ESD-TR-82-012

(12)

Semiannual Technical Summary

Distributed Sensor Networks

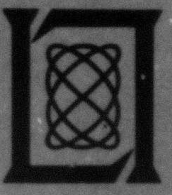
30 September 1981

Prepared for the Defense Advanced Research Projects Agency
under Electronic Systems Division Contract F19628-80-C-0002 by

Lincoln Laboratory

MASSACHUSETTS INSTITUTE OF TECHNOLOGY

LEXINGTON, MASSACHUSETTS



MASSACHUSETTS INSTITUTE OF TECHNOLOGY
LINCOLN LABORATORY

DISTRIBUTED SENSOR NETWORKS

SEMIANNUAL TECHNICAL SUMMARY REPORT TO THE DEFENSE ADVANCED RESEARCH PROJECTS AGENCY

1 APRIL — 30 SEPTEMBER 1981

ISSUED 5 MAY 1982

LEXINGTON MASSACHUSETTS

ABSTRACT

This report describes the work performed on the DARPA Distributed Sensor Networks Program at Lincoln Laboratory during the period 1 April through 30 September 1981.

CONTENTS

DISTRIBUTED SENSOR NETWORKS

I. INTRODUCTION AND SUMMARY

The Distributed Sensor Networks (DSN) program is aimed at developing and extending target surveillance and tracking technology in systems that employ multiple spatially distributed sensors and processing resources. Such a system would be made up of sensors, data bases, and processors distributed throughout an area and interconnected by an appropriate digital-data communication system. Surveillance and tracking of low-flying aircraft has been selected to develop and evaluate DSN concepts in the light of a specific system problem. A DSN test bed that will make use of multiple small acoustic arrays as sensors for low-flying aircraft is being developed and will be used to test and demonstrate DSN techniques and technology. This Semiannual Technical Summary (SATS) reports results for the period 1 April through 30 September 1981.

A plan for the evolution of the test-bed hardware configuration during the next year has been formulated and is presented in Sec. II. At the end of that period, each test-bed node will contain a Digital Communication Unit (DCU), up to two attached Motorola 68000 processors for tracking, and a signal-processing subsystem. The signal-processing subsystem also provides for data recording and playback. The test bed will also include an Experimental Control Computer that can serve to provide interim internodal communications by means of telephone lines.

The present status of the test-bed hardware and software is reported in Sec. III.

Four nodes are now operational for data acquisition purposes. One of those is installed in a vehicle for mobile operation, but the vehicle currently requires access to commercial power. An order has been placed for an acoustically quieted generator to remedy this situation. Three nodes now contain array processors that will be used to perform signal processing in

real time. With minor exceptions, all the hardware required for the signal-processing and data-recording subsystems for six nodes is now on hand.

Two test-bed nodes have had Motorola 68000 processors added to them and have been used to demonstrate real-time azimuth tracking by those two nodes, using precomputed and recorded signal-processor outputs as inputs. Azimuth track outputs from the two sites were combined using a PDP-11/70 to produce position tracks and confirm that it will be possible to do the same in real time using a 68000. Progress with the design and development of test-bed software for the 68000 processors and the signal-processing subsystem is also reported.

The results of a preliminary design effort directed at the development of an experimental Communication and Self-location subsystem for the DSN test bed are reported in Sec. IV. The system will consist of a Digital Communication Unit that is being developed within the DSN project and a Radio Unit (RU) that is being developed as part of a separate Communications Network Technology program. The DCU will be fabricated primarily of commercial board-level components, including a Motorola 68000-based single-board computer. The status of our investigation into more advanced multiprocessor computer configurations for test-bed nodes is summarized in Sec. V.

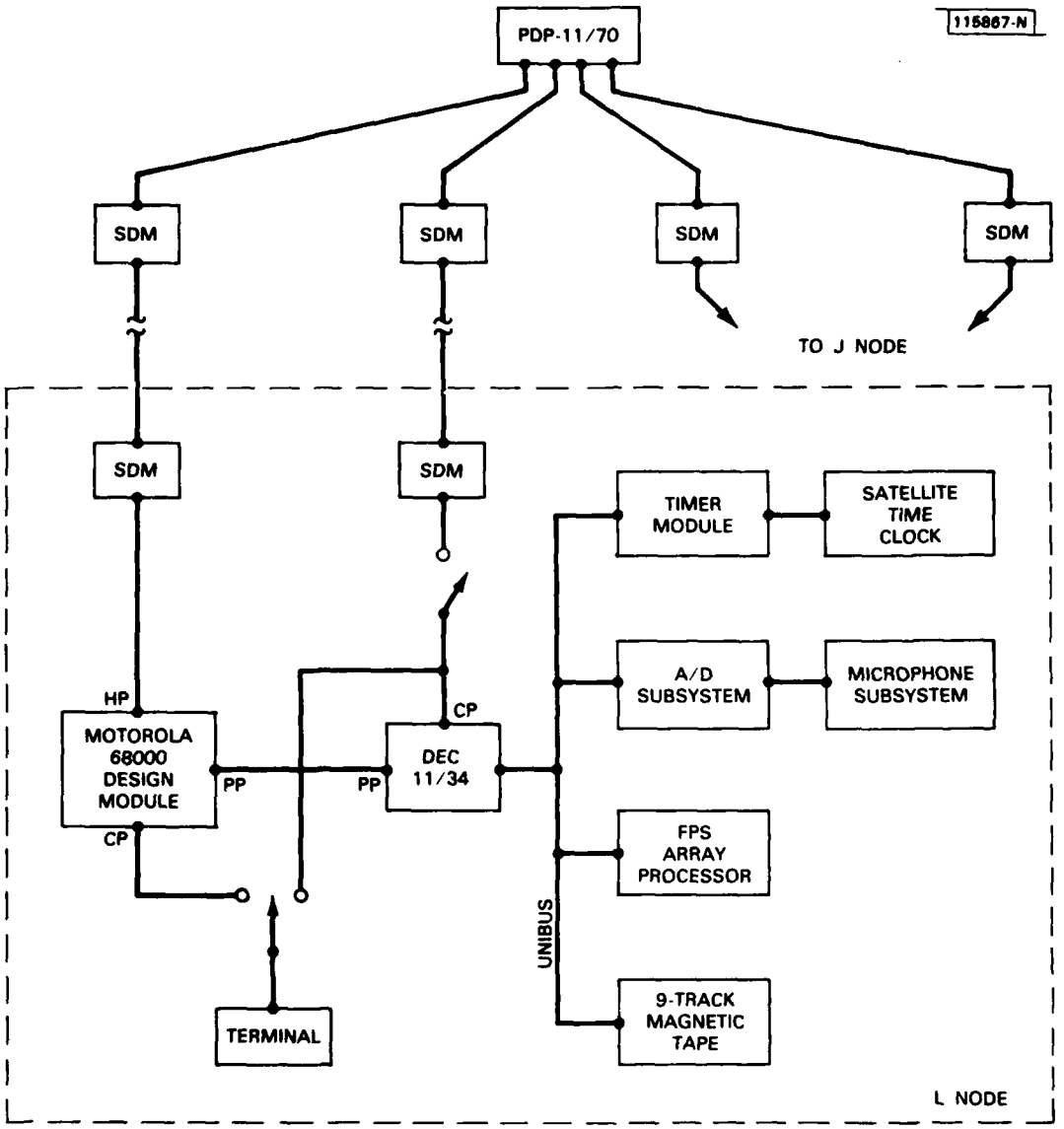
Section VI reports on papers that have been written dealing with interprocess communication in a distributed environment and with resource control in tracking processors. It also describes an approach for handling of target hypothesis trees in a distributed system and how tracking tasks might be distributed over processors.

II. TEST-BED CONFIGURATIONS PLAN

A plan has been formulated for developing the test-bed nodal hardware over the next year in such a way as to enable both single-site and multisite tracking to be accomplished in each node, and to provide necessary internodal communications using both telephone lines and, eventually, radio communications. The plan provides for expansion of each node from the present configuration to a multiple-processor configuration sufficient to experimentally support all essential DSN functions. Development of test-bed nodes beyond what is described here will depend upon the results of our investigation of new nodal architectures.

At the present time, two of the test-bed nodes are configured as shown in Fig.II-1. This configuration consists essentially of a standard nodal signal processing subsystem augmented with a Motorola 68000 computer. Other nodes now contain only the signal-processing subsystem in various stages of deployment as reviewed in Sec. III. As shown in Fig.II-1, the J and L nodes are each connected to a PDP-11/70 computer by two communication links. The PDP-11/70 provides general-purpose software development and research services to the DSN project and also provides experiment monitor and control function for the attached nodes. The test-bed plan provides for a six-node configuration with a dedicated Experiment Control Computer (ECC) replacing the PDP-11/70 and with each test-bed node connected to the ECC through only one communication line. The plan includes configurations for the ECC and its usage as a communication emulator until the time that radios are integrated into the test bed.

All nodal hardware shown in Fig.II-1, except for the 68000 Design Module and the modems, constitutes the data-acquisition and signal-processing portion of the nodes. These will remain unchanged, but the Design Module, which is a now obsolete prototype board, will be replaced with the new 68000 Versamodule systems. Six Versamodule systems have been purchased, one for each test-bed node. The configuration for the new version 1 node is shown in Fig.II-2. It consists of a signal-processing subsystem controlled by the 11/34 and a 68000 processor which is used for both communications and tracking functions. The



LEGEND SDM = SHORT-DISTANCE MODEM
HP = HOST PORT CP = CONSOLE PORT
PP = PARALLEL PORT

Fig. II-1. Present configuration of J and L nodes.

