BOLT BERANEK AND NEWMAN INC

CONSULTING · DEVELOPMENT · RESEARCH

Report No. 3236

INTERFACE MESSAGE PROCESSORS FOR THE ARPA COMPUTER NETWORK

QUARTERLY TECHNICAL REPORT No. 4

1 October 1975 to 31 December 1975

Principal Investigator: Mr. Frank E. Heart
Telephone (617) 491-1850, Ext. 470

Sponsored by:
Advanced Research Projects Agency
ARPA Order No. 2351, Amendment 15
Program Element Codes 62301E, 62706E, 62708E

Contract No. F08606-75-C-0032
Effective Date: 1 January 1975
Expiration Date: 30 June 1976
Contract Amount: $2,454,098

Title of Work: Operation and Maintenance of the ARPANET

Submitted to:

IMP Program Manager
Range Measurements Lab.
Building 981
Patrick Air Force Base
Cocoa Beach, Florida 32925

The views and conclusions contained in this document are those of the authors and should not be interpreted as necessarily representing the official policies, either expressed or implied, of the Advanced Research Projects Agency or the U.S. Government.

Report No. 3236 Bolt Beranek and Newman Inc.

January 1976

INTERFACE MESSAGE PROCESSORS FOR THE ARPA COMPUTER NETWORK

QUARTERLY TECHNICAL REPORT NO. 4

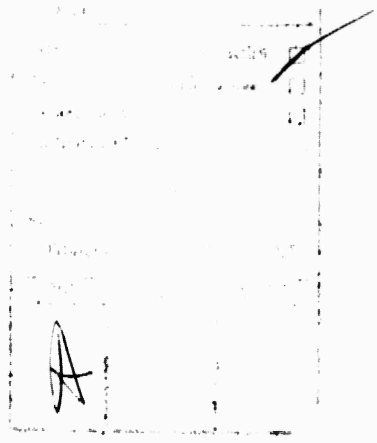
1 October 1975 to 31 December 1975

Submitted to:

IMP Program Manager
Range Measurements Lab
Building 981
Patrick Air Force Base
Cocoa Beach, Florida 32925

This research was supported by the Advanced Research Projects Agency of the Department of Defense and monitored by the Range Measurements Laboratory under Contract No. F08606-75-C-0032.

TABLE OF CONTENTS



1. OVERVIEW

This Quarterly Technical Report, Number 4, describes aspects of our work on the ARPANET under Contract No. F08606-75-C-0032 during the fourth quarter of 1975. (Work performed in 1973 and 1974 under Contract No. F08606-75-C-0027 has been reported in an earlier series of Quarterly Technical Reports, numbered 1-8; and work performed from 1969 through 1972 under Contract No. DAHC-69-C-0179 has been reported in a still earlier series of Quarterly Technical Reports, numbered 1-16.)

During the past quarter, a variety of small tasks occupied a portion of our efforts. For instance, a revision of BBN Report 1822, "Specification of the Interconnection of a Host and an IMP", was produced and distributed. The primary purpose of this revision was to present new IMP/Host message formats which permit addressing of more than sixty-three IMPs and more than four Hosts per IMP.

As the DCA ARPANET Management Branch began to become more directly involved in decisions affecting the network in the fourth quarter, our interactions with them naturally increased and, at the same time, our interactions with ARPA (as regards ARPANET operation and maintenance) decreased. For instance, we attended the ARPANET Sponsor's Group meeting in the role of

expert consultants to the DCA ARPANET Management branch, which organized the meeting; there were numerous other (probably daily) less formal instances of interactions with DCA.

For the past several quarters, the QTR has mentioned the "stressed" topological layout of the network nodes and lines, resulting sometimes in poor network performance. During this past quarter, ARPA's efforts to upgrade the network topology have born fruit and the network topology appears (to us) to be significantly improved.

Last quarter we reported that the work was essentially finished that would permit the Very Distant Host option of the IMP to reside above the 16K IMP memory boundary and thus reduce VDH pressure on IMP buffering. During the quarter very little additional work was done in this area. In particular, no VDH was actually converted to reside above the 16K IMP memory boundary. During this quarter, however, a number of TIPs which also support the IMP VDH option were expended to a total of 32K words of memory (a number of other TIPs also had their memory expanded to a total of 32K words. Therefore, by midway next quarter it is expected that the VDH option will actually be used above the 16K boundary at some of these TIP sites.

with Mr. S. Crocker of USC's Information Sciences Institute) on the subject of software certification of the PLI for the purpose of providing the PLI with a multi-address capability.

