Supporting DIS Applications using ATM Multipoint Connection Caching

Anshul Kantawala, Guru Parulkar
John DeHart

Applied Research Laboratory
Department of Computer Science
Washington University
St. Louis, Mo. 63130, USA
anshul,guru,jdd@arl.wustl.edu

Ted Marz

McDonnell Douglas Training Systems
McDonnell Douglas Aerospace
St. Louis, Mo.

Abstract

This report describes an ATM *Multipoint Connection Caching strategy (*AMCC*) to control the explosive growth of traffic within the network and at an endpoint in a large Distributed Interactive Simulation (*DIS*) application such as a battlefield simulation. For very large* DIS *applications with 100,000 entities, the current method of broadcasting information among entities will no longer be feasible due to computational and network bandwidth limitations. Our scheme divides the simulation space into grids and each grid square or a set of grid squares forms a multicast* group. *Entities join the* groups *within their perception range and thus, they receive state updates only from entities within their perception range.* AMCC *utilizes the unique capabilities of our dynamic multipoint ATM signaling protocol, CMAP. We have implemented* AMCC *and have successfully demonstrated a small* DIS *battlefield simulation application running over our* ATM *testbed.*

1 Introduction

Distributed Interactive Simulation (DIS) [8, 11] is an emerging IEEE standard which defines protocols and mechanisms necessary for interoperability between manned military simulators of various types and fidelities. In addition to virtual simulators operating over wide geographical areas, participants will be generated by automated AI-based constructive simulations and even live weapon systems. DIS is being promoted as a critical element in the development of joint doctrine, requirements definition (weapon and training systems), design, prototyping and manufacturing, contingency planning, operations, after mission review and as a tool for historical analysis. While published DIS implementations to date have focused on military applications, future extensions will support education, transportation, entertainment and other commercial applications.

DIS has progressed beyond the early definition phase; simulation networking is the highest priority for the 1990's Defense Research and Engineering Science and Technology Thrust Number 11 (Human System Interfaces) in support of Science and Technology Thrust Number Six (Synthetic Environments). We have already begun to see DIS requirements on such programs as the Army's Close Combat Tactical Trainer (CCTT) and the Air Force's Special Operations Forces Aircrew Training Systems (SOF ATS) and others. Virtually all new trainer programs will require DIS interoperability; however, there is still a great deal of research and development required in many areas to achieve the DIS vision of thousands of simulators, automated entities and live participants interacting in a "seamless virtual battle field".

Central concepts in the DIS architecture dictate that there is no central controlling computer and that each individual node maintain a complete database of the current state of the exercise, that is, locations, types, and other similar state information of each entity or player, in the simulation. State updates are then broadcast to every player as necessary. Since DIS is intended for real-time man-in-the-loop simulation, transport delay and variance of network data is of primary concern. DIS implementations result in the generation of many relatively small Protocol Data Units (PDUs), on the order of 100-200 octets each. The DIS standard specifies no particular underlying communications service and is now typically implemented on local area Ethernets, and uses a patchwork of dedicated leased telephone lines for wide area connections. Exercises on the order of 1,000 entities have been demonstrated.

Distributed hosts are used for a single simulation. The simulation may be divided among hosts by various criterion such as geographical location, entity types. The following paragraphs present two important (proposed) capabilities of DIS based simulations that will be essential for its long term success.

- **Large number of entities.** One of the "Grand Challenges" of DIS is to increase the number of entities which participate in a given exercise. Future goals established by ARPA involve exercises consisting of about 100,000 entities.

- **Other media streams.** Besides positional information, a simulation entity may send other kinds of information or media streams. Examples include digital audio, video, and electromagnetic signals. Video conferencing during DIS-based mission planning exercises is one example.

Both these capabilities lead to many challenging research and implementation problems. For example, while an individual entity in a simulation may not generate a substantial amount of data per second, the aggregate bandwidth generated by 100,000 such entities will be very large. On the other hand, including digital audio, video, and various electromagnetic signals means higher bandwidth streams with need for performance guarantees as in most other distributed imaging and multimedia applications.

The research community has been suggesting that high speed ATM networks with multipoint communication support will help support very large DIS based simulations with multimedia streams. DIS presents many demands on the communications service which ATM is well suited to handle. When an entity moves or otherwise changes its state, that information is sent to all the other players on the network. This translates directly into the requirements for high bandwidth and multipoint communication capability. Minimizing transport delay is also critical as geographically separated entities attempt tightly coupled operations such as air to air refueling. The multimedia capabilities of ATM are also well suited to the DIS visions of interactive training and after-action review. However, no research has been done in determining suitability of ATM and existing protocols for DIS applications.

