AMS: Area Message Service for SLC

M. Crane, R. Mackenzie, D. Millsom, M. Zelazny

Stanford Linear Accelerator Center, Stanford University, Stanford CA 94305 *

Abstract

The Area Message Service (AMS) is a TCP/IP based messaging service currently in use at SLAC. A number of projects under development here at SLAC require an application level interface to the 4.3BSD UNIX socket level communications functions using TCP/IP over ethernet. AMS provides connection management, solicited message transfer, unsolicited message transfer, and asynchronous notification of pending messages. AMS is written completely in ANSI 'C' and is currently portable over three hardware/operating system/network manager platforms, VAX/VMS[1]/Multinet[2], PC/MS-DOS[3]/Pathworks[4], VME 68K/pSOS/pNA[5]. The basic architecture is a client-server connection where either end of the interface may be the server. This allows for connections and data flow to be initiated from either end of the interface. Included in the paper are details concerning the connection management, the handling of the multi-platform code, and the implementation process.

1. Introduction

The principal reason for developing the AMS was to provide network services to hardware and software which was not already supported by the Stanford Linear Collider (SLC) control system. The initial demand for AMS came from a specific project, the Machine Protection System (MPS) and was soon followed by projects being developed off-line and off-site by various collaborators. In attempting to satisfy the need to access the SLC control system and database by diverse projects, the following requirements were specified.

1. The message service shall be able to be implemented on a variety of hardware and software platforms and should be easy to port to new configurations when needed. The platforms to be supported initially were VAX/VMS, Motorola 680X0/pSOS, PC/MS-DOS.

2. The message service shall rely on readily available hardware, software and protocol support. This was seen to have the following advantages: new implementations would be easier to generate, projects being developed at other sites would get easier access to the appropriate hardware and software, using widely used protocols would allow off-site developers to connect to the control system remotely for testing purposes and developers would be able to test their software off-site in the same network environment as the production environment.

3. The network services must be able to be integrated into the existing SLC network software.

4. The services shall not impose a particular paradigm: for example, server-client, master-slave. Applications shall select which ever paradigm was appropriate for them.

5. The services shall provide a flexible naming scheme which was suitable in a real-time environment.

6. AMS shall not issue error messages. Instead it shall simply return status to the caller.

2. Overview

It was decided that Ethernet and TCP/IP would provide the underlying physical, network and transport layers and that the package would rely specifically on TCP as the transport layer. Thus AMS consists of a layer between the application program and TCP/IP: calls made by the application to send and receive data to other tasks are translated into the appropriate TCP calls to set up connections and send or receive data. The application is oblivious to the connection management being undertaken on its behalf.

AMS provides two types of transfers, synchronized and unsynchronized. A synchronized message expects a reply which is bound specifically to that message. An unsynchronized request is like a reliable datagram. No reply or acknowledgement is expected at the application level although the TCP protocol provides reliable delivery.

AMS provides asynchronous notification of arrival of a message or a reply to a message. At present it is only implemented on the VAX platform. Other implementations can provide this if the operating system allows and it is required by the applications.

AMS provides peer to peer services. That is, any process can initiate or receive a data transfer at any time. If a process attempts to send data to a process to which a connection has not yet been established, AMS transparently sets up the connection. If a connection crashes, AMS attempts to re-establish the connection when the next message is sent. Thus, AMS provides the appearance of connection-

*Work supported by the Department of Energy, contract DE-AC03-76SF00515

[1]VAX and VMS are trademarks of Digital Equipment Corporation

[2]Multinet is a trademark of TGV, Inc

[3]MS-DOS is a trademark of MicroSoft Corp.

[4]Pathworks is a trademark of Digital Equipment Corporation

[5]pSOS and pNA are trademarks of ISI

0-7803-1203-1/93$03.00 © 1993 IEEE

less network services by hiding the connection management from the application.

