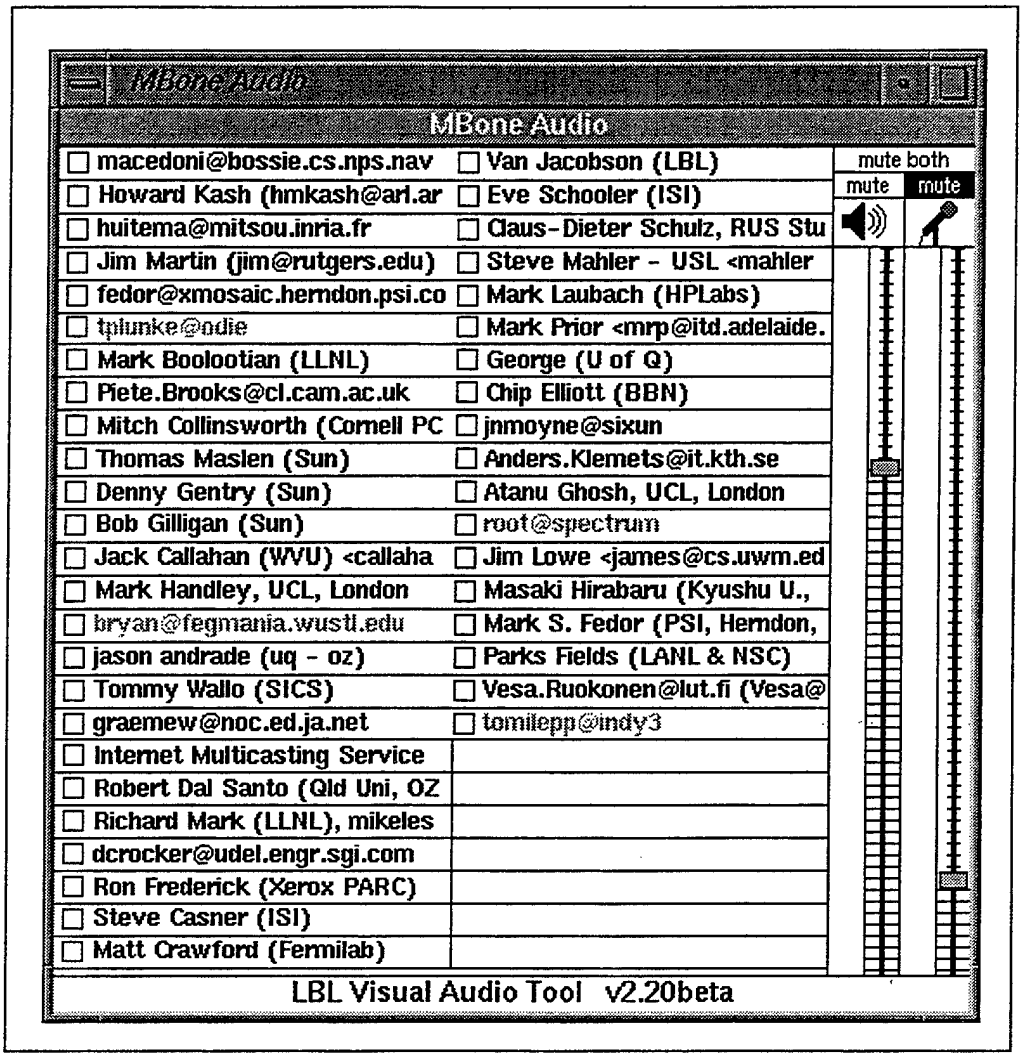


Figure 42. Visual audio tool (vat).

associated with different parts of the region. For example, the system can track persons in a building via the use of active badges. Using multicast for updates reduces aggregate message traffic [124].

Lockheed has developed a similar concept for spatial partitioning that assumes the use of ATM multicast "channels", i.e.,mapping relevant groups to ATM's Virtual Channel Identifiers [95]. Though ATM multicast technology is not yet mature (few vendors support it), these ideas present exciting possibilities.

Benford has described a concept for the spatial interaction of objects in a large-scale VE [13]. The spatial model uses different levels of awareness between objects based on their relative distance and mediated through a negotiation mechanism. An implementation using the reliable multicast version of DIVE uses "standard VR collision detection" to determine when the transitions between awareness levels should occur [24]. Van Hook has discussed a similar idea called object-based filtering in which a subscription agent informs entities that they are in range of each other. The MASSIVE project also uses this approach. However, the need for this type of collision detection, reliable communication, and strong data consistency have made it difficult for DIVE and MASSIVE to scale beyond a handful of users [13]. This may be changing as their developers pursue the use of multicast communications and weaker data consistency.

Others have suggested using multicast for DIS but, very few have actually conducted research or implemented VEs using multicast communications. SRI recommended multicast in an early 1990 White Paper and it has been recommended for IEEE 1272 standards group [135]. Van Hook also examined the idea of grid geographic filtering in which multicast groups are associated with square regions of terrain. He did an early study suggesting that an 80% reduction in traffic could be achieved using a five km square grid and 168 active multicast groups [20].

Van Hook has also proposed using a combination of grid-filtering to reduce the computational requirement of object filtering, an $O(n^2)$ operation [153]. Van Hook also suggested the idea of on-demand forwarding in which entities would send a low-rate broadcast with terse state information. Each receiver would compute a range check and send state data to the visible entities. However, object-filtering and on-demand forwarding essentially establish a multicast group for every receiver. For example, in Figure 42, Entity 1 and 2

join each other's multicast groups. Entity 3 is outside the range of 1 and 2 and therefore is a member of only its own group.

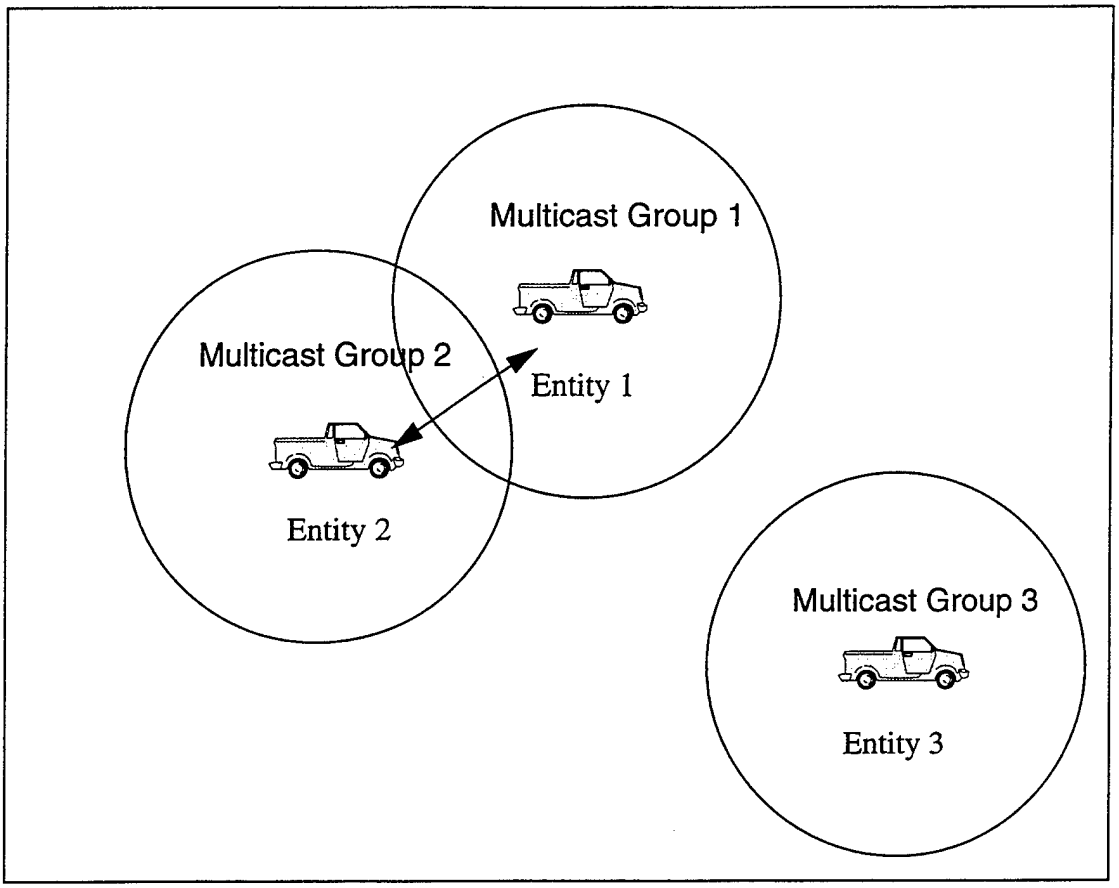


Figure 43. Object-based filtering.

Until 1994, there was reluctance in the DIS community to use IP Multicast because of ARPA support for other technologies such as ST and the DSI, the lack of a software architecture and algorithms that could exploit it, and limited hardware support. Van Hook and his team from MIT Lincoln Labs had also devoted much energy to building the Application Gateway for STOW.

The status of IP Multicast in the DIS world has changed. For example, Steve Batsell at NRL and Mark Pullen at GMU (and former Defense Simulation Internet manager) have

suggested using a two-level architecture using IP Multicast mapping to ATM multicast facilities [12,111]. They are using NPSNET to explore the use of multicast for DIS.

C. NPSNET AND MBONE

We have used MBONE to demonstrate the feasibility of IP Multicast for distributed simulations over a wide area network. In the past, participation with other sites required prior coordination for reserving bandwidth on the Defense Simulations Internet (DSI). DSI, funded by ARPA, is a private line network composed of T-1 (1.5 Mbps) links, BBN switches and gateways using the ST-II network protocol. It has been necessary to use DSI because ARPA sponsored DIS simulations used IP broadcast - requiring a unique wide area bridged network.

With the inclusion of IP Multicast, sites connected via the MBONE can immediately participate in a simulation. Sd is also used for launching multicast applications like NPSNET-IV and for automatically selecting an unused address for the new group. Furthermore, we can integrate other multicast services, such as video with NPSNET-IV. For example, participants are able to view each other's simulation with a video tool, nv.

After SIGGRAPH 93, we completed the multicast version of NPSNET-IV. After some initial tests over the Naval Postgraduate School campus network, we began experimentation over the MBONE with the help of Stephen Lau at SRI. Communicating between SRI and NPS presents a significant challenge for interactive simulation. SRI, which is on DARTNET, and NPS, attached to BARRNET, are separated by six routers with a variable delay of between 100 to 1000 ms. Figure 44 shows the routing over the MBONE. The annotations indicate physical or tunneled links and the metric associated with the links. For example, P1 indicates that a link is a direct multicast link with its metric (cost value used by the routing algorithm) set to 1. Moreover, DIS traffic must compete with MBONE video and audio multicasts. Figure 45 shows the PDU traffic generated across the MBONE between two SGI workstations running NPSNET-IV at NPS and SRI.

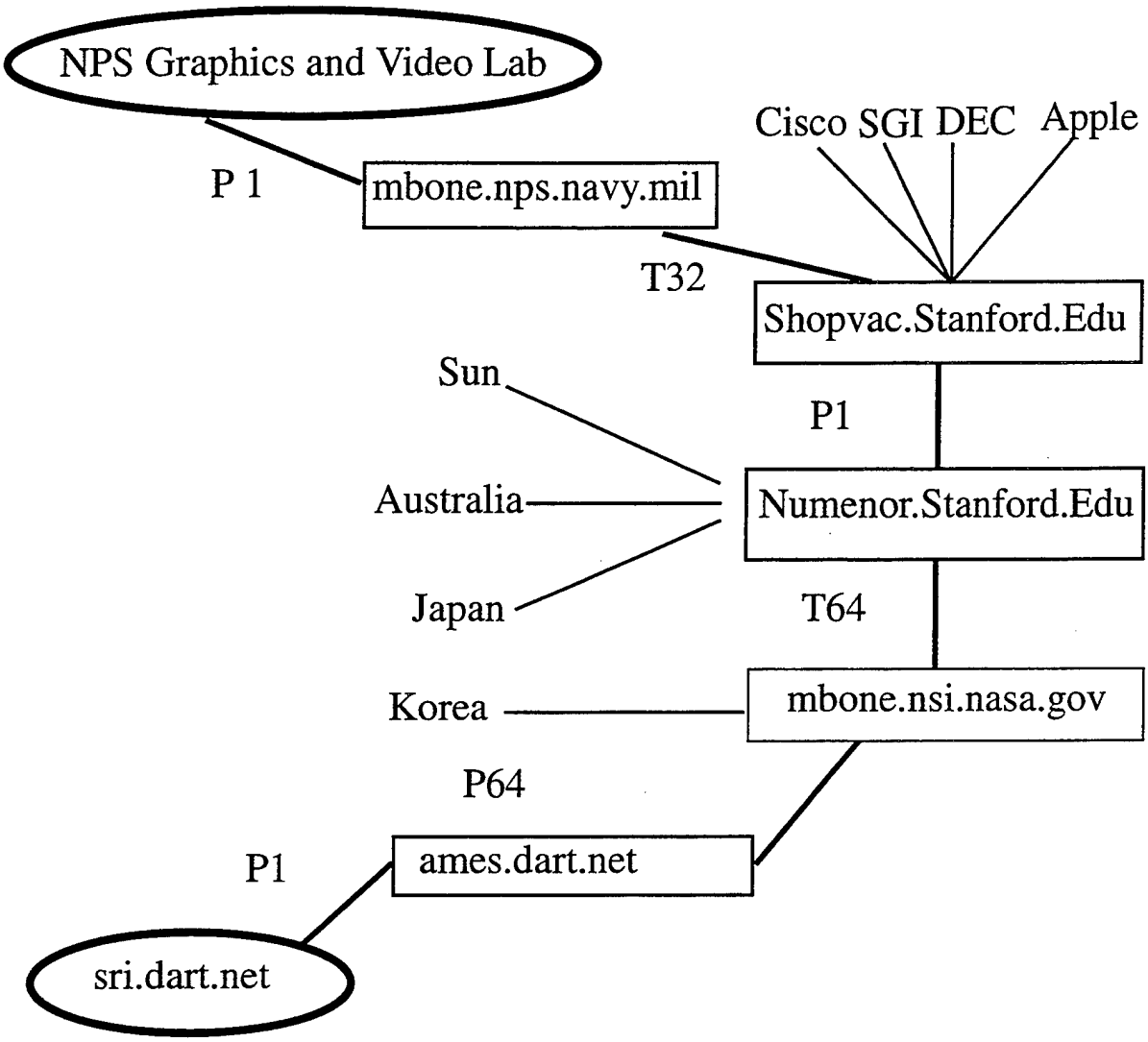


Figure 44. MBONE routing between NPS and SRI.

Despite a hostile network environment, NPSNET-IV showed little perceptual latency. We used Lau's version of a multicast video tool to transmit images of his simulation running on an SGI Onyx to the players at NPS. We could observe what was being viewed at SRI, including weapons firing on our aircraft.

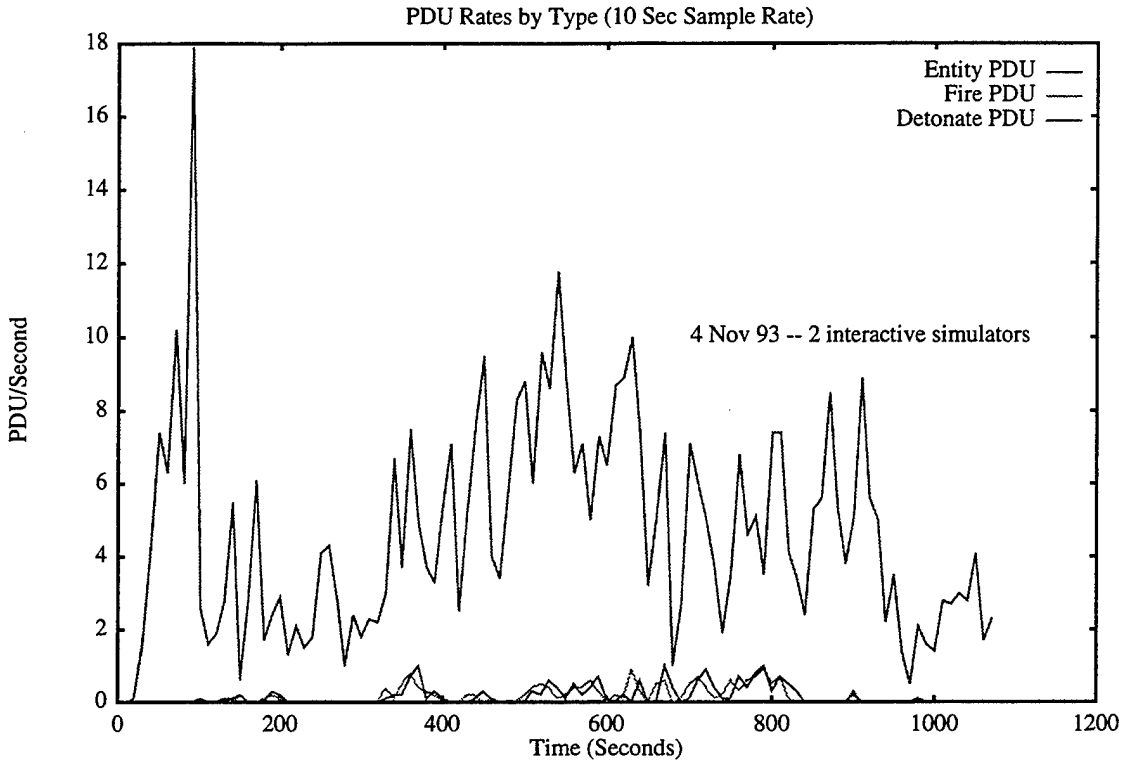


Figure 45. IP Multicast between SRI and NPS.

D. SYMPOSIUM ON 3D INTERACTIVE GRAPHICS

We continued testing involving multiple sites including the Rand Corporation, TRAC-Monterey, NRL, and GMU. In April 1995 we conducted a demonstration of NPSNET for the ACM 1995 Symposium on 3D Interactive Graphics using the MBONE.

The demonstration involved participants at MIT in Cambridge, Massachusetts, Navy Research Labs in Washington, D.C, George Mason University in Virginia, Sprint in Kansas City, SRI in Menlo Park California, and at the Naval Postgraduate School. Figure 37 shows the relative location of the participants. Remote sites simulated Apache and Hind helicopters, M1 and M2 armored vehicles, and F16 fighter, and a dismounted infantryman. Modsaf was used at NPS to simulate blimps.

The actual site of the conference was at the Hyatt Hotel in Monterey. We used a wireless Ethernet bridge at the Hyatt and Spanagel Hall at NPS to connect the local area network

we had constructed at the hotel with the MBONE. The bridge provided about 1.5 Mbps bandwidth.

We attempted to collect data regarding the network performance of the multicast session. However, we were thwarted for a number of reasons. First, we tried to gather topology information regarding the source tree of the simulation from the Hyatt using a tool called mtrace. mtrace utilizes a tracing feature implemented in multicast routers (mrouted version 3.3 and later) that is accessed via an extension to the IGMP protocol [30]. Unfortunately, an upstream router at BARRNET had not been updated with the latest release of the mroute daemon, truncating the search of the tree.

We also used a multicast version of the ping utility to get route and round trip time information. The tool sends messages to the multicast routers and receivers and attempts to record the routes, which are returned as a result of Internet Control Message Protocol (ICMP) packets. Most of the information from ping proved useless. The majority of messages indicated that duplication errors had occurred on the MBONE. We speculate that since ping packets are flooded across the MBONE, DVMRP routers would discover packets coming from multiple interfaces resulting in returned duplication messages. However, routes from NRL were returned consistently:

Our host -->

--> cs-mbone-router.cs.nps.navy.mil

--> mbone.nps.navy.mil

--> Utuma.BARRNET.NET

--> Numenor.BARRNET.NET

--> VINEGAR.SESQUI.NET

--> mae-bone.psi.net

--> iguana-fO.nrl.navy.mil

--> roosevelt.ait.nrl.navy.mil

At least seven hops were involved and the mbone was heavily utilized with at least three concurrent video and audio sessions. The round trip time was 1245 ms or approximately 625 ms network latency. Lag was noticeable when observing the NRL icon from NPSNET. However, other sites like GMU and Sprint were quite responsive. During earlier tests we consistently measured latency of less than 100 ms to GMU.

We are now examining better methods for instrumenting these experiments. However, this test highlighted some of the current problems of experimentation over the MBONE. We need better tools. Furthermore, we are at the mercy of network operators to have properly configured routers because the MBONE is a volunteer and cooperative effort. It is also difficult to recruit assistance from more than a handful of remote sites that have the appropriate connectivity, advanced graphics hardware, and network knowledge.

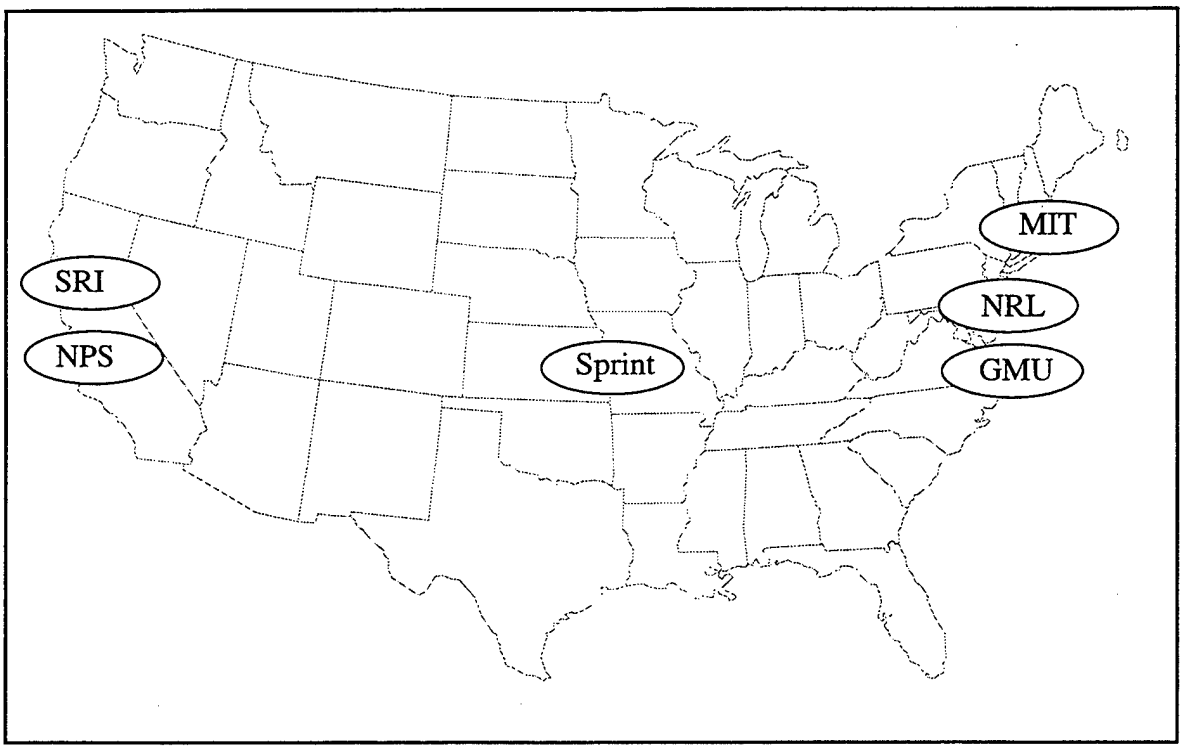


Figure 46. Location of participants.

E. SUMMARY

In the chapter we described the future network infrastructure for large-scale VEs, IP Multicast, and the Internet MBONE.We also discussed the advantages of using IP Multicast, some research related to VEs using multicast, and some of our efforts with the current multicast version of NPSNET.

VII. THEORY

A. GOALS

In this chapter, we discuss the specific goals of our architecture for construction of large-scale VEs, the heuristics employed, the aspects of real-world military environments that we can exploit, and the theory of an Area of Interest Manager that employs these principles. In the next chapter, we show the results of our experiments that demonstrate the effective of this approach.

Increasing the number of entities by more than two orders of magnitude requires us to think beyond experiences with small-scale SIMNET and DIS simulations. We believe that it is incorrect to strictly extrapolate the DIS experience (or any of the current research VEs) to large-scale VEs. However, based on the previous work discussed in this dissertation, we have the following design goals for achieving a scalable VE architecture:

1. Computational and Bandwidth Requirements

We need to reduce computational and bandwidth requirements for entity hosts while minimizing latency. The architecture must place a bound on the number of entities with which an entity communicates and maintains state. The broadcast communications model is inappropriate for this as demonstrated by the DIS experience.

2. Maintain the Current DIS Semantics

We desire to maintain the current DIS semantics because they are firmly established in the IEEE 1278 protocol, are widely used in simulators, and appropriate for many VEs. However, our architecture should not preclude changes such as the adoption of software agents (e.g., Safe-Tcl).

3. Reduced Latency for New Entrant Learning

Our architecture must minimize the time it takes for a new entrant to a VE to learn about the current state of the world while eliminating "heartbeat" messages.

4. Fully Distributed

All or at least most of the VE processing and algorithms should be distributed to client hosts to prevent computational or communications bottlenecks (e.g., Application Gateways).

5. Elimination of the Static and Dead Entity Problem

The VE should provide a persistent object protocol that allows active entities to identify static or dead entities without the use of "heartbeats".

6. Localization of Reliability Problems

An errant sender on a broadcast network can "jam" other senders, effectively bringing down an entire VE. This becomes likely for a VE with thousands of participants. Therefore, the VE must be constructed to minimize this case.

B. HEURISTICS

In examining previous work related to network VEs we developed several heuristics to deal with the scalability problem:

1. Domain Specificity

Large VEs architectures will be domain specific. By this we mean that it is difficult to construct general-purpose software architectures to support all the requirements of every domain. As we discussed previously with regard to DIS, different domains or environments demand and permit different levels of abstraction.

We can share a variety of techniques (e.g., relevance partitioning) but they will not necessarily present a general solution to all cases. Many examples abound of VEs with different demands and abstractions. A textual-based, client-server MUD may be quite appropriate if the purpose of the VE is to simply teach literacy though it can be scaled relatively easily. It is difficult to represent to a distributed group, even within the military domain, many environmental changes such as the creation of a tank ditch represented graphically as thousands of polygons in real-time but quite practical for a stand-alone simulator. On the other hand, a graphical distributed 3D VE designed to simulate tank warfare for training (e.g., CCTT) can have significantly less demands than one used to model the Pacific Ocean's weather in real-time or the hallucinations of William Gibson's *Neuromancer*.

We have limited ourselves to scaling the "traditional" SIMNET/DIS military training application by reducing the individual entity bandwidth and computational requirements. Though the architecture may serve to solve other problems such as environmental changes, our primary goal is to make current military interactive simulations capable of supporting thousands of concurrent players.

2. Behavior Characterization

An understanding of the behavior of the communicating entities is required to provide efficient architectures. This is a critical issue because it is entity actions (or inactions) that determine the demand for computational and communication resources. For example, the telephone companies for a century have measured call arrivals and holding times, developing queuing models to predict switch capacities.

The need to model a specific reality enforces the previous point. Since most of the "players" in a military VE represent the behavior of vehicles under the control of human actors in organizations it is necessary to have some idea of what the typical behavior of those organizations should be (e.g., the density of vehicles). This behavior translates into the degree of temporal and spatial coherence that exists within the VE.

Many assumptions have already been made about entity behavior in the DIS community. In this chapter we show some historical data that describes the behavior of large military organizations on the battlefield. In the next chapter, we use data from a large scale military exercises in conjunction with our heuristics that supports our work.

3. Partitioning

Large VEs must be distributed and logically partitioned so as to take advantage of temporal and geometric coherence. As we have discussed several times already, partitioning is imperative in order to make large-scale VEs tractable. This principle has been practiced by the graphics community, which for the last several years has made enormous advances in rendering speeds by partitioning graphic data bases noting that an observer has a limited view of the world [1]. For example, Manocha and Lin have developed an algorithm for collision detection that uses fixed sized bounding volumes to rule out collisions that are far apart and then use exact collision detection on the assumption that the objects are not moving swiftly [39].

As the previous chapter noted, the wide-area network infrastructure for large VEs is becoming a reality. We also showed that multicast networks like the MBONE provide a mechanism to partition group communication and, therefore, partition the virtual world for distributed processing.

Some servers will be needed for the multicast groups themselves. We use client-server relationships and the latency afforded by group changes as a way to provide object persistence and eliminate heartbeats. Additionally, some centralized services are required. For example, systems management and security control is necessary in the military domain and the commercial world may need facilities for billing. However, algorithms and processing should be distributed as much as possible to avoid bottlenecks as in the case of MUDs and with the DIS Application Gateway.

4. Real-time Efficiency

The algorithms and techniques that support the distributed VE architectures must be computationally efficient in order for VEs to operate in real-time -- at the expense of realism and generalization. For example, NPSNET has smoke with a realistic *appearance* but uses a simplistic particle model in order to permit display at 30 Hz on an SGI Reality Engine. However, more realistic models are available that display a frame every 15 minutes on a Cray-YMP [41].

5. Communications Model

VEs must have some measure of data consistency using reliable communication, but unreliable communications will predominate. A real-time system for a large number of players is not amenable to a distributed shared memory architecture -- though convenient to the programmer -- because such as system imposes too much latency because of the demand for reliable communication between different hosts.

Real-time messages require unreliable communications but large VEs need reliable mechanisms to reduce or eliminate heartbeat messages and the new entrant learning problem. Moreover, reliable communications is needed for database and model replication.

We conjecture that a large-scale real-time VE cannot guarantee strong data consistency and reliable communication among all its participants simultaneously. Instead, four types of communication can be established which, used together, allow stronger consistency than simply broadcasting state messages. They provide for a much richer world through a mechanism for sending large objects reliably and supporting VE partitioning.

In our model there exists four methods for communication within the context of VEs:

a. Light-weight interactions

These messages are composed of the same state, event, and control PDUs used in the DIS paradigm but implemented with multicast. They are light-weight because the complete semantics of the message are encapsulated in the maximum transfer unit (MTU) of the underlying data link to permit asynchronous real-time interactive use. Therefore, these PDUs (e.g.,ESPDUs) are not segmented. They are either received completely or not at all because they are communicated via connectionless and unreliable (unacknowledged data) networks. The MTU for Ethernet is 1500 bytes and 296 bytes for 9600 point-to-point (PPP) links [138].

b. Network pointers

Proposed are light-weight references to resources, in a similar way to Uniform Resource Identifier (URI) as defined in the Hypertext Transfer Protocol (HTTP) [16]. Pointers are multicast to the group so that they can be cached by members. Therefore, common queries need not be resent and the server can direct the responses to other members of the group. We make a distinction between pointers and light-weight interactions because they do not completely contain an object but rather its reference. Pointers provide a powerful mechanism for referencing not only the current aggregate state of the group but also terrain, model geometry, and entity behaviors defined by a scripting

language. In the context of the World Wide Web, network pointers have revolutionized Internet communication.

c. Heavy-weight objects

These objects require reliable, connection-oriented communication. For example, an entity may require model geometry after joining a group that does not exist in its database. The entity would multicast a request for the geometry and the response would be a multicast pointer to the source. If efforts such as the Virtual Reality Modeling Language (VRML) are successful, heterogeneous systems may be able to exchange this type of information [107].

d. Real-time streams

Video and audio traffic provide continuous streams of data that require real-time delivery, sequencing and synchronization. Moreover, these streams will be long-lasting, persisting from several seconds to days. They are multicast on a particular "channel" to a functional class. In contrast with the current DIS protocol, we propose the use of pointers to direct entities to these channels rather than, for example, forcing the VE, which may be as simple as a text-based application, to receive both light-weight DIS PDUs as well as video streams. Moreover, the VE can spawn a separate process which incorporates an adaptive receiver and which separates the handling of bursty simulation message from real-time streams.

e. Summary

In our communications model for VEs we recognize the different requirements for large-scale VEs. This DIS model, on the other hand, has attempted put all data within the same simple scheme -- unreliable, broadcast communication. However, the complexity

of interactions and the requirement for a scalable architecture also require an architecture that uses different methods for communication among entities.