With 100,000 endpoints, a single multipoint connection for the application will not be an appropriate solution because such a large multipoint connection is difficult to manage and every endpoint will have to process every message from every other endpoint leading to simply too many messages at every endpoint. We therefore propose to use ATM with its native multicast capabilities to set up connections between participating hosts in a DIS simulation. We have developed an ATM Multipoint Connection Caching Scheme (AMCC) to manage the multicast connections among the hosts. Our goal is to demonstrate the viability of and to verify, experimentally, the reduction in network traffic obtained and quantify the extra signaling overhead. Our scheme divides the simulation space into grids and each grid square or a set of grid squares forms a multicast *group*. Entities join the *groups* within their perception range and thus, they receive state updates only from entities within their perception range. We have implemented a DIS application with McDonnell Douglas which uses AMCC for setting up multicast connections over an ATM testbed. AMCC uses an ATM signaling protocol developed at Washington University for dynamically managing multicast connections between participating hosts in the DIS simulation.

In the following sections we propose a scheme which will significantly reduce network traffic for large DIS exercises by using multicast groups. Section 2 describes a signaling architecture for our ATM network. In Section 3, we describe the ATM Connection Caching Scheme. Section 4 discusses implementation issues and solutions. Section 5 outlines our experimental setup, Section 6 discusses the results obtained and Section 7 concludes the report.

2 Signaling protocols for ATM networks

Our signaling software system architecture [7] is depicted in Figure 1. This architecture attempts to solve many of the important problems that confront ATM network control engineers by providing uniform Application Programmer Interlaces (APIs) at each of the software layers, thereby encapsulating and concealing vendor-specific interfaces. The four software layers are:

- the Switch Management Layer, which provides a uniform interface to switching equipment produced by different vendors;
- the Node Management Layer, which groups multiple switches into single Nodes;
- the Connection Management Layer, which provides a homogeneous general ATM connection management interface; and,
- the Session Management Layer, which provides a high-level interface whereby applications acquire the network resources appropriate to their communications requirements.

The Connection Management Access Protocol (CMAP), the most relevant for this project, allows network clients to create, manipulate and delete multipoint, multiconnection communication channels, which we term *calls* [4]. A *call* is a distributed object maintained by the network which describes the communication paths that interconnect clients. A multipoint call is a call involving two or more clients; a point-to-point call is a special case of a multipoint call involving only two clients. Data sent over a connection by an endpoint in a call is received by all other endpoints electing to receive on this connection, although reliable delivery is not guaranteed by the network. Each call can have one or more connections and each endpoint in a call has the ability to choose which subset of connections to join. Calls are allowed to change dynamically during their lifetime, in terms of the number of endpoints, the number of connections and the reserved bandwidth of the connections. Another key feature of CMAP is that it supports **leaf-initiated** joins. To enable near optimal routing and resource utilization, when endpoints issue **join** requests, they are forwarded towards the *source* and terminated at the first node which is a member of the multipoint connection. A new branch is added from that node to the new endpoint to complete the JOIN operation. CMAP defines the interface between clients and the network which is used to create, manipulate and delete calls.

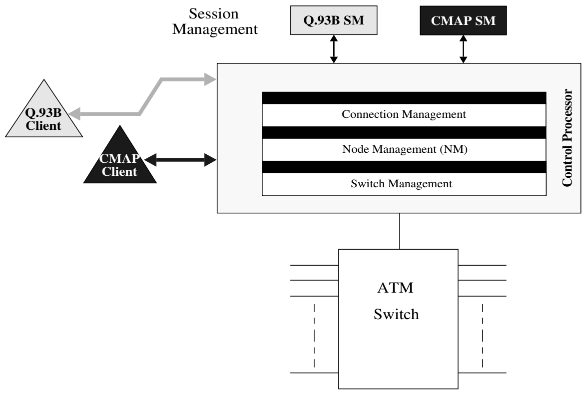

Figure 1: ATM Signaling Software System Architecture

The following is a brief description of CMAP functions that are used to support AMCC.

- *open_call.* The *open_call* operation is used by the client to create a call between itself and its access node. One or more connections can be created as part of the call using this operation and another client can be added as a second endpoint of the call.
- *close_call.* The *drop_call* operation destroys the call, removing all connections and reclaiming any resources used by the connections.
- *add_ep.* The *add_ep* operation requests that an additional client be added to an ongoing call. Once a client has been successfully added, it can receive and transmit on all connections within the call.
- *drop_ep.* The *drop_ep* operation removes the specified endpoint from the call.
- *open_con.* The *open_con* operation requests that a new connection be added to the ongoing call. This operation can be initiated by the owner of the call or any client.
- *close_con.* The *close_con* operation destroys a specified connection and reclaims all resources used by the connection.

For large DIS simulation exercises, we propose to use multicast ATM connections for data transfer. Since the nature of DIS entities is inherently dynamic, it is essential that our underlying ATM signaling protocols support dynamic change in the call structure in terms of the number of endpoints, connections, transmitters and receivers. For example, when entities move or change their perception range, they need to join new multicast connections and/or drop themselves from old multicast connections. When the number of multicast calls is large, source initiated joins and drops for endpoints creates a bottleneck which significantly increases signaling overhead. There are also substantial resource cost savings associated with leaf-initiated joins over source initiated joins. First, the *source* of a multicast connection does not need to know or keep track of all the endpoints in the connection. This leads to savings in processing resources and memory. Second, a join signaling message to a multipoint connection can be terminated at the first node in its path towards the SOURCE. This leads to reduced latency in signaling along with near optimal multipoint connection tree[12]. In [3, 5], further discussions are presented for the necessity of leaf-initiated joins for a scalable multicast service. Thus another key aspect required of the signaling protocol is that it support **leaf-initiated** joins and drops. We believe that CMAP is the only ATM signaling protocol that provides support for fully dynamic call structure, multipoint calls with multiple transmitters and leaf initiated joins and drops. The following section describes our multipoint connection caching scheme in detail.