A name translation service is provided so that local name table maintenance is not necessary. This service can be provided from more than one source, to avoid having a single point of failure, and it allows for dynamic address assignment so that in the event of a system failure the translation for a name, 'ONLINE' for example, can be reassigned to the address of the current online host. This name server currently runs only under VMS.

A naming convention is used to allow the application program to send and receive messages to AMS peers using ASCII node and task names. In this convention the node name corresponds to the IP address and task name corresponds to the TCP port number. The translation of these node and task names to IP addresses and port numbers is provided by the name server.

3. Connection Management

One design goal of AMS is to hide connection management from the user providing peer to peer networking in keeping with the current SLC control system message service. Each AMS peer which uses AMS has a server socket to passively accept connection requests and a client socket to actively connect a client socket to a target AMS peer. AMS initialization is performed by calling AMS_INIT with a number of configuration arguments such as the maximum size of the messages to be sent/received, the maximum number of nodes to connect to, lists of AMS peer names to send or receive from, etc. The initialization routine: allocates memory space to use at run time; sets up a linked list of records which track the status of each connection; sets up the local server socket to accept incoming connections; allocates a client socket for each possible remote connection; tries to connect to each possible remote peer; registers the peer with the name server; and then returns to the user. To remove AMS from a process the AMS_KILL routine is used which closes all allocated socket structures including the server socket, frees the AMS allocated memory space and removes the AMS peer from the name server.

During runtime the code checks to see if there are any outstanding incoming connection requests at the server port and connects them as required. If a data send is required and there are no connections to the peer, AMS tries to set up a connection to the peer and complete the data transfer.

4. Message Transfer

The calling interface provides two modes for sending messages, **"synchronized"** and **"unsynchronized"**. Synchronized mode provides a mechanism by which a sending task can bind an outbound message with a specific reply. If a reply to a synchronized message is not received within a specified timeout but is later generated by the receiver, because, for example, the receiver's host is slow or the function requested takes a long time, it will be discarded. Thus, an application can be sure that a synchronized reply really "belongs" to the message last sent. It also ensures that the reply comes from the instantiation of the task which received the message. Unsynchronized mode provides a simple message transfer without regard to synchronization and without regard to the current instantiation of the receiving task. Thus two unsynchronized messages could be received by different sequential executions of the same task.

Synchronized messages are supported by the services AMS_SEND_SYNCH, AMS_SEND_REPLY, AMS_GET_REPLY and AMS_RECEIVE. Synchronized messages are sent by calling the routine AMS_SEND_SYNCH. This allows the caller to send multiple synchronized messages in the one call and primes the message service to expect replies. Message destinations are identified uniquely by the triplet (node, task, command) and only one oustanding message to a specific triplet is allowed at any time.

Once a synchronized message has been sent, it can be "cancelled" by receipt of a reply from the target or a timeout where the timeout period for a reply starts after AMS_GET_REPLY has been called. After sending a synchronized message, the sender can receive a reply by calling AMS_GET_REPLY. This service allows the caller to specify a list of messages sent using one or more previous calls to AMS_SEND_SYNCH. In AMS_GET_REPLY a timeout can be specified after which any replies to the messages specified are discarded. In addition, for each message in the list, a status is returned which specifies the fate of the reply. Once a timeout has expired, a new message can be sent to the same triplet. In this case it is possible that the receiver is still holding onto messages from a previous "send". In fact, messages can "stack up" in the receiver and will be presented to the application in the order received. If a receiver replies to messages which have been timed-out, the replies are discarded by AMS at the sender end.

Replies are sent using the service AMS_SEND_REPLY. AMS checks that replies correspond with synchronized messages previously received and for given node/task, it always generates replies to the earliest synchronized message received.

Unsynchronized messages are supported by the services AMS_SEND, AMS_RECEIVE and AMS_RECEIVE_NOWAIT. To send an unsynchronized message, the routine AMS_SEND is called.

To receive the next available message, AMS_RECEIVE or AMS_RECEIVE_NOWAIT is called. These routines return both synchronized and unsynchronized messages. It is up to the application to decide which type of message it has received. The "nowait" version provides a mechanism for asynchronous notification under VMS.