C. THE BATTLEFIELD ENVIRONMENT

After arriving at these principles and investigating the *scalability problem,* we realized that we might take advantage of our knowledge about real military combat which our NPS-NET virtual environment attempts to emulate.

1. Limited Entity Area of Interest

Military entities have a limited area of interest that varies with the capabilities of their sensors, environment, and relationship to other objects in the world. One way we can define this area of interest is spatially. Entities can be associated with particular areas or volumes in the VE. Entities whose spatial interest are associated with the same cell can be considered part of the same *spatial class or partition.*

We can take advantage of the fact that humans on the ground can only see a limited distance because views are obstructed by terrain, foliage, or buildings. Some VE systems take this into account in order to reduce processing requirements. We have found that Modsaf limits intervisibility checks to less than 3500 meters from simulated entities [85].

For most likely scenarios, the value used by Modsaf is appropriate. Table 4 shows in meters a variety of intervisibility values for different types of military terrain based on expected defensive scenarios for ground combat units. In-view is the distance travelled by an attacker in which the attacker is visible. Out-of-View is the opposite. First Open is the dis-

Location	In-View	Out-of-View	First Open	Expected Open	Expected PLOS
Fulda Gap, GE	553	761	2201	1561	.398
Quasrod Dasht, Iran	878	703	2776	2024	.547
South Korea	452	1008	1864	1484	.311
Ft. Hunter Ligget, CA	507	1158	1551	1214	.286
Ft. Irwin, CA	741	535	3052	1973	.579
Fort Hood TX	611	1554	1332	1093	.279
Yakima, WA	1238	429	3508	2434	.748

Table 4: Military intervisibility values in meters[159].

tance at which the attacker becomes visible. In the table, Expected Open is the mean range at which the defender obtains line-of-site visibility of the attacker. Probability Line of Sight (PLOS) is the likelihood that intervisibility exists between a defender and attacker. *These values are all less than 4 km* [159]. Ft. Irwin, the home of the National Training Center, has steep mountains surrounding large desert valleys that provide excellent intervisibility for defenders. The NTC is a major site for STOW 97 [112].

Entities also may belong to a *functional class* in which an entity may communicate with a subset of entities. Soldiers communicate hierarchically on broadcast radio networks. For example, each company in a battalion has its own frequency assigned for internal communicating. Therefore, simulated radio traffic should be restricted only to the interested parties of the group as is the case with teleconferencing on the MBONE. In Figure 47, it is

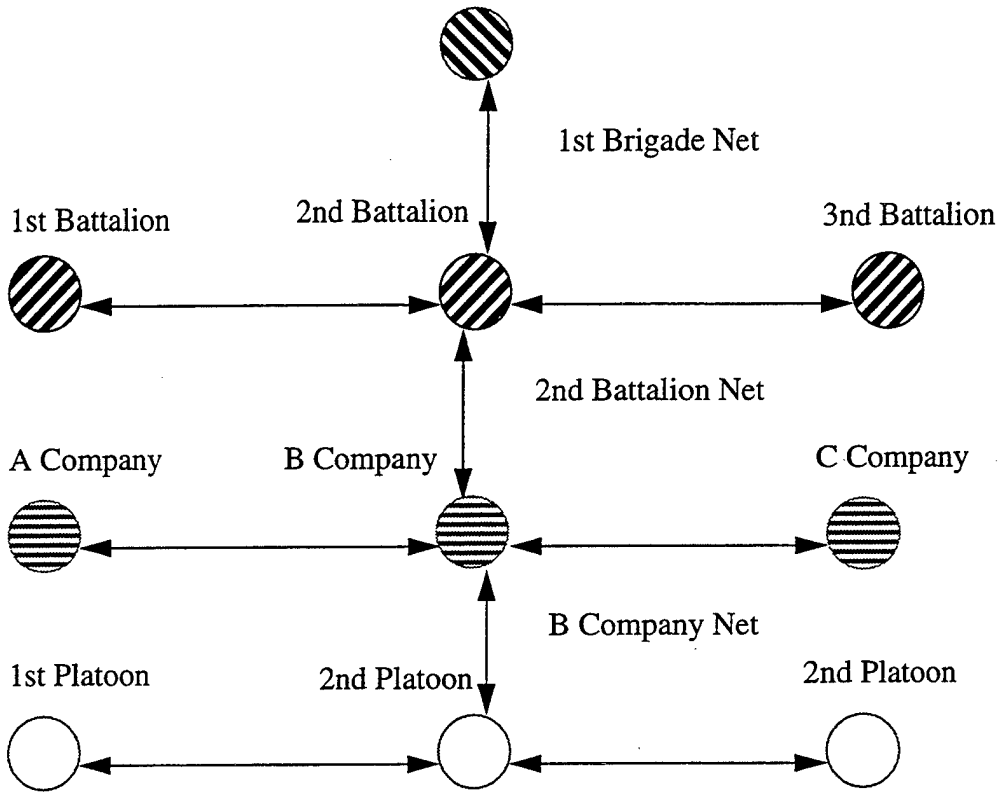


Figure 47. Military communication networks.

obvious to see that military communications is partitioned by organization and relationship. We can take advantage of that by mapping radio channels to multicast groups, e.g. the B company, 2nd Battalion, 1st Brigade Net --> 63.25 MHz --> 224.35.67.12.

Other types of functional classes could be related to system management or services such as time. (The Network Time Protocol has already been assigned a well-know IP Multicast address by the Internet address authority.) Another example of a functional class in the military domain would be a VE "air control" group. The group would include entities that are primarily concerned with entities or events occurring in the air. Therefore, air defense and aircraft entities would comprise the majority of the group. Aircraft and air defense systems are relatively sparse in the whole as compared to other combat systems such as tanks. Air defense systems would also belong to a small subset of the spatial class. Aircraft which are interested in a particular area of ground can "focus" and join a spatial group associated with its area of interest.

Finally, entities can belong to a *temporal class*. Entities have different real-time communication requirements. Tank entities simulating combat need to communicate in real-time with respect to human perception but a VE system management agent might only need updates every several minutes. Similarly, a simulator of a space-borne sensor only needs a general awareness of ground vehicle entities and therefore can accept low-resolution updates. When there is a need for more resolution, the simulator, like aircraft entities, can focus and become part of a spatial group.

2. Aggregate Behavior

The aggregate behavior of military entities changes with organizational size, scope and mission. For example, we know that for large units (e.g. a divisions and corps) in the real world, battles have a relatively low density of combat systems compared to small units such as platoons and companies. The median historical density for all armies is 82 weapon systems per kilometer of *front* [93]. However, as shown in Figure 48 which depicts a US division on Central European terrain, smaller units like battalions are usually organized in a linear front while the division itself extends in depth with large gaps among units.

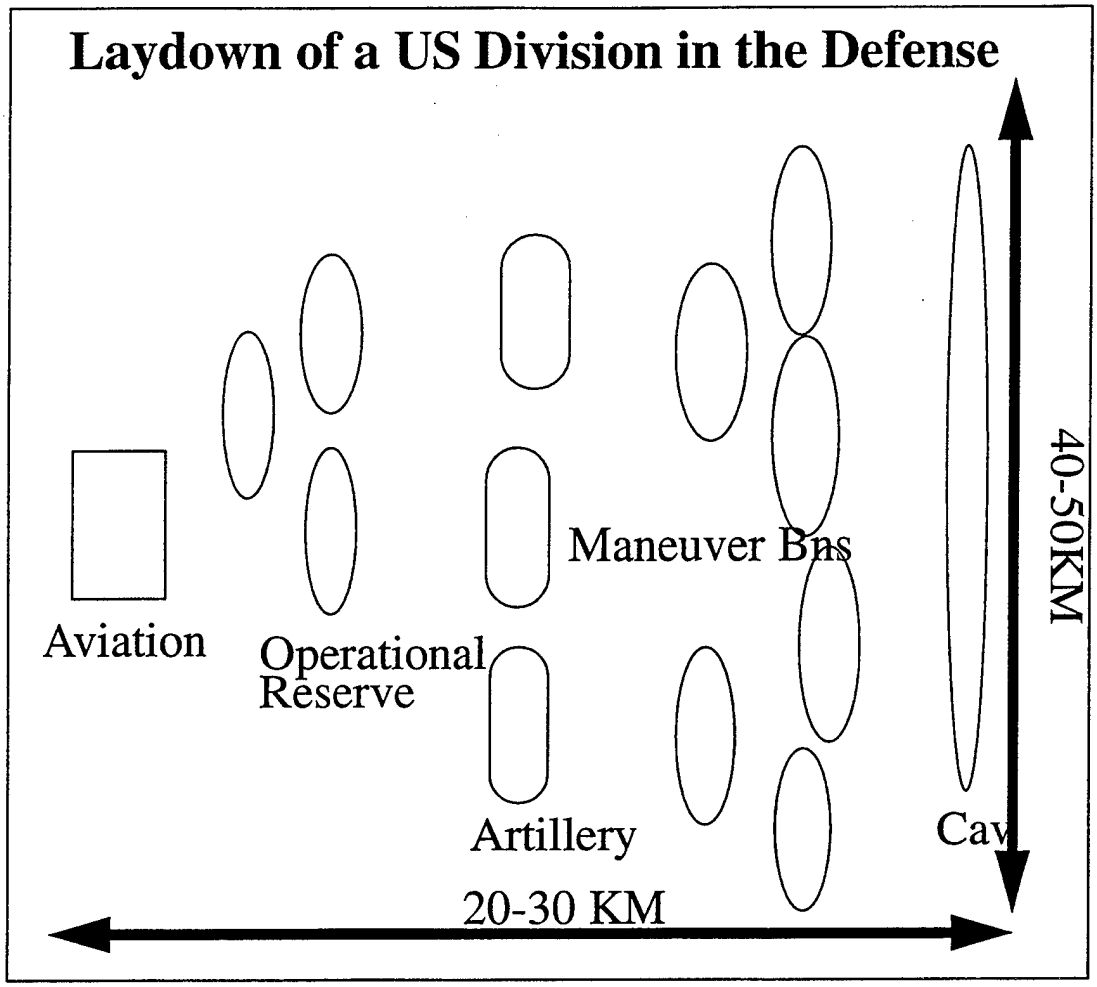


Figure 48. US division in the defense.

A conservative estimate for the width and depth of a United States heavy division in the defense is 40x20 km or 800 km^2. Using McQuie's numbers we can derive the density of systems [93]:

82 weapons systems/km x 40 km = 3280 weapon systems

3280 weapons systems/800 km^2 = **4.1 weapon systems/km^2**

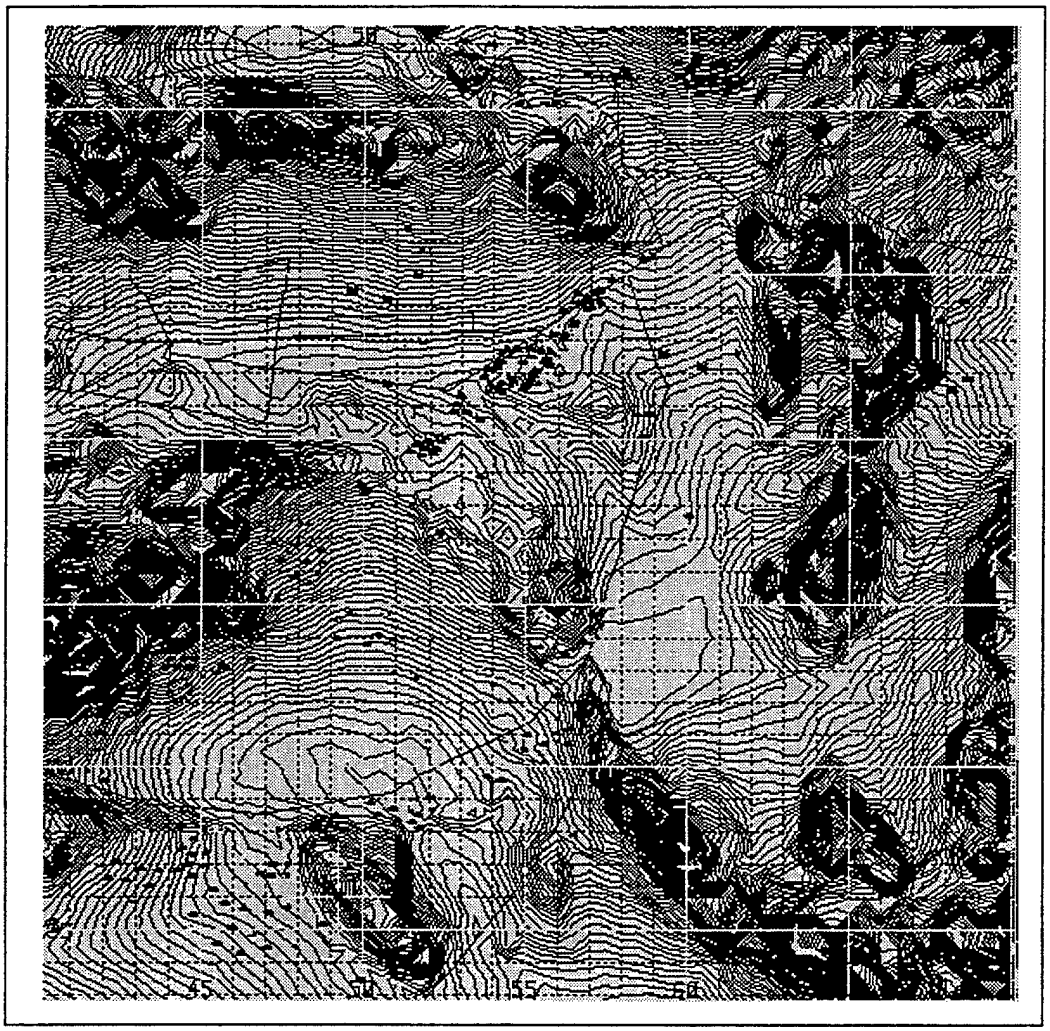


Figure 49. Battlefield at NTC.

Another way to calculate densities is to observe that a modern heavy U.S. division has approximately 1500 weapon systems (excluding machine guns and small arms) for a density of **2 weapons systems/km^2**. Aircraft have even lower densities. During Desert Storm, roughly 1000 aircraft were in the air over Iraq with an average density of 1 aircraft per 400 km^2. Obviously some areas of the battlefield are much more congested than what these

numbers suggest. *However, these average densities imply that many parts of the battlefield, and hence our large-scale virtual world, at any particular time, are largely unoccupied.* This is shown in Figure 49 which depicts a brigade-size force in the attack at the National Training Center, Ft. Irwin, CA. The area is 60 x 50 km. Friendly vehicles are blue; enemy vehicles are red. (Note that early DIS analyses used 10,000 vehicles in a similar sized area).

These areas remain unoccupied for long periods because ground systems do not advance very fast or often relative to the size of the battlefield. Helmbold in his study on the rates of advance for land operation found that they are not predictable [65]. Furthermore, he determined that land combat operations stand still 90-99% of the time. However, we do know modern-day limits. The slowest modern US advance was 100 m per day at Okinawa and that the world's record in modern warfare was 92 km per day for 4 days by the 24th Mechanized Infantry Division in Desert Storm [65,50]. Assuming that the division moved constantly for 16 hours per day this rate of advance translates to 5.8 km per hour. Individual vehicles may move much faster but, they would not continue at high rates very long because they fight as part of units in which movement must be coordinated. *With respect to the military domain, group membership within a spatial class should change relatively slowly with respect to other state changes or events.*

Related to density (or what Depuy calls the dispersion factor) is intervisibility. As intervisibility increases, military tactics dictate that systems become more dispersed to reduce vulnerability from indirect and direct fire weapons [48]. Moreover, in terrain where intervisibility is diminished (valleys, jungle) units cluster together because the ranges of direct fire weapons are reduced and command and control becomes problematic.

D. DIS AREA OF INTEREST MANAGER

To take advantage of the principles outlined above and the real world attributes of large military environments, we propose the use of a software "glue" or middleware between the DIS event and state PDU paradigm and the network layers that is wedded to that reality. The area of interest manager (AOIM) partitions the VE into a set of workable, small scale environments or classes to reduce the computational load on hosts, minimize communications on network tail links, and localize reliability problems. Furthermore, the AOIM exists with every simulator to distribute partitioning processing among hosts.

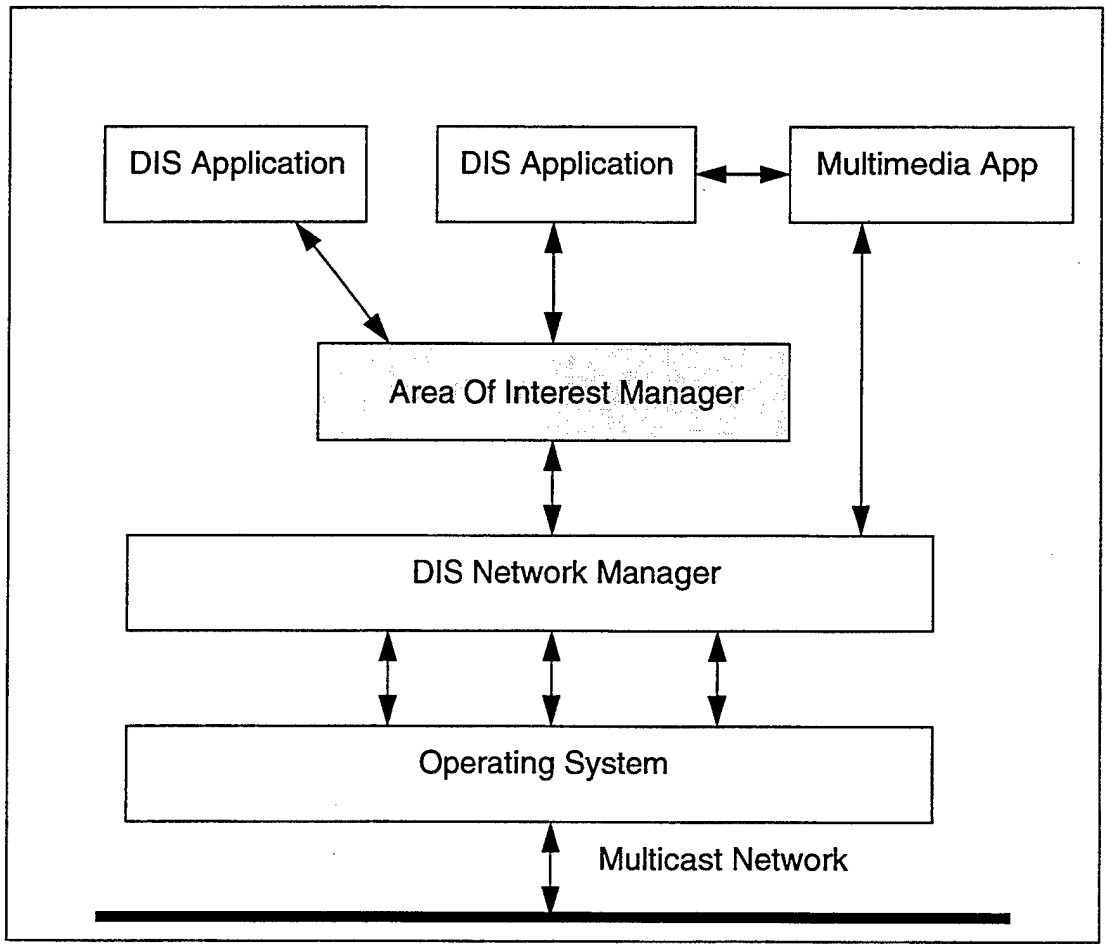


Figure 50. Area of Interest Manager.

The AOIM uses spatial, temporal, and functional classes for establishing membership in multicast network groups. From the perspective of the AOIM, IP Multicast allows the creation of transient multicast groups that can be associated with an entity's area of interest (AOI). Therefore, a host communicates only with entities which are relevant to its perception of the VE. Additionally, multiplexing and demultiplexing is done at the network level.

This naturally provides a way of separating classes of traffic such as audio, video and simulation data. For example, the radio communications functional class would be mapped to a particular multicast group address or "channel".

1. Spatial Associations

As noted before, some researchers have suggested using simple collision detection to associate entities. However, this requires some service to provide the mapping using $O(n^2)$ complexity. In addition, a multicast address for each entity is presumed. Though there are currently 2^{28} IP Multicast and IP version 6 will have a vastly increased address space, we are concerned that it will be difficult for network routers to handle 100,000 active dynamic multicast routes simultaneously. To illustrate our idea, we examine using the AOIM to associate spatial classes with multicast addresses. We partition the VE with fixed-sized hexagonal cells.

More formally, if we use a 2-dimensional hexagonal grid system we can define S, the set of all grids or cells, $A_{i,j}$,where i and j are the integer coordinates of the area. Let $\{A_{i,j} | ((i,j) \in I)\}$ and $\bigcup_{i,j \in I} A_{i,j} = S$. We can have a function, w, that maps the set of entities to a set of a hex cells (from their Cartesian coordinates), $E \rightarrow S$, such that $w(e_{k \in I}) = A_{i,j}$. It is a partition because the relation, R, is an equivalence relation.

Proof. Let R be the relation on S where (x,y) belong to the same subset, $A_{i,j}$. We see that $(w(e), w(e)) \in R$ for every $e \in E$, since w(e) is a subset of itself. Hence R is reflexive. If $(w(e), w(f)) \in R$ then w(e) and w(f) are in the same subset of the partition, so that $(w(f), w(e)) \in R$. Hence R is symmetric. If $(w(e), w(f)) \in R$ and $(w(f), w(g)) \in R$, w(e) and w(f) are in the same partition, X, and w(f) and w(g) are in the same subset of the partition, Y. Since the subsets of the partition are disjoint, and w(f) belongs to X and Y, it follows that X = Y. Consequently, w(e) and w(g) belong to the same subset of the partition, so that $(w(e), w(g)) \in R$. Thus, R is transitive.

We then map S to M, the set of IP Multicast addresses:

$$E \rightarrow S \rightarrow M \qquad \text{Equation 1}$$

For example, in application, we take the cartesian coordinates, convert them to hex coordinates, and then map them to a multicast address:

$$82.5,\ 101.2 \rightarrow 2,\ 4 \rightarrow 224,\ 003,\ 002,\ 004$$

2. Hexagonal Tessellation

We borrow the use of hexagons from the mobile telecommunications and the military gaming world. Hexagonal tessellations have a number of advantages. Samet notes that hexagons are regular, have a uniform orientation, and have uniform adjacency [118]. Hexagons have been traditionally used for war games because they permit movement along six axes as opposed to only four with squares.

Cellular telephony received its name from the use of hexagonal cells in the design of mobile telephone systems. MacDonald, in his classic paper on the cellular concept, described Bell Laboratories choice of hexagons for cellular geometry:

Although propagation considerations recommend the circle as a cell shape, the circle is impractical for design purposes, because an array of circular cell produces ambiguous areas which are contained either in no cell or in multiple cells. On the other hand, any regular polygon approximates the shape of a circle and three types, the equilateral triangle, the square, and the regular hexagon, can cover a plane with no gaps or overlaps. A cellular system could be designed with square or equilateral triangles, *but,... to serve a given total coverage area, a hexagonal layout requires fewer cells* [86, p. 15].

We use an algorithm developed by the Joint Propulsion Laboratory to convert an entity's Cartesian coordinates to hex coordinates in constant time. The algorithm was originally used for satellite image processing [116]. We initially tessellate the region with rectangular cells whose northwest and southeastern vertices correspond to the center of the hexagons. We compute through translation which cell a given cartesian point is in. We can then determine two candidate hexes. The cartesian coordinates of the hex centers are calculated and a range check is done to determine the closest hex. See Appendix A for a listing of the source code.

The algorithm is made even more efficient by noting that once we have done an initial determination of which cell an entity is active in, a new conversion is only necessary if it at a distance of more than the height (the radius of the inscribed circle) of the hexagon. Therefore we can store the old cartesian coordinates of the hex and do a simple range check to the entity's location and compare that value with the height. If the distance is less than the height then a conversion is unnecessary. Since the height^2 is a constant, the total number of instructions is six (two subtractions, one addition, two multiplies, and one compare):

```
if (height^2 >((current_hex_x - entity_x)^2 + (current_y - entity_y)^2))
{
        convert_cartestian_to_hex
}
```

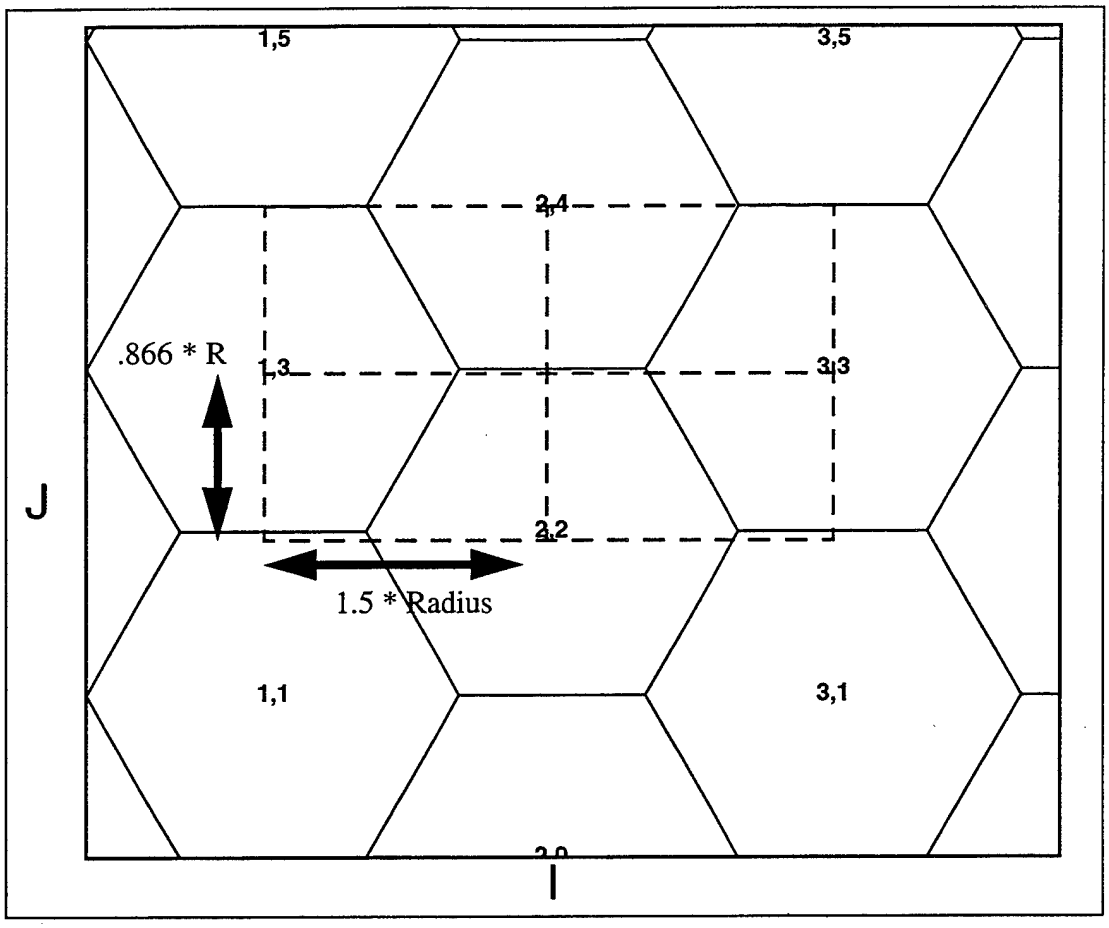


Figure 51. Mapping Cartesian to Hex coordinate system.

In the example in Figure 52, we associate a vehicle with seven hexagons that represent its AOI. Hence, it is also a member of seven network multicast groups. The entity's host listens to all seven groups but, with two exceptions, it sends PDUs only to the one associated with the cell in which it is located.

A vehicle's AOI is typically defined by a radius -- much like the range of an omnia-directional signal of transmitter in a cellular telephone system. If squares were used, we would either need to include more area than was necessary (and thus include more entities in our AOI) or use smaller grids - requiring more multicast groups - and compute which grids the vehicle should be associated with.

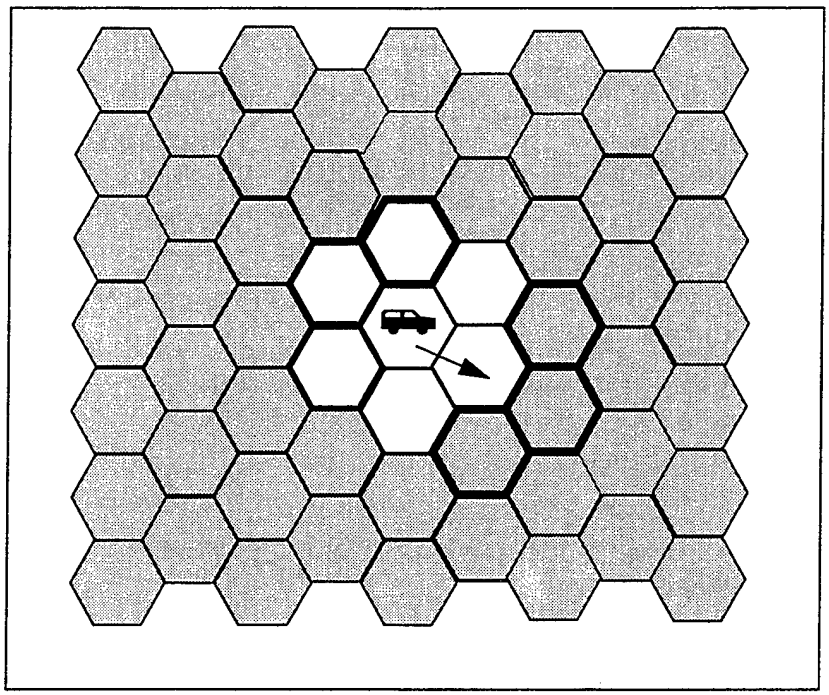


Figure 52. Area of Interest for vehicle mapped to a subset of multicast groups.