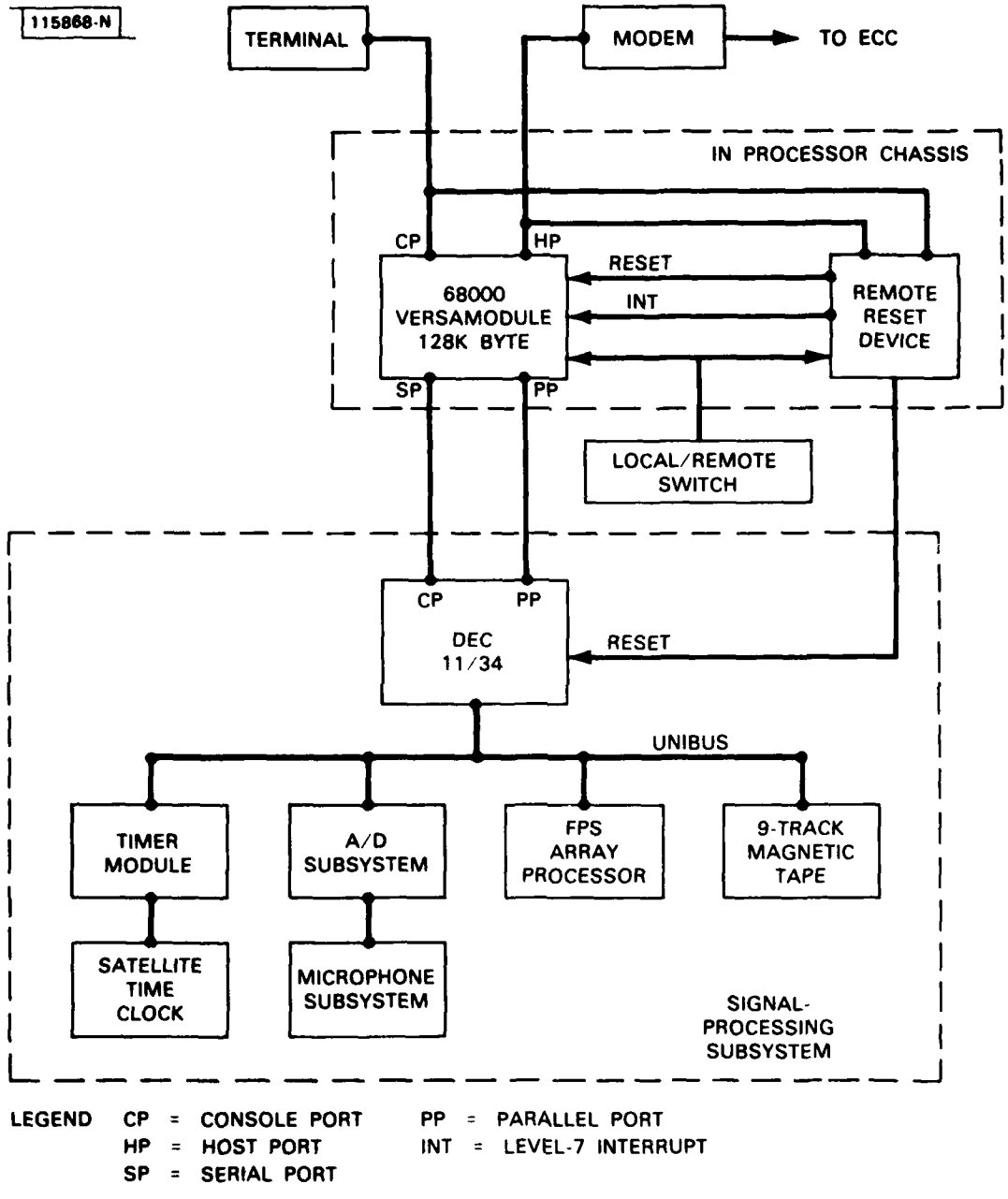


Fig. II-2. Version 1 node configuration.

node is capable of local operation by means of its attached terminal and of remote operation over the network. In addition, the node is equipped with a Remote Reset Device (RRD) which monitors the communication line for reset codes. Upon receipt of an appropriate code, the RRD can interrupt the 68000, reset the 68000, or reset the 11/34. This facilitates a recovery from a crash in a remote node.

The main functional advances represented by the version 1 node and the ECC are: (1) Improved nodal reliability and suitability for remote operation, (2) ability to function with only one communication link per node, (3) ability to operate the test bed as a truly distributed system with the ECC providing internodal communications by means of message switching between attached communication links as indicated in Fig.II-3. The ECC will also be used to initialize the system and to monitor distributed-tracking experiments.

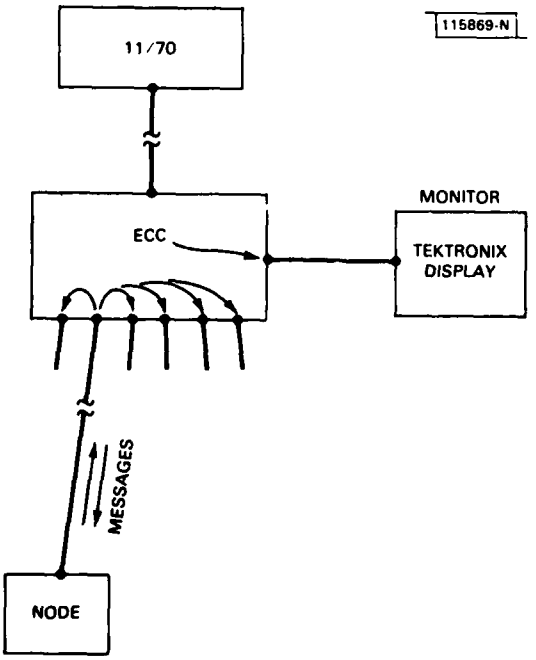


Fig.II-3. Role of the Experiment Control Computer (ECC) in communications emulation and test-bed monitoring.

The configuration of the ECC is shown in Fig.II-4. It will use a 68000 processor with 128K bytes of memory. The serial data links to each of the nodes are interfaced to the ECC processor by means of Motorola MCCM cards which support four DMA channels each. This will allow the links to operate at a full 9600 baud while minimizing the CPU time required. The ECC can be controlled either locally, using the Tektronix display, or remotely from the 11/70 computer. All program downloading into the nodes will be performed over the link to the 11/70 via the ECC. Data logging can also be performed using the link to the 11/70 although an architectural provision has been made for the future addition of an ECC disk for this purpose.

For the version 1 nodal configuration, a single 68000 processor on each node must be time shared to perform both single-site and multisite tracking as well as simple communications through the ECC. This is a large load for a single unit, and we are now considering whether we should evolve the nodes directly into version 2 nodes. The version 2 nodal configuration shown in Fig.II-5 provides for up to two tracking processors and a Digital Communications Unit that will be dedicated to communication functions. The DCU (which is discussed in more detail in Sec. IV) and the Radio Unit and Modem will constitute the Communication and Self-location subsystems for the test-bed nodes. The RU is shown dotted because it will not be integrated into the nodes until FY 83.

As shown in the figure, a remote maintenance module is included in version 2. The function of this remote maintenance module is to allow remote checkout of a node over the network and to reset or interrupt processors attached to the DCU. Some of the remote maintenance functions planned for implementation are:

(a) Turn on and off the battery charger.
(b) Turn on and off calibration speaker.
(c) Monitor battery power to microphones.