3 ATM Multipoint Connection Caching

We propose to use an ATM Multipoint Connection Caching (AMCC) scheme on CMAP to support large battle field simulations using DIS. The key observations that led to our proposal is the finite perception range of entities and the support for dynamic multicast connections in ATM. For example, Figure 2 shows a typical mix of entities that would exist in a battlefield simulation along with their perception ranges. We can see that entities like soldiers with very small perception ranges do not need many state updates of other entities. We can thus make substantial gains in reducing network traffic by eliminating unnecessary transmission of data of entities which are beyond the perception range of the simulated entities. We model entity movement and their dynamic perception range by dynamic ATM multicast connections. Entities join multicast connections which are within their perception range and as they move or their perception range changes, they change their membership to multicast connections accordingly. The basic idea of this scheme is explained in the following paragraphs.

In general, only a small percentage of the data in a large exercise is needed by any one application, the technical challenge is to eliminate unwanted data as soon as possible so unwanted data will have minimal effect on host performance. If possible, the network itself can be used to help in this filtering process. The proposed solution is to "grid" the database geographically and associate each grid square with a unique multipoint connection. A simulated entity joins a set of multipoint connections based on its location and its *perception range*. Note that each individual entity will also be able to define its "perception range". This allows different types of entities to customize their area of interest. For example, tanks may only need to know about other entities within 10KM, but a jet fighter may need data out to 200KM. Typically, an entity will join all the multipoint connections associated with grid squares within its perception range. For example, consider Figure 3 which shows a 2D grid with two entities A and B. $M_{i,j}$ represents a multipoint connection

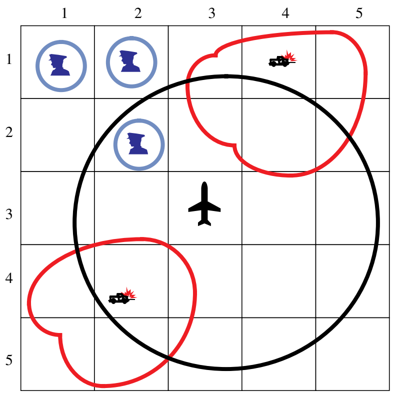

Figure 2: Sample DIS Application

associated with grid element (i, j). Entities A and B are the transmitters on the connection $M_{3,3}$ and $M_{1,4}$ respectively. Given the perception range of the entity A, shown as a cloud, it acts as a receiver on multipoint connections: $M_{2,2}, M_{2,3}, M_{2,4}, M_{3,2}, M_{3,4}, M_{4,2}, M_{4,3}$, and $M_{4,4}$. Note that B is a receiver on, among other connections, A's connection $M_{3,3}$, and so B gets all messages from A as expected. Also note that if there is more than one entity in a grid square, then all of them become transmitters on the connection associated with the grid square. As the entity moves in space, it requests to be dropped from certain multipoint connections (associated with its past perception range) and to join others (associated with its new perception range). For example, if the entity A moves from location $(3, 3)$ to $(4, 4)$, it becomes a transmitter on connection $M_{4,4}$, changes from being the transmitter to a receiver on the connection $M_{3,3}$, drops from connections $M_{2,2}, M_{3,2}, M_{4,2}, M_{2,2}, M_{2,3}$, and $M_{2,4}$, and joins as a receiver on connections $M_{5,3}, M_{5,4}, M_{5,5}, M_{3,5}$, and $M_{4,5}$. Because multipoint connection manipulation operations, such as join and drop take time, it may be necessary that an entity join appropriate multipoint connections before it really needs to send or receive any data on those connections. This requires precomputing expected position of the entity and initiating the join operation(s) on the appropriate connections in anticipation. Again, consider Figure 3. In anticipation of entity A's move from location $(3, 3)$ to $(4, 4)$, all the join operations mentioned above could have been initiated while the entity A was still in location $(3, 3)$.

This scheme, called ATM Multipoint Connection Caching, provides the following benefits which are crucial if the network were to support simulations with very large number of endpoints:

- It will drastically reduce the number of packets (PDUs) to be processed at every endpoint.

- It will also significantly reduce the amount of redundant traffic within the network because PDUs are sent on only the multipoint connections whose endpoints need them.