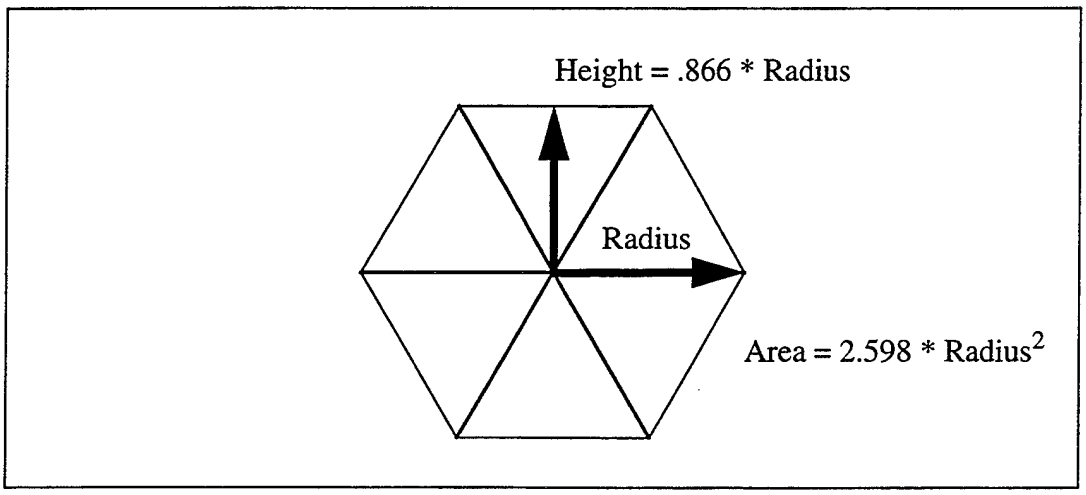


Figure 53. Hexagon geometry.

Using hexagons with a four km radius, the AOI in Figure 52 ranges from 10.39 km to 8 km from the center and the area is 208 km^2. If the average density of vehicles is two per km^2, then the entity host communicates with approximately 400 other entities.

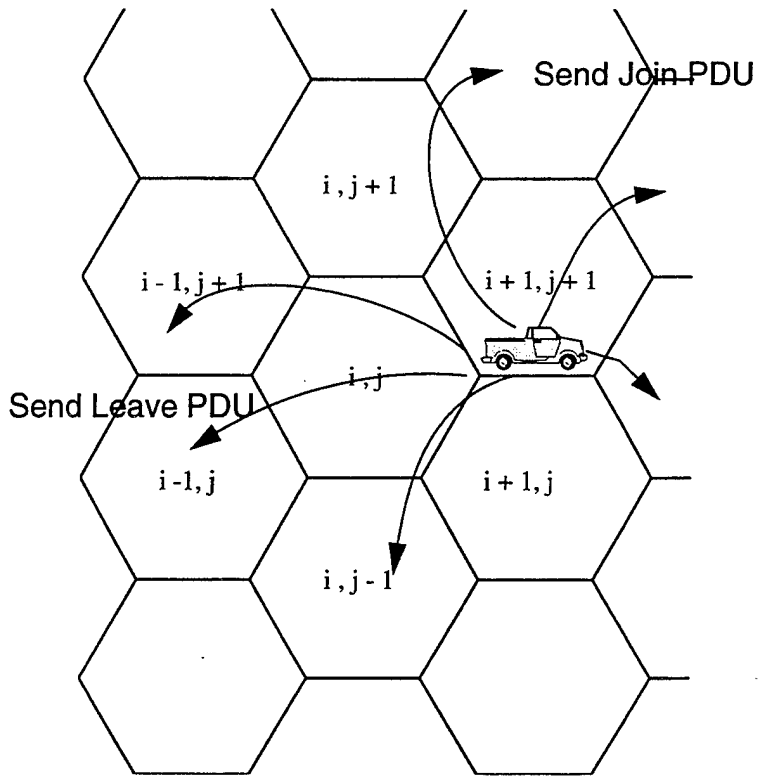
3. Persistent Object Protocol

Entities can belong to several groups at a time to avoid boundary or temporal aliasing. The advantage of belonging to multiple cells mapped to multicast groups is that as the vehicle moves through the VE, it is always aware of every entity within its range, even when crossing cell boundaries. A problem with this is in determining the appropriate size hex and cluster that preserves this awareness without including too many entities (load balancing). Our experimental data discussed later suggests a four km radius hex as a good candidate for exercises like STOW.

Another issue is how quickly and how many ground vehicles move from hex to hex. This affects the stability of multicast group and, as we discuss below, network traffic. We expect that there will be few group transitions by a ground-based entity within an hour (which we demonstrate in later chapters) because, on average, groups of vehicles will move slowly relative to the entire VE. If a vehicle was moving at the Desert Storm record advance rate, it would transition on average one cell an hour.

Moreover, the vehicle portrayed in Figure 52 must join and leave three multicast groups which are associated with cells at the periphery of its AOI where change is less critical - ameliorating the effects of latency caused by joining and leaving new groups. As the vehicle moves through the VE, it uniformly adds and deletes the same number (three) of cells/multicast groups. This is not true for geometry like squares. Active multicast groups only exist for which an entity occupies a grid. The outlined clear cells are removed and the outlined grey cells are added as the entity transitions to a new cell.

We use group changes as an opportunity for database updates -- similar to a paged memory scheme -- in order to eliminate regular ESPDU updates. We do this in a logical, distributed manner using knowledge about the age of entities with respect to their particular group.




```
WHEN ne:{entity object transitions to the north east}
      ASK hexTable[i+1][j+1] SendJoinPDU(); {active}
      ASK hexTable[i+1][j+2] SendJoinPDU(); {rest are passive}
      ASK hexTable[i+2][j+1] SendJoinPDU();
      ASK hexTable[i+2][j] SendJoinPDU();

      ASK hexTable[i][j] SendLeavePDU(); {leave active}
      ASK hexTable[i-1][j+1] SendLeavePDU();
      ASK hexTable[i-1][j] SendLeavePDU();
      ASK hexTable[i][j-1] SendLeavePDU();

      ASK hexTable[i+1][j+2] SendAllPDU(number);
      ASK hexTable[i+2][j+1] SendAllPDU(number);
      ASK hexTable[i+2][j] SendAllPDU(number);
```

Figure 54. Cell transition from current active to northeast cell in MODSIM II.

An entity joins a group as a passive or active member. Active members send as well as receive PDUs within the group, are located in the cell associated with the group, and can become the group leader. Passive members normally do not send PDUs to the group except when they join or leave. They are associated with the group because the cell is within their AOI, yet they are not located within the cell.

When an entity joins a new group it notes the time it entered and issues a *Join Request* PDU to the cell group. The PDU has a flag indicating whether it is active or passive. The group leader replies with a *Pointer* PDU that references the request and in turn multicasts a PDU containing a pointer to itself or another active entity. The new member sends a *Data Request* PDU to the referenced source which issues a *Data* PDU containing the aggregate set of active entity PDUs. A passive entity becomes an active member of a group by reissuing the *Join Request* PDU with a flag set to active when entering a cell. Departures from the group are announced with a *Leave Request* PDU (Figure 54).

We use the oldest member of the group as the election method for group leader. We make use of timestamps to determine the oldest member. The first active member of a group will issue several Join Request PDUs before concluding that it is the sole member of the group and therefore the oldest. When a passive entity determines that there is no leader, it merely listens for active members. A new active member of an established group issues a Join Request PDU, receives the Data PDU, notes the join timestamps of the members, and keeps track of those who enter and leave.

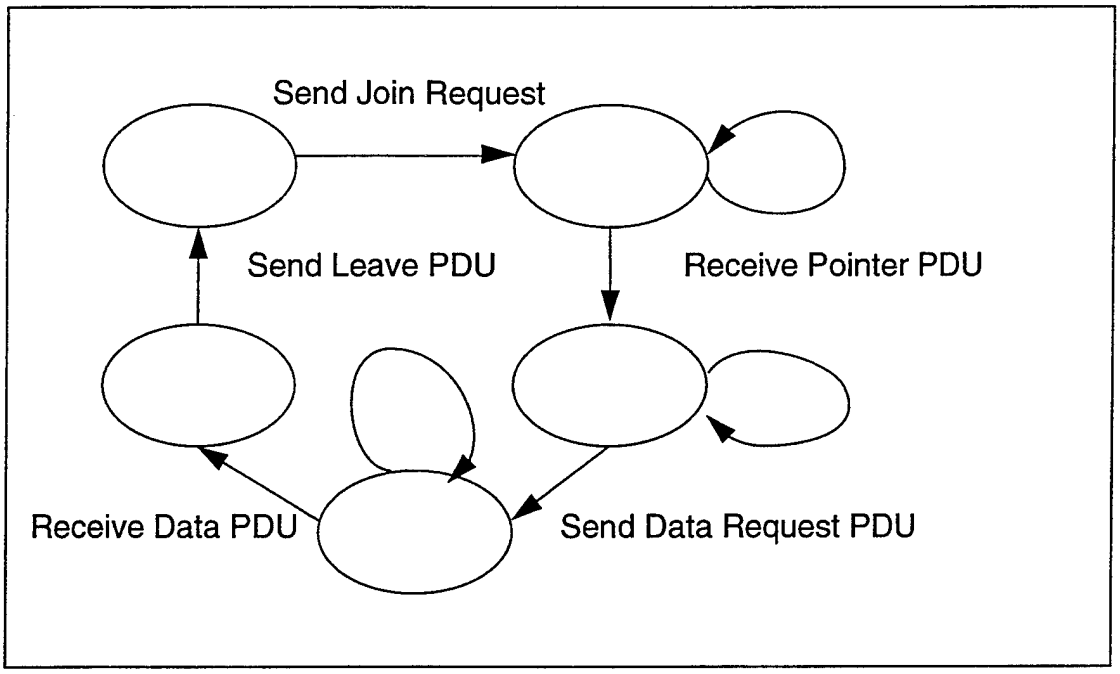


Figure 55. Entity transition protocol.

4. Rationale

The Data PDU may be sent reliably to the issuer of the Join Request PDU via a unicast protocol as a heavy-weight object. With a large member distributed simulation, reliability, as provided in the Transmission Control Protocol (TCP), would normally penalize real-time performance merely by having to maintain timers for each host's acknowledgment. Moreover, flow control is also not appropriate for DIS since systems with humans in the loop can recover from a lost state message more gracefully than from late arrivals. Fortunately, within the context of DIS, a certain amount of unreliability is tolerable and is mediated through the use of the dead-reckoning and smoothing algorithms [97]. Other applications such as packet voice and video can use adaptive techniques to handle lost packets and delays [103]. However, we can reliably send the Data PDU because the entity will normally be joining a group that is at the periphery of its AOI where latency is not as critical.

5. Entity Interactions

Entities can only interact if they are aware of and can communicate with each other. Entity A becomes aware of entity B only if B is an active member of a group that A belongs to -- and therefore, in the AOI of A. If both are only passive members of the same groups then each one is beyond the view or influence of the other.

In a combat simulation, it is possible that if tank A fired a non-guided munition (which is not instantiated as an entity) at tank B, then B's AOI might not overlap the cell in which A was an active member tank. A must become an active member of the target area cell and forward a detonation PDU to that cell. According to the DIS protocol, entities assess for themselves the effects of the detonation and report via an ESPDU any state changes which are the result.

E. SUMMARY

In this chapter we have presented our design objectives for a scalable virtual environment, the principles that we employ, the concept of the Area of Interest Manager, and the communications model it uses. In the next chapter we will present experimental data showing the results of simulating the Area of Interest Manager with data from a large scale exercise based on a real-world scenario.

VIII. SIMULATION

A. OBJECTIVES

This chapter presents data on a simulation used to evaluate our architecture for large-scale VEs. We designed our simulation to examine these questions:

- Does partitioning using our architecture reduce bandwidth and computational requirements for large-scale VEs as compared to the DIS model?
- Does the architecture scale and if so how well?

We were interested in the effect the behavior of entity distribution and maneuver had on our architecture -- particularly as compared to the current DIS broadcast scheme.

B. MEASURES OF EFFECTIVENESS

We translated the above questions to the following *measures of effectiveness*:

1. Communication Traffic Rate

The overall traffic to each individual entity as the result of state messages and entity transitions is compared to DIS broadcast traffic to indicate the relative effectiveness of the architecture. Recall that in the DIS model an entity receives the traffic generated by all entities, including heartbeats. Additionally, the aggregate traffic is measured as this value represents the bandwidth needs of backbone network.

2. Multicast Groups

The peak and average number of cells occupied during the simulation determine the number of multicast addresses that the network must support. In the DIS model there is only one group.

3. Entities in Multicast AOI

Because an entity must maintain state on the entities within its groups, we need to observe the combined peak and average number of entities in each entity's group. Therefore, we determine for every entity, the total number of entities in its groups. For the broadcast model, the AOI consists of the entire environment.

4. Group Transition Rate

The rate of transitions affects the stability of the multicast network and routing (e.g., location of RPs, shortest-path computation, etc.). Moreover, the value is an indicator of network traffic generated by entity transitions.

C. HYPOTHESES

The following are our null hypotheses:

- $H_0 1$: The peak network bandwidth is unaffected by using our multicast AOI model with respect to the DIS broadcast model.
- $H_0 2$: The number of active multicast groups is unaffected by spatial partitioning.
- $H_0 3$: The peak number of entities that an entity must maintain state on is unaffected by the multicast AOI architecture with respect to the DIS broadcast model.
- $H_0 4$: The total number of entity transitions is unaffected by spatial partitioning size.
- $H_0 5$: The peak network bandwidth for multicast AOI model increases at the same or higher rate than for the DIS broadcast model as the number of entities in the simulation increase.
- $H_0 6$: The peak number of entities that an entity must maintain state on for the multicast AOI model increases at the same or higher rate than for the DIS broadcast model.

We expected the null hypothesis would be rejected. However, we were uncertain about the impact of entity transitions on network traffic as result of the combat scenario.

D. EXPERIMENTAL DESIGN

We constructed a simulation that tested the above hypotheses. Figure 56 illustrates our simulation development process. Briefly, we developed our simulation by collecting entity behavior data from a large constructive simulation based on a real-world military scenario (similar to one proposed for use in STOW 97). After post-processing, we then took the military entity data and applied our partitioning algorithms with a model called HexSim. We then evaluated the HexSim output to test the above hypotheses. Below is a detailed description of the design components.

1. Step One: Combat Scenario

An important requirement for our simulation was to evaluate our architecture with respect to data that represented the behavior of real combat entities. Therefore, we used as input to our simulator data from a brigade-size combat simulation. The simulation was created and sponsored by the Battle Command Battle Labs (BCBL) at Ft. Leavenworth, Kansas that:

- was based on a real-world exercise with real-world data,
- portrayed military combat organizations and maneuver at NTC,
- occurred on terrain that represented the worst case for our architecture (discussed below).

The purpose of the scenario developed by BCBL was to drive a combat analysis called the "April experiment". The "April experiment" was part of an effort to design and evaluate future combat organizations under an Army-wide program called Force XXI. The organization under study for the experiment was the Mobile Strike Force (MSF). MSF is being

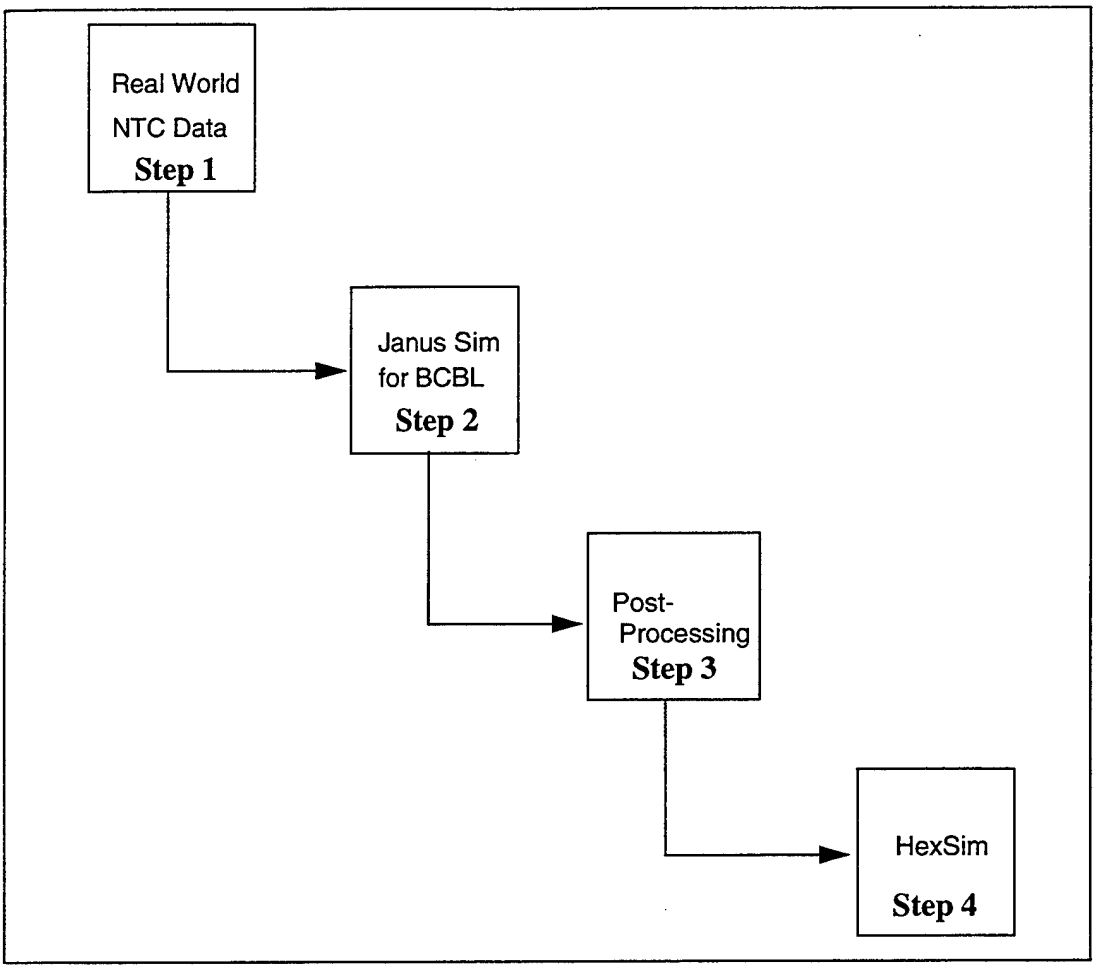


Figure 56. Simulation development.

developed by the Army as a highly deployable force to support a wide range of military contingency missions. The organization is almost identical to the composition of man-in-the-loop SAF for STOW 97 [139]. Figure 57 shows the task force organization.

BCBL developed the scenario based on an actual brigade task force exercise at the National Training Center (NTC Rotation 93-03) [54]. This was a large exercise for the NTC which normally supports only battalion task force training. *Moreover, a brigade typically represents the largest self-contained maneuver element in the Army.* The exercise included an armor battalion task force, a mechanized infantry battalion task force, a light infantry

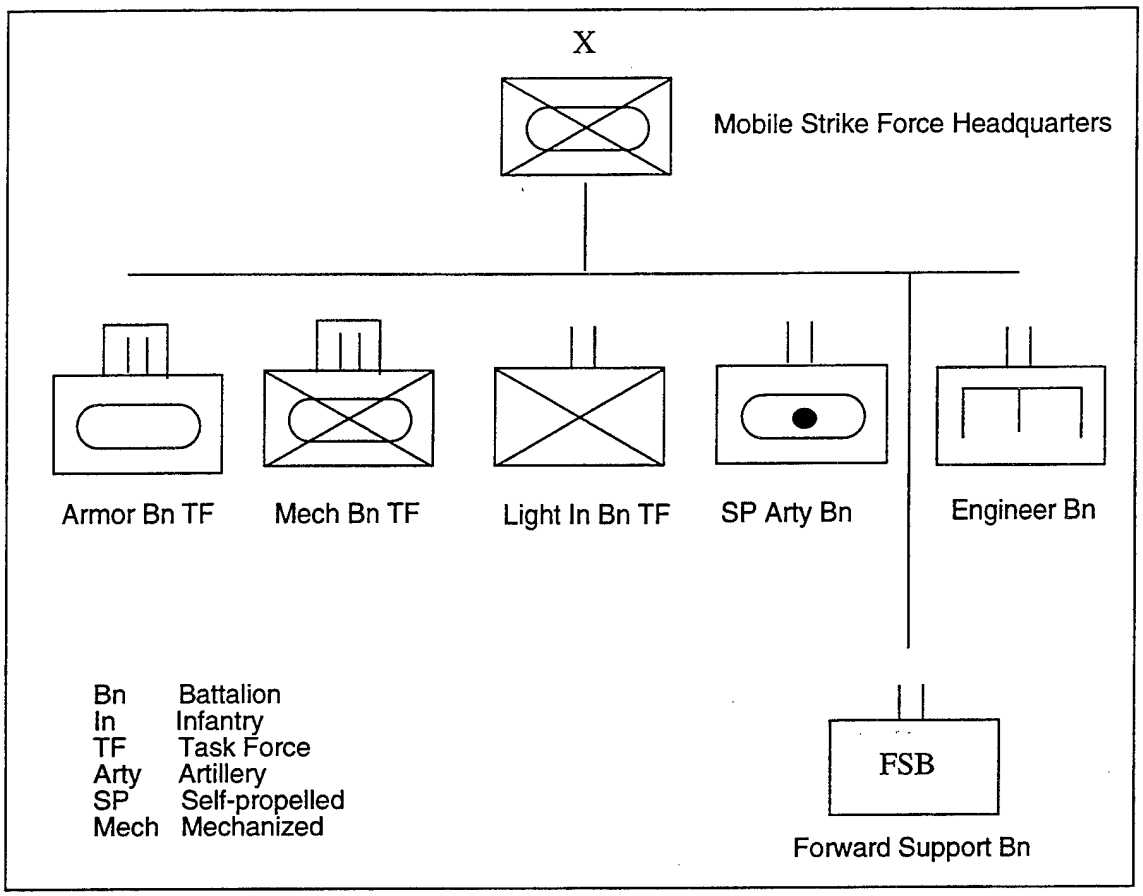


Figure 57. Brigade task force organization.

battalion, a self-propelled artillery battalion, an engineer battalion, and a forward support battalion.

BCBL used from the NTC rotation:

- an exact task organization of forces to include cross attachments,
- original mission statements, commander's intent, critical events and times, and a copy of the maneuver overlay,
- vehicle location data from the NTC instrumentation system.

NTC instruments the vehicles that participate in such exercises and records their location during the conduct of simulated battle. We considered using this data directly. However, it offers low resolution information (one to fifteen minute periodic reports). The Army

Research Institute compiled the instrumentation data and in turn provided the information to TRADOC Analysis Center-Monterey (TRAC). TRAC and BCBL used the initial real-world locations of the vehicles and the scenario from the NTC rotation to build a simulation using the Janus simulator.

Intervisibility is excellent in the wide open desert "bowls" at NTC as shown in the previous chapter. This normally implies that vehicles become tactically dispersed. On the other hand, NTC also has a number of narrow valleys that represent choke points in which vehicles cluster. Therefore, NTC presents the worst case for choosing cell sizes. Cells must be large to account for intervisibility but they may include choke points which result in large concentrations of maneuver elements and hence, the possibility of many vehicles or systems in an entity AOI.

2. Step Two: Janus Modeling

The Janus Combat Modeler, as described in [47],is a monolithic combat model used primarily by the United States Army since 1983 at over fifty locations worldwide. It is used as a combat development tool, analyzing both weapon systems and tactics. Janus is also employed as a tactics training and staff planning tool.

Modeling military systems in ground combat is the primary focus of Janus. Users of Janus develop combat scenarios consisting of short, scripted, closed, force-on-force engagements. Janus portrays both entity level systems as well as aggregate level such as infantry platoon or artillery batteries. Janus employs high resolution algorithms which accurately model numerous weapons platforms.

Janus has been judged as a valid environment for combat development by the US Army. Mel Parish, the Chief of the Janus Development Division, states that the maneuver of Janus entities has been compared and calibrated against real combat and training exercises such as the Desert Storm battle 73 Easting and NTC rotations [102].

Janus is monolithic in nature, operating on one computer system. Its two-dimensional displays provide the military expert with a 2D graphic display of the battlefield. The Janus model is made up of sixteen FORTRAN executable programs and associated databases. Janus can operate on two platforms. One version uses Digital VAX machines utilizing the VMS operating system and X-Windows workstations. The second version runs using the UNIX operation system and employs either Dextranase or X-Windows workstations. TRADOC Analysis Command at White Sands Missile Range (TRAC-WSMR) is the Army agency responsible for maintenance and modification of Janus. BCBL and TRAC used the UNIX based Janus version 3.X for the April experiment.

BCBL ran the MSF scenario with Janus in April 1994. The battle simulated was a deliberate attack by a blue brigade task force against approximately two battalions of enemy (red) forces. Advanced military students from the Command and General Staff School (CGSC) acted as the brigade and battalion commanders and staff for the blue force. They could choose unit routes or missions during the conduct of the exercise. Red forces were fought by a specially trained team called the "world-class OPFOR" from the Battle Command Training Program at Ft. Leavenworth [54].

The battle was approximately ten hours long. Blue forces were positioned in the northeast of the NTC playbox with red forces in the defense in the southwest. Little activity occurred in the early portion of the battle -- mostly during the night -- when blue reconnaissance units probed enemy lines. The last three hours (0700 -1000) were the most active when the MSF attacked with its main force at dawn. Janus was run non-interactively and was stopped as units reached phase lines pre-planned by the students. New routes were chosen if demanded by the tactical situation.

3. Step Three: Janus Post-processing

Janus records entity activity into several separate files [47]. We used the PPRUNXXX.DAT file for initial force positions, PPMOVXXX.DAT for movement data, and PPKILLXXX.DAT for information on entity kills in conjunction with the Janus force listing to produce an input file for our simulator. We developed a set of Awk language scripts to prepare the data for use with our simulator. Air entities, including helicopters and close support aircraft such as the A-10, were removed because of our concerns that Janus does not portray aircraft accurately and to simplify our analysis by focusing only on ground systems.

4. Step 4: HexSim

Our simulation, called HexSim (Appendix B), deterministically modeled our architecture (spatial partitioning and persistent object protocol) and the DIS broadcast model. The input to HexSim was the Janus data which indicated the behavior (e.g., distribution, movement, kills) of the entities during the course of a ten hour period. Hexsim was constructed using the CACI MODSIM II language. MODSIM II is a modular, object-oriented, strongly typed simulation language [23]. The language has facilities for discrete event simulation such as queue objects. MODSIM II also permits polymorphism so that user defined objects can inherit attributes from others such as the system stack, queue, and group objects.

In HexSim, each entity and hex cell are treated as objects with their own data structures and methods (Figure 58 shows an entity object definition). When entity activity is read from the post-processed Janus files, the simulator determines its associated hex object and if the entity's activity results in network traffic. The appropriate entity and hex objects are updated with the computed results. For example, each hex maintains a list of entities present within its boundaries. If an entity transition between cells occurs, the persistent ob-

```
DEFINITION MODULE Entity;

TYPE

            directionType = (n,ne,se,s,sw,nw);

  EntityObj = OBJECT
    timestamp : REAL;
    side      : INTEGER;
    event     : INTEGER;
    id        : INTEGER;
    active    : BOOLEAN;
    trans     : INTEGER;
    moves     : INTEGER;
    evermove  : BOOLEAN;
    count     : INTEGER;
    xcoord    : REAL;
    ycoord    : REAL;
    icoor     : INTEGER;
    lasti     : INTEGER;
    jcoord    : INTEGER;
    lastj     : INTEGER;
    direction : directionType;
    apdubitcount: INTEGER;

            ASK METHOD SetTime( IN t : REAL);
            ASK METHOD SetSide( IN s : INTEGER);
            ASK METHOD SetEvent( IN e : INTEGER);
            ASK METHOD SetId( IN i : INTEGER);
            ASK METHOD SetActive( IN a : BOOLEAN);
            ASK METHOD SetCount( IN c : INTEGER);
            ASK METHOD SetXYcoord( IN x,y: REAL);
            ASK METHOD SetHexcoord( IN x,y : REAL);
            ASK METHOD IncTrans();
            ASK METHOD IncMoves();
            ASK METHOD SetIJcoord(IN i,j: INTEGER);
            ASK METHOD PrintEntity();
            ASK METHOD SetMove(IN m : INTEGER);
            ASK METHOD SetTrans(IN tr : INTEGER);
            ASK METHOD ObjInit();
            ASK METHOD SetDirection();
            ASK METHOD SetEverMove();
            ASK METHOD SendAllPDU(IN entitynumber : INTEGER);
            ASK METHOD Resetcount();
            ASK METHOD Bitcount(): INTEGER;

  END OBJECT;

            entityListtype  = ARRAY INTEGER OF EntityObj;

END MODULE.
```

Figure 58. Entity object and module definitions.

ject protocol is initiated. Furthermore, HexSim keeps track of dead entities, primarily for computing their effect on broadcast traffic.

The output of HexSim was a variety of statistics that indicated network traffic and entity counts in Area of Interests. The following statistics were generated every second during each simulation run:

- number of entity transitions among cells
- number of hexes occupied
- the maximum number of entities in a cell
- the cell with the maximum number of entities
- maximum entities in a group for an entity
- the entity with the most entities in its groups
- maximum bits per second to an entity
- the entity with maximum bits per second
- minimum bits per second to an entity
- the entity with minimum bits per second
- the mean bits per second
- the broadcast bandwidth

In addition, we periodically collected data about every cell including:

- number of entities in the cell
- the maximum number of entities during the simulation
- the mean number of entities in the cell
- the standard deviation, variance and bit count

We compared some HexSim outputs to calculations done by hand and to previous data collected from tests of NPSNET to insure that the results were both logical and correct for a specific input [91].(See the DIS and NPSNET chapters for some expected values.) For example, DIS broadcast traffic was computed by determining how many entities presented any activity during a period, multiplying the total by the size of an ESPDU encapsulated in a UDP packet and adding the traffic represented by dead or stationary entities.

5. Hypothesis Testing

We selected partition size as the independent variable to test hypotheses $H_0 1$ to $H_0 4$. Our reasoning is that if hex size has no correlated effect on the dependent values (peak bandwidth, number of multicast groups, peak entities in AOI, entity transitions) then our architecture is ineffective at limiting them relative to the DIS model values. On the other hand, if there is a correlation than the architecture is effective, particularly if the values are lower for the multicast model.

We initially used one, two, three, and four km radius hexes in seven hex clusters per entity in our architecture. We compared each partitioning size with each other and with the broadcast model. Figures 59 and 60 show the relationship of hex size to the terrain and entities for a portion of the playbox. Note the large number of one km hexes relative to four km cells. An entity cluster (five hexes) is highlighted in each image.

We used the following criteria for rejection:

- $H_0 1$: Reject if peak network bandwidth for the multicast AOI model positively correlates with partitioning size.
- $H_0 2$: Reject if the number of active entity groups for the multicast AOI model correlates with partitioning size.
- $H_0 3$: Reject if the peak number of entities that an entity must maintain state on correlates with partitioning size.