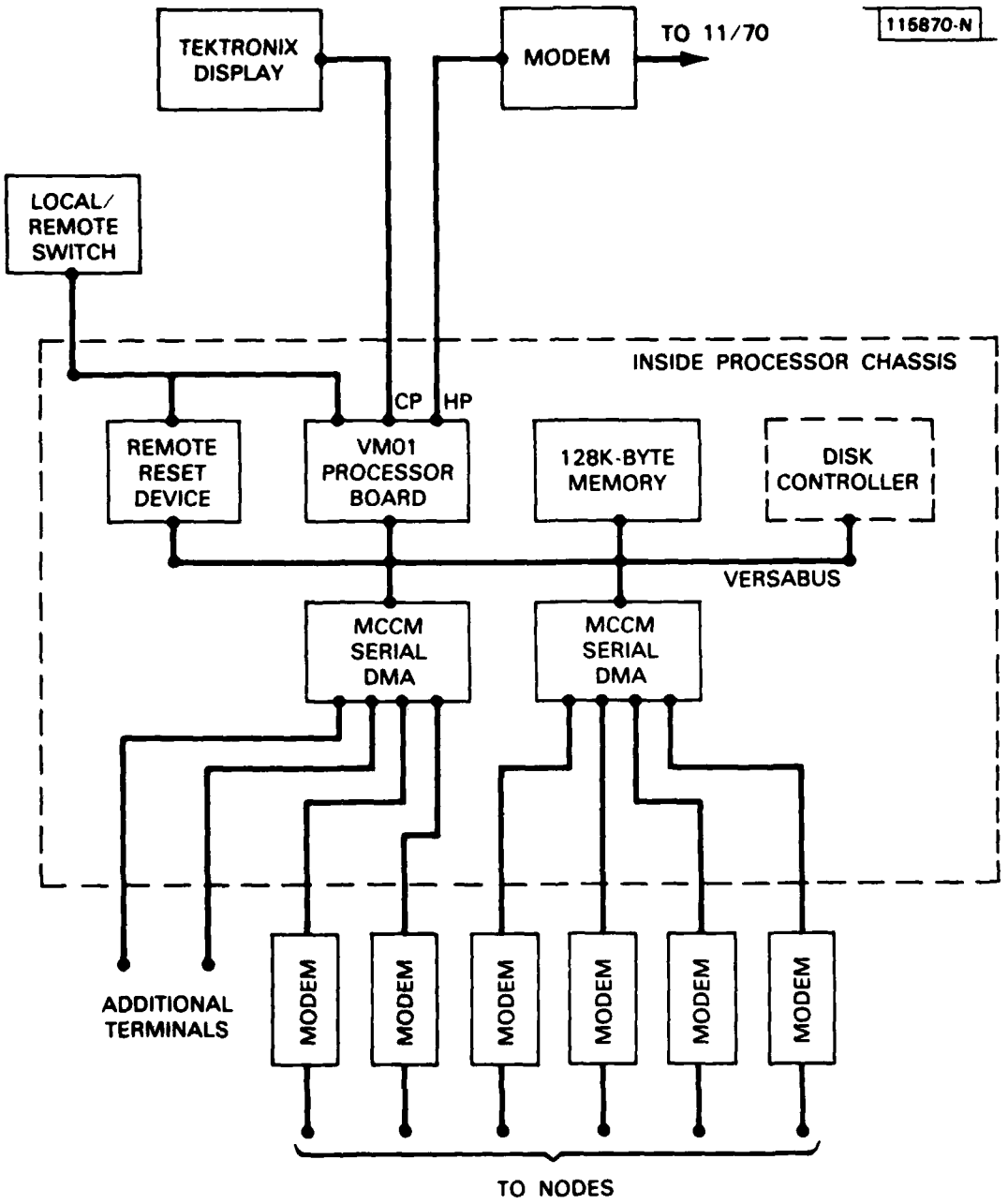
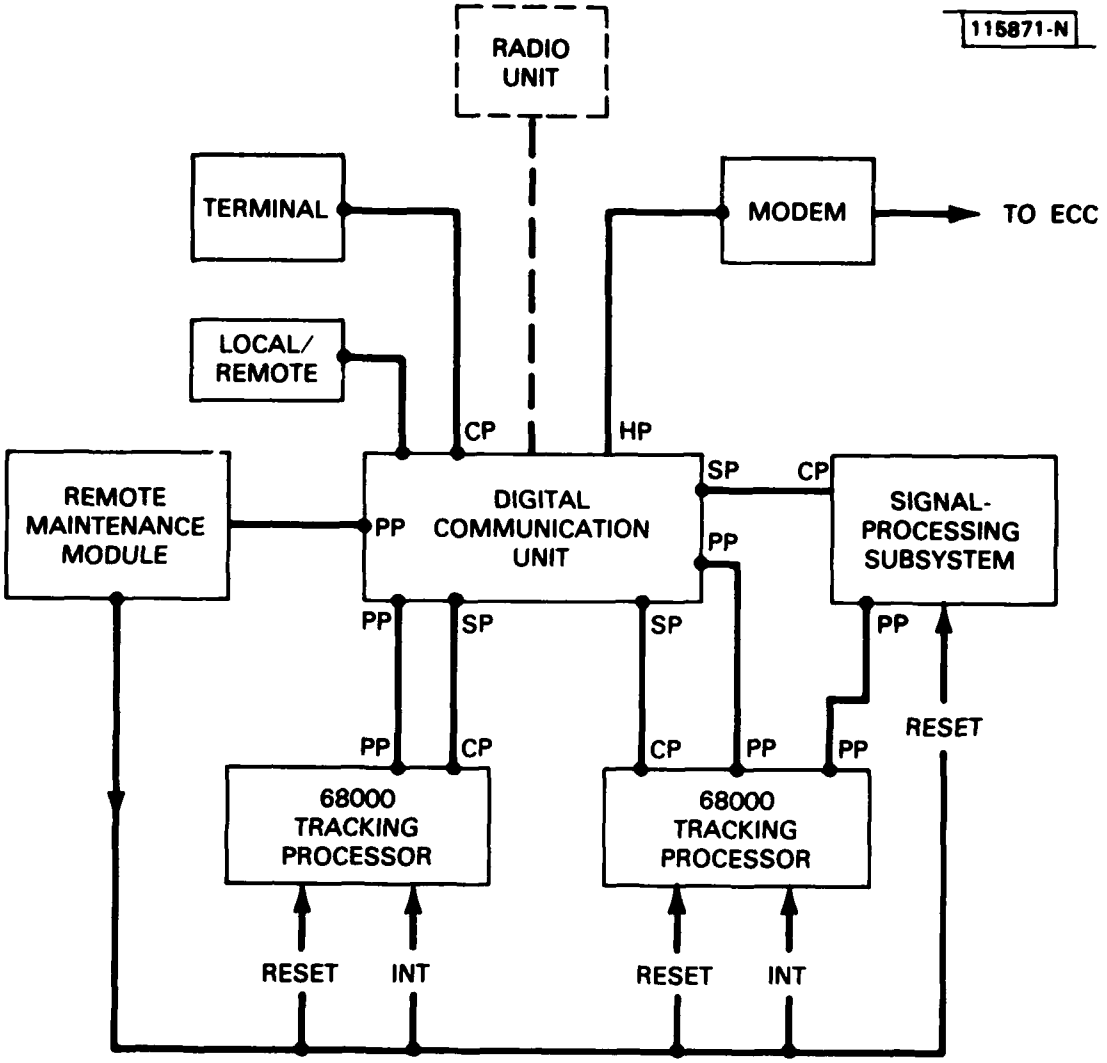


Fig. II-4. Experiment Control Computer configuration.



LEGEND PP = PARALLEL PORT
SP = SERIAL PORT
CP = CONSOLE PORT

Fig.II-5. Version 2 node configuration with two tracking processors.

While most of the new features of the nodes are oriented toward operating the system as a network, provision will be retained for operating nodes in a data-collection mode. It will continue to be possible to load the 11/34 from magnetic tape and operate it in stand-alone mode under local terminal control.

III. TEST-BED STATUS

A. NODAL SIGNAL PROCESSING SUBSYSTEM

The following summarizes the hardware status of the test-bed signal processing subsystems. These subsystems are comprised of the elements shown within the dotted line of Fig. II-2. The basic elements provide sensing, data recording, time stamping, and signal-processing capabilities for all node configurations.

As of this writing, four signal processing subsystem nodes are operational for data acquisition purposes. Three of these are also operational for signal processing. The microphone array and calibration speaker for a fifth node have been deployed on an unused radar tower in preparation for the installation of the electronics for that node. One of the operational data acquisition nodes is the first of three planned mobile nodes. This mobile node was assembled after the first of three custom vehicles, being purchased from Poindexter Industries in Houston, Texas, was delivered in early September.

With the exception of one PDP-11/34 computer, which was returned to the factory for replacement; one magnetic tape transport, presently on order; and one array processor which is to be ordered, all hardware to complete the fifth and sixth nodes is on hand. This hardware will be installed in the next two DSN vehicles as they are delivered from Houston, Texas. At that time we will have three fixed and three mobile nodes.

The first mobile node does not yet contain a self-contained power system. Outlets to supply power are being installed at outdoor locations in close proximity to presently deployed microphone arrays. This will allow any or all of the mobile systems to be checked out and used with any of the established arrays.

Specifications for a quiet engine/generator power system were submitted to four manufacturers of such equipment. Bids were received and evaluated, and a purchase order for one such system was issued to J & A Enterprises of Swampscott, Massachusetts. The power system, scheduled for delivery in December 1981, consists of an Onan 15-kW gasoline-driven, air-cooled

Fig. III-1. Mobile DSN node.

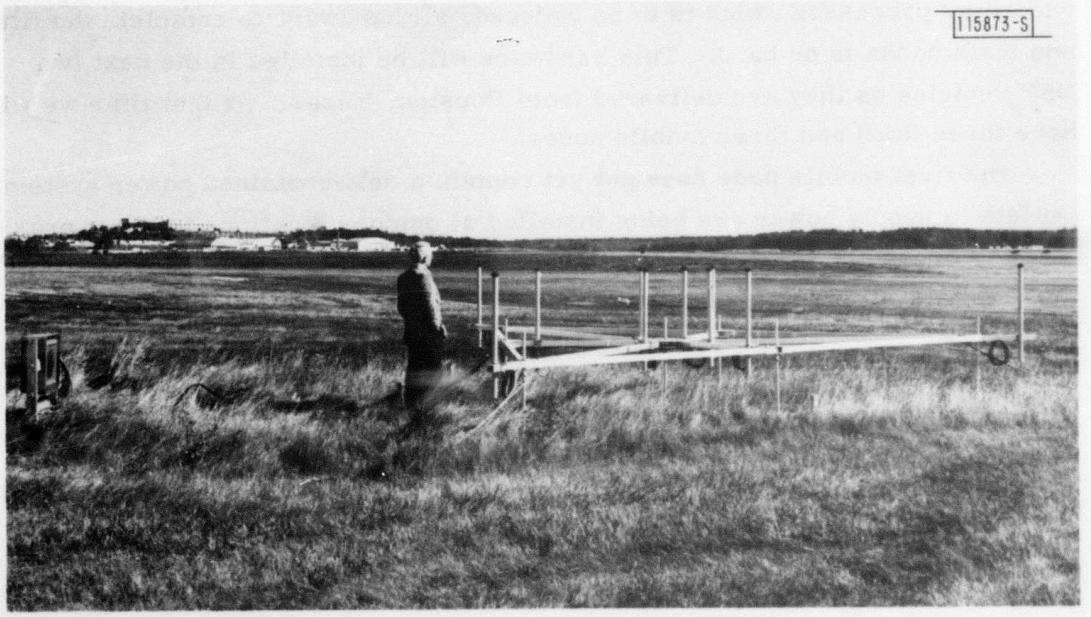


Fig. III-2. Deployed DSN acoustic array.

engine/generator (E/G) set housed in a custom-designed sound-attenuating enclosure. The E/G power system will mount on the rear platform of the DSN vehicle. Figure III-1 is a picture of the vehicle showing the rear platform, measuring 6 × 8 ft, which will accommodate the E/G power system. Equipment racks are visible through the open door.