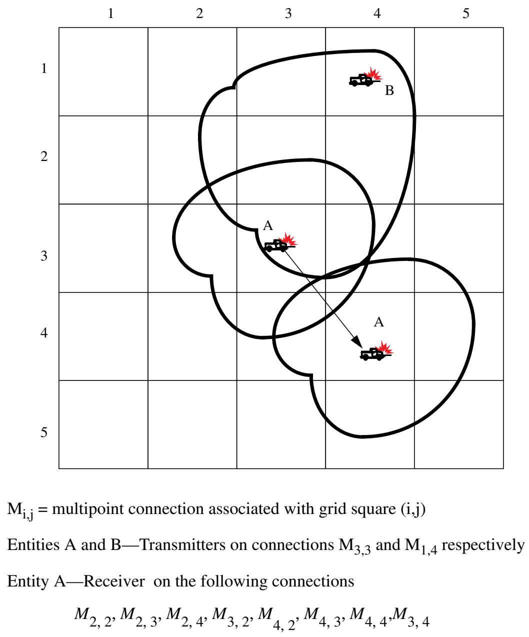

Figure 3: ATM Multipoint Connection Caching

The proposed scheme is more complex than simply using a single multipoint connection. However, a single multipoint connection with as many as 100,000 endpoints is very difficult to manage and could lead to significant performance degradation because of the amount of unneeded data that would be transmitted to endpoints. Thus, it is essential that other more complex solutions be pursued for such cases. Complexity in the AMCC scheme results from the fact that it requires maintaining more multipoint connections and repeated join and drop operations on these connections as entities move or as their geographic attention span changes. The scheme also requires that the underlying connection management system allow a very generalized multipoint connection: a connection with endpoints (entities) that have different read and write permissions and join and drop operations initiated by an endpoint. Though the ATM signaling standards are moving in this direction, Washington University's ATM signaling protocols are probably the only ATM protocol suite that meet these requirements today.[1]

4 Many-to-Many Connections over ATM

Figure 4 illustrates a scenario which has one multicast connection with three endpoints: A, B and C over an ATM network. Hosts A and B are transmitters in this connection and host C is a receiver. Consider the case when a single Virtual Circuit (VC) connection ($VPI = 0$, $VCI = 1$) is setup for this multi-

[1] Of course, if the revised ATM standard protocols provide the necessary capabilities, we would use them as opposed to our own non-standard protocols.

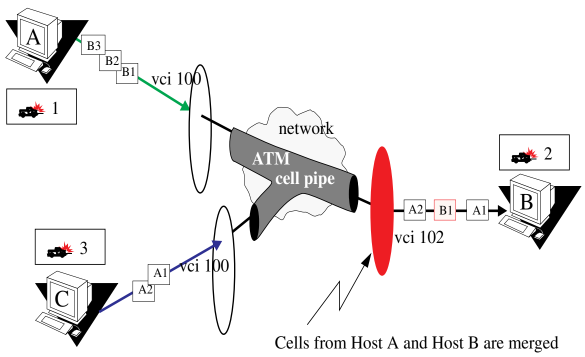

Figure 4: VCs with multiple transmitters

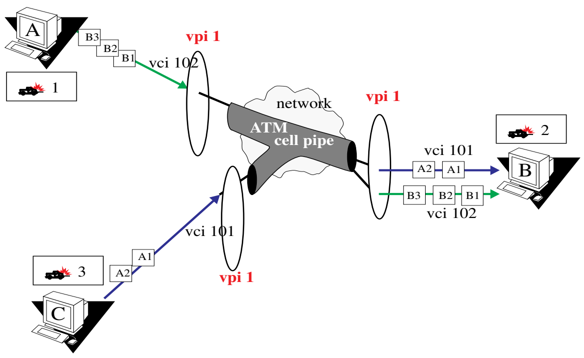

Figure 5: Using VPs for multicast calls

cast call. ATM cells contain only VPI/VCI numbers in their headers which are used by the switch for routing. The cells from both host A and host B contain identical VPI/VCI numbers since they are part of the same VC and thus they are not differentiated at the switch. Due to this, cells from both hosts can get interleaved before they reach host C which means that host C can no longer use hardware re-assembly which would lead to severe performance degradation. This problem of source discrimination in a many-to-many connection over ATM has been discussed in [3, 5, 6]. In [10], the authors propose to use "packet" switching at nodes which have a multicast connection with multiple transmitters. In this approach, they take advantage of the fact that AAL5 has an end-of-packet identifier as part of the ATM header. Cells arriving at different input ports on a switch which belong to a single multicast connection are *packet-switched* to the output port, i.e., all the cells from one input port are switched until an end-of-packet identifier is seen before cells from another input port are serviced. For our implementation, we used a commercial Bay Networks switch and thus did not have the luxury to implement the solution outlined above since it requires a change in the switch architecture. Currently, the ATM Forum [1, 2] has proposed using point-to-multipoint connections only. The problem of source discrimination is addressed by defining a separate connection for each transmitter but no inherent abstraction, like the CMAP call, is supported by the network to assist applications in organizing their connections.

In the following discussion we present several strategies which can be used to overcome this problem of source discrimination in a many-to-many connection over ATM along with their merits and demerits, and finally we present the scheme we adopted for our project.