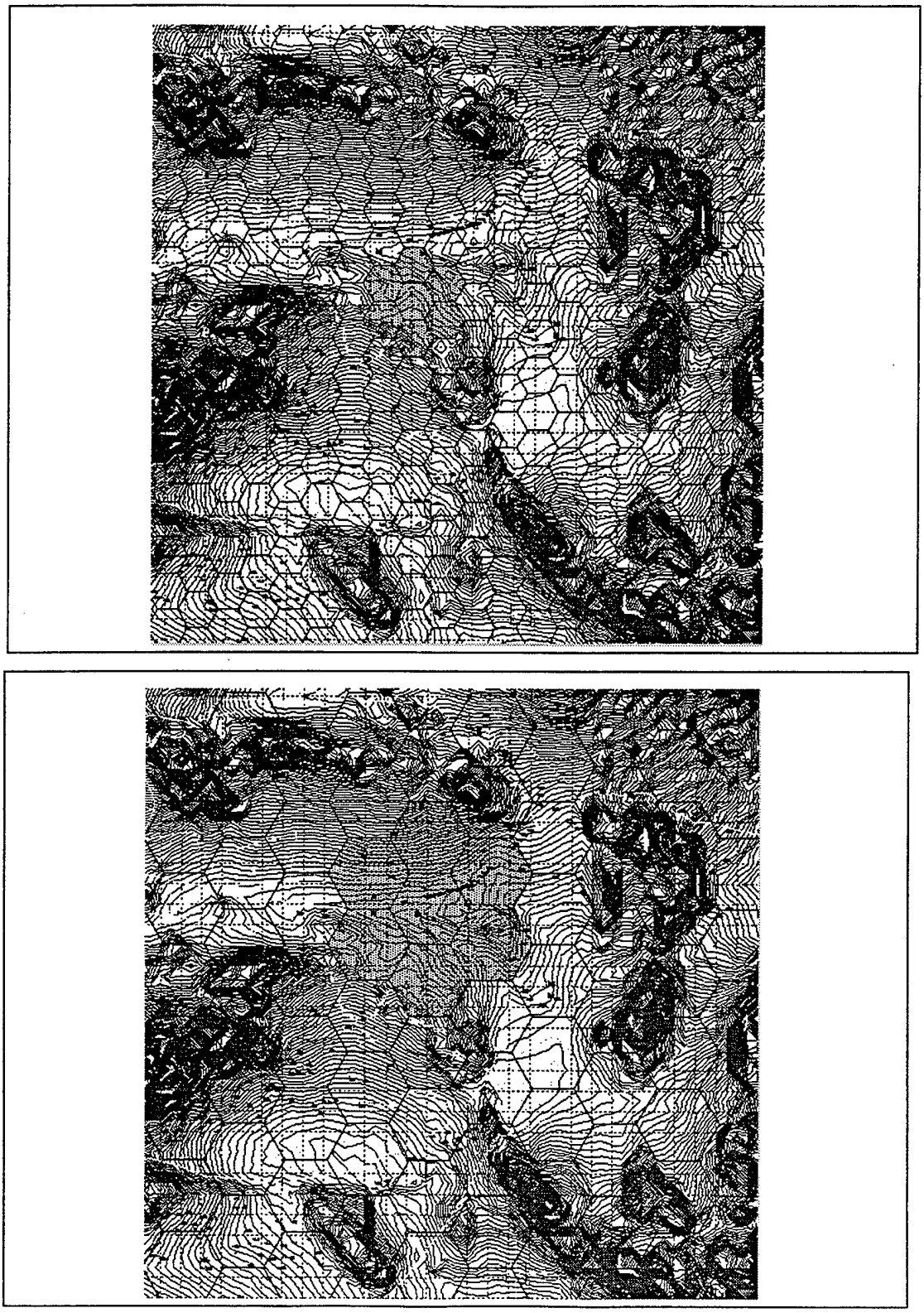


Figure 59. One and two km hexes.

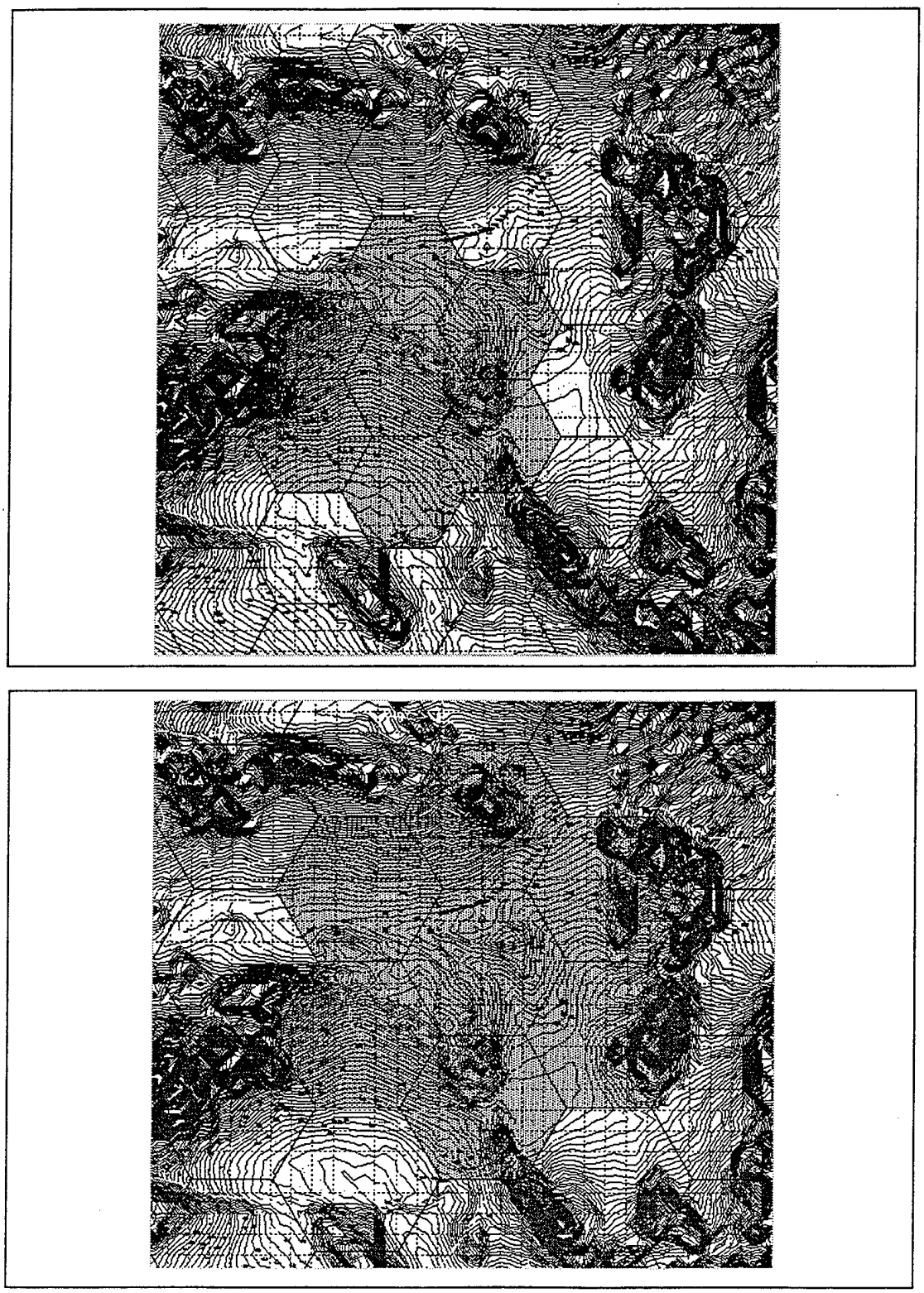


Figure 60. Three and four km hexes.

- $H_0 4$: Reject if the total number of entity transitions correlates with partitioning size.

For $H_0 5$ and $H_0 6$ the dependent value was the number of entities. A four km hex radius partitioning was chosen to evaluate the ability of the architecture to scale with the addition of more entities in comparison with broadcast. We chose four km because the size is the most doctrinally correct for the terrain (see our discussion in the previous chapter) and because it appeared after the first set of comparisons to show substantial improvement over the DIS broadcast model. We increased the number of entities by 2, 3, and 4 times the original 2191 count. We used the following criteria for rejection of these hypotheses:

- $H_0 5$: Reject if the peak network bandwidth for the multicast AOI model increases at a lower rate than for the DIS broadcast model as the number of entities in the simulation increase

- $H_0 6$: Reject if the peak number of entities that an entity must maintain state for the multicast AOI model increases at a lower rate than for the DIS broadcast model.

E. EXPERIMENTAL ASSUMPTIONS

- We assumed that Janus portrayed entities in a realistic and tactically correct manner. As discussed previously, the scenario represented a real NTC exercise and was conducted by tactical experts.

- Janus represents some entities as aggregates (e.g., infantry platoons). HexSim de-aggregates these entities to individual weapons system for computing live and dead entity densities and their effect on multicast and broadcast traffic. (For example, instead of an infantry platoon we represented it as fifteen individual entities.)

- Broadcast heartbeats were treated stochastically using a normal distribution with a mean period of five seconds -- a default value in the IEEE 1278 standard.

- Only Entity State PDU (ESPDU), Join PDU, Leave PDU and Data PDU traffic were considered. Though other network traffic may come to dominate in future DIS exercises (e.g., Signal PDUs) ESPDUs currently consume over 90% of typical exercise traffic (see Chapter III). The Data, Leave, and Join PDUs are associated with our persistent object protocol.

- Janus only provides one second time resolution. Therefore, we assumed that an entity move would generate an expected value of eight ESPDUs for the second (see Chapter III). In this regard we treated the multicast and broadcast cases the same.

- DIS entities can have articulated components (e.g., turret and gun tube) which add to the size of the ESPDU but are not represented in Janus. Therefore, each entity was assumed to have two articulated components in determining the size of the ESPDUs.

- When examining scaling, we replicated the original entity data and shifted each replication by seventeen kilometers -- the width of the brigade in the scenario and roughly the doctrinal width of a standard mechanized brigade's boundaries on this type of terrain.

- Overhead for reliable communications (e.g., acknowledgments and re-sends) was set at 10000 bits per transfer of group information on an entity Join.

- We assumed that network bandwidth was unlimited and that there was no network congestion or contention in order to determine the application requirements.

- When using our multicast architecture we assumed that entities would belong to seven spatial groups or cluster.

Some of these assumptions could negatively effect the performance of our architecture. For example, the infantry are densely concentrated on the battlefield when they are de-aggregated. Therefore, an entity with an infantry battalion (~500 entities) in its AOI may have many more systems that it must maintain state on than if an armor battalion is portrayed (~ 100) in the AOI. Furthermore, as mentioned before, terrain at NTC combines both wide open areas with narrow choke points. Many of the other assumptions, including ESPDU rates, articulated components and available bandwidth, should not have an important impact since they apply equally to our architecture and the DIS model.

F. DATA AND ANALYSIS

a. *Communication traffic among entities*

$H_0$1.*The peak network bandwidth is unaffected by using our multicast AOI model with respect to the DIS broadcast model.* The simulation computed traffic on a per-entity basis. Different entities would receive different amounts of network traffic based on the activity and size of the groups to which they belong. We determined every second which entity out of all 2191 received the most network traffic (peak entity bps). We also computed the max and the mean peaks for over the entire 10 hour period (Table 5). For the DIS broadcast traffic (no partitioning, one group) the maximum peak was 302515 bps and the mean peak was 801058 bps.

Parameter	1 km	2 km	3.5 km	4 km
Peak entity bps	2063360	2076672	2169856	2209792
Mean max entity bps	55432	69718	73016	75799

Table 5: Peak multicast communications traffic.

Analysis. Peak multicast AOI traffic strongly correlates with hex size -- 0.9645. (Figure 61 shows the regression equation and significance tests). Therefore, we reject $H_0$1. Furthermore, peak AOI traffic is smaller than peak broadcast communication by 27% in the case of 4 km hexes. Moreover, the mean multicast traffic overall is much less than the broadcast traffic. Figure 62 shows this during the most active period of the simulation. The mean multicast peak strongly also correlates with the hex size -- 1.0 -- while the max peak grows relatively slowly with hex size.

```
Regression Equation for REG:
Bps =  1.747e-05 A  +  -34.7093

Significance test for prediction of REG
    Mult-R  R-Squared      SEest      F(1,2)    prob (F)
    0.9645     0.9302     0.4176     26.6667     0.0355
Significance test(s) for predictor(s) of REG
Predictor       beta        b       Rsq       se       t(2)
Hex size        0.9645     0.0000 0.0000  0.0000    5.1640
```

Figure 61. Regression analysis for peak multicast traffic.

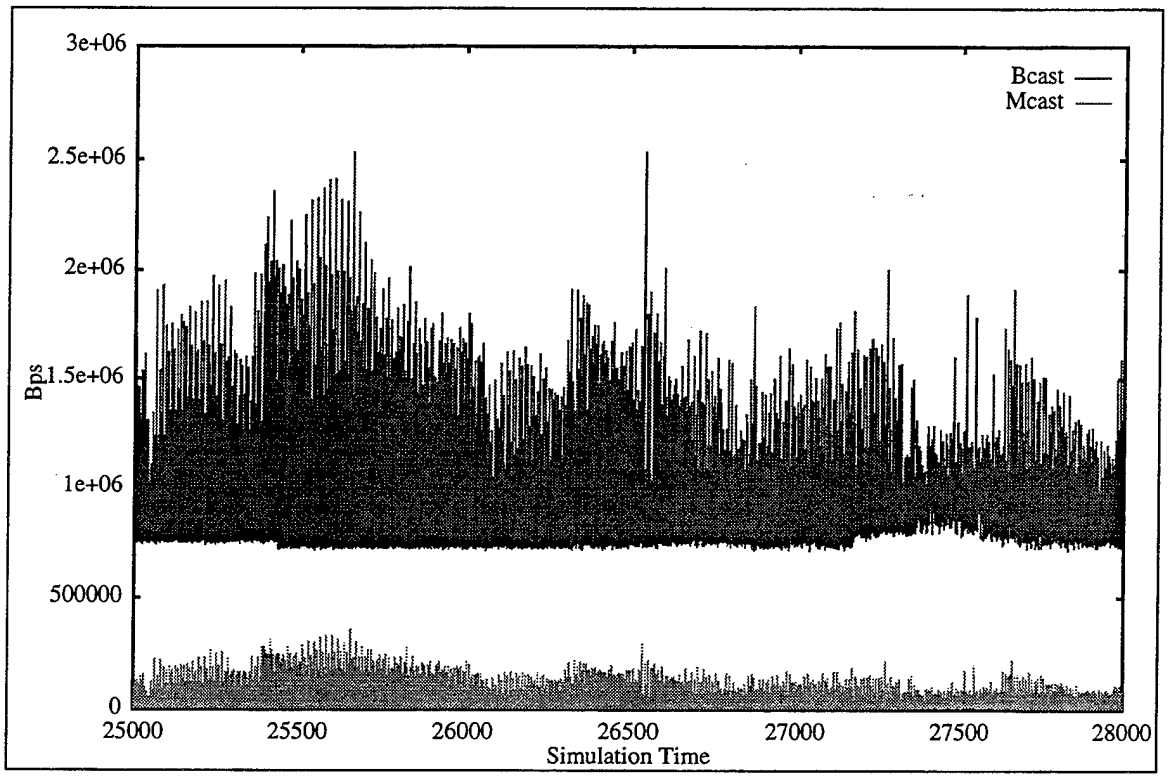


Figure 62. Mean multicast (4 km hex) vs. broadcast traffic.

However, as shown in Figure 63, we observed that because of entity transitions, large peaks occurred using our model. These spikes are not real-time data, but reliable traffic that is flow controlled -- Data PDUs via TCP. On the other hand, as shown in Figure 64, for *less than ten seconds of the ten hour simulation does peak bandwidth exceed T-1*

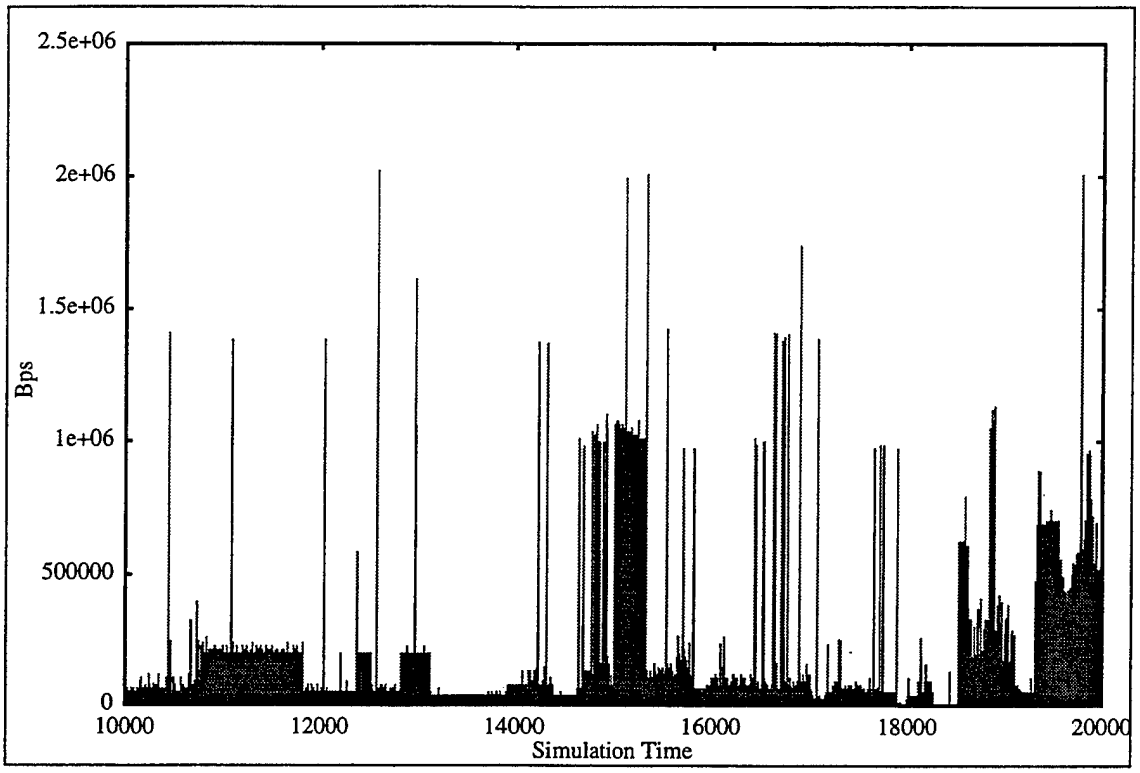


Figure 63. Traffic using four km radius cells.

rates. In a congested network, we conjecture that would be smoothed out by the congestion and flow control mechanisms.

We also found the minimum traffic for an entity using our architecture was 0 bps. Therefore, at times during the simulation some entity AOIs had no activity (e.g. command posts on the periphery of the battle). Mean traffic is less than 300 kbs while broadcast traffic is approximately 1.5 Mbps.

b. Active multicast groups

$H_0$2. *The number of active multicast groups is unaffected by spatial partitioning.*Every one thousand seconds, we determined which cells were active. At most only 13% of the cells were occupied with a 1 km hex radius (Table 6). Figure 66 shows the distribution of entities in the playbox. The playbox was defined by the most distant entities in the scenario.

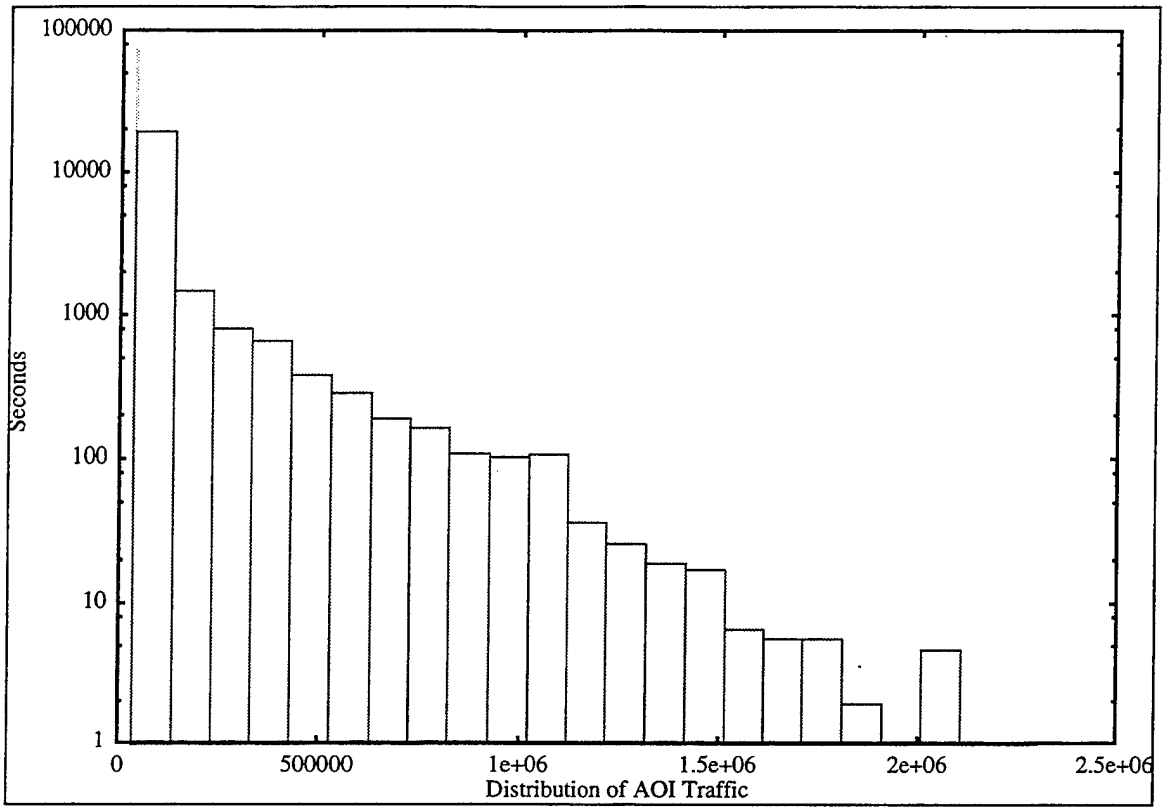


Figure 64. Distribution of entity peak bandwidth.

Parameter	1 km	2 km	3 km	4 km
Total number of hexes	1240	336	154	99
Peak hexes occupied	162	64	42	30
Mean hexes occupied	151	60	38	27
Percent peak occupied	13%	19%	27%	30%

Table 6: Cell occupancy.

Analysis. The number of active entity groups for the multicast AOI model correlates with partitioning size (Figure 65). Therefore, we reject $H_0 2$. Note in the Figure 66 that one cell has about 250 entities. This cell has the light infantry battalion and illustrates the problem of representing different types of entities in a VE. While armored vehicles tend to disperse, dismounted infantry soldiers and their weapons cluster in relatively dense groups.

```
Regression Equation for REG:
Groups  =  -0.01935 A  +  3.94131

Significance test for prediction of REG
    Mult-R  R-Squared      SEest      F(1,2)    prob (F)
    0.8993     0.8087     0.6916     8.4538     0.1007

Significance test(s) for predictor(s) of REG
Predictor        beta        b        Rsq        se       t(2)
Hex size       -0.8993   -0.0193    0.0000    0.0067    2.9075
```

Figure 65. Regression analysis for active multicast groups.

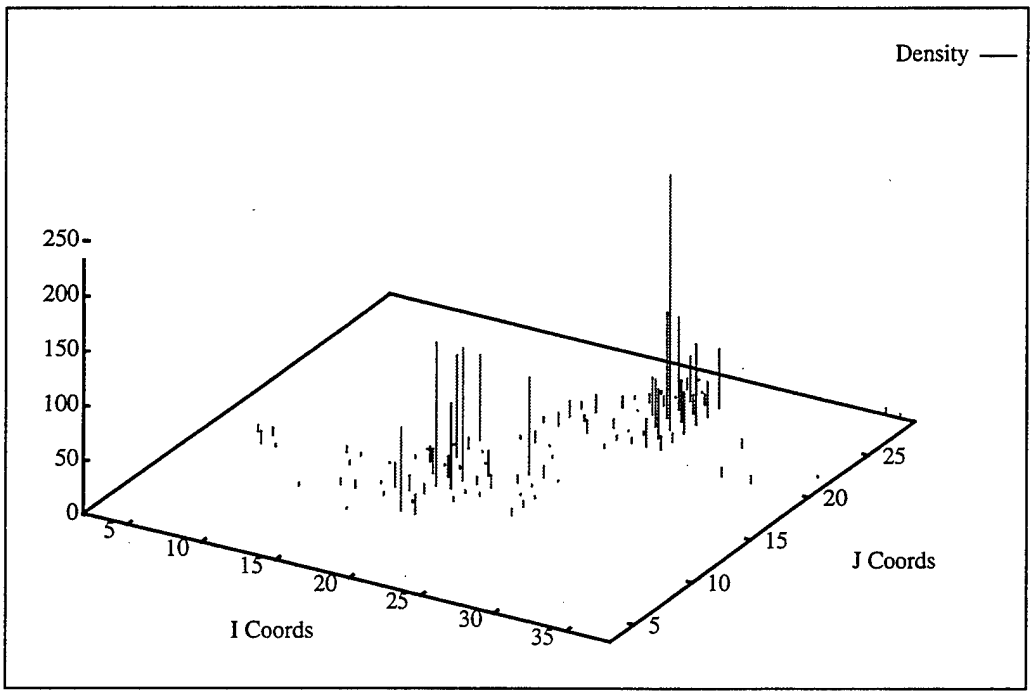


Figure 66. Entity distribution using hex coordinates.

The implication of this distribution is that much of the battlefield is unoccupied and that a relatively small subset of the multicast addresses will be used. Therefore, using 4 km cells for STOW 97, with a size of 800 by 800 km, approximately 15,000 hex cells (area of playbox / area of cell) would be required, of which only 5000 would be active.

c. *Maximum Entities in AOI*

$H_0$3. *The peak number of entities that an entity must maintain state on is unaffected by the multicast AOI architecture with respect to the DIS broadcast model.* We computed every second the entity with the most number of entities in its AOI and then determined the peak and mean peak values for each size hex.

Analysis. Hex size strongly correlates to the peak number of entities in an entity's AOI -- 0.975 (Figure 67). Therefore, we reject $H_0$3. Figure 68 also demonstrates this relationship. Table 7 shows that even with the 4 km hex the peak number of entities in any AOI is less than the 2191 for the DIS broadcast model. Therefore, less state maintenance is required by entities in our model (32% less for a brigade at NTC) as opposed to the DIS broadcast model. Figure 69 shows this with the different size groups over time. The rise at the end of the graph is due to the concentration of entities as the majority of the blue forces come in contact with the red forces.

```
Regression Equation for REG:
Entities  =  0.003049 A  +  -0.892565

Significance test for prediction of REG
    Mult-R  R-Squared      SEest     F(1,2)   prob (F)
    0.9752     0.9509     0.3503    38.7534     0.0248

Significance test(s) for predictor(s) of REG
Predictor        beta          b        Rsq         se        t(2)
Hex size         0.9752     0.0030     0.0000     0.0005     6.2252
```

Figure 67. Regression analysis for peak entities in AOI.

Parameter	1 km	2 km	3 km	4 km
Peak entities in AOI	583	994	1380	1494
Mean max entities in AOI	522	870	1016	1199

Table 7: Peak number of entities in AOI for multicast.

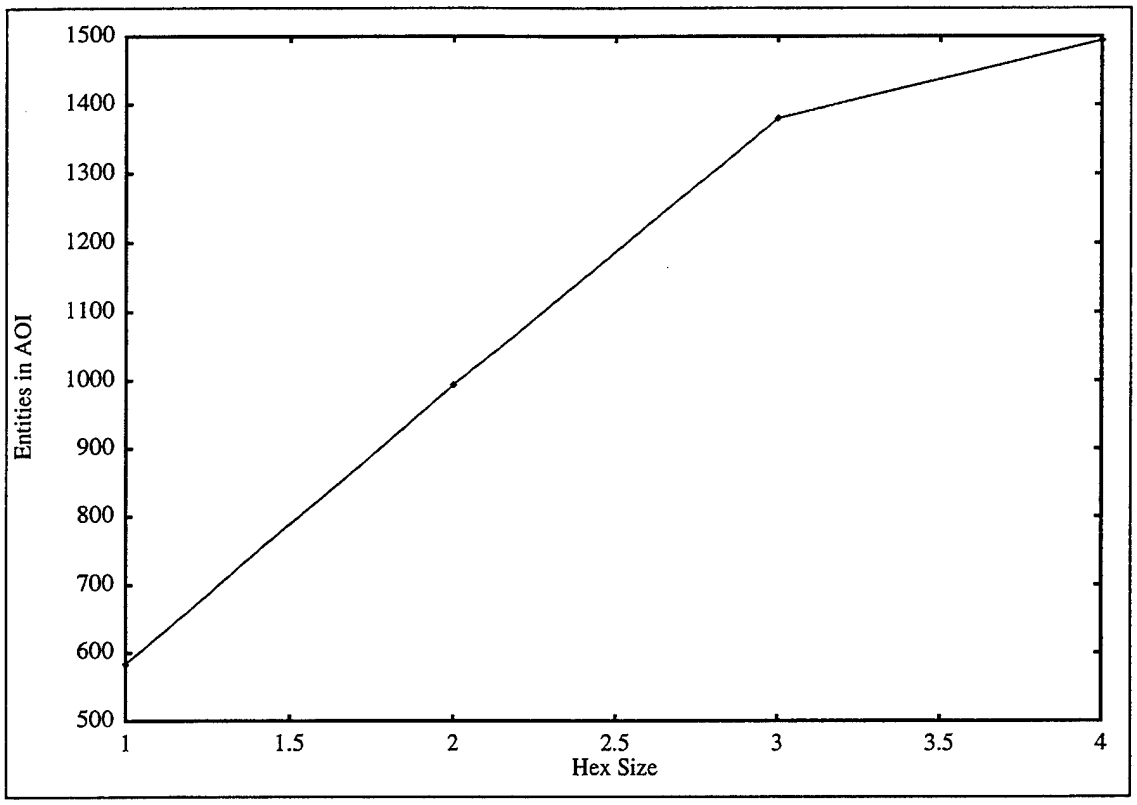


Figure 68. Peak entities in AOI vs. hex size.

d. Entity transitions

$H_0$4. *The total number of entity transitions is unaffected by spatial partitioning size.* We computed the peak and total number of transitions by entities as they moved from one hex cell to another.

Analysis. The total number of transitions correlates with the hex size (Figure 70). Therefore, we reject $H_0$4. Table 8 and Figure 71 also show that as hex size increases the total number of entity transitions decrease. The peak number of entities for a cell naturally would be expected to increase with cell size. This implies that, even though larger cells may have more entities, this is offset from a communications standpoint by a lower likely number of transitions which trigger the sending of group data.