The microphone array at the Lincoln Laboratory Flight Facility was moved from the hangar roof to an open grassed area west of the hangar area. This required a 1000-ft signal cable run from the control electronics to the array. Calibration and noise-background recording tests have shown this node to be completely operational and in a much better location than was the case before the move. Figure III-2 shows the array in its new location with the calibration speaker off to the left of the array.

Time-of-day (TOD) timing systems using both NBS WWVB time broadcasts and Geostationary Operational Environmental Satellite (GOES) time transmissions were evaluated for use as a common source to time synchronize the six DSN nodes (for time tagging data tapes, etc.). The satellite TOD clock system was found far superior to the WWVB radio system for our purposes. The primary reason for this was the weak WWVB signal levels in this area. With the satellite system, time synchronization was never lost after the system was turned on. The True Time Instrument's model 468-DC TOD digital clock satellite unit was evaluated, and six of these units with appropriate antennas have since been ordered and received. Installation of the clock systems at the nodes is under way at the present time. The construction of six DSN timer units was completed during the past quarter, and all have been checked out with the 11/34 computer interface. It is through these timer units that the TOD clocks communicate with the 11/34 computer.

B. REAL-TIME TRACKING

In the prior SATS,[1] it was reported that the single-site tracking algorithms had been converted to run on a Motorola 68000 processor. This work has been expanded to allow the demonstration of real-time tracking of targets by two nodes. In this demonstration, prerecorded outputs from signal-processing algorithms are input to single-site trackers running in the two nodes at a

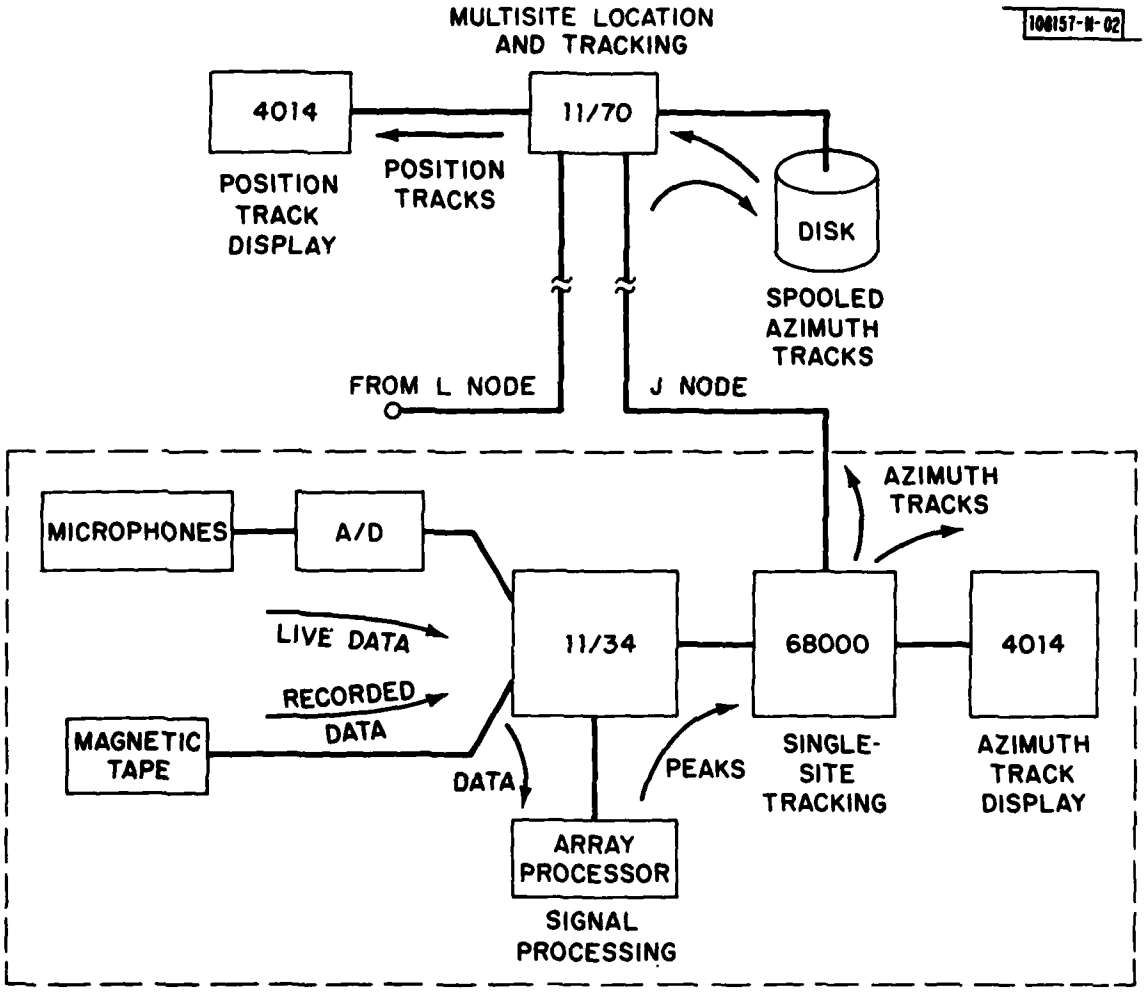


Fig. III-3. Experimental data flow.

real-time rate. The output from the trackers is then transmitted to a central 11/70 computer where the azimuth tracks are combined in real time to generate target location tracks.

The experimental configuration used from this demonstration is shown in Fig. III-3, which is similar to Fig. II-1, but with emphasis placed upon the use of that configuration for the real-time demonstrations. The two nodes are designated L and J, corresponding to the buildings on which the acoustic arrays for these nodes are located. Each of these nodes has a Motorola 68000 processor which performs single-site tracking and communicates track outputs to the PDP-11/70 over a 9600-baud line. Each node contains a PDP-11/34 computer, which controls the data acquisition and the signal processing. The 11/34 is also connected to the PDP-11/70 by a 9600-baud line for program loading. The J node is equipped with a Tektronix 4014 graphics display which is used for displaying azimuth tracks in real time. The PDP-11/70 computer is also equipped with a Tektronix display on which the locations and position tracks of the targets can be displayed.

The hardware configuration of Fig. III-3 will support processing live data from microphones, recording of live and processed data, and processing of previously recorded data. Pending completion of the real-time signal processing software, raw microphone data are recorded onto magnetic tape. This is then processed off line to produce a tape containing peaks (estimated target power levels and azimuths). The peak tape is then mounted, and the peak records are read by the 11/34 and passed to the 68000 in simulated real time. The peak data are then processed by the single-site tracking software to produce azimuth tracks as a function of time. Tracking load and performance are exactly as they would be with live data being processed through the signal processor in real time. Azimuth tracks are transmitted over 9600-baud links to the 11/70. In addition, the azimuth tracks for node J can be displayed on the graphics display attached to that node.

Azimuth track data are spooled into disk files upon being received by the PDP-11/70. The multisite location and tracking algorithm takes its inputs from these spool files. Spooling is used so that the multisite tracking need not run in real time. This facilitates testing when other users are sharing the

11/70. With no other users on the 11/70, the multisite location tracker operates in real time, including the display of position tracks on an attached Tektronics 4014 terminal.

The multisite location algorithm uses the location technique described previously.[2] In this algorithm, each azimuth track point received from a node is compared with azimuth tracks from other nodes to form locations. These locations are then formed into position tracks, which are plotted on the graphics display. The algorithm now runs in the 11/70 computer. It will be converted to run in a nodal 68000 processor for future experiments and demonstrations in the test bed. The algorithm is implemented using dynamically allocated lists for nodal azimuth tracks and for target tracks. As a result it can accommodate different scenarios and variations in the number of nodes in the network.

The multisite location algorithm used less than 12 percent of the 11/70 CPU time when processing two-node data, with an average of two azimuth tracks per node as input. The processing load will increase with the number of nodes and the number of tracks per node. We estimate that it should be possible to process inputs from at least four nodes in real time with a single 68000 processor.

For the existing algorithms, the single-site tracker is a larger consumer of CPU time than the two-site location tracker. Depending upon its parameter settings, it can consume more than the available CPU time in a single 68000 processor. Consequently, a study of processor resource allocation methods was undertaken as part of the initial tuning of the single-site tracker to make it run in real time in a single processor. That study is reported separately in Sec. VI-2.

To assist in the study of processing bottlenecks in the single-site tracker, a special piece of hardware was constructed to non-intrusively measure the average time consumed within any section of code. The device is triggered to start and stop elapsed-time measurements based upon access to specific addresses on the 68000 bus. The addresses are settable using DIP switches.

Using this hardware, it was found that the clustering routine used far more CPU time than expected. As a result, the clusterer was rewritten from a

"combine the closest pair" algorithm to a stream clusterer, which improved the speed considerably.