4.1 VP based Solution

One solution for source discrimination is to use Virtual Paths (VPs) for multicast calls. Each transmitter in the multicast call can use a different VCI to allow source discrimination at the receivers. Figure 5 shows an example using VPs for multicast calls. Unfortunately, we discovered that the existing host interface cards do not support Virtual Paths [9] . They only accept $VPI = 0$ and a 16 bit VCI field for setting up virtual circuits. Thus, we could not use this approach in our solution.

VPs have been originally designed to be used for trunk routing or at the network backbones and the range of available VPs is extremely scarce. Hence, the use of VPs for source discrimination would incur the liability of being able to setup only a small number of multipoint connections. It would also raise political problems, since VPs have not been designed to be used for this purpose.

4.2 Distribute Entities based on Multicast Group

All transmitting entities within a multicast *group* must be simulated on a single host. This would ensure that there would be no interleaving of packets from different entities in any multicast connection since the operating system (or the network driver) serializes packets arriving from an application. Initially, this would entail dividing the geographical area among the host running the simulation and making sure that all entities reside on the host serving the corresponding location. When an entity crosses a grid boundary, if the new *group* it joins already has another transmitter on a different host, the entity must be physically moved to the other host.

This strategy is straightforward in terms of adding or deleting members from groups. However, it involves a significant modification to DIS simulation engines. Implementation of this scheme would require all DIS applications to be tailored to allow movement of entities across different hosts on demand. This will increase the complexity of the simulations exorbitantly making this scheme infeasible for practical implementations. Also, DIS is a standard and an implementation as described above would not be considered by the community to be a standard implementation.

4.3 Software Re-assembly

In this scheme, if there are multiple transmitting entities within a *group* on different hosts, we can use "raw" ATM mode and software reassembly at the receivers. This would allow entities within a *group* to be simulated by different hosts. This strategy again adds minimal overhead to the maintenance of multicast groups after entity motion across *group* bound-

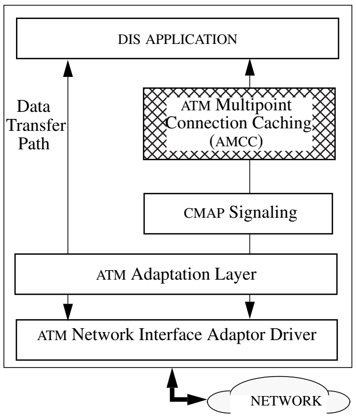

Figure 6: Implementation within each host

aries. However, this scheme will suffer from a severe performance degradation due to the much slower speeds of software reassembly.

4.4 Point-to-Multipoint connections within a Multicast Call

We recall from Section 2, that CMAP supports a multicast CALL that can have multiple endpoints and multiple connections. Also, the number of endpoints and connections within a call are dynamic, allowing endpoints to join or leave calls and add or drop connections within a call. We use the feature of multiple connections with a multicast call to address the problem of source discrimination. Transmitting entities within the same *group* on different hosts establish new point-multipoint connections within the same multicast call. Only one point-multipoint connection is required per host per group. If there are multiple entities in a single multicast call being simulated on the *same* host, they can transmit on the same connection. The application will *sequentially* send data packets to the network interface which performs the segmentation into ATM cells thus precluding any interleaving of cells.

This strategy increases the overhead of establishing and maintaining multicast groups by causing transmitting entities on different hosts within a single *group* to set-up new point-multipoint connections instead of being able to simply add the entity as a new member of the existing *group*. Another overhead is maintaining tables listing entity/group pairs on each host. We note, however, that the maximum number of connections summed over all *groups* cannot exceed $m \times n$, where m is the number of hosts running the DIS exercise and n is the total number of grid squares in the area. We also note that the additional signaling overhead incurred due to the use of this scheme is significantly lower than using software re-assembly for all data packets.

Given our constraints regarding lack of support for VPIs and the excessive overhead needed for the other schemes, we decided to use point-to-multipoint connections within our multicast calls to solve the problem of source discrimination. Another big advantage of CMAP is that it provides us the capability to create abstract multicast calls among all the participating hosts in a grid square and within each call create multiple point-to-multipoint connections. Without this feature, we would have had no choice but to revert to software re-assembly.

5 Implementation Details

Figure 6 is a block diagram of our implementation on each machine. On the signaling path, the DIS application sends entity location and perception range updates to AMCC. Based on these updates, AMCC changes the multicast connection membership for the entity and returns new file descriptors which the DIS application uses to send and receive data. The multicast connection management is done by AMCC using CMAP to add, delete and modify existing multicast calls and connections within calls. On the data transfer path, the application receives and sends data directly from the ATM host adaptor driver using read and write calls. We used AAL5 as our ATM Adaptation Layer and the Aruba network driver over the ENI host adaptor for our network interface. All our implementation work was done in the Solaris operating system. The following sections describe in detail the implementation of the different components of our system.