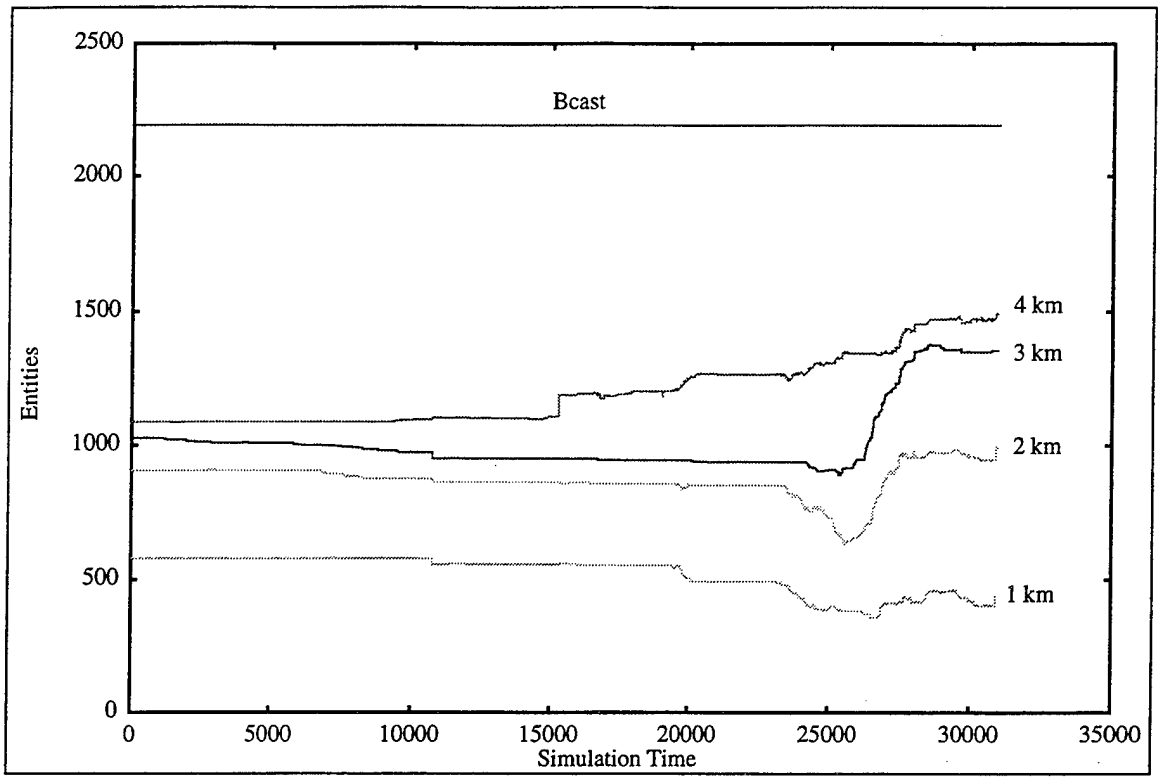


Figure 69. Max entities in AOI by hex size.

```
Regression Equation for REG:
Transitions  =  0.003049 A  +  -0.892565

Significance test for prediction of REG
    Mult-R  R-Squared      SEest     F(1,2)    prob (F)
    0.9752     0.9509     0.3503    38.7534     0.0248

Significance test(s) for predictor(s) of REG
Predictor        beta          b        Rsq         se        t(2)
Hex size         0.9752     0.0030     0.0000     0.0005     6.225
```

Figure 70. Regression analysis for total entity transitions.

Interestingly, we found that 694 or 32% of the entities did not move during the entire simulation. 99% to 95% of the entities were stationary at any time. This is in line with Helmbold's study discussed in our theory chapter. The vehicles that did not move included artillery systems, command vehicles, and entities in the defense.

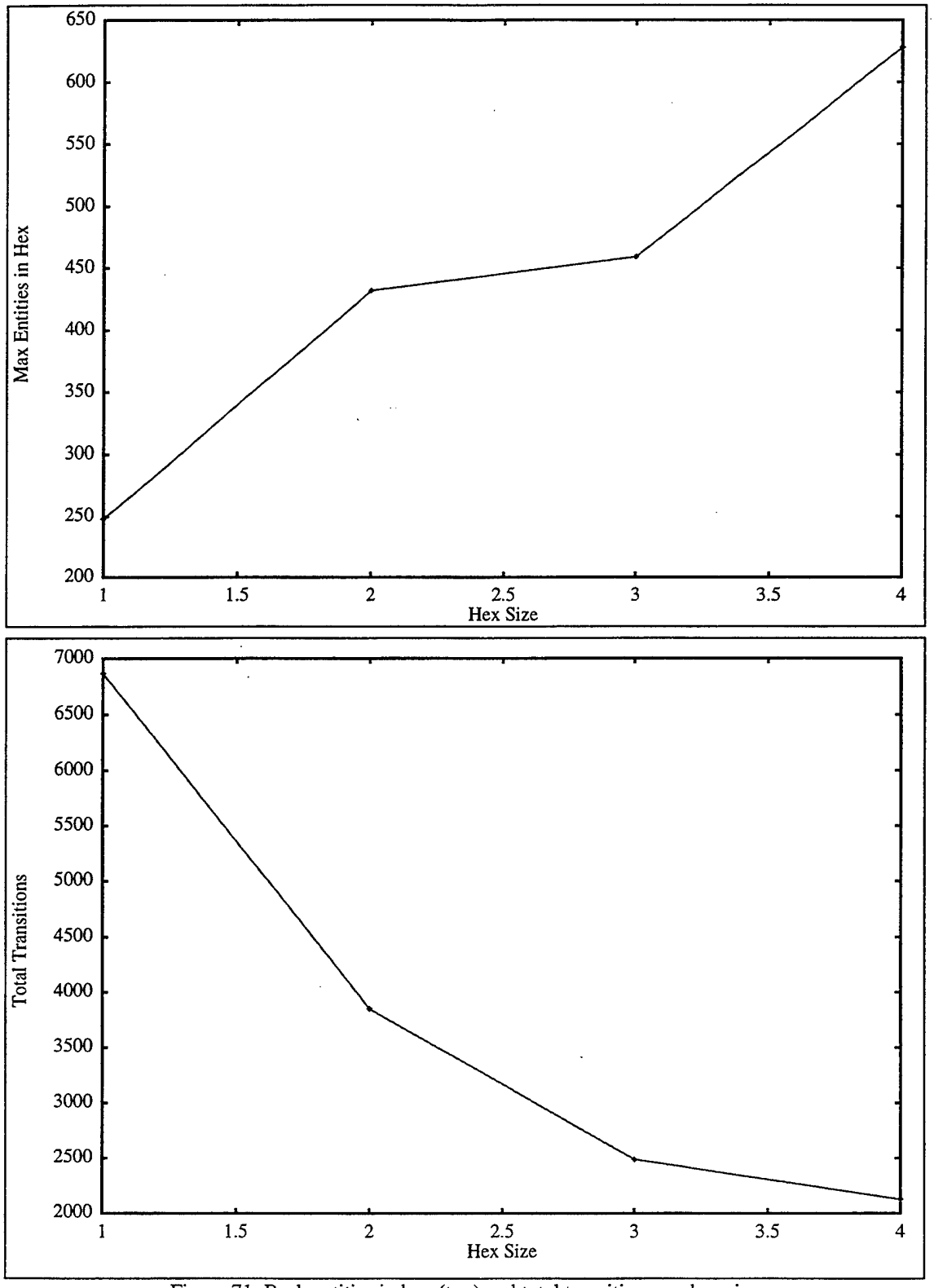


Figure 71. Peak entities in hex (top) and total transitions vs. hex size

Parameter	1 km	2 km	3.5 km	4 km
Peak entity transitions	76	22	22	76
Total entity transitions	6866	3850	2490	2125
Peak entities in hex	248	432	459	628
Mean max entities in a hex	210	388	411	526

Table 8: Entity transitions and density.

e. Bandwidth scalability

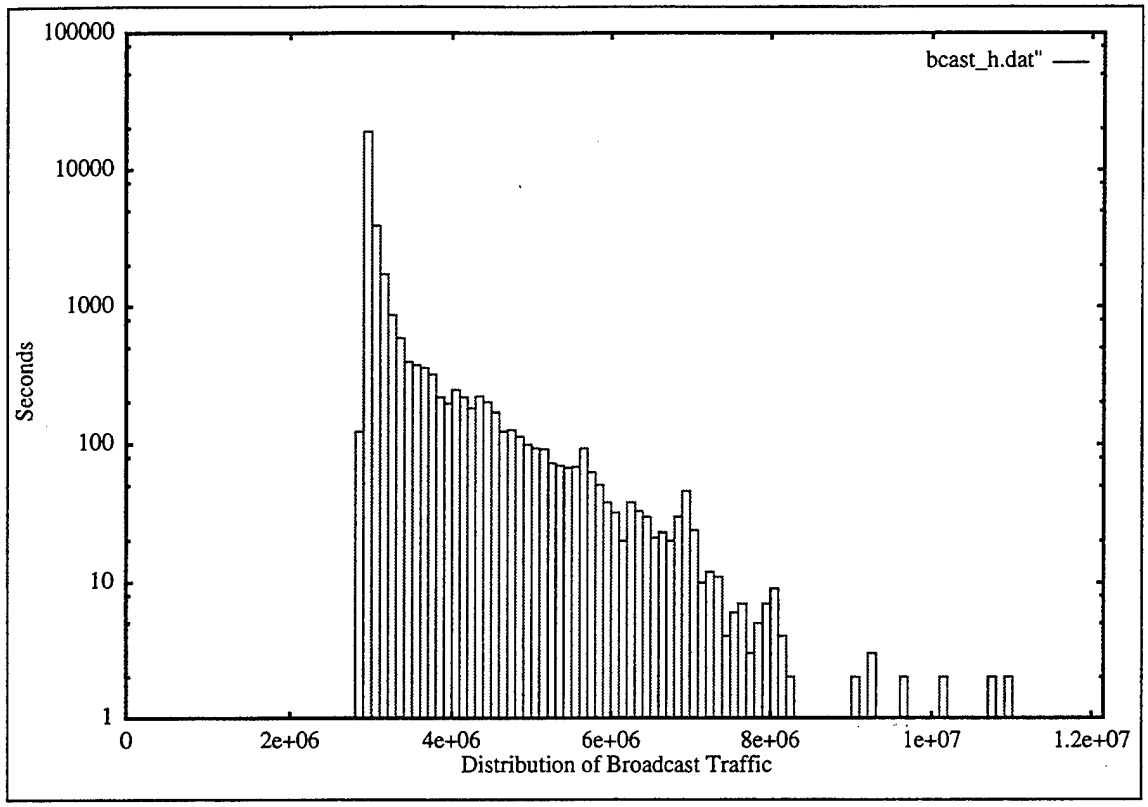
$H_0$5. *The peak network bandwidth for multicast AOI model increases at the same or higher rate than for the DIS broadcast model as the number of entities in the simulation increase.*

Analysis. The peak multicast AOI traffic is relatively constant, as shown in Table 9 (its flat), while the DIS traffic increases with a linear regression equation of 3.304×10^7 x Entities + 0.000274967. Therefore, we reject $H_0$5.

While the mean maximum broadcast bandwidth increases linearly using the DIS model, our architecture's mean maximum entity bandwidth increases at a far lower rate. For example, with 2191 entities the ratio between the multicast and broadcast peaks is 2.2 to 3 while at 8764 entities the ratio is 1 to 3 (Table 9).

The two histograms in Figure 72 again demonstrate that the advantage of using our architecture. They show the distribution of traffic for 8764 entities using the DIS model and the AOIM. In our architecture there is only a single one second peak over 4 Mbps while the remainder of the traffic is less than 2.5 Mbps.

The peak aggregate bandwidth used by our architecture does exceed the peak broadcast traffic. This is important for determining the backbone network requirements. However, it is less critical to the tail links since we assume that entities would be hosted at different sites, rather than concentrated on one subnet.



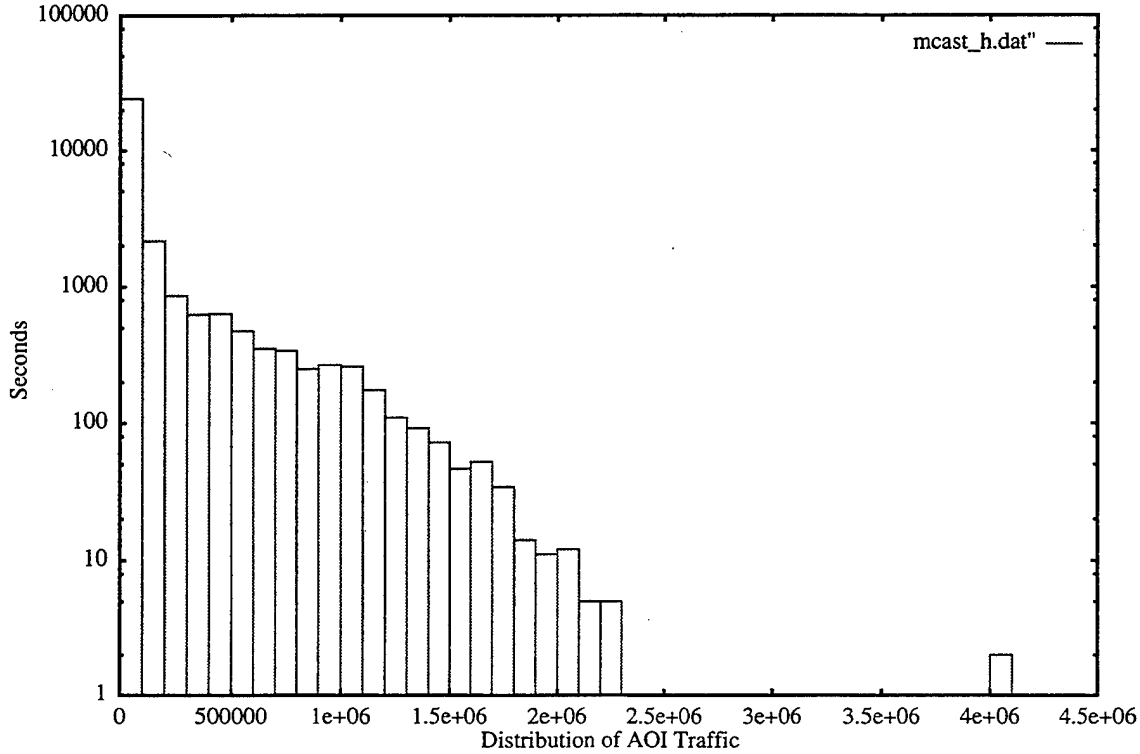


Figure 72. Distribution of peak traffic.

Parameter	2191 entities	4382 entities	6573 entities	8764 entities
Peak entity traffic	2209792	4193280	4193280	4193280
Mean max entity bps	75799	101755	111714	121140
Peak broadcast bps	3025152	6051968	9078784	12103936
Mean max broadcast bps	801058	1602923	2404792	3206665
Peak aggregate bps	2760708	7422292	11856556	18662480
Mean max aggregate bps	80496	168666	253374	340984

Table 9: Bandwidth vs. number of entities

f. Entities in AOI

$H_0$6. *The peak number of entities that an entity must maintain state on for the multicast AOI model increases at the same or higher rate than for the DIS broadcast model.*

Analysis. Table 10 shows that the peak AOI size grows slowly for the multicast model as the number of entities in the simulation increase. The average of the four multicast peaks is 1711 with a standard deviation of 147. In contrast, as the number of entities increase by 2, 3, and 4 times, the DIS broadcast model increases by the same number because an entity must maintain state on all the entities in the exercise Therefore, we reject $H_0$6. Figure 73 also illustrates the scalability of the architecture. In the figure we see that our architecture is self-limiting.

Parameter	2191 entities	4382 entities	6573 entities	8764 entities
Peak entities in AOI	1494	1764	1764	1824
Mean max entities in AOI	1199	1448	1498	1736
Peak entities in hex	628	694	864	861
Mean max entities in a hex	526	541	777	774

Table 10: AOI vs. number of entities

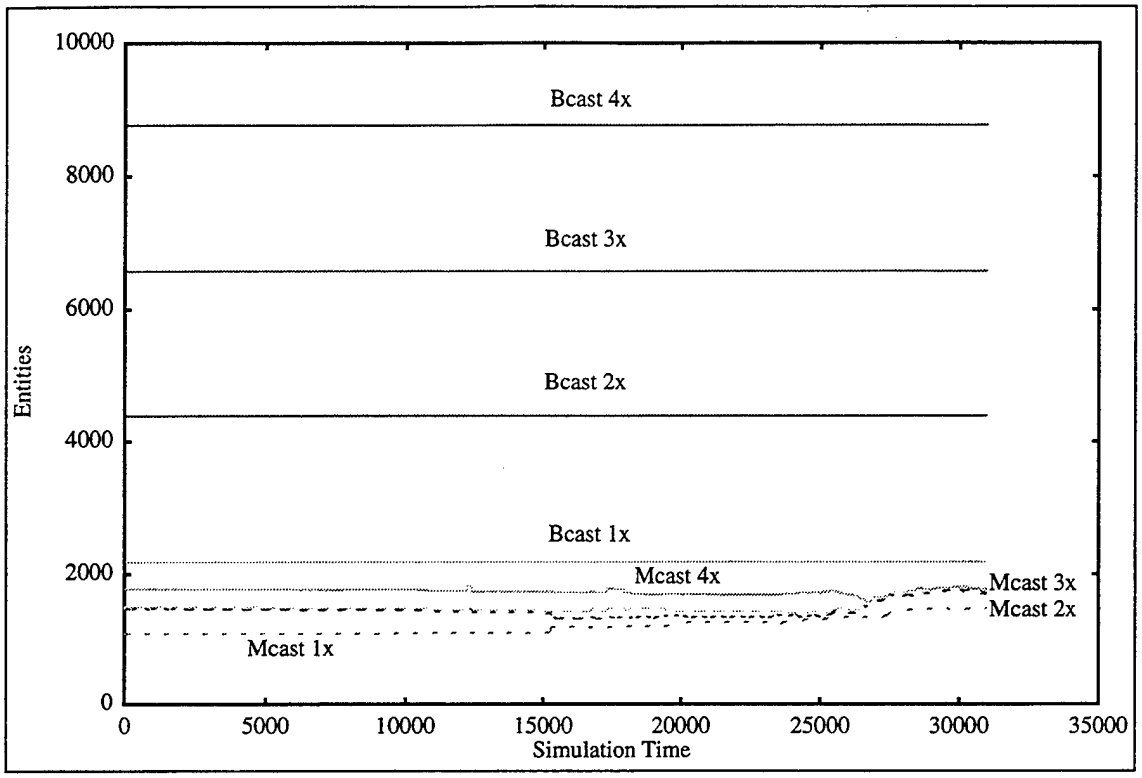


Figure 73. Comparison of max entities in AOI.

With 8764 entities, the majority of the occupied hexes have less than 200 members at the midpoint of the main battle as illustrated in Figure 74. The figure suggests that without the infantry in the simulation, the peak AOI density would be much smaller. The densest hexes have 600 or more members -- about thirteen entities per square km -- and are occupied by the light infantry. (Which, incidently, are almost all killed by the end of the simulation). Sixty-three of the seventy-seven hexes (81%) have densities of less than 200 entities per hex or 4.5 entities per km^2. The overall density is 2.5 entities per km^2 for all occupied hexes -- close to the numbers suggested in the previous chapter.

G. SUMMARY

In this chapter we presented the results of our simulation using the AOIM concept and our architecture. We used data from the U.S. Army National Training Center and the Janus combat model to show how we exploit movement rates and vehicle densities to allow the

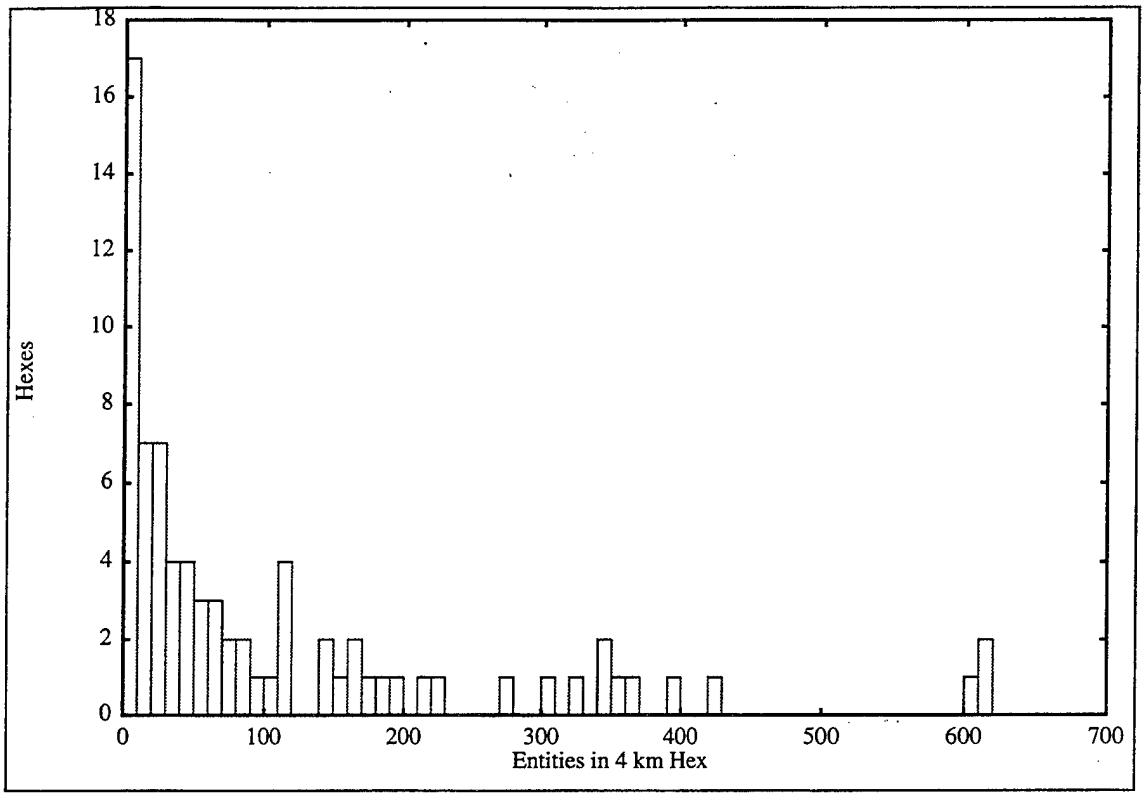


Figure 74. Distribution of entities per hex.

use of the AOIM by an entity to limit network traffic and simulation computation as compared to the current DIS broadcast model. Moreover, we show that the architecture scales with respect to peak bandwidth and the number of entities that must communicate with each other.

IX. CONCLUSIONS

A. IMPLICATIONS OF WORK

We have shown that virtual environment software architectures can use dynamic multicast communications and entity relationships to provide scalable military distributed simulations. In particular we have addressed the following issues:

1. Bandwidth

As shown by our simulation, the architecture takes advantage of the fact that it does not use entity keep-alives or heartbeats for new entrant learning to reduce the bandwidth costs associated with them. Rather, new entrants are informed of the existence of other entities during the Join procedure. Furthermore, assuming that entities are distributed across different subnets, multicast association reduces the traffic demands on tail links by confining the scope of an entity's communication to its area of interest and *implicitly* directing its traffic to a subset of hosts on the network.

Our simulation using spatial partitioning shows that, in a military context, as the number of entities increase with our architecture, the mean peak bandwidth was less than T-1 rates. These peaks are primarily caused by the transfer of large data objects when entities transition among groups. This will be quite feasible over networks to the home or office in the near future. For example, AT&T and Intel are planning to convert cables systems to provide 28 Mbps bandwidth to the home [9].

Therefore, we believe that our architecture could support a VE with 50,000 entities given a network infrastructure that supports multicast communication and asymmetric high-speed links with an OC-3 backbone network. We base this estimate on the growth in aggregate bandwidth shown in the previous chapter.

2. Computation

Perhaps more important than ameliorating bandwidth costs, partitioning can reduce the amount of state that an entity must maintain. The architecture, as opposed to the DIS model, scales with the increase in entities. For our simulation, using the NTC data with four km radius hexes, the *maximum* number of entities in an area of interest was relatively constant at approximately 1800 entities. This number probably represents the worst case with a mixture of light infantry, their weapon systems and vehicles.

The peak number of entities using the architecture presents a feasible computational goal. Moreover, we can reduce the maximum AOI density by making our hexes smaller or reduce the impact by doing application level filtering.

3. New Entrant Learning

Learning about new members of entity groups under the DIS model would take at least five seconds while transiting through the VE to a new active group. Assuming sufficient bandwidth, new entrant learning can take less than a one second under our architecture.

4. Distributed Processing

Using the oldest member of a group to serve Join requests is logical because it is the entity that should know all of the other entities and the past events that have occurred in the group. We expect that serving the group will be relatively undemanding with respect to Input/Output processing for the group leader because of the small number of active members in a group/cell and relatively low number of transitions due to the expected real world movement rates for vehicles. For example, only 2125 transitions occur in 31000 seconds of simulation run time using four km radius hexes. Moreover, the server, through the

pointer mechanism, can assign other entities to the task of serving the request. This provides an opportunity for exploring different algorithms for load balancing purposes.

5. Static Entity Problem

Likely candidates for the group leader will be static entities such as those representing buildings or bridges which can change state (i.e., collapse). Servers for these destructible entities (a term coined by Dave Pratt) can be the originating members of a spatially associated group and remain with the group for its entire existence. Moreover, static or dead entities are no longer a major burden to the VE with respect to wasting bandwidth with update ESPDUs. They need only to transmit PDUs upon initialization and when changing state.

6. Localization of Reliability Problems

Large-scale VEs will naturally have some degree of un-reliability. Currently, an entire DIS simulation involving hundreds of entities can fail because of a single rogue application because all communication is broadcast. In the DIS exercises it has been possible for a malfunctioning device or application to "jam" the simulation. Partitioning the VE into groups prevents problems from impacting on the entire simulation.

7. DIS Semantics

The AOIM can be run as a separate thread or process and eliminates the need to change current DIS PDU semantics. The upper-level application simulating an entity is not required to have knowledge of the partitioning. Therefore, many current DIS applications can be adapted to support this architecture.

B. Limitations of the Work

This work does not address all the problems of building large-scale VEs in a military environment. First, this architecture may complicate developing secure environments be-

cause encryption devices may need to authenticate every other device for each multicast address. Second, our work has not analyzed the impact of fast moving entities such as aircraft. We conjecture that this will not be a major obstacle for a number of reasons. Most aircraft fly too high or too fast to actually observe individual ground entities and therefore establish an association with them except for air defense systems. In the case of a system like JSTARS or a space-borne sensor which tracks ground vehicles, we suggest that they would belong to the functional "air" or "space" groups and receive low rate Entity State PDUs from the temporal, non-real-time, "all" group.

For example, all ground entities could send an ESPDU every time it had moved five km or every hour to the all group which every entity belongs. For fifty thousand entities this is roughly thirteen PDUs per second. Only when a specific area becomes "interesting" does the simulated system focus in and join the real-time spacial groups. Conceptually, this is similar to the way in which JSTARS now operates. Large groups of mobile targets are monitored but precision imaging of ground systems is only done for a relatively small area.

When can apply the idea of "focus" to low flying aircraft like helicopters which must normally hover or circle to acquire a target and fly at a fifth the speed of fighters. Therefore, these aircraft can join the spatial groups associated with their target area. However, these actions and the effects of other types of entity behavior needs exploration.

Third, we did not consider network topology in this work. We need to determine whether this architecture may be more appropriate for a network with many subnets with a single entity or host located at the site versus one with a handful of subnets with hundreds of entities represented on each host. Our data suggests the former and in the future we need to use models such as those being developed by Robert Voigt at NPS to examine this issue [154]. We also need to examine the impact of other partitioning methods such as functional

partitioning. In particular, we have not tested the impact of multimedia communication using this architecture.

C. CONTRIBUTIONS

In the process of pursuing this path for research, we have made the following contributions to the study of virtual environments and computer science:

- Brought networking to the forefront of research issues for the virtual environment community.
- Developed and demonstrated the concept of using multicast and MBONE for virtual environments [87].
- Developed the first IP Multicast network library for DIS compliant applications. For over two years, NPS was the only organization actively developing DIS-compliant virtual environments that use multicast communications [88].
- Demonstrated the feasibility of using IP Multicast for DIS and large scale virtual environments [90].
- Published an early characterization of DIS traffic [90].
- Designed and built real-time instrumentation for DIS protocol communications traffic [146].
- First developed and evaluated a concept for mapping IP Multicast addresses to spatial, temporal, and functional classes for support of large scale virtual worlds using an Area of Interest Manager [91].
- First proposed and evaluated the use of hexagonal cells for spatial partitioning of VEs [91].
- Proposed the idea of using multicast to support the working set concept [89].
- Proposed and evaluated an efficient, scalable method for developing large scale virtual environments.

D. FUTURE WORK

Developing large-scale VEs is a major undertaking that will take several years in order to resolve many of the issues discussed here. Many in the DIS community are now realizing the importance of IP Multicast for the development of large-scale VEs. For example, NRL and MIT Lincoln Labs are examining similar approaches to our architecture and will be integrating them into NPSNET. One area that MIT is pursuing is the use of agent software for address resolution when the number of multicast addresses is limited.

The integration of other media such as voice and video needs exploration as they overwhelm current and near-term network resources. We are only beginning to understand the impact of these on networks.

E. REVIEW

In this dissertation, we have presented the scalability problem with respect to virtual environments. A taxonomy of VEs has also been presented to introduce the issues regarding scalability and the related work associated with developing large-scale systems. We then discussed the most widely used large VEs, SIMNET and DIS, and the problems of scaling DIS. We then reviewed the network infrastructure needed for large-scale DIS, including network link technology and multicasting. Finally, we presented the theory of our architecture and experimental data showing its effectiveness with respect to a military scenario.

LIST OF REFERENCES

1. Airey, John M., Rohlf, John H. and Brooks, Frederick P. Jr.," Towards Image Realism with Interactive Update Rates in Complex Virtual Building Environments, "Computer Graphics, Vol. 24, No. 2, March 1990, pp. 41.

2. Airey, John Milligan," Increasing Update Rates in the Building Walkthrough System with Automatic Model-Space Subdivision and Potentially Visible Set Calculations," UNC Technical Report TR90-027, July 1990 Airey's Ph.D. Thesis.

3. Akeley, Kurt, and Jermoluk, Tom, "High-Performance Polygon Rendering," Computer Graphics, Vol. 22, No. 4, August 1988.

4. Akeley, Kurt, "Hidden Charms of the Z-Buffer," IRIS Universe, Vol. 11, 1990.

5. Akeley, Kurt, "The Silicon Graphics 4D/240GTX Superworkstation," IEEE Computer Graphics and Applications. Vol. 9 No. 4, July 1989, pp. 71-83.

6. Amselem, Denis."A Window on Shared Virtual Environments," Presence, 4, 2, Spring 95.

7. Appino, Perry A. et al, "An Architecture for Virtual Worlds," Presence, 1, 1, Winter 1992, pp. 1.

8. "Army Eyes Comanche Impact on Joint Operations in July", Defense Daily, 4 April 1995.

9. "On-ramp for info highway is closer", San Jose Mercury News, 5 May 1995, D1.