Figure III-4 shows the position tracks obtained from a 3-min. run using data recorded during the fly-by of a T-28 aircraft. The track of the T-28 is also shown for comparison as is the map showing highways and other major features in the vicinity of the experiment. Graphic displays of two-site tracks, with a background map and scales, are produced as part of the output of the two-site tracking algorithm.

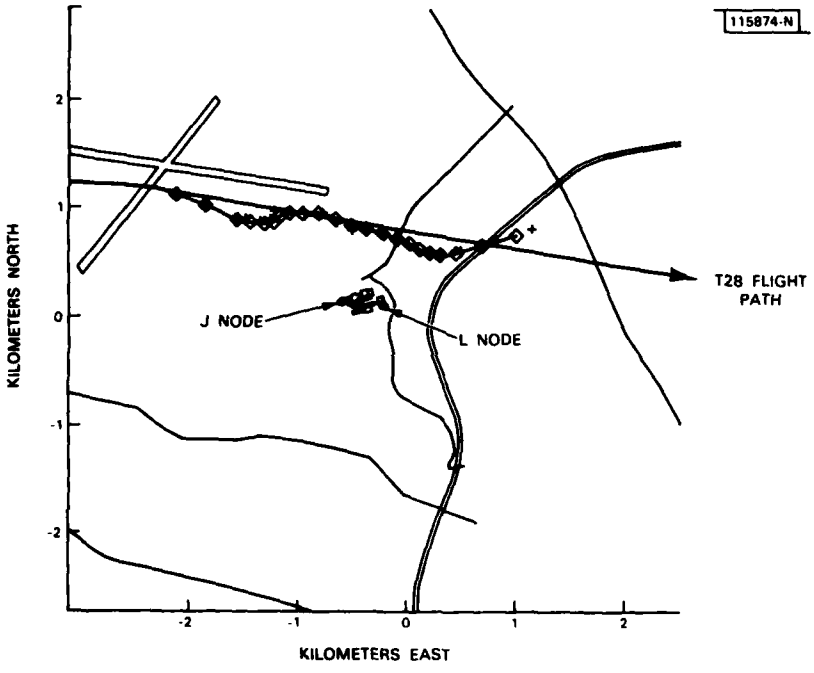


Fig. III-4. Target position track plot for T-28 aircraft data.

As a result of these activities we have:

(1) Confirmed that we can operate our single-site tracking algorithms in real time in a single 68000 processor, provided that the number of targets is small.

(2) Confirmed that our multisite locator algorithm will operate in real time in a single 68000 processor for a small number of nodes. This conclusion is based upon the fact that a two-node PDP-11/70 version operates in real time using only a small fraction of the PDP-11/70.

In addition, considerable insight has been gained concerning the performance of tracking algorithms and the interactions between the algorithms, the estimation of azimuths by the signal-processing software, and the geometrical configuration of the targets and tracking nodes. This can now be applied to improve algorithms and system performance.

C. MC68000 SYSTEM SOFTWARE

Extensions are being made to the 68000 executive software to support DMA and interrupt-driven I/O, as well as multitasking. These extensions are being made to provide the support necessary for the development of applications-level software for the Experiment Control Computer (ECC) and the Digital Control Unit (DCU). The philosophy of the planned extensions is to make them as simple as possible.

We now operate our 68000's without a resident operating system. User programs obtain operating system level services by means of calls to executive subroutines. These subroutines are contained in the run-time library which is linked with the user's program prior to downloading from the 11/70 computer. This same philosophy will be continued, with the extensions taking the form of added subroutine calls.

Programs are developed using the PDP-11/70 computer where they are edited, compiled, assembled, and link edited with the run-time library. The resultant absolute load module is downline loaded into the 68000 over a serial ASCII link. The code is executed in the 68000 under the control of a small ROM resident monitor that provides breakpoint debugging facilities. Programs are mostly written in the C language with executive routines patterned after UNIX routines whenever possible.

A load module is able to run on any configuration 68000 system provided that the 68000 is equipped with suitable ROMs. These ROMs contain a device table and other configuration data which are interrogated by the executive-level routines. These routines are then able to modify their actions to be appropriate to the configuration on which they are being run. This technique avoids having to generate separate load modules for each configuration and thereby simplifies software development.

The current 68000 I/O package supports programmed I/O. That is, data are read from, or written to, the I/O control circuitry directly by the processor. Interrupts are not used and data are transferred only at the time an I/O request is made by the user's program. This has been adequate for existing tracking and display programs that talk to one I/O port at a time. However, the DCU and the ECC must handle multiple asynchronous data streams with moderate to high data rates. At the same time, they must be capable of communicating with their operator console terminals. Interrupt-driven I/O software is being implemented to handle such data streams in parallel and to handle the DMA devices that will be required to provide some of these services.

The console and host ports will be run under interrupts using ring buffers. For DMA devices, the user processes will provide the necessary buffers. In all cases, the I/O control will be by means of queues and queued objects. The queued objects will be requests to fill or empty the user I/O buffers. Each I/O channel will have separate input and output request queues. Execution of user requests will be done in the background by the interrupt I/O routines.

Multitask operation of the 68000 is necessary for the ECC, the DCU, and for running single-site and multisite tracking in one CPU. In each case, the tasks will operate as a group to achieve a common goal. Also they will operate in a sympathetic environment where the actions of one task can be modified to meet the needs of another. This situation has allowed us to plan a simple implementation in which all the tasks are loaded as one module and in which tasks are not protected from one another.

A design document for these extensions has been prepared and is being used as the basis for implementation. The present implementation status is that most of the interrupt-driven I/O routines have been completed and

rudimentary multitasking has been tested. The major items still to be completed are serial and parallel DMA I/O drivers and the use of time functions in multitasking operations.

D. DATA-RECORDING AND SIGNAL-PROCESSING SOFTWARE

A new Data Acquisition System (DAS) has been designed for the nodal PDP-11/34 processors and will be implemented to replace the previous Data Acquisition System[2,3] and the Analysis Server.[4] The new design incorporates nodal signal processing and data recording in one package. The design is more general than the previous one and provides flexibility for algorithm and performance experimentation. In addition, the primitive operations needed to make use of the nodal array processors for real-time signal processing have been designed and implemented.

Data format changes have been made as follows. First, time stamp and node identification information are added to the output data. Second, checksum and resynchronization information are added to allow the data to be sent to intercomputer communications ports as well as to magnetic tape. Third, modifications were made to allow handling of data other than unprocessed acoustic time series. This includes cross spectra, single-channel power spectra, wavenumber spectra, and peaks in wavenumber spectra, as well as parametric information about the nodal hardware and software configuration. Fourth, provision was made for links between parametric data records and other data records whose interpretation depends upon the parameters.

The data formats designed for the new DAS represent an experimental use of computer information storage and communication concepts that are quite different from those usually employed in computer systems. Our general idea is to think in terms of fixed value sets rather than in terms of variables that take on different values at different times. For example, much of the data in a DSN can naturally be considered as time series. Such a time series is a set of values, or value set, where each value has a unique identifier containing the time of the value and a name for the conceptual variable to which the value belongs. The values are constants, they never change, and they can be stored and rebroadcast indefinitely without any logical inconsistency arising in the information structure.

In the new DAS the data values are called external objects, and are fairly large structures. They contain a 16-byte external object identifier, and at least one 8-byte external structure header (external structures are explained below), in addition to any communications protocol information. The size of the external objects will range from several tens of bytes to several kilobytes.

An external object is a modular entity composed of a sequence of structures called external structures. The first external structure in an object serves as a header and contains the object identifier along with sequencing information used in magnetic tape and simple one-way communications media protocols. The next external structure is the principal structure in the object, i.e., the data. Following the principal structure are optional print names, time-and-node stamp, comment, and error message structures.

In defining external object formats it was necessary to address the issue of computer-independent standards for binary data representation. Our solution, suggested in part by the INTERNET protocol,[5] was to adopt the byte-oriented format of the IBM 360, in which numbers are 1, 2, 4, or 8 bytes long and are aligned at a displacement within their containing structure which is an exact multiple of their length. For numerical representation, we have adopted the IBM standard for integers and the IEEE standard for floating point. Both standard and computer-specific external objects are allowed. Since a 1-byte integer is stored in the same way by all computers, a part of the external object header is used to indicate if the object is standard or for a specific computer type.

In addition to formulating a new DAS design, the primitives needed to interact with the FPS-120B array processors attached to the PDP-11/34's in the test-bed nodes have been defined and implemented. With these tasks completed, we can now proceed to implement real-time signal-processing algorithms in the test-bed nodes and implement improved data collection software that will provide for processed as well as unprocessed data collection.

The primitives for dealing with the FPS-120B copy data and code between the PDP-11/34 and array processor memories, start an array processor function, and wait for the array processor to complete a processing step. They are executed by the PDP-11/34, and they provide the programmer with

considerable flexibility in using the array processor. Array-processor and general-purpose code can be easily mixed.

The memory copy primitive copies blocks of data between addresses in a single 32-bit address space into which both PDP-11 and array processor memory have been incorporated. Input and output routines can use memory copy to input directly to the array processor memory or to output directly from that memory. The same array processor memory can be accessed under different addresses to get the same information in different formats, e.g., an array processor word may appear as a 32-bit PDP-11 floating-point number when accessed with one address, or as a 16-bit fixed-point number when accessed with another address. Memory copy is implemented as a sequence of atomic units that copy up to 32 bytes and then stop to allow the PDP-11 to interrupt. This avoids potential timing problems in a real-time system.