5.1 DIS Application

The DIS application developed for this effort is a simple Entity Generator and a Plan View Display (PVD) as shown in Figure 7. The Entity Generator creates and manages up to a maximum of 64 entities per machine. We have three different entity type classes: Aircraft, Tank, and Infantry. Each entity type is then further subdivided into specific aircraft and tank models. All entities can be placed on one of four motion path types: stationary, linear, circular, or waypoint. In waypoint motion paths, we specify a list of locations or *waypoints* along which the entity must traverse. Upon reaching the final waypoint, the entity moves back towards the first waypoint and traces the same route. For simulation runs, we create initial configuration files which list the entities, types, motion paths and error criteria. The error criteria define the threshold difference between entity locations which would generate an entity state Protocol Data Units (PDU).

The PVD is a 2-dimensional display developed using X-windows and Motif. Entities and their perception ranges are displayed in the simulation area of the PVD along with the overlaid grid squares. The Entity Generator shares information about entity state with the PVD using a shared memory entity database. Entity updates from local and remote entities are stored in the entity database which is used by PVD for updating the display. The PVD can also be used to choose an *entity of interest* and change its perception range. Further details about the PVD layout are laid out in Section 7 along with screen dumps of the display.

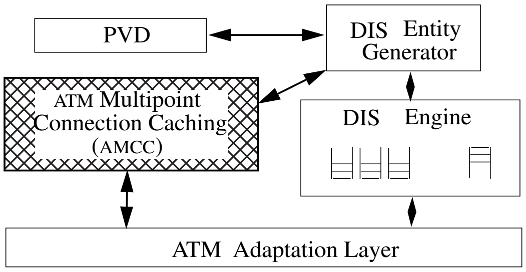

Figure 7: DIS application

The McDonnell Douglas Training Systems DIS Interface Unit (MDTS DIU) handles sending and receiving data from the network. The DIU consists of the DIU Engine, the DIU Interface Library, and the Hardware Interface Package. The Engine and Interface Library consist of libraries of highly portable C code and together perform the required DIS specific processing. All machine specific processing (with the exception of endian conversion) is performed in the Hardware Interface Package. This facilitates the porting of the DIU to various processor, operating system, and network platforms. For this effort, we created a Hardware Interface Package which interfaced directly with the ATM host adaptor via the ATM device drivers. Virtual circuits are created by AMCC and assigned their own unique file descriptor. The DIU then uses UNIX[2] read and write calls to fetch and place packets on the network.

5.2 Interface between AMCC and DIS Application

The DIS application provides AMCC with entity location, velocity and range of regard (perception range) whenever entities surpass their specified error criteria (i.e., the difference in their current and previous location is greater than the value specified in the configuration file). Based on these parameters, AMCC joins new multicast calls (creating them if necessary) and drops out of calls that no longer have any local entities participating. It then opens new data channels if required for transmission and reception of data and provides the DIU with a **transmit** file descriptor for transmission of entity updates for current entity along with a list of **receive** file descriptors for reception of entity updates within its perception range.

5.3 Interface between AMCC and CMAP

The following discussion describes in detail, via an example, implementation of the interface between AMCC and CMAP. As described in Section 2, CMAP provides a flexible multicast call management model to a user application. The following example will discuss how CMAP was used by AMCC to maintain the list of multicast calls in each host required for the DIS application. For our test setup, we do not create any multicast calls apriori but only create them on demand as entities move into the particular grid squares.

[2] Unix is a trademark of AT&T.

Entity Position	Host	CMAP calls	Multicast groups
Initial	Host 1	open_call	$M_{2,2}$,$M_{2,3}$,$M_{2,4}$,$M_{3,2}$, $M_{3,3}$,$M_{3,4}$,$M_{4,3}$,$M_{4,4}$
		add_con	$M_{3,3}$
	Host 2	open_call	$M_{1,2}$,$M_{1,3}$,$M_{1,4}$
		add_ep	$M_{2,2}$,$M_{2,3}$,$M_{2,4}$,$M_{3,2}$, $M_{3,3}$,$M_{3,4}$
		add_con	$M_{1,4}$
Entity A moves from $M_{3,3}$ to $M_{4,4}$	Host 1	drop_ep	$M_{2,2}$,$M_{2,3}$,$M_{2,4}$,$M_{3,2}$
		drop_con	$M_{3,3}$
		open_call	$M_{5,3}$,$M_{5,4}$,$M_{5,5}$,$M_{3,5}$, $M_{4,5}$
		add_con	$M_{4,4}$

Table 1: Example of AMCC call management using CMAP

Consider Figure 3 which shows a simple application with two entities A and B being simulated on hosts 1 and 2 respectively. For this example we shall assume that the simulation on host 1 of entity A started prior to that of entity B on host 2.[3] Table 1 lists the CMAP calls made by AMCC for each machine given the position of two entities. The first two columns show the entity position and the machine number respectively. The third column lists the CMAP calls made by AMCC to join and drop from multicast groups and to add or drop connections within them. We can see from the table that initially host 2 receives entity updates from host 1 since it is a member of $M_{3,3}$ and host 1 is transmitting on a connection in $M_{3,3}$. When entity A moves from $M_{3,3}$ to $M_{4,4}$, host 2 will no longer receive any entity updates from host 1 since host 1 is transmitting on a connection in $M_{4,4}$ and host 2 is *not* a member of group $M_{4,4}$. This is exactly the behavior that we require, since entity A is no longer in the perception range of entity B and thus B should not receive any updates from A.