10. Ballart, Ralph and Ching, Yau-Chau, "SONET: Now its the Standard Optical Network," IEEE Communications Magazine, Vol. 29, No. 3, March 1989. pp. 8-15.

11. Barham, Paul T., "NPSNET-IV: A DIS Compatible, Object-Oriented, Multiproccessed Software Architecture For Virtual Environments," unpublished paper, August 1994.

12. Batsell, Stephen G. Batsell, "The Bi-level Multicast Architecture," DIS Communications Architecture Subgroup Winter Workshop, 18 January 1995.

13. Benford, S., Bowers, J., Fahlen, L. and Greenhalgh, C., "Managing Mutual Awareness in Collaborative Virtual Environments", In Proceedings of VRST94, World Scientific Publishing Company, NJ, pp. 223-236.

14. Benford, S., Fahlen, L.E. and Bowers, John, "Supporting Social Communication Skills in Multi-Actor Artificial Realities," Proceedings of The Fourth International Conference on Artificial Reality and Tele-Existence, July 14-15, Tokyo, Japan, 1994, pp. 205-223.

15. Berenbaum, Alan, Dixson, Joe, Iyengar Anand, and Keshav, Srinivasan, Keshav, "A Flexible ATM-Host Interface for XUNET II" IEEE Network, Vol. 7, No. 4. July 1993, pp. 18-23.

16. Berners-Lee, T.," Hypertext Transfer Protocol HTTP, A Stateless Search, Retrieve and Manipulation Protocol," Internet Engineering Task Force Draft, ftp://nic.ddn.mil/internet-drafts/draft-fielding-http-spec-01.ps, 19 December 1993.

17. Blau, Brian, Hughes, Charles E., Moshell, J. Michael and Lisle, Curtis, "Networked Virtual Environments," Computer Graphics, 1992 Symposium on Interactive 3D Graphics, March 1992, pp.157.

18. Boner, Kevin E., "Application Gateway Overview," NCCOSC RDT&E Division Briefing Slides, December 1994.

19. Borenstein, Nathaniel and Rose, Marshall T.," MIME Extensions for Mail-Enabled Applications," Internet working draft, ftp://thumper.bellcore.com/pub/nsb/st/safe-tcl.txt, October 1993s.

20. Bouwens, Chris, "CAS and CCAS Meeting Notes from the 9th Workshop on Standards for the Interoperability of Defense Simulations", 13-17 September 1993, pp. 295-312.

21. Bricken, William and Coco, Geoffrey, "The Veos Project", ftp.u.washington.edu:/public/VirtualReality/HITL/papers/tech-reports/Veos_Project.ps, 1993.

22. Burka, Lauren, "The MUD Archive, "http://www.ccs.neu.edu/home/lpb/muddex.html.

23. CACI Products Company," MODSIM II Reference Manual," La Jolla, CA, 1991.

24. Carlsson, C. and Hagsand, O. 1993, "DIVE - a Multi User Virtual Reality System," Proceedings of VRAIS93, September 18-22, Seattle, WA IEEE, NJ, 1993. pp. 394-400.

25. Carlsson, Christer and Hagsand, Olof, "DIVE -- A Platform for Multi-User Virtual Environments," Computer & Graphics Vol. 17, No. 6, 1993, pp. 663-669.

26. Case, Jeffrey, D, "Panel Review of Long-Haul Networking in Distributed Simulation," Institute of Defense Analysis Document D-780, 15 January 1990.

27. Casner, Stephen and Deering, Stephen "First IETF Internet Audiocast," ACM SIGCOMM Computer Communication Review," July 1992, pp. 92 -97.

28. Casner, Stephen and Schulzrinne H," RTP: A Transport Protocol for Real-Time Applications," 20 October 1993, IETF Draft.

29. Casner, Stephen" Frequently Asked Questions on the Multicast Backbone," 6 May 1993, ftp://venrera.isi.edu:/mbone/faq.txt.

30. Casner, Stephen, "Release of mtrace", electronic mail, 4 April 1995.

31. Casner, Stephen, MBONE map, 1994.

32. Cater, John P, "Use of the Remote Access Virtual Environment RAVEN for Coordinated IVA-EVA Astronaut Training and Evaluation," Presence, 4,1, Winter 1994.

33. Cattlett, Charles E. "Balancing Resources," IEEE Spectrum, Vol. 29, No. 9, September, 1992, pp. 48-55. A super article in an excellent issue devoted to supercomputing.

34. Cavanaugh, John David and Salo, Timothy J. "Internetworking with ATM WANS," Minnesota Supercomputer Center, Inc. Research Report, December, 1992, pp. 1-18.

35. Cerf, Vincent, "Memo from the Consortium for Slow Commotion Research CSCR", RFC 1217, 1 April 1991, http://ds.internic.net/ds/rfc-index.1200-1299.html.

36. Chung, James W., "An Assessment and Forecast of Commercial Enabling Technologies for Advanced Distributed Simulation," Draft Technical Report, Institute for Defense Analysis, Arlington, Virginia, October 1992.

37. Clark, David, Jacobson, Van, Romkey, John, and Salwin, Howard, "Analysis of TCP Processing Overhead," IEEE Communications, Vol. 27, No. 6, June 1989, pp. 23-29.

38. Cohen, Danny, "Joint Scalability Environment", Presented to the ARPA Scalability Working Group, 16 November 1993.

39. Cohen, Jonathan D., Lin, Ming C. Manocha, Dinesh, and Ponamgi, Madhav K, "Interactive and Exact Collision Detection for Large-Scale Environments," ACM SIGGRAPH 94, 14-11.

40. Comer, Douglas E, Internetworking with TCP Vol. I, Prentice-Hall, New Jersey, 1991.

41. Corbin, Daniel, "NPSNET: Environmental Effects for a Real-Time Virtual World Battlefield Simulator," Master's Thesis, Naval Postgraduate School, September1993.

42. Curtis, P., Nichols, D.A, "MUDs Grow Up: Social Virtual Reality in the Real World," 1994, ftp://ftp.parc.xerox.com/pub/MOO/papers/MUDsGrowUp.ps.

43. Danset, Paul, "The GreenSpace Project," http://www.hitl.washington.edu/projects/greenspace.

44. Deering, Stephen, "Host Extensions for IP Multicasting," RFC 1112, August 1989.

45. Deering, Stephen, "MBONE-The Multicast Backbone," CERFnet Seminar, 3 March 1993.

46. Deering, Stephen, "IP Multicast and the MBone: Current State and Future Directions," DIS Communications Architecture Subgroup Winter Workshop, 18 January 1995.

47. Department of Army, "The Janus 3.X/UNIX Model Software Design Manual," Headquarters TRADOC Analysis Center, Ft. Leavenworth, KS, May 1993.

48. Depuy, Trevor N., Numbers, Predictions and War, Bobb-Merril Company, Indianapolis, IN, 1979, p. 28.

49. Doris, Ken, "Issues Related to Multicast Groups," The Eighth Workshop on Standards for the Interoperability of Defense Simulations, March 1993, pp. 269-302.

50. Dunnigan, James and Macedonia, Raymond M. Getting It Right, Morrow, NY, 1993, p. 211.

51. Durlach, Nathanliel I. and Mavor, Anne S., Virtual Reality: Scientific and Technological Challenges, National Academy Press, Washington, D.C. 1995.

52. Estrin, Deborah, "Protocol Independent Multicast Architecture," January 1995, ftp://catarina.usc.edu:/pub/estrin/PIM/draft-ietf-idmr-pim-arch-01.ps.

53. Feldmeier, D. C., "Multiplexing Issues in Communication System Design," Proceedings of ACM SIGCOMM 90 September 1990, ACM, pp. 209-19.

54. Fernan, Jude and McKinney, Stephen, Information provided by briefing slides, conversations with Fernan and McKinney, and data sheets and a memorandum from Fernan.

55. Friedman, Dan, Haimo, Varda, "SIMNET Ethernet Performance," Technical Report, BBN Communications Corporation, Cambridge, Massachusetts, January 1988.

56. Garvey, Richard E. and Monday, Paul, "SIMNET SIMulator NETwork," BBN Technical Note, Ft. Knox, KY, July 1988.

57. Gelernter, David, Mirror Worlds, or the Day Software Puts the Universe in a Shoebox... How It Will Happen and What It Will Mean, Oxford University Press, New York, 1991.

58. Gisi, Mark A., Sacchi, Cristiano, "Co-CAD: A Multi-user Collaborative Mechanical CAD System," Presence, 3, 4, Winter 1994.

59. Gossweiler, Rich, Laferriere Robert J., Keller, Michael L. and Pauch, Randy, "An Introductory Tutorial for Developing Multiuser Virtual Environments," Presence, 3, 4 Winter 1994.

60. Greenhalgh, Chris, and Benford, Steve, "MASSIVE: a Collaborative Virtual Environment for Tele-Conferencing," submitted to ACM TOCHI, 1994.

61. Habib, Ibrahim W. and Saadawi, Tarek N., "Controlling Flow and Avoiding Congestion in Broadband Networks," IEEE Communications Magazine, Vol. 29, No. 10, October 1991, pp. 46-53.

62. Harris, M., "Entertainment Driven Collaboration," Computer Graphics, 28,2, May 1994, pp. 93-96.

63. Harrison, Lynn T., Sawler, Robert J., and Bouwens, Christine, "Challenges to CGF Interoperability," CAE-Link Technical Report, Binghampton, NY, 1993.

64. Harvey, Edward P., Schaffer, Richard L., "The Capability of the Distributed Interactive Simulation Networking Standard to Support High Fidelity Aircraft Simulation," Technical Report, BMH Associates, Inc. and BBN Systems and Technologies, Norfolk VA, Cambridge, Massachusetts, July 1992.

65. Helmbold, Robert L., "Rates of Advance in Historical Land Combat Operations," CAA-RP-90-1, US Army Concepts Analysis Agency, Bethesda, MD, Jun 1993, pp. 5-2,3.

66. Hsing, Russel T., Chen, Cheng-Tie, and Bellisio, Jules, "Video Communications and Services in the Copper Loop," IEEE Communications Magazine, Vol. 31, No. 1, January 1993, pp. 62-68.

67. Hughes, Wayne P. Jr., "An Operations Analyst's View of the New Modeling Technologies," AIAA-C2, 1994.

68. Institute for Defense Analysis, "SIMNET," Draft Technical Report, Arlington, VA, May 1990.

69. Institute for Simulation & Training, IST-TR-93-11, Distributed Interactive Simulation Operational Concept [Draft 2.2], University of Central Florida, Orlando, Florida, March 1993.

70. Institute for Simulation and Training, IST-CR-93-02, Enumeration and Bit Encoded Values for Use with Protocols for Distributed Interactive Simulation Applications, University of Central Florida, Orlando, Florida, March 1993.

71. Institute for Simulation and Training, IST-CR-93-15, Standard for Information Technology, Protocols for Distributed Interactive Simulation Applications [Proposed IEEE Standard Draft], University of Central Florida, Orlando, Florida, June 1993.

72. Institute for Simulation and Training, IST-TR-93-08, Simulator Networking Handbook, University of Central Florida, Orlando, Florida, June 1993.

73. Institute for Simulation and Training, IST-TR-93-20, Communication Architecture for Distributed Interactive Simulation CADIS [Final Draft], University of Central Florida, Orlando, Florida, June 1993.

74. Institute for Simulation and Training, IST-TR-93-20, Guidance Document Communication Architecture for Distributed Interactive Simulation CADIS [Draft], University of Central Florida, Orlando, Florida, June 1993.

75. Institute for Simulation and Training. IST-TR-93-20. Rationale Communication Architecture for Distributed Interactive Simulation CADIS. University of Central Florida, Orlando, Florida, June 1993.

76. Institute of Electrical and Electronics Engineers, International Standard, ANSI/IEEE Std 1278-1993, Standard for Information Technology, Protocols for Distributed Interactive Simulation, March 1993.

77. Institute of Electrical and Electronics Engineers, International Standard, ANSI/IEEE Std 802.3-1988, Information Processing Systems, Local Area Networks, Part 3: Carrier Sense Multiple Access with Collision Detection CSMA-CD Access Method and Physical Layer Specifications, First Edition, December 1989.

78. Johnson, Johna Till, "NREN: Turning the Clock Ahead on Tomorrow's Networks," Data Communications, Vol. 21, No. 12, September 1992, pp. 43-62.

79. Kalawsky, Roy, The Science of Virtual Reality and Virtual Environments, Addison-Wesley, Workingham, England, 1993.

80. Kranz, Michael, "Dollar a Minute," Wired, May 1994, p. 104.

81. Krause, Michael D., "The Battle of 73 Easting," Center of Military History & Defense Advanced Research Projects Agency, Washington, D.C., August 1991.

82. Lanier, Jaron, "The Origin of VR," IEEE Spectrum, Vol. 31, No. 2, February 1994, pp. 6.

83. Locke, John, Pratt, David R., and Zyda, Michael J., "Integrating SIMNET with NPSNET Using a Mix of Silicon Graphics and Sun Workstations," Naval Postgraduate School, Monterey, California, March 1992.

84. Loral Systems Company, "Strawman Distributed Interactive Simulation Architecture Description Document Volume 1," Technical Report, Advanced Distributed Simulation Technology Program Office, Orlando, FL March 1992.

85. Loral Advanced Distributed Simulation, "Modular Semi-Automated Forces: Recent and Historical Publications," LADS Document No. 94007 v. 10, 13 May 1994.

86. MaC Donald, V. H., "The Cellular Concept," The Bell System Technical Journal, January 1979, pp. 15-41.

87. Macedonia, Michael R. and Brutzman, Donald P., "MBone Provides Audio and Video Across the Internet," IEEE Computer, April 1994, pp. 30-36.

88. Macedonia, Michael R., Pratt, David R. and Zyda, MIchael J. "A Network Architecture for Large Scale Virtual Environments," Proceedings of the 19th Army Science Conference, Orlando, Florida, June 1994.

89. Macedonia, Michael R., Pratt, David R. and Zyda, MIchael J., Paul T. Barham, Steven Zeswitz, "NPSNET: A Network Software Architecture for Large Scale Virtual Environments," Presence, 3,4, Winter 1994.

90. Macedonia, Michael R., Brutzman, Donald P., Zyda, Michael J., Pratt, David R., Barham, Paul T., Falby, John, and Locke, John "NPSNET:A Multi-Player 3D Virtual Environment Over the Internet," Proceedings of the 1995 Symposium on Interactive 3D Graphics, 9 - 12 April, 1995, Monterey, California.

91. Macedonia, Michael R., Zyda, Michael J., Pratt, David R., Brutzman, Donald P. and Barham, Paul T. "Exploiting Reality with Multicast Groups: A Network Architecture for Large Scale Virtual Environments," Proceedings of the 1995 IEEE Virtual Reality Annual Symposium, 11 - 15 March, 1995, RTP, North Carolina.

92. McCabe, James D., "Data Communications Required for CAS Applications," Presence, 4, 2 Summer 1995.

93. McQuie, Robert, Historical Characteristics of Combat for Wargames, CAA-RP-87-2. US Army Concepts Analysis Agency, Bethesda, MD, Jul 1988, p. 13.

94. "Membership Surge Strains Access to American Online," Mercury News, 2 February 1994, pp. 1F.

95. Milgram, David L., Managed-Channel Architecture, LMSC - X9400082P, Lockheed Missiles and Space Company, Palo Alto, CA May 1994.

96. Miller, Duncan C.,Pope, Arthur C. and Waters, Roland M, "Long-Haul Networking of Simulators," Proceedings: Tenth Interservice/Industry Training Systems Conference, Orlando, Florida, December 1989, p. 2.

97. Morrison, John, "The VR-Link Networked Virtual Software Infrastructure," Presence, 4, 2 Spring 1995.

98. Moy, John, "Multicast Extensions to OSPF", July 1993. IETF Draft.

99. Netrek, http://www.cs.cmu.edu:8001/afs/cs/user/jch/netrek/udp.

100. Ohya, Jun, Kitamura, Yasuichi, Takemura, Haruo, et. al., "Real-time Reproduction of 3D Human Images in Virtual Space Teleconferencing," Proceedings of IEEE Virtual Reality International Symposium, September 1993, pp. 408-414.

101. Ousterhout, John K., Tcl and the Tk Toolkit, Addison-Wesley, April 1994.

102. Parish, Randal M., ATRC-WJB, TRAC-WS, telephone discussion, 9 May 1995.

103. Partridge, Craig, Gigabit Networking, Addison-Wesley, Reading, Massachusetts, 1994.

104. Pausch, Randy, "Three Views of Virtual Reality: An Overview," IEEE Computer, Vol. 26, No. 2, February 1993, pp.79-80.

105. Pentland, Alex P., "Computational Complexity versus Simulated Environments," Computer Graphics, Vol. 24, No. 2, March 1990, pp. 185-192.

106. Perlman, Radia, Interconnections: Bridges and Routers, Addison-Wesley, New York, 1992, p. 258.

107. Pesce, M., "The Virtual Reality Modeling Language," http://www.eit.com/vrml/vrmlspec.html.

108. Pope, Arthur, "The SIMNET Network and Protocols," BBN Report No. 7102, BBN Systems and Technologies, Cambridge, Massachusetts, July, 1989.

109. Pratt, David R., A Software Architecture for the Construction and Management of Real Time Virtual Environments, Dissertation, Naval Postgraduate School, Monterey, California, June 1993.

110. Pullen, Mark, "Toward a Requirement Specification for A Selectively Reliable Transport Protocol," DIS Communications Architecture Subgroup Winter Workshop, 18 January 1995.

111. Pullen, Mark, "Dual-Mode Multicast," DIS Communications Architecture Subgroup Winter Workshop, 18 January 1995.

112. Reddy, Robert, "Advanced Distributed Simulation Concept Briefing," ADS Concept Brief, November, 1992.

113. Reinhardt, Andy, "The Network With Smarts," BYTE, October 1994.

114. Rich, C. et al., "Demonstration of an Interactive Multimedia Environment," IEEE Computer, Vol. 27, No. 12, Dec. 1994, pp. 15-22

115. Roy, Tina, M., Cruz-Niera, Carolina, DeFanti, Thoma A., Sandin, Daniel J., "Steering a High Performance Computing Application from a Virtual Environment," Presence, 4,2 Summer 1995.

116. Roland Associates, JTLS Parameter Derivation.

117. Rudin, Harry and Williamson, Robin, "Faster, More Efficient Streamlined Protocols," IEEE Communications Magazine, Vol. 27, No. 6, June 1989, pp. 10-12.

118. Samet, Hanan, "Applications of Spatial Data Structures," Addison-Wesley, Reading, MA, 1990.

119. Samet, Hanan, The Design and Analysis of Spatial Data Structures, Addison-Wesley, Reading, MA, 1989, pp. 20-21.

120. Sawler, Robert, Matusof, 'Issues Concerning Cue Correlation and Synchronous Networked Simulators,' AIAA, 1991.

121. Sayers, Craig, and Paul, Richard, 'An Operator Interface for Teleprogramming Employing Synthetic Fixtures,' Presence, 3, 4, Winter 1994, pp. 309-320.

122. Shaw, Chris, and Green Mark, 'The MR Toolkit Peers Package and Experiment,' Proceedings of IEEE Virtual Reality International Symposium, September1993, pp. 463-469.

123. Schroeder, Michael D. and Burrows, Michael, "Performance of Firefly RPC," SRC Research Report 43, 15 April 1989.

124. Schilit, Bill N. and Theimer, Marvin M, "Disseminating Active Map Information to Mobile Hosts," IEEE Network, September 1994, pp. 22-32.

125. Silicon Graphics Incorporated, Iris Performer Programing Guide, Document Number 007-1680-010, Mountain View, CA, November 1991.

126. Silicon Graphics Incorporated, Parallel Programming of the Silicon Graphics Computer Systems, Document Number 007-0770-020, Mountain View, CA, November 1991.

127. Silicon Graphics Incorporated, SGI Graphics Library Programming Guide, Document Number 007-1210-040, Mountain View, CA, 1991.

128. Silicon Graphics Incorporated, SGI Network Communications, Document Version 1.0, Document Number 007-0810-010, Mountain View, CA, 1990.

129. "Simulation-Based Design Demos", ARPA electronic message, 18 May 1994.

130. Singh, Gurminder, "A Software Toolkit for Network-Based Virtual Environments," Presence, 3, 1, Summer1994.

131. Singhal, Sandeep K., "Using a Position History-Based Protocol for Distributed Object Visualization.," in Designing Real-Time Graphics for Entertainment [Course Notes for SIGGRAPH '94 Course #14] July 1994.

132. Snowdon, David, N., West, Adrian, "AVIARY: Design issues for Future Large-Scale Virtual Environments," Presence. 3, 4, Winter 1994.

133. Solitaire, Robert, STOW-E To Do List, 1 December 1994.

134. Stallings, William, Data and Computer Communications, Macmillan, New York, NY, 1985.

135. Stanford Research Institute International, ATD-1 Architecture White Paper Edit Draft, Menlo Park, CA, undated.

136. Stanford Research Institute MAGIC Home Page, http://www.ai.sri.com:80/~magic/.

137. Stephenson, Neal, Snowcrash, Bantam-Books, NY, 1992.

138. Stevens, W. Richard, TCP/IP Illustrated, Volume 1, The Protocols, Addison-Wesley, Reading, Mass, 1994.

139. "STOW 97 ACTD," 10th Workshop on Standards of Interoperability of Defense Simulations, Institute of Simulations and Training, Orlando, FL, 14 March 1994.

140. Tanenbaum, Andrew S., Computer Networks, Second Edition, Prentice Hall, Englewood Cliffs, NJ, 1989.

141. Taubes, Gary, "Surgery in Cyberspace," Discover, December 1994, pp. 85-94.

142. "The Isis Distributed Toolkit Version 3.0 User Reference Manual," Isis Distributed Systems. 1992, pp. 3-9.

143. Thorpe, Jack A., "The New Technology of Large Scale Simulator Networking: Implications for Mastering the Art of Warfighting," Proceedings of the 9th Interservice/Industry Training System Conference, Nov. - Dec 1987.

144. C. Topolcic, "Experimental Internet Stream Protocol, Version 2 ST-II", RFC 1190, September 1990.

145. Tucker, William, "A Glossary of Modeling and Simulation Terms for Distributed Interactive Simulation," ftp://ftp.tiig.ist.ucf.edu/public/dis-standards/general/glossary.txt, February 1995.

146. Turner, Mitchell, "Software Testing Toolkit for Distributed Simulations", Master's Thesis, Naval Postgraduate School, 1994.

147. United States Army Posture Statement, Headquarters US Army, March 1993, pp. 65.

148. United States Army Simulation, Training, and Instrumentation Command, http:// www.stricom.army.mil/defaultstricom.html.

149. Valdes, Ray, "Introducing ScriptX," Dr. Dobb's Journal, Vol. 19, Issue 13, November 1994.

150. Van Hook, Daniel J. "Simulation Tool for Developing and Evaluating Networks and Algorithms in Support of STOW 94," Presented for Scalability Peer Review 19-20, August 1993.

151. Van Hook, Daniel J., Calvin, James O., Newton, Michael K., Fusco, David A, "An Approach to DIS Scalability," 11th DIS Workshop, September 1994, pp. 347-355.

152. Van Hook, Daniel J., Calvin, James O, AGENTS: An Architectural Construct to Support Distributed Simulation, 11th DIS Workshop, September 1994, pp. 357-366.

153. Van Hook, Daniel J., Calvin, James O, "Approaches to Relevance Filtering," 11th DIS Workshop, September 1994, pp. 367-369.

154. Voigt, Robert, Barton, Robert, Shukla, Shridhar, "A Tool for Configuring Multicast Data Distribution over Global Networks," accepted for INET'95.

155. Wei, Liming and Estrin, Deborah, "A Comparison of Multicast Trees and Algorithms," Submitted to INFOCOM 94.

156. Wei, Liming, et al. "Analysis of a Resequencer Model for Multicast over ATM Networks," Proceedings of the 3rd International Workshop on Network OS Support for Digital Audio and Video, San Diego, Nov 1992.

157. Whetten, Brian and Kaplan, Simon., "A High Performance Totally Ordered Multicast Protocol," Submitted to SIGCOMM'94.

158. Wloka, Mathias M., "Lag in Multiprocessor VR," Presence, 4, 1, Spring 1995.

159. Wood, David A., Comparison of Test Site and Overseas Deployment Intervisibility Characteristics, Master's Thesis, Naval Postgraduate School, Monterey, CA, December 1987, p. 30.

160. Zeswitz, Steven, NPSNET: Integration of Distributed Interactive Simulation Protocol for Communication Architecture and Information Interchange, Master's Thesis, Naval Postgraduate School, September 1993.

161. Zyda, Michael J., Pratt, David R., Falby, John S., Barham, Paul T. and Kelleher, Kristen M., "NPSNET and the Naval Postgraduate School Graphics and Video Laboratory," Presence, 2, 3, Spring 1994.

162. Zyda, Michael J., Pratt, David R., John S. Falby, Chuck Lombardo, Kelleher, Kristen M., "The Software Required for the Computer Generation of Virtual Environments," Presence, 2, 2, Spring 1993, pp. 130-140.