REFERENCES

1. Distributed Sensor Networks Semiannual Technical Summary, Lincoln Laboratory, M.I.T. (31 March 1981), DTIC AD-A108275.
2. Ibid. (31 March 1980), DTIC AD-A091766/6.
3. Ibid. (30 September 1979), DDC AD-A086800/0.
4. Ibid. (30 September 1980), DTIC AD-A103045/1.
5. "DoD Standard Internet Protocol," Comput. Commun. Rev. 10, 12-51 (1980).

IV. COMMUNICATION AND SELF-LOCATION SUBSYSTEM

A Communication and Self-location subsystem is being developed for use in the DSN test bed. In the short term it will be used to provide internodal DSN communications by means of telephone lines interconnecting nodes through a single switching computer. In the longer term it will provide all necessary test-bed communications and self-location functions by means of digital radios. The subsystem will consist of two parts: a Digital Communications Unit (DCU) and a Radio Unit (RU). The results of a preliminary design effort directed at the development of the communication subsystem are reported below. The RU is being developed under a companion Communications Network Technology program.

A. DCU HARDWARE DEVELOPMENT

Hardware options for the implementation of a DCU have been investigated and a decision made to maximize the use of off-the-shelf hardware for the test-bed DCUs. A tentative decision was made to use the Motorola Versamodule family of boards for the DCU and the following report reflects that decision, but the more recently available Stanford University Network (SUN) single-board processors appear to be an interesting alternative. We will evaluate the use of those processors in the DCU before completing a detailed design and starting implementation. The SUN card interfaces to the very widely used Multibus, whereas the Versamodule interfaces to the less widely used Versabus.

Each DCU in the test bed will consist of monoboard computers, additional memory boards, boards with high-throughput parallel DMA channels, and an additional experimental board to provide extra serial I/O ports and remote maintenance and control services for test-bed nodes. The minimum essential configuration will be with one computer, one memory board, one DMA board, and one experimental board.

The Versamodule monoboard computer includes two serial 9600-baud ports, 32K of RAM, 32K of PROM, timers, and four programmed parallel I/O ports. One serial port can be used to interface to a local terminal and the other for a phone-link connection through modems to the test-bed Experiment

Control Computer. The four parallel I/O ports can be used to monitor or control other subsystems in the node.

A four-channel parallel DMA board has been announced by Motorola, and its suitability for use in the test-bed DCU is being investigated. The tentative plan is to use three of the channels to interface the DCU to the RU. Two 16-bit parallel channels will be reserved for receiving and sending packet text. Although the RU will operate in half-duplex mode, two channels will be used because a transmit packet might be preempted by a reception. The third channel is allocated to half-duplex status and command communications between the DCU and RU. It will alternately pass command and status records that allow the DCU to control the RU and that inform the DCU of RU packet receptions and transmissions as well as other RU status information. The fourth channel is a candidate for a high-speed interface between the DCU and the rest of the test-bed node.

The memory board in a DCU will provide for data transfer between the DCU and RU on a non-interference basis with the DCU processor board and will provide the extra memory needed to support all self-location as well as communication functions. If necessary, an additional processor board will be added to support self-location. In the early stages of use in the test bed, with communications provided by telephone lines, the DCU will also support a limited amount of tracking software.

Versamodule back planes now come in four-slot modules so there is a size, weight, and cost penalty in going beyond four boards in a single DCU. But to minimize the design and development of special boards and maximize use of commercial boards, it appears that the test-bed DCU may contain as many as six Versamodule boards. It does not appear that there would be a corresponding problem with the use of the Multibus and SUN processors. That option offers the potential for reduced size, weight, and cost with equivalent or improved functionality.

B. TIMING CONSIDERATIONS

The RU will be a pseudonoise spread-spectrum unit with code changing for each bit throughout a packet. The spreading code for the first bit of a

packet is called the seed code. The seed determines the rest of the sequence in a packet, and the seed will be changed under DCU control as a function of time. The seed will be changed no more often than once each 10 ms, and no individual node will broadcast more than one packet in a 10-ms interval. This allows for enough broadcasts to provide the needed DSN communications and, because the seed changes often and is not reused by any transmitter, will provide good resistance against repeat jammers.

Consistent with the 10-ms minimum time between possible changes in code seeds and transmission of packets, time in the communication subsystem will be decomposed into two components. A fast component will keep track of times smaller than 10 ms, and a slow component will keep track of time in units of 10 ms. The slow time will be maintained by software in the DCU, and the fast time will be maintained in hardware by the RU. The 10-ms beat will be provided to the DCU by the RU hardware clock. The use of 10-ms code changing intervals and a number of other implementation issues have led to this decomposition. The DCU will select the slots in which the RU is to attempt transmissions. It will initiate a transmission attempt by the RU in a selected slot by means of a command packet sent to the RU in the previous slot. The command will specify the desired start time within the slot. When a received packet is sent from the RU to the DCU, the RU will also report the arrival time of the first bit in the packet. The reported time will specify only the time of arrival within the 10-ms slot in which the bit arrived.

The fast clock will advance in increments of 10.81081 ns (the reciprocal of the system clock of 92.5 MHz). A 20-stage binary counter in the RU will provide the clockword. (A count of 925,000 corresponds to 10 ms and in the absence of time slew commands is the maximum count before reset.) As mentioned, the RU hardware clock provides the 10-ms "beat" for the DCU software clock. Conversely, the DCU will shift the node's perception of the slot boundary by providing a slew adjustment word to the fast clock. Part of each command packet from the DCU will specify the length of the next 10-ms interval in terms of a count of 10.81081 fast ticks. The purpose of clock slewing in a node is to align the 10-ms ticks of a node with those of its neighbors so all the nodes will be transmitting and receiving with the correct seeds to facilitate communication.

The following outlines one way that new nodes can achieve synchronization with an operating network although the node may initially have no a priori knowledge of network time. At random times and at random nodes, an operating network can broadcast timing packets using a seed known to all potential participants in the network. Such a packet would contain both network time (the slow clock) and the packet transmission time within whatever 10-ms interval it was transmitted. A new unsynchronized node need only listen using the known seed until it receives a packet. It then can set its DCU clock exactly and can compute the slew needed to align its 10-ms ticks with those of the network. The alignment accuracy will be determined primarily by the unknown propagation time of the timing packet that was used. This will be on the order of 50 μs or less, depending upon the distance between nodes. A subsequent bilateral packet exchange between the new member and any synchronized older member can be used to determine the absolute range between the two nodes as well as provide a vernier estimate of the remaining clock offsets. The remaining clock difference will be eliminated via the slew mechanism, and the internode range will be added to self-location tables. Clock alignment after such a sequence will be on the order of a single tick of the fast clock. Of course, clocks will slowly drift relative to each other and distributed procedures will be required to routinely monitor such drift and provide corrections to keep synchronization within acceptable bounds.

Startup, network synchronization, and self-location measurements are ongoing DSN research areas. The approach outlined above is the one being considered for initial test-bed implementation, with a single node arbitrarily selected at startup to be the network nucleus.

V. ADVANCED NODE ARCHITECTURE

Alternatives are being investigated for multiple-processor DSN nodes beyond the version 2 configuration shown in Fig. II-5. The overall specification and preliminary design for an advanced multiple-processor DSN nodal processor has been completed. Motorola 68000's have been used as the basic processor in the system. From the architectural point of view, each nodal processor is flexible both in number of 68000 processors and in the amount of memory these processors may share. The system can be viewed as groups of processors having both private and shared memory, with groups further interconnected by a packet bus.

During the next two quarters we plan to review our design, investigate other alternatives as outlined below, and select an alternative that will be used for experimental purposes in the test bed. Implementation of our present specification for a DSN nodal processor would require the detail design and development of a new 68000-based single-board computer as part of the nodal processor development. Major objectives of the coming months will be to simplify the existing design, adopt commercially available processor boards if possible, and formulate a plan for developing advanced nodes and incorporating them into the test bed. The following summarizes the status of our investigation of alternatives to our existing preliminary design.

We have just received detailed documentation for BBN's Butterfly Processor for consideration as an alternative advanced node processor element. We will review this documentation and evaluate the BBN processor in terms of the DSN application. Some of the areas of concern are the lack of a priority mechanism in messages, the possible inability of the memory map to rapidly expand virtual segments, and the likelihood that the microcode in the Processor Node Controller co-processor will not meet our needs and may be difficult to modify.

Not unexpectedly, several new single-board computers have recently appeared on the commercial market and one in particular may be well suited to the DSN context. The board is now a commercial offering by Forward Technology, and other sources are expected to be available shortly. It is based on

the Stanford University SUN machine whose development was sponsored by DARPA. It is comparatively inexpensive ($3500), has 256 kB (kilobytes) of local memory with parity, a memory map which will rapidly expand virtual segments, and a counter/timer IC which provides one 16-bit timer for the user. The board provides an interface to an IEEE 796 bus (Multibus) and is immediately capable of supporting a shared-memory communication system with a centrally arbitrated access to the shared memory. This leads to the obvious advantage that commercial memory boards, which are already produced in volume, may be used in this system, and, in fact, large numbers of I/O devices are also readily available for installation on an IEEE 796 bus.

As mentioned in Sec. IV, the SUN processor is also being considered as the basis for the Digital Communication Unit (DCU) that is to be part of the DSN test bed. The use of this processor for the DCU as well as for advanced nodal multiprocessor experimentation is attractive since the DCU could then be just another homogeneous element of an advanced nodal configuration.