5.4 AMCC

In implementing AMCC, we were faced with a number of challenges. The first one was to choose a data structure which would be efficient and allow fast access for call management. We decided to create a 2-dimensional array which would model the grid square layout of our actual simulation space and store a **transmit** file descriptor along with the number of entities currently transmitting and receiving in each grid square. Although the actual grid layout may be large, we store the minimal required data in each cell in the array and we have the advantage of very fast indexed array access.

Another challenge we faced was creating a global database of all the multicast calls in the simulation. This database is accessed when determining if a call already exists for a particular multicast group, or if it needs to be created. Accesses to the database had to be mutually exclusive among the different hosts to maintain consistency. Since we did not have the luxury of distributed shared memory, we used an RPC

[3] If the simulations start together, requests for call creation are sequentialized at the global call database which will be discussed in the next section.

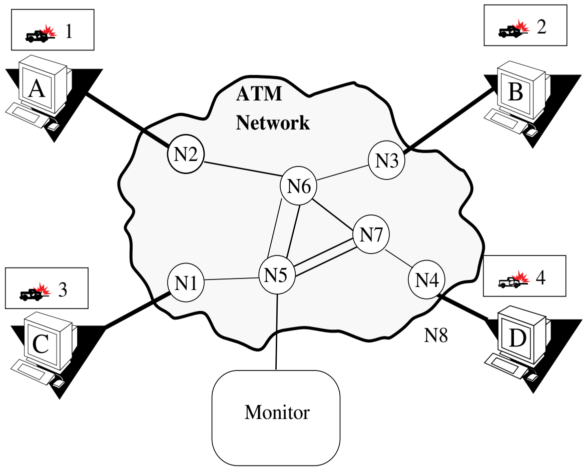

Figure 8: Overall Experimental Setup

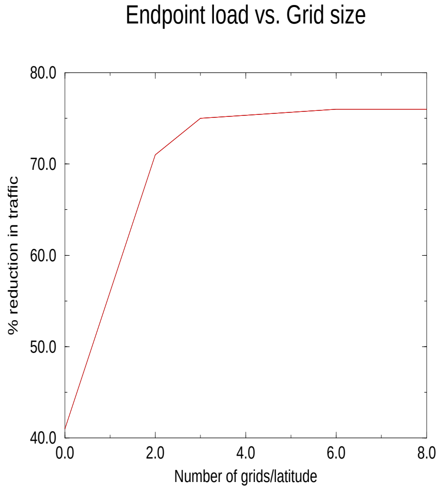

Figure 9: Reduction in Endpoint Traffic

server with locking for mutual exclusion to serve as a global repository of multicast calls. However, using RPC calls did lead to a degradation in performance. We found that the time spent in AMCC per entity update was about 100 ms due to the RPC overhead (this is excluding the time spent in CMAP for setting up and modifying calls). We realize that this limits our ability to handle 10 entity updates/s, but since we did not have distributed shared memory, this was a cost we had to accept.

For quantitative results, we have introduced a probe into the DIU which keeps track of aggregate data transmitted and received during the length of the simulation. To keep an account of the signaling overhead we timestamp every call to AMCC by the DIU along with its return. We also have a *monitor* which is used to display the entire simulation grid space. It maintains default VC connections with all the other simulating host and receives *all* entity updates. This monitor is used only for debugging and demonstrating and plays no integral part in the actual simulation. We present the results we have obtained in the following section.

6 Experimentation and Results

Figure 8 shows our overall experimental setup for evaluating the benefits against the costs of AMCC. All the machines used are Sun SPARCs running Solaris. Hosts A through D run the DIS application. The monitor is used to provide a global view of the entire simulation.

6.1 Display of DIS Application over ATM Multicast

On every host participating in the simulation, we have a display which shows the locally simulated entities, their perception ranges and all external entities within the perception ranges. We can select whether we want the display centered on a particular location or on an entity. If we choose to center the display on an entity, the entity we select becomes the entity of interest and we can change its perception range using a slide bar. We also have a global display that shows the current *multicast calls* between a pair of hosts that have at least one transmitting entity.

Simple and complex simulation scenarios have been tested and proven to work correctly over ATM using AMCC. With complex simulations, the dynamic nature of the DIS application tests the utilization of dynamic connections by AMCC. Due to space limitations, we have not shown here displays from our simulation runs. These displays can be accessed at the URL: http://www.arl.wustl/ anshul/INFOCOM97/

6.2 Experimental Results

We conducted several experiments to evaluate the reduction in traffic at the endpoints by using multicast connections. For our experiments, we used a fixed set of configuration files to start the simulations and varied the grid sizes. We started by using a grid square size equal to one degree latitude and longitude. Our final run was done using 64 grid squares to cover the same area as a degree of latitude and longitude. Each simulation was run for 360 seconds which allowed time for the transients to stabilize. A typical simulation would consist of 16 entities distributed across 4 hosts.