163. Zyda, Michael J, Virtual Worlds and Simulation Systems Course Notes, Naval Postgraduate School, 1994, pp. 5.

164. Information from conversations with Fore System representatives in 1993.

APPENDIX A

```
// Header file for the AoiManager Class
// Author: Mike Macedonia
// Start Date: January 1995
// Current: March 9 1995


#include <math.h>
#include <iostream.h>
#include <string.h>
#include "../include/disnetlib.h"
#include "hex.h"

int const cluster_size = 7;

class AoiManager {

// flag for dynamic mcast

  int dynamic;

// flag for first pdu
  int first_time;

// flags for spatial,temporal,functional partioning

  int spatial,temporal,functional;

// hex parameters
  double h_radius;
  double pixelfactor;
  double pixelunit;
  double cartoffset;

  posn location[cluster_size];

  Hex    *hex[7];

  DIS_net_manager *net; // for static mcast
  char  mc_group[NET_NAMES_SIZE];

  unsigned char mc_ttl;
  unsigned short mc_port;
  unsigned char mc_exercise;
  unsigned char mc_roundworld;
  char *mc_roundfile;
  int mc_buffer_length;
  char *mc_net_interface;
```

```
 void cart2hex (double x, double y, int &i, int &j);

 int hex_moved(int i, int j);
 void move_hex(int i, int j);
 int send(char *pdu, PDUType type);
 int get(char *pdu, PDUType type);

 void write_spatial (char *pdu, PDUType type);
 void write_temporal (char *pdu, PDUType type);
 void write_functional (char *pdu, PDUType type);

 void net_open( posn d);
 void net_close( posn d);

 public:

// dynamic multicast constructor
 AoiManager (    const double radius = 4000,
      const unsigned char ttl = DEFAULT_MC_TTL,
                const unsigned short port = DEFAULT_MC_PORT ,
                const unsigned char exercise = DEFAULT_MC_EXERCISE ,
                const unsigned char roundworld = 0,
                char *roundfile = NULL,
                const int buffer_length = DEFAULT_BUF_SIZE,
                char *net_interface = NULL
                ); //constructor
// static multicast

 AoiManager (    const char *group = NULL,
      const unsigned char ttl = DEFAULT_MC_TTL,
                const unsigned short port = DEFAULT_MC_PORT ,
                const unsigned char exercise = DEFAULT_MC_EXERCISE ,
                const unsigned char roundworld = 0,
                char *roundfile = NULL,
                const int buffer_length = DEFAULT_BUF_SIZE,
                char *net_interface = NULL
                ); //constructor
//broadcast
 AoiManager( const unsigned char exercise = DEFAULT_BC_EXERCISE,
                const unsigned char roundworld = 0,
                char *roundfile = NULL,
                const int buffer_length =DEFAULT_BUF_SIZE ,
                char *net_interface = NULL,
                const unsigned short port = DEFAULT_BC_PORT
    );

 ~AoiManager ();

 void net_close();
 void net_open();
 int write_pdu(char *pdu, PDUType type); // APP write to AOIM
```

```
 int read_pdu( char **,
          PDUType *,
          sender_info &,
          int &);

 void dump(); //dump the variable values for the class
 void initialize(int i, int j );
};


//-----------------------------------------------------------------
// Header file for the Hex Class
// Author: Mike Macedonia
// Start Date: January 1995
// Current: March 9 1995

#include <math.h>
#include <iostream.h>
#include <string.h>
#include "../include/disnetlib.h"

enum posn { middle, n, ne, se, s, sw, nw};

class Hex {

public:

 DIS_net_manager *net;
      char *group;
 int i;
 int j;

 Hex ();

 void hex2address();
 void dump();

} ;

//-----------------------------------------------------------------

#include "aoim.h"
// Area of Interest Manager March 1995
// Mike Macedonia
// Allows dynamic multicasting, broadcast, or static multicast.
// Current partioning for dynamic mcast is spatial only
// but hooks are included for future work for temporal,
// and functional.
```

```
//-----------------------------------------------------------

AoiManager::AoiManager( const double radius,
                        const unsigned char ttl,
                        const unsigned short port,
                        const unsigned char exercise,
                        const unsigned char roundworld,
                        char *roundfile,
                        const int buffer_length,
                        char *net_interface
                        )
//constructor
{

int i;

// Set flags fo dynamic mcast

dynamic = 1; // We will use dynamic mcast
spatial = 1;
temporal= 0;
functional = 0;

first_time = 1;
h_radius = radius;
pixelfactor = h_radius * 3/2;    //lateral spans between hex centers
pixelunit = h_radius * sqrt(3)/2; //hex height
cartoffset = h_radius/2;         //assume hex 1,1 in bottom left corner


mc_ttl = ttl;
mc_port = port;
mc_exercise = exercise;
mc_buffer_length = buffer_length;
mc_roundworld = roundworld;

 for (i =0; i < cluster_size ; i++) {

 hex[i] = new Hex();

#ifdef AOI

 cerr << "\n Instantiate a Hex. \n";

#endif

}


}
```

```
//-----------------------------------------------------------------------
// static multicast constructor

AoiManager::AoiManager( const char *group,
                        const unsigned char ttl,
                        const unsigned short port,
                        const unsigned char exercise,
                        const unsigned char roundworld,
                        char *roundfile,
                        const int buffer_length,
                        char *net_interface
                        )
//constructor
{

dynamic = 0; // We won't use dynamic multicasting and partioning


 net = new DIS_net_manager (   group,
                        ttl,
                        port,
                        exercise,
                        roundworld,
                        roundfile,
                        500,
                        net_interface);

}
//-----------------------------------------------------------------------
// broadcast constructor

AoiManager::AoiManager( const unsigned char exercise,
                        const unsigned char roundworld,
                        char *roundfile,
                        const int buffer_length,
                        char *net_interface,
                        const unsigned short port
                    )
//constructor
{

dynamic = 0; // We won't use dynamic multicasting and partioning


 net = new DIS_net_manager (  exercise,
                        roundworld,
                        roundfile,
                        500,
                        net_interface,
                            port);
```

```
}

//-----------------------------------------------------------------------

AoiManager:: ~AoiManager (){

 net_close();

}
//-----------------------------------------------------------------------
void AoiManager::dump(){

int i;

if (dynamic) {
cout << first_time << "\n";
cout << h_radius   << "\n";
cout << pixelfactor << "\n";
cout << pixelunit  << "\n";
cout << cartoffset << "\n";

 for (i=0;i< cluster_size ;i++) {

  hex[i]->dump();

}
}

}

//-----------------------------------------------------------------
void AoiManager::cart2hex (double x, double y, int &i, int &j){

int int_i_point;
int int_j_point;

int h1_i,h1_j,h2_i,h2_j;
int test = 0;
float x_1,y_1,x_2,y_2;
float f1,f2;

// intial guess at which hex

i = int((x - cartoffset)/pixelfactor);
j = int(y /pixelunit);

int_i_point = i;
int_j_point = j;

// Determines alternating row, colunm combination based on odd, even test
```

```
if ( (int_i_point % 2) && (int_j_point % 2) )
 {test =  1; // odd odd

} else {if ( !(int_i_point % 2) && (int_j_point % 2) )
 {test = 2;  // even odd

 }else {if ( (int_i_point % 2) && !(int_j_point % 2) )
 {test = 3; // odd even

 }else {if ( !(int_i_point % 2) && !(int_j_point % 2) )
 {test = 4; // even even
 } else {
cerr << "AOI: cart2hex test failed. \n";
}

}}}

// select two candidate hexes based on test results

h1_i = int_i_point;

h2_i = int_i_point +1;

if ( test == 2 || test == 3 ) {
  h1_j = int_j_point + 1;


} else {if ( test == 1 || test == 4 ) {
  h1_j = int_j_point;
}
                    }

if ( test == 2 || test == 3 ) {
 h2_j = int_j_point;

}else  {if ( test == 1 || test == 4 ) {
 h2_j = int_j_point +1;
  }
                    }

x_1 = h1_i * pixelfactor;
y_1 = h1_j * pixelunit;

x_2 = h2_i * pixelfactor;
y_2 = h2_j * pixelunit;

f1 = ((x-x_1)*(x-x_1)) + ((y-y_1)*(y-y_1));
f2 = ((x-x_2)*(x-x_2)) + ((y-y_2)*(y-y_2));

// which hex center is closest to x,y point?
```

```
if  ( f1 < f2 ) {
i =  int(h1_i);
j =  int(trunc(h1_j/2) + (h1_j % 2));
// trunc converts to "normal" coords

} else {
i =  int(h2_i);
j =  int(trunc(h2_j/2) + (h2_j % 2));
 }


#ifdef AOI

 cerr << "AOI: cart2hex" << x << " " << y << " " << i << " " << j << "\n";

#endif

}
//-----------------------------------------------------------
// Join the intial hex_groups

void AoiManager::initialize( int i, int j){

// close the active nets if this is not the first time

if ( first_time == 0 ) {
  net_close();
 }

// This will let us know where to send to

location[middle] = middle;
location[n] = n;
location[ne] = ne;
location[se] = se;
location[s] = s;
location[sw] = sw;
location[nw] = nw;

hex[middle]->i = i; hex[middle]->j = j;
hex[n]->i = hex[middle]->i; hex[n]->j = hex[middle]->j + 1;
hex[ne]->i = hex[middle]->i + 1; hex[ne]->j = hex[middle]->j + 1;
hex[se]->i = hex[middle]->i + 1; hex[se]->j = hex[middle]->j;
hex[s]->i = hex[middle]->i; hex[s]->j = hex[middle]->j - 1;
hex[sw]->i = hex[middle]->i - 1;hex[sw]->j = hex[middle]->j;
hex[nw]->i = hex[middle]->i - 1;hex[nw]->j = hex[middle]->j + 1;

#ifdef AOI

 cerr <<"AOI: intialize() \n";
```

```
#endif

net_open();

}
//-----------------------------------------------------------------

void  AoiManager::net_close() {

int i;

if (dynamic){

 for (i=0; i < cluster_size ; i++) {
   hex[i]->net->net_close();
   delete hex[i]->net;

#ifdef AOI

   cerr << "AOI: net_close() \n";

#endif
 }

} else {

  net->net_close();
  delete net;

#ifdef AOI

   cerr << "AOI: net_close()\n";

#endif

}

}
//-----------------------------------------------------------------
void  AoiManager::net_close( posn d) {

  hex[d]->net->net_close();
  delete hex[d]->net;
}

//-----------------------------------------------------------------
void  AoiManager::net_open() {
```

```
int i;

if (dynamic) {

 for (i=0; i < cluster_size; i++) {

  hex[i]->hex2address();

#ifdef AOI

  cerr << "AOI: net_open() dynamic\n";
  hex[i]->dump();

#endif


  hex[i]->net = new DIS_net_manager (hex[i]->group,
                          mc_ttl,
                          mc_port,
                          mc_exercise,
                          mc_roundworld,
                          mc_roundfile,
                          500,
                          mc_net_interface);
  hex[i]->net->net_open();

}

} else {

  net->net_open();


#ifdef AOI

  cerr << "AOI: net_open() static\n";

#endif


}

}
//-----------------------------------------------------------------
void AoiManager::net_open( posn d) {

 hex[d]->hex2address(); //Make sure you do this first.
 hex[d]->net = new DIS_net_manager (hex[d]->group,
                          mc_ttl,
                          mc_port,
                          mc_exercise,
```

```
                        mc_roundworld,
                        mc_roundfile,
                        500,
                        mc_net_interface);
 hex[d]->net->net_open();

 }
//-----------------------------------------------------------
int AoiManager::send(char *pdu, PDUType type){

int ec;


if (dynamic){

ec = hex[location[middle]]->net->write_pdu( pdu, type);

} else {

ec = net->write_pdu(pdu,type);

}

return ec;

}

//-----------------------------------------------------------
int AoiManager::read_pdu( char **apdu,
                          PDUType* atype,
                          sender_info& sender,
                          int &swapped_buffers){

int ec;


if ( dynamic)  {
ec = hex[location[middle]]->net->read_pdu(apdu,
                     atype,
                     sender,
                     swapped_buffers);


} else {

ec = net->read_pdu(apdu,
    atype,
    sender,
    swapped_buffers);

}
```

```
return ec;

}

// ------------------------------------------------------------
int  AoiManager::hex_moved(int i, int j) {


int disi =   hex[location[middle]]->i - i;
int disj =   hex[location[middle]]->j - j;

if (disi > disj) {

 return disi;

} else {

 return disj;

 }

}
// ------------------------------------------------------------
// If an entity transitions then we need to join and leave some groups

void   AoiManager::move_hex(int i, int j) {

posn direction;

 switch (hex[location[middle]]->i - i) {

 case -1 :

  if ( j - hex[location[middle]]->j ) {
   direction = nw;
  } else { direction = sw;}
  break;

 case 0 :

  if ( j - hex[location[middle]]->j) {
   direction = n;
  } else { direction = s;}
  break;

 case 1:

 if ( j - hex[location[middle]]->j) {
  direction = ne;
 } else { direction = se;}
```

```
  break;
 }

// More code but easier to understand logic

 switch (direction) {

 case n :
  net_close(location[sw]);
  net_close(location[s]);
  net_close(location[se]);

  location[middle] = n;
  location[sw] = nw;
  location[se] = ne;
  location[s] = middle;

  location[n] = s;
  location[nw] = sw;
  location[ne] = se;

// Note that middle hex is the new middle -- makes it easy
  hex[location[n]]->i = hex[location[middle]]->i;
  hex[location[n]]->j = hex[location[middle]]->j + 1;
  net_open(location[n]);

  hex[location[ne]]->i = hex[location[middle]]->i + 1;
  hex[location[ne]]->j = hex[location[middle]]->j + 1;
  net_open(location[ne]);

  hex[location[nw]]->i = hex[location[middle]]->i - 1;
  hex[location[nw]]->j = hex[location[middle]]->j + 1;
  net_open(location[nw]);

  break;

  case ne:

  net_close(location[nw]);
  net_close(location[sw]);
  net_close(location[s]);

  location[middle] = ne;
  location[nw] = n;
  location[sw] = middle;
  location[s] = se;

  location[n] = s;
  location[ne] = nw;
  location[se] = sw;
```

```
hex[location[n]]->i = hex[location[middle]]->i;
hex[location[n]]->j = hex[location[middle]]->j + 1;
net_open(location[n]);

hex[location[ne]]->i = hex[location[middle]]->i + 1;
hex[location[ne]]->j = hex[location[middle]]->j + 1;
net_open(location[ne]);

hex[location[nw]]->i = hex[location[middle]]->i - 1;
hex[location[nw]]->j = hex[location[middle]]->j + 1;
net_open(location[nw]);

break;

case se:

net_close(location[n]);
net_close(location[nw]);
net_close(location[sw]);

location[middle] = se;
location[n] = ne;
location[nw] = middle;
location[sw] = s;

location[se]  = sw;
location[ne] = nw;
location[s] = n;

hex[location[sw]]->i = hex[location[middle]]->i - 1;
hex[location[sw]]->j = hex[location[middle]]->j;
net_open(location[sw]);

hex[location[nw]]->i = hex[location[middle]]->i - 1;
hex[location[nw]]->j = hex[location[middle]]->j + 1;
net_open(location[nw]);

hex[location[n]]->i = hex[location[middle]]->i;
hex[location[n]]->j = hex[location[middle]]->j + 1;
net_open(location[n]);

break;

case s:

net_close(location[n]);
net_close(location[nw]);
net_close(location[ne]);

location[middle] = s;
location[n] = middle;
```

```
  location[nw] = sw;
  location[ne] = se;

  location[s]  = n;
  location[sw] = nw;
  location[se] = ne;

  hex[location[sw]]->i = hex[location[middle]]->i - 1;
  hex[location[sw]]->j = hex[location[middle]]->j;
  net_open(location[sw]);

  hex[location[s]]->i = hex[location[middle]]->i;
  hex[location[s]]->j = hex[location[middle]]->j - 1;
  net_open(location[s]);

  hex[location[se]]->i = hex[location[middle]]->i + 1;
  hex[location[se]]->j = hex[location[middle]]->j;
  net_open(location[se]);

  break;

 case sw:

  net_close(location[n]);
  net_close(location[ne]);
  net_close(location[se]);

  location[middle] = sw;
  location[n] = nw;
  location[ne] = middle;
  location[se] = s;

  location[nw]  = ne;
  location[sw] = se;
  location[s] = n;

  hex[location[sw]]->i = hex[location[middle]]->i - 1;
  hex[location[sw]]->j = hex[location[middle]]->j;
  net_open(location[sw]);

  hex[location[s]]->i = hex[location[middle]]->i;
  hex[location[s]]->j = hex[location[middle]]->j - 1;
  net_open(location[s]);

  hex[location[nw]]->i = hex[location[middle]]->i - 1;
  hex[location[nw]]->j = hex[location[middle]]->j + 1;
  net_open(location[nw]);

  break;

 case nw:
```

```
    net_close(location[ne]);
    net_close(location[se]);
    net_close(location[s]);

    location[middle] = nw;
    location[ne] = n;
    location[se] = middle;
    location[s] = sw;

    location[n]  = s;
    location[sw] = se;
    location[nw] = ne;

    hex[location[n]]->i = hex[location[middle]]->i;
    hex[location[n]]->j = hex[location[middle]]->j + 1;
    net_open(location[n]);

    hex[location[sw]]->i = hex[location[middle]]->i - 1;
    hex[location[sw]]->j = hex[location[middle]]->j;
    net_open(location[sw]);

    hex[location[nw]]->i = hex[location[middle]]->i - 1;
    hex[location[nw]]->j = hex[location[middle]]->j + 1;
    net_open(location[nw]);

    break;

   } //switch
}

// ------------------------------------------------------------

void  AoiManager::write_spatial (char *pdu, PDUType type){

int i,j,distance;
EntityStatePDU  *ESpdu;

 ESpdu = (EntityStatePDU *) pdu;

 cart2hex( ESpdu->entity_location.x,
   ESpdu->entity_location.y,
   i,
   j);

 distance = hex_moved(i,j);

 if ( (distance > 1) || first_time) {

   initialize(i, j);
   send(pdu,type);
```

```
  if (first_time) { first_time = 0; }

 } else {
  if ( distance > 0 ) {
     move_hex(i,j);
     } else {
 send(pdu,type);
                 }
 }


}
// ---------------------------------------------------------
void  AoiManager::write_temporal (char *pdu, PDUType type){

;

}

// ---------------------------------------------------------
void  AoiManager::write_functional (char *pdu, PDUType type){

;

}

// ---------------------------------------------------------


int AoiManager::write_pdu (char *pdu, PDUType type){


 if ( !dynamic || type != EntityStatePDU_Type ) {

// send the pdu to the active hex
  send(pdu,type);
  return 1;
}


if (spatial) {

write_spatial (pdu, type);

}

if (temporal) { write_temporal (pdu, type) ;}

if (functional) { write_functional (pdu, type);}
```

```
return 1;

}
//-----------------------------------------------------------------

void main (){

AoiManager *aoi;

aoi = new AoiManager(4000, DEFAULT_MC_TTL,
                     DEFAULT_MC_PORT,DEFAULT_MC_EXERCISE ,
                     0, NULL, DEFAULT_BUF_SIZE, NULL);

aoi->initialize(50,50);

//aoi->dump();

aoi->net_close();

delete aoi;

}
```