In the Forward Technology board (as in the BBN implementation), protection of the shared memory is specified by the originator of a read or write operation. This means that if a process in one processor crashes, it may very well corrupt shared data. If other processors rely upon the validity of that data, the crash may propagate. On the other hand, erecting a firewall against this crash propagation by requiring validation of all shared data before use will impose a fairly heavy load on the processors. It is likely that the same firewall in a message-based system, such as our packet bus design, will load the processors less heavily because the overhead for checking an entire message is paid only once, rather than once per word.

Neither the Butterfly nor the SUN processors have any of the special software and hardware maintenance features which we incorporated in our own design. However, if the processors could be augmented with these features, then the major difference between them is essentially the inter-processor communication mechanism. The Forward Technology processor supports shared-memory communication, and the BBN machine emulates this mode in addition to providing a block copy function. The BBN processor, however,

provides a higher potential aggregate bandwidth for communications than either the Forward Technology machine or our design, but without any available prioritized access to communications. Neither the Forward Technology nor the BBN machines provide multidrop or broadcast modes for messages as our design does.

VI. MISCELLANEOUS

A. INTER-PROCESS COMMUNICATION IN A DISTRIBUTED ENVIRONMENT

We are continuing general investigations into software aspects of inter-processor communication systems for real-time distributed environments. A paper[1] that deals with this problem area has been written and will appear in the IEEE Transactions on Communications. It discusses the important issues of reliability, process congestion, throughput and response time and then proposes an inter-process communication system based upon queueable objects. It deals with the logical aspects of inter-process communication, not with the details associated with any particular communication medium. The idea is to devise a uniform software interface for inter-process communication so that processes can communicate in the same way over a range of physical interconnections from shared virtual memory to radios. The inter-process communication system discussed in the paper is an outgrowth of the queueable-object communication system which we have implemented and use in the PDP-11/34 computers in the test-bed nodes.

B. CPU AND MEMORY RESOURCE CONTROL IN A TRACKING PROCESSOR

While developing tracking software to run in the test bed, it became apparent that the number of hypotheses could exceed the memory available to hold them and the CPU time available to process them. This motivated us to investigate how to control these limited resources to obtain good performance while operating within the constraint of available resources. The results of this investigation will be published,[2] and are summarized here.

Figure VI-1 shows a simple hierarchical hypothesis generation model that corresponds to our acoustic tracking algorithms. In this model, processing is done in frame intervals of T seconds. Sensor inputs arrive every T seconds, and are processed to generate level-1 hypotheses, which are used to generate level-2 hypotheses, and so forth. All the level-k hypotheses are created or updated at one pass before proceeding to level k + 1. The hypotheses at level k can be pruned before proceeding to level k + 1.

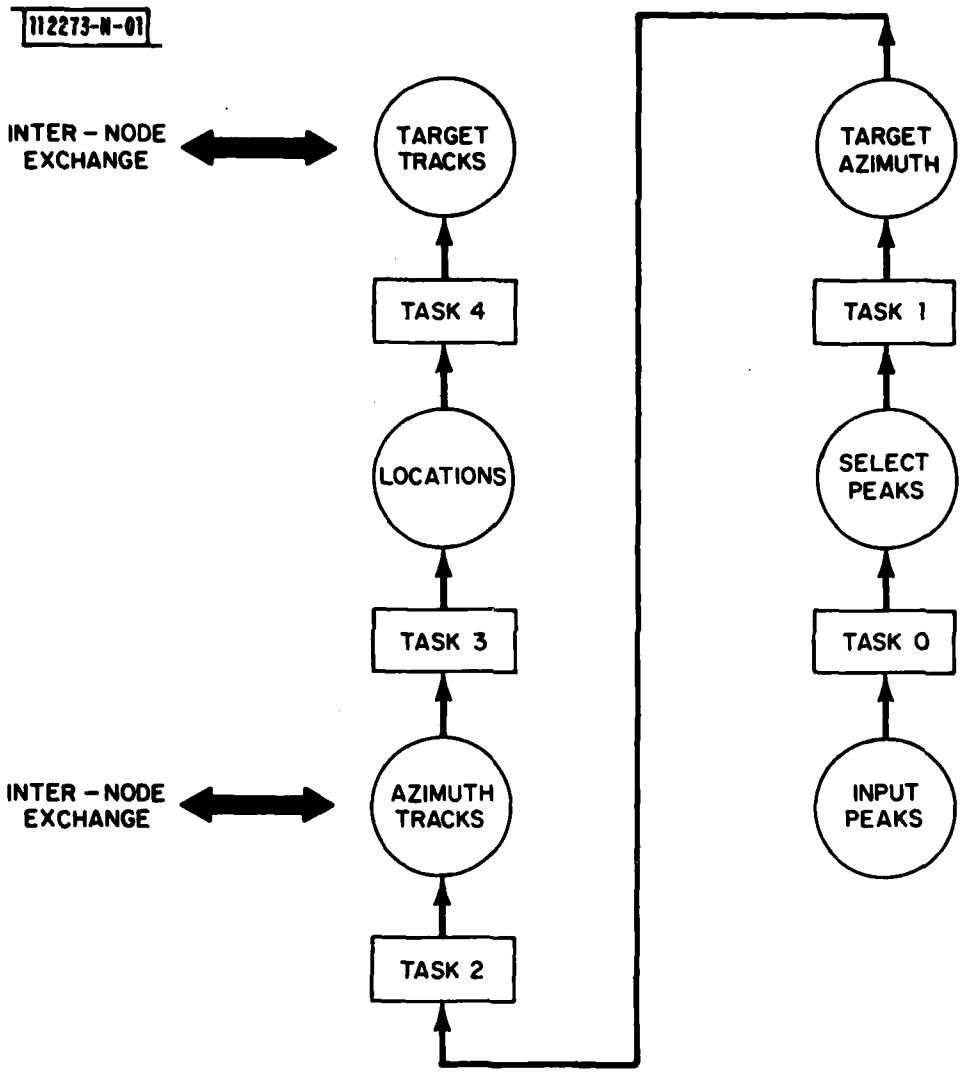


Fig. VI-1. Hierarchical structure of acoustic tracking software.

The nodal DSN tracking software is organized with one task responsible for the generation, evaluation, and pruning of hypotheses at one level of the hierarchy. A single such task performs the functions of the hypothesis Pruner and the Knowledge Sources that generate and modify hypotheses in the Hearsay system,[3,4] but only for a single level of the hierarchy.

We have used a simple model for the consumption of CPU time by tasks to evaluate CPU control strategies. The model for the time required at level k is:

$$T(k) = Cin(k)\ N(k-1) + Cout(k)\ N1(k)$$

where N(k) is the number of hypotheses output from level k and input to level k + 1, N1(k) is the number of hypotheses generated at level k before pruning to N(k), and Cin(k) and Cout(k) are proportionality constants expressing the processing time per hypothesis input and generated at level k, respectively. This model assumes that the processing time is a linear function of the hypothesis counts, which is a reasonable approximation for our DSN algorithms.

For the system to operate in real time, the sum of the T(k) within a frame interval T must be less than T. Some approaches that might be used to do this are:

(1) Generate all feasible hypotheses, but save only those better than some threshold V(k).

(2) Randomly generate feasible hypotheses up to some maximum limit M(k).

(3) Generate all feasible hypotheses, keeping only the best M(k).

(4) Generate a random set of M1(k) feasible hypotheses, keeping only the best M(k).

Obviously it is desirable to retain hypotheses at any level in the hierarchy that are the support for high-quality hypotheses at higher levels. Our analysis of resource control strategies assumed that we can formulate local quality measures for each level such that the selection of good hypotheses will tend to produce hypotheses that are the support for good hypotheses at higher levels.

In order to evaluate the alternative strategies, we required a model for the generation of feasible hypotheses. A Poisson model was selected. The appropriateness of this choice was tested using 10 min. of acoustic data recorded at one of our test-bed arrays. The distribution of the peaks generated by the signal-processing algorithms, each peak corresponding to a feasible hypothesis at the bottom of the hypothesis hierarchy, was measured and found to be well approximated by a Poisson distribution.

The four control techniques suggested above have been reviewed using the Poisson assumption and the linear CPU time model. Method (4) was judged to be the best alternative for initial use by us in the test bed on the basis of performance and implementation simplicity. With M1(k) and M(k) fixed, there are absolute upper limits on resource usage so real-time operation can be assured. But M1(k) can be selected large enough so that very few high-quality hypotheses are completely missed. A number of runs with experimental data have been used to select the two parameters for each task.

The organization of memory to hold hypotheses was also considered. If the memory required to store one hypothesis at level k is H(k), then the total memory required is the sum of H(k) M(k) for all levels plus the largest value of $[M1(k) - M(k)]\,H(k)$, to allow memory for generating hypotheses prior to pruning. By suitably choosing the parameters M(k) and M1(k), we can ensure that the number of hypotheses does not exceed the amount of memory available to hold them. This is provided that the memory space occupied by hypotheses that are eliminated is made available for reuse. We could achieve total reuse by preallocating the space for the M(k) hypotheses at each level, plus a shared buffer used to temporarily hold the excess hypotheses that are generated prior to pruning. For the DSN tested, we are using a dynamic memory allocation scheme for the hypotheses. This offers the benefit of allowing the system to accommodate changes in M1(k) and M(k) while it is running. Such changes might eventually be made in response to changes in performance goals.