AMS_RECEIVE_NOWAIT will set up an Asynchronous System Trap (AST) for when a message arrives unless there is already a message waiting. When a message arrives, the AST set up by AMS_RECEIVE_NOWAIT will copy

the message into the user's buffer, remove its own internal copy, set the user's event flag and call the user's AST.

5. Name Server

The name server process currently runs on the VAX platform only. It has a hardcoded, well known IP port and IP address but future plans include the ability to move the server from node to node. TCP/IP was chosen for the server connection protocol since there was already a base of TCP code implemented for AMS. This will eventually change to UDP to allow less network overhead, quicker response, and the ability to use multicast features. The server simply loops accepting new connections from AMS clients, servicing name translation requests and closing the connections. Utilities to support server diagnostics and routine shutdown and startup procedures are in development now.

The AMS name server client code resides along with the rest of the AMS code in each platform's libraries. The client connects to the name server, sends a name translation request to the name server, and returns. There are no special features in these client operations.

6. Security

Security is required to protect an AMS peer from receiving connections from unauthorized clients. Extra connections including accidental and malicious attempts to talk to an peer are not allowed. Each peer provides a list of permitted peers to AMS at initialization time. This list is checked at connect time to ensure that the incoming peer is valid and permitted. The notion of a ALL* (or total wildcard) is used to tell AMS that any peer may be connected and received/sent to. The use of ALL* bypasses the use of the permitted peer list, but does check the name server to validate the peer name. The name server is the central place where security is checked. Before a peer is recognized it must register with the name server. After it has registered, other peers may attempt connections to it. When AMS no longer exists for a particular node, the peer can be removed from the name server by calling AMS_KILL.

7. Implementation

All of the AMS source code is stored on the VAX/VMS system using the Digital Equipment Corporation Code Management System (CMS) as the code management tool. This allows multiple programmers to work on the same bits of code with a minimum of conflicts. The code is shared amongst the differing platforms by using a special include file, one for each platform. This include file provides a means to translate file names, differing function return codes and differing function call names. Very few 'C' language #ifdef statements are actually used in the code which makes maintainence and readability much easier.

The first goal in the implementation process was to get a simple connect and data passing skeleton up and working. The requirement was to implement the basic connection philosophy as the foundation for the send/receive portions of the code. AMS_INIT was the first routine to be coded along with the multi-platform include files. VAX to VAX were the first connections, followed by the 68K to VAX. The MS-DOS port followed soon after to ensure that the multi-platform coding philosophy was correct. After connections were established, the passing of data was the next step. The send routines were simple since it is an active type of transaction. Receive was more difficult because of the polling nature of time independent data receives. The TCP socket select call was implemented along with buffer allocation routines to allow receiving data with a minimum of CPU overhead. Studies of CPU and network performance were done soon after the intial releases of AMS. It was found that each VAX process using AMS consumed substantial CPU time calling the TCP select call to see if any new data was available. The no_wait receive routines were then coded specifically for the VAX platform to cut this CPU time down. The name server was the last part of the project to be implemented. Previous to this time, all AMS peer names were stored in hard coded tables internal to AMS.

8. Future Plans

There are plans to improve AMS as the user base grows. The most important plan is to change the name server communications from TCP to UDP. This will reduce network traffic and server node CPU usage. There also needs to be failover procedures in software to gracefully handle the transfer of the SLC control system from on VAX to another with the AMS impact being the node location of the name server and the translation of the node names "PRODUCTION" and "DEVELOPMENT" which are used by the AMS peers to distinguish between the SLC VAX'es. There are also a number of diagnostic tools to support the name server which need development.

AMS is currently being used by a number of development projects here at SLAC and has had nearly a year of satisfactory service. It has proven itself to be a reliable messaging service and has met all of it's design goals. The first production release of projects using AMS are due in the very near future.