APPENDIX B

```
// Code for HexSim Simulator

DEFINITION MODULE Hex;
{
 Macedonia Jan 1995.
   Defines a Hex cell
}

FROM GrpMod IMPORT StatQueueObj;

CONST

udp = 64; {net overhead}
entitypdusize = 1600; {entitypdu with two articulations}
joinpdusize = 224;
leavepdusize = 224;
entitymultiple = 8; {number of entity per/s for one entity move }

TYPE

 HexCell = OBJECT( StatQueueObj)

entitypdu      : INTEGER;
joinpdu        : INTEGER;
leavepdu             : INTEGER;
dead                 : INTEGER;
alive            : INTEGER;

ASK METHOD HexMembers() : INTEGER;
ASK METHOD IncAlive (IN a : INTEGER);
ASK METHOD DecAlive (IN a : INTEGER);
ASK METHOD IncDead  (IN d : INTEGER);
ASK METHOD SendEntityPDU(IN m : INTEGER);
ASK METHOD SendJoinPDU(IN a: INTEGER);
ASK METHOD SendLeavePDU(IN a: INTEGER);
ASK METHOD Resetcount();
ASK METHOD Bitcount(): INTEGER;

END OBJECT;

hexTabletype    = ARRAY INTEGER OF ARRAY INTEGER OF HexCell;

END MODULE.

// ------------------------------------------------------------------
IMPLEMENTATION MODULE Hex;

 OBJECT HexCell;
```

```
ASK METHOD IncAlive (IN a : INTEGER);
BEGIN
        alive := a + alive;

END METHOD;

ASK METHOD DecAlive (IN a : INTEGER);
BEGIN

        alive := alive - a;
        IF alive < 0
         alive := 0;
        END IF;

END METHOD;

ASK METHOD IncDead (IN d : INTEGER);
BEGIN

        dead := dead + d;
END METHOD;

ASK METHOD HexMembers() : INTEGER;
BEGIN

        RETURN (dead + alive);

END METHOD;

ASK METHOD SendEntityPDU(IN m : INTEGER);
BEGIN

      entitypdu := entitypdu + (m);

END METHOD;

ASK METHOD SendJoinPDU(IN a : INTEGER);
BEGIN

        joinpdu := a + joinpdu;

END METHOD;

ASK METHOD SendLeavePDU(IN a : INTEGER);
BEGIN

        leavepdu := a + leavepdu;

END METHOD;
```

```
ASK METHOD Resetcount();
BEGIN

entitypdu := 0;
     joinpdu   := 0 ;
     leavepdu  := 0;

END METHOD;

ASK METHOD Bitcount(): INTEGER;
VAR
total    : INTEGER;

BEGIN

total := (entitymultiple * entitypdu * (entitypdusize + udp)) +
         (joinpdu * (joinpdusize + udp)) +
         (leavepdu * (leavepdusize + udp));

RETURN total;

END METHOD;

END OBJECT;

END MODULE.

//-----------------------------------------------------------------
DEFINITION MODULE Cart;

{ Conversion of cartisian to hex routines }

FROM MathMod IMPORT SQRT;

CONST

{ this is the lower left-hand corner of the playbox
maxx = 579.00;
maxy = 928.00;
yorigin = 877.00;
xorigin = 521.00;
}

VAR

hratio, cartoffset,sqrthratio:  REAL;
pixelfactor,pixelunit  :      REAL;  {lateral spans between hex centers, hex
                        height}
maxx, maxy:  REAL;
xorigin, yorigin: REAL;
```

```
PROCEDURE InitCart(IN hradius,xo,yo,mx,my: REAL; OUT pf, pu : REAL);

PROCEDURE UtmToFlat(INOUT x, y : REAL);

PROCEDURE CartToHex(IN x, y : REAL; OUT i, j : INTEGER);

END MODULE.
//-----------------------------------------------------------------------
IMPLEMENTATION MODULE Cart;

FROM MathMod IMPORT SQRT;

PROCEDURE InitCart(IN hradius,xo,yo,mx,my: REAL; OUT pf, pu : REAL);
BEGIN
{ for example
 radius = 1000
 then lateral distance between hexes is 1000 * 3/2 = 1500
 hex heigth from bottom to top is sqrt(3)* radius = 1732 m
}

xorigin := xo;
yorigin := yo;
maxx:= mx;
maxy:= my;

hratio := 3.0/2.0;
pixelfactor := hradius * hratio;    {lateral spans between hex centers}
pixelunit := hradius * (SQRT(3.0)/2.0); {hex height}
cartoffset := hradius/2.0;

pf:= pixelfactor;
pu := pixelunit;

END PROCEDURE;

{-----------------------------------------------------------}
PROCEDURE UtmToFlat(INOUT x,y : REAL);
{ make conversions to hexs a bit easier by having origin at corner
of playbox }
BEGIN

x := (x - xorigin) * 1000.0;
y := (y - yorigin) * 1000.0;

END PROCEDURE;

{-----------------------------------------------------------}
PROCEDURE CartToHex( IN x, y : REAL; OUT i,j : INTEGER);


VAR
```

```
intipoint,intjpoint : INTEGER;
h1i,h1j,h2i,h2j: INTEGER;
test            : INTEGER;
x1,y1,x2,y2: REAL;
f1,f2           : REAL;

BEGIN

{ intial guess at which hex }

UtmToFlat(x,y);

intipoint := TRUNC((x - cartoffset)/pixelfactor);
intjpoint :=  TRUNC(y/pixelunit);

{ Determines alternating row, colunm combination based on odd, even test }

IF  (intipoint MOD 2 = 1) AND (intjpoint MOD 2 = 1)
     test :=  1; { odd odd }
ELSIF  NOT(intipoint MOD 2 = 1 ) AND (intjpoint MOD 2 = 1)
     test := 2;  { even odd }
ELSIF  (intipoint MOD 2 = 1) AND NOT(intjpoint MOD 2 = 1 )
     test := 3; { odd even }
ELSIF  NOT(intipoint MOD 2 = 1 ) AND NOT(intjpoint MOD 2 = 1)
     test := 4; { even even }
ELSE
OUTPUT("CartToHex failed 1");
END IF;


{ select two candidate hexes based on test results}

h1i := intipoint;

h2i := intipoint +1;

IF ( test = 2) OR (test = 3 )
       h1j := intjpoint + 1;

ELSIF ( test = 1) OR (test = 4 )
       h1j := intjpoint;

ELSE OUTPUT("Cart2Hex failed 2");

END IF;

IF ( test = 2) OR (test = 3 )
      h2j := intjpoint;

ELSIF ( test = 1) OR (test = 4 )
```

```
      h2j := intjpoint +1;

ELSE OUTPUT("Cart2Hex failed 3");

END IF;

x1 := FLOAT(h1i) * pixelfactor;
y1 := FLOAT(h1j) * pixelunit;

x2 := FLOAT(h2i) * pixelfactor;
y2 := FLOAT(h2j) * pixelunit;

f1 := ((x-x1)*(x-x1)) + ((y-y1)*(y-y1));
f2 := ((x-x2)*(x-x2)) + ((y-y2)*(y-y2));

{ which hex center is closest to x,y point?}

IF ( f1 < f2 )
     i := h1i;
     j := TRUNC(FLOAT(h1j)/2.0) + (h1j MOD 2);

{trunc converts to "normal" coords}

ELSE
     i := h2i;
     j := TRUNC(FLOAT(h2j)/2.0) + (h2j MOD 2);

END IF;

END PROCEDURE;

END MODULE;
//-----------------------------------------------------------------------

DEFINITION MODULE Bcast;
{
 Macedonia Jan 1995.
Computes bcast traffic with heartbeats
}

CONST
udp = 64; {net overhead}
     entitypdusize = 1600; {entitypdu with two articulations}
entitymultiple = 8; {number of entity per/s for one entity move }

TYPE

 Bcast = OBJECT

entitypdu : INTEGER;
```

```
    ASK METHOD SendEntityPDU(IN m : INTEGER);
    ASK METHOD Resetcount();
ASK METHOD Bitcount(IN totalheartbeat: INTEGER): INTEGER;

 END OBJECT;


END MODULE.

//-----------------------------------------------------------------
IMPLEMENTATION MODULE Bcast;

FROM RandMod IMPORT RandomObj;

 OBJECT Bcast;

ASK METHOD SendEntityPDU(IN m : INTEGER);
BEGIN

      entitypdu := entitypdu + (m);

END METHOD;

ASK METHOD Resetcount();
BEGIN

entitypdu := 0;

END METHOD;

ASK METHOD Bitcount( IN totalheartbeat :INTEGER): INTEGER;
VAR
total    : INTEGER;
BEGIN

{OUTPUT(totalheartbeat);}

total := (entitymultiple * entitypdu * (entitypdusize + udp)) +
(totalheartbeat * (entitypdusize + udp));

RETURN total;

END METHOD;

END OBJECT;

END MODULE.

//-----------------------------------------------------------------
DEFINITION MODULE Entity;
```

```
{
 Macedonia Jan 1995.
  Defines an Entity
}

CONST
 tcp = 198;
 allpdusize = 10000; { overhead from using TCP assuming some losses}
  entitypdusize = 1600; {entitypdu with two articulations}

TYPE

directionType = (n,ne,se,s,sw,nw);
EntityObj = OBJECT
timestamp   : REAL;
side      : INTEGER;
event     : INTEGER;
id        : INTEGER;
active    : BOOLEAN;
alive     : INTEGER;
trans:   : INTEGER;
moves: INTEGER;
evermove: BOOLEAN;
count      : INTEGER;
xcoord      : REAL;
ycoord : REAL;
icoord  : INTEGER;
lasti     : INTEGER;
jcoord      : INTEGER;
lastj     : INTEGER;
direction: directionType;
apdubitcount: INTEGER;


ASK METHOD SetTime( IN t : REAL);
ASK METHOD SetSide( IN s : INTEGER);
ASK METHOD SetEvent( IN e : INTEGER);
ASK METHOD SetId( IN i : INTEGER);
ASK METHOD SetActive( IN a : BOOLEAN);
ASK METHOD SetAlive (IN al: INTEGER);
ASK METHOD DecAlive (IN d: INTEGER);
ASK METHOD SetCount( IN c : INTEGER);
ASK METHOD SetXYcoord( IN x,y: REAL);
ASK METHOD SetHexcoord( IN x,y : REAL);
ASK METHOD IncTrans();
ASK METHOD IncMoves();
ASK METHOD SetIJcoord(IN i,j: INTEGER);
ASK METHOD PrintEntity();
ASK METHOD SetMove(IN m : INTEGER);
ASK METHOD SetTrans(IN tr : INTEGER);
ASK METHOD ObjInit();
```

```
ASK METHOD SetDirection();
ASK METHOD SetEverMove();
ASK METHOD SendAllPDU(IN entitynumber : INTEGER);
ASK METHOD Resetcount();
ASK METHOD Bitcount(): INTEGER;

END OBJECT;

entityListtype  = ARRAY INTEGER OF EntityObj;

END MODULE.

//-----------------------------------------------------------------
IMPLEMENTATION MODULE Entity;

FROM Cart IMPORT CartToHex, maxx, maxy, yorigin,xorigin;

TYPE

OBJECT EntityObj;

ASK METHOD SetTime( IN t : REAL);
BEGIN
        timestamp := t;

END METHOD;

ASK METHOD SetSide( IN s : INTEGER);
BEGIN
        side := s;
END METHOD;

ASK METHOD SetEvent( IN e : INTEGER);
BEGIN
        event := e;
END METHOD;

ASK METHOD SetId( IN i : INTEGER);
BEGIN
        id := i;
END METHOD;

ASK METHOD SetActive( IN a : BOOLEAN);
BEGIN
        active := a;
END METHOD;

ASK METHOD SetAlive(IN al: INTEGER);
BEGIN
        alive := al;
END METHOD;
```

```
ASK METHOD DecAlive(IN d: INTEGER);
    BEGIN
        alive := alive - d;
        IF alive < 0
       alive := 0;
       END IF;

END METHOD;

ASK METHOD SetCount( IN c : INTEGER);
BEGIN
       c := count;
END METHOD;

ASK METHOD SetXYcoord( IN x,y: REAL);
BEGIN

       IF ((x > xorigin) AND (x < maxx)) AND
            ( (y > yorigin) AND (y < maxy))
       xcoord := x;
       ycoord := y;
       ELSE
       {OUTPUT("xy coord out of range  ", x);}
       END IF;

END METHOD;

ASK METHOD SetHexcoord( IN x,y: REAL);
BEGIN
       CartToHex(x,y,icoord,jcoord);
END METHOD;

ASK METHOD SetIJcoord(IN i,j: INTEGER);

BEGIN
       lasti  := icoord;
       lastj  := jcoord;
       icoord := i;
       jcoord := j;
       SetDirection;

END METHOD;

ASK METHOD SetDirection;
{Figures out the direction of movement}
BEGIN

       IF icoord = lasti
         IF jcoord > lastj
              direction := n;
```

```
        ELSE
            direction := s;
        END IF;
      ELSIF icoord > lasti
        IF jcoord > lastj
            direction := ne;
        ELSE
            direction := se;
        END IF;
      ELSE
        IF jcoord > lastj
            direction := nw;
        ELSE direction := sw;
        END IF;
      END IF;


END METHOD;

ASK METHOD IncMoves();
BEGIN
        INC(moves);
END METHOD;


ASK METHOD IncTrans();
BEGIN
        INC(trans);
END METHOD;

    ASK METHOD PrintEntity();
BEGIN

OUTPUT(timestamp,
    " ",id,
    " ",side,
    " ",event,
    " ",xcoord,
    " ",ycoord,
    " ",icoord,
    " ",jcoord
);
END METHOD;


ASK METHOD ObjInit();
BEGIN
        timestamp  := 0.0 ;
        side       := 0;
        event      := 0;
        id         := 0;
```

```
        active     := TRUE;
        alive     := 1;
        count      := 0;
        trans     := 0;
        moves     := 0;
        xcoord     := 0.0;
        ycoord     := 0.0;
        icoord     := -1;
        jcoord     := -1;
        evermove   := FALSE;

END METHOD;

ASK METHOD SetMove(IN m : INTEGER);

BEGIN

 moves := m;

END METHOD;

ASK METHOD SetTrans(IN tr : INTEGER);
BEGIN

      trans:= tr;

END METHOD;

ASK METHOD SetEverMove();
BEGIN

evermove := TRUE;

END METHOD;

 ASK METHOD SendAllPDU(IN entitynumber : INTEGER);
CONST
BEGIN

     apdubitcount:= apdubitcount + tcp + allpdusize +
              (entitynumber * entitypdusize) ;

END METHOD;


ASK METHOD Resetcount();
BEGIN

     apdubitcount := 0;

END METHOD;
```

```
ASK METHOD Bitcount(): INTEGER;
BEGIN
RETURN  apdubitcount;

      END METHOD;
 END OBJECT; {Entity Object}

END MODULE.


//-----------------------------------------------------------------
MAIN MODULE HexSim;

{   Macedonia Jan 1995

Simulates hexagonal partioning of a dis exercise.

size of the playbox is the min and max of all entity locs

min x = 522.83
max x = 578.54
min y = 878.48
max y = 927.40
maxveh    = 2191;
0-50k meters, x, y
    }
FROM Bcast   IMPORT Bcast;
FROM UtilMod IMPORT GetCmdLineArg,GetNumArgs;
FROM MathMod IMPORT CEIL;
FROM IOMod   IMPORT StreamObj, FileUseType(Input), FileUseType(Output);
FROM Entity  IMPORT EntityObj, entityListtype,
                    directionType(n,ne,se,s,sw,nw);
FROM Hex     IMPORT HexCell, hexTabletype;
FROM Cart    IMPORT InitCart, CartToHex;
FROM RandMod IMPORT RandomObj;

CONST

timehexdump = 1000.0;
timedatadump = 1.0;
heartbeat = 5.0; {DIS default}
yorigin = 877.00;
xorigin = 521.00;

TYPE


VAR

 lasttimehexdump:REAL;
```

```
 difftime:     REAL;
 inputfile:    StreamObj;
 unitfile        :     StreamObj;
 entityfile:     StreamObj;
 hexTable:   hexTabletype;
 entityList        :entityListtype;
 bcast        :      Bcast;
 entity :      EntityObj;
 maxk,maxj,hfcount:     INTEGER;
 clock        :      INTEGER;
 randomheartbeat:RandomObj;
 maxx,maxy:REAL;
 maxentity:  INTEGER; {includes some empty ids because of number scheme}
 entitytotal:  REAL;{actual total}
 totalmoves: REAL;

{-----------------------------------------------------------}
{Copy the contents of the entity to the appropriate on in the list}

PROCEDURE SetEntity();
BEGIN
        ASK entityList[entity.id] SetTime(entity.timestamp);
          ASK entityList[entity.id] SetSide(entity.side);
          ASK entityList[entity.id] SetEvent(entity.event);
        ASK entityList[entity.id] SetId(entity.id);
          ASK entityList[entity.id] SetXYcoord(entity.xcoord,entity.ycoord);
        ASK entityList[entity.id] SetIJcoord(entity.icoord,entity.jcoord);

END PROCEDURE;

{-----------------------------------------------------------}
{ Initalize lists and table, open files}

PROCEDURE Init();

VAR
 i,j,k     :    INTEGER;
 hradius    :     REAL;
 pixelunit  :REAL;
 pixelfactor :REAL;
 str,inname:STRING;
 unitname:STRING;
 size    :       INTEGER;
 index  :       INTEGER;
 field    :       STRING;
 id,unit,side:INTEGER;
 alive    :       INTEGER;
```

```
BEGIN

clock := 1; {starting  time}
hradius := 4000.0;
maxy := 927.40;
maxx := 578.54;
maxentity := 1280;
entitytotal := 2191.0;
inname := "mov.dat";
unitname := "unit.dat";


IF (GetNumArgs > 0)
GetCmdLineArg(1,str);
size := STRTOINT(str);

CASE size
WHEN 1:
maxx := 578.54;
maxentity := 1280;
entitytotal := 2191.0;
inname := "mov.dat";

WHEN 2:
 maxx := 578.54 + 40.0;
     maxentity := 1280 + 2000;
     entitytotal := 2191.0 * 2.0;
inname := "2.dat";
unitname := "2unit.dat";

WHEN 3:
     maxx := 578.54 + 80.0;
     maxentity := 1280 + 4000;
     entitytotal :=  2191.0 * 3.0;
inname := "3.dat";
unitname := "3unit.dat";

 WHEN 4:
     maxx := 578.54 + 120.0;
     maxentity := 1280 + 6000;
     entitytotal :=  2191.0 * 4.0;
inname := "4.dat";
unitname := "4unit.dat";

OTHERWISE
;
END CASE;
END IF;

InitCart(hradius,xorigin,yorigin,maxx,maxy,pixelfactor,pixelunit);
```

```
maxj := CEIL((maxx-xorigin)* 1000.0/pixelfactor) + 1;
maxk := CEIL((maxy-yorigin) * 1000.0/(2.0 * pixelunit)) + 1;

{Sanity Check}
OUTPUT("#Radius is ",hradius);
OUTPUT("#Playbox is ",maxj, "x", maxk," hex.");
OUTPUT("#Pixelfactor ",pixelfactor," Pixelunit ", pixelunit);

OUTPUT("Opening inputfile");

NEW(inputfile);
  ASK inputfile TO Open(inname, Input);

OUTPUT("Opening entity.dat");

NEW(entityfile);
  ASK entityfile TO Open("entity.dat",Output);

OUTPUT ("Openting unit.dat");

NEW(unitfile);
  ASK unitfile TO Open(unitname, Input);

NEW(entity);

NEW(randomheartbeat);

NEW(bcast);

NEW(entityList, 1..maxentity);

FOR j:= 1 TO maxentity
NEW(entityList[j]);

END FOR;

OUTPUT("Reading unitfile");

WHILE NOT (ASK unitfile eof)
FOR  index:= 1 TO 3
        ASK unitfile TO ReadString(field);
        CASE index

         WHEN 1:
              unit := STRTOINT(field);
         WHEN 2:
              side := STRTOINT(field);
         WHEN 3:
              alive := STRTOINT(field);

         OTHERWISE
```

```
            ;
        END CASE;


END FOR;

        id := unit + ((side  - 1) * 1000);
          ASK entityList[id] TO SetAlive(alive);

END WHILE;

ASK unitfile TO Close;

DISPOSE(unitfile);

NEW(hexTable, 0..maxj, 0..maxk);
 FOR j:= 0 TO maxj
FOR k := 0 TO maxk
        NEW(hexTable[j][k]);
END FOR;
 END FOR;

END PROCEDURE;

{-----------------------------------------------------------}
PROCEDURE DumpData(IN ct : REAL;IN hd : BOOLEAN); FORWARD;
{-----------------------------------------------------------}
{ Read the input file, check the time                    }

PROCEDURE ReadData();
VAR

 index,unit,id :     INTEGER;
 x,y    :      REAL;
 hexdump    :BOOLEAN;
 field   :      STRING;

BEGIN

     FOR index:= 1 TO 6
        ASK inputfile TO ReadString(field);
        {OUTPUT(field);}
         CASE index
           WHEN 1:
{ lets convert time to seconds}
           ASK entity TO SetTime(STRTOREAL(field)*60.0);

           WHEN 2:
              unit := STRTOINT(field);

           WHEN 3:
```

```
            ASK entity TO SetSide(STRTOINT(field));

          WHEN 4 :
            ASK entity TO SetEvent(STRTOINT(field));

          WHEN 5:
            x := (STRTOREAL(field));

          WHEN 6:
            y := (STRTOREAL(field));
          OTHERWISE
            ;
          END CASE;

      END FOR;

{Explanation: before we dump data and copy
the entity to the list  we check to see if all the events for a particular
time have all occured.
}
IF ( entity.timestamp > 0.000010){initial psns}
      WHILE ((entity.timestamp - FLOAT(clock)) > timedatadump )

         IF (entity.timestamp - lasttimehexdump) > timehexdump
           lasttimehexdump := entity.timestamp;
               hexdump := TRUE;
         ELSE hexdump := FALSE;
         END IF;
           {OUTPUT("clock ",clock);}

         DumpData(FLOAT(clock),hexdump);
         INC(clock);


END WHILE;
END IF;
{ seperate the red id's from the blue id's }

      id := unit + ((ASK entity side  - 1) * 1000);
ASK entity TO SetId(id);
      ASK entity TO SetXYcoord(x,y);
ASK entity TO SetHexcoord(entity.xcoord,entity.ycoord);

END PROCEDURE;
{-----------------------------------------------------}
{ This simulates net traffic to the hex cell that an entity
belongs to. }

PROCEDURE MoveTraffic(IN i,j,moves : INTEGER);

BEGIN
```

```
ASK hexTable[i][j] TO SendEntityPDU(moves);

END PROCEDURE;

{-----------------------------------------------------}
PROCEDURE BcastMove(IN moves: INTEGER);
BEGIN

ASK bcast TO SendEntityPDU(moves);

END PROCEDURE;
{-----------------------------------------------------}
PROCEDURE TransitionTraffic(IN i,j,id : INTEGER);


BEGIN
{ this is where we send traffic to all the affected hexes
when with do a transition

1 join active
3 join passive
1 leave active
3 leave passive
3 all pdus (big ones based on the count in the hex)

}
CASE entity.direction

WHEN n:
ASK hexTable[i][j+1] SendJoinPDU(entityList[id].alive); {active}
ASK hexTable[i][j+2] SendJoinPDU(entityList[id].alive);
        {rest are passive joins}
ASK hexTable[i-1][j+2] SendJoinPDU(entityList[id].alive);
ASK hexTable[i+1][j+2] SendJoinPDU(entityList[id].alive);

ASK hexTable[i][j] SendLeavePDU(entityList[id].alive); {leave active}
ASK hexTable[i-1][j] SendLeavePDU(entityList[id].alive);
ASK hexTable[i][j-1] SendLeavePDU(entityList[id].alive);
ASK hexTable[i+1][j] SendLeavePDU(entityList[id].alive);

ASK entityList[id]  SendAllPDU(hexTable[i][j+2].HexMembers);
ASK entityList[id]  SendAllPDU( hexTable[i-1][j+2].HexMembers);
ASK entityList[id]  SendAllPDU(hexTable[i+1][j+2].HexMembers);

 WHEN ne:
ASK hexTable[i+1][j+1] SendJoinPDU(entityList[id].alive); {active}
ASK hexTable[i+1][j+2] SendJoinPDU(entityList[id].alive);
                        {rest are passive joins}
ASK hexTable[i+2][j+1] SendJoinPDU(entityList[id].alive);
ASK hexTable[i+2][j] SendJoinPDU(entityList[id].alive);
```

```
ASK hexTable[i][j] SendLeavePDU(entityList[id].alive); {leave active}
ASK hexTable[i-1][j+1] SendLeavePDU(entityList[id].alive);
ASK hexTable[i-1][j] SendLeavePDU(entityList[id].alive);
ASK hexTable[i][j-1] SendLeavePDU(entityList[id].alive);

ASK entityList[id]  SendAllPDU(hexTable[i+1][j+2].HexMembers);
ASK entityList[id]  SendAllPDU( hexTable[i+2][j+1].HexMembers);
ASK entityList[id]  SendAllPDU(hexTable[i+2][j].HexMembers);

 WHEN se:
ASK hexTable[i+1][j] SendJoinPDU(entityList[id].alive); {active}
ASK hexTable[i+2][j] SendJoinPDU(entityList[id].alive);
                    {rest are passive joins}
ASK hexTable[i+2][j-1] SendJoinPDU(entityList[id].alive);
ASK hexTable[i+1][j-1] SendJoinPDU(entityList[id].alive);

ASK hexTable[i][j] SendLeavePDU(entityList[id].alive); {leave active}
ASK hexTable[i-1][j] SendLeavePDU(entityList[id].alive);
ASK hexTable[i-1][j+1] SendLeavePDU(entityList[id].alive);
ASK hexTable[i][j+1] SendLeavePDU(entityList[id].alive);

ASK entityList[id]  SendAllPDU(hexTable[i+2][j].HexMembers);
     ASK entityList[id]  SendAllPDU( hexTable[i+2][j-1].HexMembers);
     ASK entityList[id]  SendAllPDU(hexTable[i+1][j-1].HexMembers);

WHEN s:
ASK hexTable[i][j-1] SendJoinPDU(entityList[id].alive); {active}
ASK hexTable[i-1][j-1] SendJoinPDU(entityList[id].alive);
                    {rest are passive joins}
ASK hexTable[i][j-2] SendJoinPDU(entityList[id].alive);
ASK hexTable[i+1][j-1] SendJoinPDU(entityList[id].alive);

ASK hexTable[i][j] SendLeavePDU(entityList[id].alive); {leave active}
ASK hexTable[i-1][j+1] SendLeavePDU(entityList[id].alive);
ASK hexTable[i][j+1] SendLeavePDU(entityList[id].alive);
ASK hexTable[i+1][j+1] SendLeavePDU(entityList[id].alive);

ASK entityList[id]  SendAllPDU(hexTable[i-1][j-1].HexMembers);
ASK entityList[id]  SendAllPDU( hexTable[i][j-2].HexMembers);
ASK entityList[id]  SendAllPDU(hexTable[i+1][j-1].HexMembers);


WHEN sw:
ASK hexTable[i-1][j] SendJoinPDU(entityList[id].alive); {active}
ASK hexTable[i-1][j-1] SendJoinPDU(entityList[id].alive);
                    {rest are passive joins}
ASK hexTable[i-2][j-1] SendJoinPDU(entityList[id].alive);
ASK hexTable[i-2][j] SendJoinPDU(entityList[id].alive);

ASK hexTable[i][j] SendLeavePDU(entityList[id].alive); {leave active}
```

```
ASK hexTable[i][j+1] SendLeavePDU(entityList[id].alive);
ASK hexTable[i+1][j+1] SendLeavePDU(entityList[id].alive);
ASK hexTable[i+1][j] SendLeavePDU(entityList[id].alive);

ASK entityList[id]  SendAllPDU(hexTable[i-1][j-1].HexMembers);
ASK entityList[id]  SendAllPDU(hexTable[i-2][j-1].HexMembers);
ASK entityList[id]  SendAllPDU(hexTable[i-2][j].HexMembers);

WHEN nw:
ASK hexTable[i-1][j+1] SendJoinPDU(entityList[id].alive); {active}
ASK hexTable[i-2][j] SendJoinPDU(entityList[id].alive);
                    {rest are passive joins}
ASK hexTable[i-2][j+1] SendJoinPDU(entityList[id].alive);
ASK hexTable[i-1][j+2] SendJoinPDU(entityList[id].alive);

ASK hexTable[i][j] SendLeavePDU(entityList[id].alive); {leave active}
ASK hexTable[i+1][j+1] SendLeavePDU(entityList[id].alive);
ASK hexTable[i+1][j] SendLeavePDU(entityList[id].alive);
ASK hexTable[i][j-1] SendLeavePDU(entityList[id].alive);

ASK entityList[id]  SendAllPDU(hexTable[i-2][j].HexMembers);
ASK entityList[id]  SendAllPDU(hexTable[i-2][j+1].HexMembers);
ASK entityList[id]  SendAllPDU(hexTable[i-1][j+2].HexMembers);

OTHERWISE
;
END CASE;

END PROCEDURE;

{-----------------------------------------------------}
PROCEDURE ComputeEntities (IN id : INTEGER): INTEGER;
{Check the number of entities in each member group}

CONST

VAR

i,j :     INTEGER;
total:    INTEGER;

BEGIN

i := entityList[id].icoord;
j := entityList[id].jcoord;

IF (i <> -1) AND (-j <> -1)

total := hexTable[i][j].HexMembers +
         hexTable[i][j+1].HexMembers +
         hexTable[i][j-1].HexMembers +
```

```
        hexTable[i-1][j+1].HexMembers +
        hexTable[i-1][j].HexMembers +
        hexTable[i+1][j+1].HexMembers +
        hexTable[i+1][j].HexMembers;
ELSE

total := 0;

END IF;

RETURN total;

END PROCEDURE;

{-----------------------------------------------------}
PROCEDURE ComputeMcastTraffic (IN i,j : INTEGER): INTEGER;
{When entity multicasts the hex takes note.}

VAR

total:   INTEGER;

BEGIN

IF (i <> -1) AND (-j <> -1)


total := hexTable[i][j].Bitcount +
        hexTable[i][j+1].Bitcount +
        hexTable[i][j-1].Bitcount +
        hexTable[i-1][j+1].Bitcount +
        hexTable[i-1][j].Bitcount +
        hexTable[i+1][j+1].Bitcount +
        hexTable[i+1][j].Bitcount;
ELSE

total := 0;

END IF;

RETURN total;

END PROCEDURE;
{-----------------------------------------------------}
PROCEDURE ComputeUnicastTraffic( IN id : INTEGER): INTEGER;
{When entity does a SendAllPDU it keeps track of the traffic}

VAR

total : INTEGER;
```

```
BEGIN

total := ASK entityList[id] TO Bitcount;


ASK entityList[id] TO Resetcount;

RETURN total;

END PROCEDURE;

{-----------------------------------------------------}
PROCEDURE FinalStats();

VAR

notmoved : INTEGER;
n : INTEGER;

BEGIN

 FOR n := 1 TO maxentity

 IF (entityList[n].evermove = FALSE ) AND NOT (entityList[n].id =  0)
notmoved := notmoved + entityList[n].alive;
END IF;

 END FOR;

OUTPUT("Not moved = ",notmoved," ",FLOAT(notmoved)/ entitytotal *100.0, "%");

END PROCEDURE;

{-----------------------------------------------------}
{ Dump to data files the info on entities and hex cells   }

PROCEDURE DumpData(IN ct : REAL;IN  hd : BOOLEAN);

CONST
format1 = "* ******.*******<";
format2 = "*** *** *** *** ***.** *****.** *****.** *******<";
format3 = "*******.******* **** **** **** **** **** **** ********* **** ********* **** ******* **** **** **********
*********";

VAR

entitymoved,entitytrans,n : INTEGER;
j,k,bigj,bigk  : INTEGER;
maxbpsn,minbpsn  : INTEGER;
maxoccupied,currentoc  : INTEGER;
hexoccupied  : INTEGER;
```

```
str1,str2,str3,str4,str5  : STRING;
bps              : INTEGER;
maxbps,minbps  : INTEGER;
totalbps  : INTEGER;
meanbps        : INTEGER;
hexfile          : StreamObj;
entitynumber  : INTEGER;
maxentitynumberentity  : INTEGER;
maxentitynumber        : INTEGER;
bcastbps  : INTEGER;
unicast,totalunicast  : INTEGER;
totalmcast  : INTEGER;
aggregate  : INTEGER; {unicast + mcast}
nomove         : REAL;

BEGIN

{Data collection every second}

     FOR n := 1 TO maxentity

IF (entityList[n].id <> 0)
        entitynumber := ComputeEntities(n);

        IF entitynumber > maxentitynumber
        maxentitynumber := entitynumber;
        maxentitynumberentity := n;
        END IF;

{We compute the bandwidth per entity, combining the multicast traffic
and unicast traffic}

        unicast :=  ComputeUnicastTraffic(entityList[n].id);
        totalunicast := totalunicast + unicast;

        bps := ComputeMcastTraffic(entityList[n].icoord,
                    entityList[n].jcoord) +
              unicast;

        totalbps := totalbps + bps;

        IF bps > maxbps
              maxbps := bps;
              maxbpsn := n;
        END IF;

        IF bps < minbps
              minbps := bps;
              minbpsn := n;
        END IF;
```

```
          IF (entityList[n].moves > 0)
         entitymoved := entitymoved +  entityList[n].alive;
           ASK entityList[n] SetMove(0);
        END IF;

        IF (entityList[n].trans) = 1
         entitytrans := entitytrans + entityList[n].alive;
         ASK entityList[n] SetTrans(0);
        END IF;
END IF;
     END FOR;

 meanbps := TRUNC(FLOAT(totalbps)/entitytotal);


IF hd {Is is time to write a hexfile?}

INC(hfcount);
NEW(hexfile);
str4 := INTTOSTR(hfcount);
str5 := ".hex";
INSERT(str5,0,str4) ;

ASK hexfile TO Open(str5,Output);

str1 := SPRINT("#",ct) WITH format1;
ASK hexfile TO WriteString(str1);
END IF;

FOR j:= 0 TO maxj
     FOR k := 0 TO maxk


        IF (hexTable[j][k].numberIn > 0)

        IF hd
          str2 := SPRINT(j ,k,
                    hexTable[j][k].HexMembers,
                    hexTable[j][k].Maximum,
                    hexTable[j][k].Mean,
                    hexTable[j][k].StdDev,
                    hexTable[j][k].Variance,
                    hexTable[j][k].Bitcount) WITH format2;
          ASK hexfile TO WriteString(str2);
          ASK hexfile TO WriteLn;
          END IF;

        INC(hexoccupied); {count how many entities in a hex}

{Count total mcast traffic}
```

```
        totalmcast := totalmcast + hexTable[j][k].Bitcount;

    currentoc := hexTable[j][k].HexMembers;

            IF maxoccupied < currentoc
            bigj := j;
            bigk := k;
            maxoccupied := currentoc;
            END IF;


        END IF;


{Reset bitcount}

        ASK hexTable[j][k] TO Resetcount;

    END FOR;

END FOR;

aggregate := totalunicast + totalmcast;

nomove := entitytotal - totalmoves;

totalmoves := 0.0;

{broadcast is computed by adding movement traffic with static heartbeats}

bcastbps := ASK bcast TO Bitcount(TRUNC(ASK randomheartbeat Normal(nomove/
heartbeat,4.0)));

ASK bcast TO Resetcount;

 str3 := SPRINT(ct,
        entitymoved,
        entitytrans,
        hexoccupied,
        maxoccupied,
        bigj,
        bigk,
        maxbps,
        maxbpsn,
        minbps,
        minbpsn,
        meanbps,
        maxentitynumber,
        maxentitynumberentity,
        bcastbps,
        aggregate
```

```
        ) WITH format3;

  ASK entityfile TO WriteString(str3);
  ASK entityfile TO WriteLn;

IF hd

  ASK hexfile TO Close;
  DISPOSE(hexfile);

END IF;

{Santity Check}
{  OUTPUT(str3);
}
END PROCEDURE;

{-----------------------------------------------------}
{ Close files and disposes objects.                  }

PROCEDURE Cleanup();
BEGIN

  ASK inputfile TO Close;
  ASK entityfile TO Close;

  DISPOSE(bcast);
  DISPOSE(randomheartbeat);
  DISPOSE(entity);
  DISPOSE (inputfile);
  DISPOSE (entityfile);

END PROCEDURE;

{-----------------------------------------------------}
{ Where the action is.                               }

PROCEDURE SimLoop();

VAR

n: INTEGER;

BEGIN

{ read the records of events in }

WHILE  NOT (ASK inputfile eof)

{read data unitil data is in range}
```

```
 REPEAT
  ReadData;
 UNTIL (ASK entity xcoord <> 0.0) AND (ASK entity ycoord <> 0.0);

{first time, intialize }
 IF (entityList[entity.id].icoord = -1 ) OR  (entityList[entity.id].jcoord = -1 )

SetEntity;

ASK hexTable[entityList[entity.id].icoord,entityList[entity.id].jcoord]
                TO Add( entityList[entity.id]);

     ASK hexTable[entityList[entity.id].icoord,entityList[entity.id].jcoord]
                 TO IncAlive( entityList[entity.id].alive);

{entity has not not transitioned to a new hex}
 ELSIF ((entityList[entity.id].icoord = entity.icoord) AND
     (entityList[entity.id].jcoord = entity.jcoord))

{Has entity really moved?}
IF entity.event > 0
     ASK entityList[entity.id] TO IncMoves;
ASK entityList[entity.id] TO SetEverMove;

END IF;

{are they dead?}

IF entity.event < 0
     {reduce the alive entity count}
ASK entityList[entity.id] TO DecAlive(ABS(entity.event));
{tell the hex to remember the dead}
ASK hexTable[entityList[entity.id].icoord,entityList[entity.id].jcoord]
         TO IncDead( ABS(entity.event));
ASK hexTable[entityList[entity.id].icoord,entityList[entity.id].jcoord]
            TO DecAlive( ABS(entity.event));
END IF;

MoveTraffic(entity.icoord,entity.jcoord,entityList[entity.id].alive);
totalmoves := FLOAT(entityList[entity.id].alive) + totalmoves;

BcastMove(entityList[entity.id].alive); {for bcast stats}
SetEntity;

{entity has moved to a new hex}
 ELSIF ((entityList[entity.id].icoord <> entity.icoord) OR
     (entityList[entity.id].jcoord <> entity.jcoord))

ASK entityList[entity.id] TO IncTrans;
ASK entityList[entity.id] TO IncMoves;
ASK entityList[entity.id] TO SetEverMove;
```

```
{are they dead?}
     IF entity.event < 0
     ASK entityList[entity.id] TO DecAlive(ABS(entity.event));
     {tell the hex}
     ASK hexTable[entityList[entity.id].icoord,entityList[entity.id].jcoord]
          TO IncDead(ABS(entity.event));
     ASK hexTable[entityList[entity.id].icoord,entityList[entity.id].jcoord]
          TO DecAlive( ABS(entity.event));
     END IF;

BcastMove(entityList[entity.id].alive); {for bcast stats};
 totalmoves := FLOAT(entityList[entity.id].alive) + totalmoves;

ASK hexTable[entityList[entity.id].icoord,entityList[entity.id].jcoord]
          TO DecAlive(entityList[entity.id].alive);

ASK hexTable[entityList[entity.id].icoord,entityList[entity.id].jcoord]
               TO RemoveThis( entityList[entity.id]);
SetEntity;

TransitionTraffic(entity.icoord,entity.jcoord,entity.id);

ASK hexTable[entityList[entity.id].icoord,entityList[entity.id].jcoord]
               TO Add( entityList[entity.id]);

     ASK hexTable[entityList[entity.id].icoord,entityList[entity.id].jcoord]
               TO IncAlive( entityList[entity.id].alive);


 ELSE

OUTPUT("Test conditions failed");

 END IF;

 INC(n);

 END WHILE;

END PROCEDURE;

{ - - - - - - - - - - - - - - - }
{ Main section.                 }

BEGIN

Init;

SimLoop;
```

```
FinalStats;

Cleanup;

END MODULE.
```

APPENDIX C

```
# Hex.tcl
# Developed December 1994 Mike Macedonia
# Tcl/Tk code to generate hex images

image create photo .i -file 502.ppm

set high [image height .i]
set wide [image width .i]

set auto_path "$tk_library/demos $auto_path"

canvas .h -width $wide  -heigh $high

set radius [expr 3.0 * $wide/30.0]

.h create image  [expr $wide/2] [expr $high/2] -image .i

pack .h

set s3 [expr sqrt(3)/2]

#-----------------------------------------------------------
# Creates a hex outline given the cart coords and radius.

proc hex {x y radius} {

global s3

set r  [expr $radius*1.0]
set r2 [expr $r/2]
set r3 [expr $r*($s3)]


set x1 [expr $x - $r ]
set x2 [expr $x - $r2]
set x3 [expr $x + $r2]
set x4 [expr $x + $r ]
set x5 $x3
set x6 $x2
set y1 $y
set y2 [expr $y + $r3]
set y3 $y2
set y4 $y
set y5 [expr $y - $r3]
set y6 $y5


.h create line ${x1} ${y1} ${x2} ${y2} -fill black
```

```
.h create line ${x2} ${y2} ${x3} ${y3} -fill black
.h create line ${x3} ${y3} ${x4} ${y4} -fill black
.h create line ${x4} ${y4} ${x5} ${y5} -fill black
.h create line ${x5} ${y5} ${x6} ${y6} -fill black
.h create line ${x6} ${y6} ${x1} ${y1} -fill black
}

#-----------------------------------------------------------
# Creates a filled hex polygon give the the center cart coords and the radius

proc hex_xy {x y radius} {

global s3
global tk_library

set r [expr $radius*1.0]
set r2 [expr $r/2]
set r3 [expr $r*($s3)]


set x1 [expr $x - $r ]
set x2 [expr $x - $r2]
set x3 [expr $x + $r2]
set x4 [expr $x + $r ]
set x5 $x3
set x6 $x2
set y1 $y
set y2 [expr $y + $r3]
set y3 $y2
set y4 $y
set y5 [expr $y - $r3]
set y6 $y5

.h create polygon ${x1} ${y1} ${x2} ${y2} ${x3} ${y3} ${x4} ${y4} \
${x5} ${y5} ${x6} ${y6} -fill red -stipple @$tk_library/demos/images/grey.25
}

#-----------------------------------------------------------
# Test routine that displays a fancy hex pattern.

proc eye {} {

global wide high

for { set i 0} {$i <= 100 } {incr i 1} {
hex [expr $wide/2] [expr $high/2] [expr $i * .25]

}
}

#-----------------------------------------------------------
```

```
# Creates a hex map on the canvas with TPDS coord numbers

proc map { r } {

global wide high cellX cellY r2 h radius

set radius [expr $r*1.0]
set h2 [expr $radius*sqrt(3)]
set h  [expr $h2/2]
set r2 [expr $radius*3/2]
set x $radius
set y 0
set x_coord 1
set right 0

while {$x < $wide + $radius} {


set c [expr $x_coord % 2]

if { $c == 1} {
                set y [expr $high - $h]
                set y_coord 1
                        } else {
                set y $high
                set y_coord 0
                        }


                set right $x

                while {$y > 0}  {

                set up $y

                hex $right $up $radius

                set new [.h create text ${right} ${up}]
#                       -text "$x_coord,$y_coord" -fill black]

                set y [expr $up - $h2]

    incr y_coord 1

                }

                set x [expr $right + $r2]
                incr x_coord 1

}
}
```

```
#-----------------------------------------------------------------
# Dumps the canvas to a postscript file.

proc print_ps {} {.h postscript -colormode gray -file "~/print/test.ps"}

#-----------------------------------------------------------------
# Finds the hex that cart coords are in.

proc cart2hex { x y } {

global wide high i_hex j_hex h r2 radius pixelfactor pixelunit

set pixelfactor $r2
set pixelunit  $h
set cartoffset [expr $radius/2.0]

#puts " h= $h , r2 = $r2 "

set i [expr ($x-($cartoffset))/$pixelfactor]
#set i [expr $x/$pixelfactor]
set j [expr $y/$pixelunit  ]

#puts "i = $i , j = $j"

set int_i_point [expr int($i)]
set int_j_point [expr int($j)]

#puts "int_i = $int_i_point , int_j = $int_j_point"

if { [expr $int_i_point % 2] && [expr $int_j_point % 2] } {
set test 1
# odd odd
} elseif { [expr !($int_i_point % 2)] && [expr $int_j_point % 2] } {
set test 2
# even odd
} elseif { [expr ($int_i_point % 2)] && [expr !($int_j_point % 2)] } {
set test 3
# odd even
} elseif { [expr !($int_i_point % 2)] && [expr !($int_j_point % 2)] } {
set test 4
# even even
} else {
error "Hex test failed"
}

#puts "test i= $test"

set h1_i $int_i_point

set h2_i [expr $int_i_point +1]
```

```
if { $test == 2 || $test == 3 } {
 set h1_j [expr $int_j_point + 1]
}
if { $test == 1 || $test == 4 } {
 set h1_j $int_j_point
}
if { $test == 2 || $test == 3 } {
 set h2_j $int_j_point
}
if { $test == 1 || $test == 4 } {
 set h2_j [expr $int_j_point +1]
}

#puts "h1_i = $h1_i , h1_j = $h1_j , h2_i = $h2_i , h2_j = $h2_j "

set x_1 [expr $h1_i*$pixelfactor]
set y_1 [expr $h1_j*$pixelunit]

set x_2 [expr $h2_i*$pixelfactor]
set y_2 [expr $h2_j*$pixelunit]
set f1 [expr ($x-$x_1)*($x-$x_1) + ($y-$y_1)*($y-$y_1)]
set f2 [expr ($x-$x_2)*($x-$x_2) + ($y-$y_2)*($y-$y_2)]

# Commented out test code
#puts "$x_1 , $y_1  $x_2 , $y_2 "

#.h create oval [expr $x_1-$cartoffset - 1]m [expr ($high*10)-$y_1 - 1]m [expr $x_1-
$cartoffset + 1]m [expr ($high*10)-$y_1 + 1]m -fill blue -tag dot

#.h create oval [expr $x_2-$cartoffset - 1]m [expr ($high*10)-$y_2 - 1]m [expr $x_2-
$cartoffset + 1]m [expr ($high*10)-$y_2 + 1]m -fill blue -tag dot

#puts "f1 = $f1, f2 = $f2"

if { $f1 < $f2} {
set i_hex $h1_i
set j_hex $h1_j

} else {
set i_hex $h2_i
set j_hex $h2_j
}
puts "i_hex = $i_hex"
puts "j_hex = $j_hex"
}

#-----------------------------------------------------------
# Hilites a hex when given hex coords.

proc hex_ij { i_hex j_hex } {
```

```
global h r2 radius high

set pixelfactor $r2
set pixelunit $h
set cartoffset [expr $radius/2.0]

set x_coord [expr ($i_hex*$pixelfactor)-$cartoffset]
set y_coord [expr ($high) - ($j_hex*$pixelunit)]

hex_xy [expr $x_coord] [expr $y_coord] $radius

}

#-----------------------------------------------------------------
# This procedure finds hex closest to cart coords and hilites

proc find_hex { x y } {

global i_hex j_hex wide high r2 h radius pixelfactor pixelunit

puts "$x $y"
set cartoffset [expr $radius/2.0]
#.h create oval [expr $x - 5] [expr ($high)-$y - 5] [expr $x + 5] \
                [expr ($high)-$y + 5] -fill yellow -tag dot
cart2hex $x $y

hex_ij $i_hex $j_hex

}

#-----------------------------------------------------------------
# Display map with defaults on startup.

map $radius

# find pixel ratio

bind .h <Button-1> {find_hex %x [expr ($high - %y)]}
```

INITIAL DISTRIBUTION LIST

1.	Defense Technical Information Center Cameron Station Alexandria, VA 22034-6145	2
2.	Dudley Knox Library Code 52 Naval Postgraduate School Monterey, CA 93943-5001	2
3.	Director of Research Administration Code 08 Naval Postgraduate School Monterey, CA 93943-5001	1
4.	Dr. Michael J. Zyda, Code CS/ZK Computer Science Department Naval Postgraduate School Monterey, CA 93943-5001	2
5.	Dr. Michael G. Sovereign, Code OR/MS Operations Research Department Naval Postgraduate School Monterey, CA 93943-5001	2
6.	Dr. Dan C. Boger, Code SM/BO Systems Management Department Naval Postgraduate School Monterey, CA 93943-5001	2
7.	Dr. David R. Pratt, Code CS/PR Computer Science Department Naval Postgraduate School Monterey, CA 93943-5001	4
8.	Dr. Ted Lewis, Code CS Computer Science Department Naval Postgraduate School Monterey, CA 93943-5001	2

9. LTC Jim Wall 1
ASHPC
Army Research Labs
APG, MD 21005-5067