C. DISTRIBUTION OF TRACKING TASKS OVER MULTIPLE PROCESSORS

The hierarchical task structure discussed in the preceding section leads to a straightforward decomposition over multiple processors with one or more

layers assigned to each processor. The tasks can be assigned to minimize interprocessor traffic while maximizing the use of the combined CPU time of the processors. This will typically result in tasks residing in the same processor as their input hypothesis level, although this is by no means essential.

In the existing tracking software, hypotheses are kept on linked lists with one list per level of the hypothesis tree. The tracking tasks utilize a List Management System (LMS) to access the hypotheses. To create a new hypothesis, the routine lwrite(list, position, &structure) is used, where "structure" is the hypothesis structure to be entered into the list. To retrieve a hypothesis the routine lread(list, position, &structure) is used, where "structure" is the structure into which the hypothesis is to be copied. Hypotheses may be modified by the lrewrite routine.

The following describes how the LMS system will be modified to handle the situation when the hypothesis tree is distributed over several processors. The LMS software running in each processor will maintain tables of the assignment of different lists in the hypothesis tree to different processors. When an access request to a particular list is made, the LMS software will determine if it is in the local processor or in another. If the list is in the local processor, the access request will be fulfilled locally. If the list is in a remote processor, a message requesting the list function will be sent to that processor to obtain the requested service. This message will contain the hypothesis in the case of an lwrite, or simply the request in the case of an lread. The remote processor will reply with a message containing the hypothesis, in the case of an lread, or an acknowledgment in the case of an lwrite. The result will be returned to the tracking task that initially requested the service. As the tracking tasks use the same calls to LMS for both local and remote accesses, the tracking code will be independent of which processor it runs in.

A general software structure to implement this is shown in Fig. VI-2. At the top level are the tracking tasks which call LMS subroutines to access the hypothesis database. These routines communicate with a guardian of the hypothesis lists being maintained in the local CPU. The guardian has a table

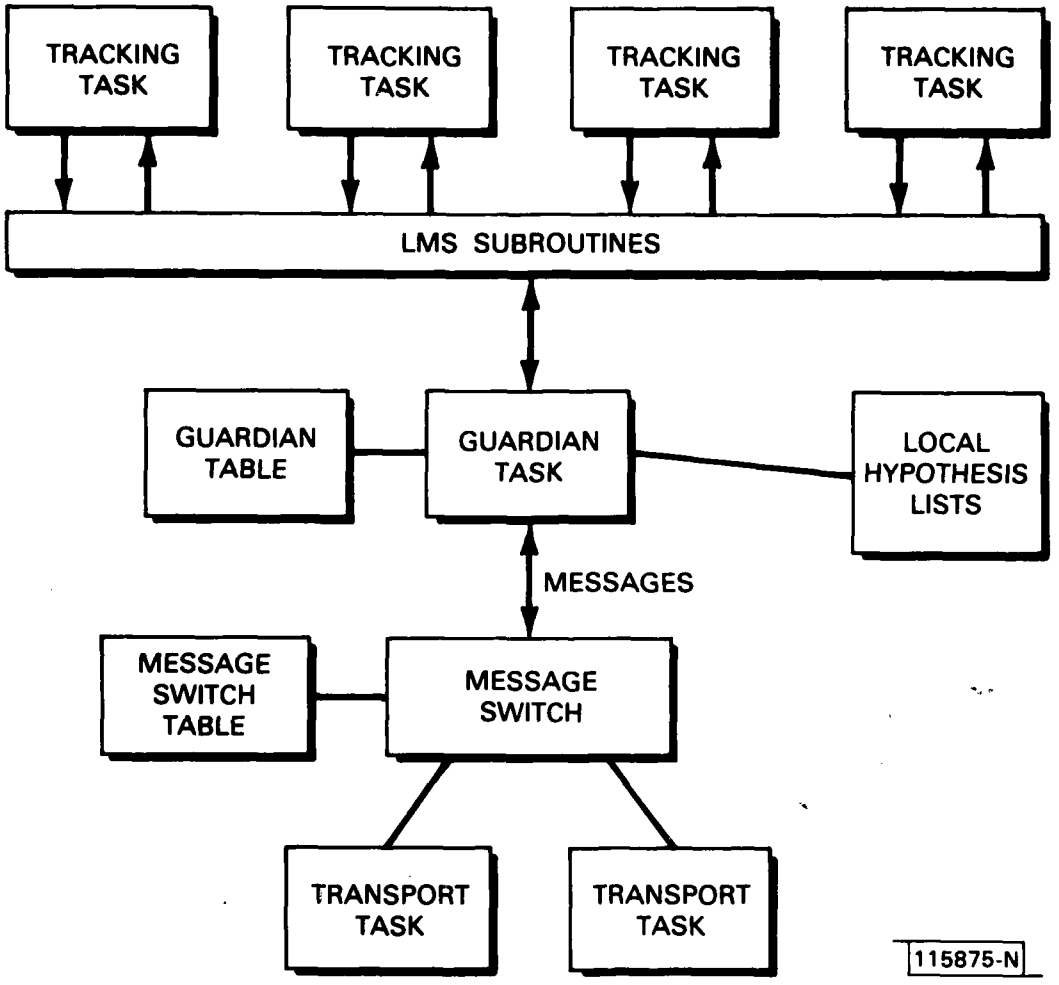


Fig. VI-2. Software structure for a distributed List Management System (LMS).

indicating which lists are maintained by which processor. If the list is in the local processor, the guardian performs the requested action. If the list is not local, the guardian will send a message to the guardian in the appropriate processor.

Messages between guardians will be handled by a general Message Switch in each processor. It will handle messages on the basis of DSN node identification, processor within the node, and task within the processor. Messages will be switched between local tasks. When a message is destined for another processor, it will be switched to the local task that interfaces to that processor. For destinations within the same node, the task will be the one handling the local hardware connection to the processor. For processors on other nodes, the task will be an interface to an appropriate communication network. Such tasks are indicated by the Transport Tasks in the figure.

Message switch tables and guardian tables will initially be assigned manually and will be compiled into the load modules for each individual processor. This is simple to implement and will suffice for many DSN experiments. The dynamic allocation and modification of these tables is a topic for future DSN work.

REFERENCES

1. R. L. Walton, "Rationale for a Queueable Object Distributed Inter-Process Communication System," to be published in the IEEE Transactions on Communications.
2. P. E. Green, "Resource Control in a Real-Time Tracking Processor," to be published in the Conf. Record, Fifteenth Asolimar Conf. on Circuits, Systems, and Computers, Pacific Grove, California, 9-11 November 1981.
3. L. D. Erman and V. R. Lesser, "The Hearsay-II System: A Tutorial," Chap. 16 in Trends in Speech Recognition, W. A. Lea, Ed. (Prentice-Hall, Englewood Cliffs, New Jersey, 1980).
4. V. R. Lesser and L. D. Erman, "Distributed Interpretation: A Model and Experiment," IEEE Trans. Comput. C-29, 1144-1162 (1980).

UNCLASSIFIED

SECURITY CLASSIFICATION OF THIS PAGE *(When Data Entered)*

REPORT DOCUMENTATION PAGE		READ INSTRUCTIONS BEFORE COMPLETING FORM
1. REPORT NUMBER ESD-TR-82-012	2. GOVT ACCESSION NO. AD-A115 729	3. RECIPIENT'S CATALOG NUMBER
4. TITLE *(and Subtitle)* Distributed Sensor Networks		5. TYPE OF REPORT & PERIOD COVERED Semiannual Technical Summary 1 April – 30 September 1981
		6. PERFORMING ORG. REPORT NUMBER
7. AUTHOR*(s)* Richard T. Lacoss		8. CONTRACT OR GRANT NUMBER*(s)* F19628-80-C-0002
9. PERFORMING ORGANIZATION NAME AND ADDRESS Lincoln Laboratory, M.I.T. P.O. Box 73 Lexington, MA 02173-0073		10. PROGRAM ELEMENT, PROJECT, TASK AREA & WORK UNIT NUMBERS ARPA Order 3345 Program Element Nos. 61101E and 62708E Project Nos. 1D30 and 1T10
11. CONTROLLING OFFICE NAME AND ADDRESS Defense Advanced Research Projects Agency 1400 Wilson Boulevard Arlington, VA 22209		12. REPORT DATE 30 September 1981
		13. NUMBER OF PAGES 44
14. MONITORING AGENCY NAME & ADDRESS *(if different from Controlling Office)* Electronic Systems Division Hanscom AFB, MA 01731		15. SECURITY CLASS. *(of this report)* Unclassified
		15a. DECLASSIFICATION DOWNGRADING SCHEDULE

16. DISTRIBUTION STATEMENT *(of this Report)*

Approved for public release; distribution unlimited.

17. DISTRIBUTION STATEMENT *(of the abstract entered in Block 20, if different from Report)*

18. SUPPLEMENTARY NOTES

None

19. KEY WORDS *(Continue on reverse side if necessary and identify by block number)*

multiple-sensor surveillance system
multisite detection
target surveillance and tracking
communication network
acoustic sensors
low-flying aircraft
acoustic array processing
digital radio

20. ABSTRACT *(Continue on reverse side if necessary and identify by block number)*

This report describes the work performed on the DARPA Distributed Sensor Networks Program at Lincoln Laboratory during the period 1 April through 30 September 1981.

DD FORM 1 JAN 73 1473 EDITION OF 1 NOV 65 IS OBSOLETE

UNCLASSIFIED

SECURITY CLASSIFICATION OF THIS PAGE *(When Data Entered)*