Figure 9 depicts the change in network load at each endpoint due to changes in grid size. The X-axis shows the number of divisions per degree latitude (longitude). We ran four sets of experiments with grid square sizes ranging from one grid square per latitude/longitude to 64 grids per latitude/longitude. We can see that even with very large grid square sizes, we get a reduction of about 41% in endpoint load. When we moved to very small grid squares (64 grids/degree latitude(longitude)) we get a 76% reduction in network load. This significant reduction in network usage will be a key factor in allowing larger simulations to be run. Without it, scaling to large numbers of entities would be impossible.

Figure 10 shows the overhead incurred by AMCC while setting up, modifying and tearing down multicast calls on demand. When grid square sizes de-

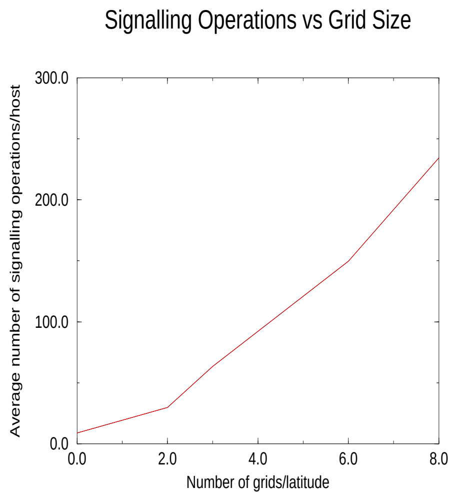

Figure 10: Signaling overhead

crease, an entity's perception range will cover a much greater number of grid squares leading to an increase in number of multicast calls. The signaling overhead is presented as the average number of CMAP operations per host per simulation. We have evaluated that each CMAP operation takes about 40 ms and thus at very small grid square sizes, the amount of time spent in signaling is very high. Our signaling overhead is too high and this limited us in both the number of entities that we could simulate and the granularity of the grid squares. From both the graphs, we see that even with large grid squares which give us a relatively low signaling overhead, we get a significant reduction in network load.

The reason for the steep slope of the plot of signaling operations versus grid size is that as the grid squares get smaller, entities are moving across grid boundaries more frequently. It is the crossing of boundaries, or perception ranges crossing boundaries, that cause signaling operations to be performed.

We have identified problems that we need to address in our signaling software which will help us reduce the signaling overhead and thus improve the performance of the simulation. The first implementation of the signaling system was done as a set of heavy-weight Unix processes. We are investigating the performance improvements that can be gained by reducing the number of processes necessary to implement the system. We believe that we can reduce the time per operation to below 10ms.

7 Conclusion

We have demonstrated the capability of running DIS applications over an ATM network. With the implementation of AMCC to support the DIS applications, we have shown that the significant reduction of network traffic realized through the use of dynamic ATM multicast connections can make a real difference in running large DIS simulations. This is an important result for future DIS applications that will be simulating on the order of 100,000 entities and use various media streams which require a much higher network bandwidth.

Our implementation and experimentation has helped us gain further insights into problems that have yet to be addressed. The next issue that we propose to pursue is the reduction of signaling overhead. One possible approach to this is the use of background anticipatory joins. This will allow signaling operations to be performed prior to the time at which they are required. We will also be investigating modifications to our signaling protocols to allow easier and more efficient management of the DIS global call database.

References

[1] The ATM Forum Technical Committee, "ATM User-Network Interface Specification-Version 3.1", September 1994.

[2] The ATM Forum Technical Committee, "User-Network Interface (UNI) Signaling Specification, Version 4.0", June 1996.

[3] Ballardie, T., Francis, P., and Crowcroft, J., "Core Based Trees (CBT)", in Proc. ACM SIGCOMM 1993.

[4] Cox, K.C., DeHart, J.D., "Connection Management Access Protocol (CMAP) Specification," Technical Report WUCS-94-21, Washington University, 1994.

[5] Deering, S. et al., "An Architecture for Wide-Area Multicast Routing", in Proc. ACM SIGCOMM 1994.

[6] DeHart, J.D., Gaddis, M.E., and Bubenik, R.G., "Virtual Paths and Virtual Channels," IEEE INFOCOM '92, May 1992.

[7] DeHart, J.D., "Connection Management Software System (CMSS) Architecture," Working Note ARL-95-03, ARL, Washington University, 1995.

[8] "The DIS Vision - A Map to the Future of Distributed Simulation," DIS Steering Committee, October 1993.

[9] "ENI ATM Host Adapter Card" ENI.

[10] Grossglauser, M., and Ramakrishnan, K. K., "SEAM: Scalable and Efficient ATM Multipoint-to-Multipoint Multicasting" in NOSSDAV 1996.

[11] "Standard for Distributed Interactive Simulation - Application Protocols (Version 2.0 Fourth Draft)," University of Central Florida Institute for Simulation and Training, IST-CR-94-15, February 1994.

[12] Waxman, B.M., "Routing of Multipoint Connections," IEEE J. Selected Areas in Communications, Vol.6, No.9, December 1988.