During the quarter, the IMP software modification which permits more than sixty-three IMPs and more than four Hosts per IMP progressed nicely. By the end of the quarter, a version of the software containing all the necessary new formats (with respect to inter-IMP operation) was up and running in our test cell. We are hopeful that this version will be released about midway through next quarter. The further revisions of the IMP software to make the necessary changes to the IMP/Host formats (i.e., those described in the December 1822 revision) now look as if they will be the major IMP development task of the next quarter.

During the quarter the first Private Line Interface was shipped to the field. However, due to circumstances at the field site, this PLI is now not scheduled to go into service until late in the next quarter. Early in the next quarter a second PLI is scheduled to be shipped to the field, presumably to go into service at the same time as the first. In the PLI development area, we considered and, we hope, solved a problem dealing with purging of PLI memory when an operational machine is taken off-line for maintenance by non-cleared personnel or when the center where a PLI resides shifts down the security classification level at which it will operate. We also continued to meet with ARPA and DCA (and instituted informal communications

2. TIP SOFTWARE DEVELOPMENT

During the quarter the new TIP software to support the new Telnet protocol was extensively debugged and we now expect initial release of the TIP software containing both the new and old Telnet protocols to occur early next quarter. In addition to the new Telnet protocol, the new TIP software release contains another change, namely, the "Host scheduled down" messages that the TIP prints to TIP users will now be in TIP-local time rather than in GMT. As for the new TELNET option itself, the new release implements the new Telnet Protocol while maintaining the old Telnet Protocol in parallel; the user is allowed to switch a device from one to the other on command. Thus, no TIP functions will be lost although they may not yet be available under the new protocol. The following changes are all a result of this implementation.

(1) The basic new Telnet Protocol is implemented including: a) the new commands @OLD TELNET and @NEW TELNET; b) revision of the @OPEN and @LOG commands to ICP to the appropriate socket for old or new Telnet; c) changes to the output (net to terminal) path to allow the TIP to receive multi-character Telnet commands intact over message boundaries and pending error messages; d) changes to the

input (terminal to net) path to allow the TIP to insert multi-character Telnet commands intact over message boundaries and without interference from terminal input; e) addition of a process (at interrupt level) for interpreting received Telnet commands, taking appropriate action, and queuing them for response; f) addition of a process (at background level) for handling queued items including replying to known and unknown option requests, initiating requests, matching replies to requests sent, and taking option specific actions; g) addition of timeout code to remove inactive queue entries; h) changes in the code that interprets the synch-datamark sequence to handle both old and new protocol.

(2) The Timing Mark option is implemented when requested by another Host, providing a Telnet level synch. The TIP will not initiate the option.

(3) The Echo option is implemented. The Binary and Remote Controlled Transmission and Echoing (RCTE) options are coded but not fully checked out and thus have not yet been enabled. Each of these options required the following changes:

a) revision of user level commands to determine which protocol the device is using and act accordingly; b) revision and addition of status bits for each device to record both which state of an option the device desires and which state the device (or connection) is actully in; c) addition of code to negotiate an option taking into account current status, what the terminal desires, and other states (for instance the RCTE and Echo options are incompatible); d) addition and/or modification of code to perform the option; e) addition of code to set the correct defaults on opening a connection and to automatically initiate desired states different from defaults (for instance, local echo is the default state for a connection, but the TIP will immediately request remote echo for any device that desires it).

(4) The reset and initialization code is modified to reflect the above changes.

3. PACKET SATELLITE DEMONSTRATION AND SATELLITE IMPS

Over the entire last quarter we were very actively participating in the Packet Satellite Program Working Group which is carrying out experimentation on and demonstration of the concept of packet braodcast by satellite using Satellite IMPs and the Atlantic Intelsat satellite. Our efforts in this area have fallen into several categories: 1) continuing development of and modifications to the Satellite IMP program to make it more suitable for the planned experiments and demonstrations; 2) liaison with the other members of the working group; 3) aiding other members of the working group in carrying out experiments; 4) studying certain aspects of the packet-broadcast-by-satellite technology; 5) installing the Satellite IMPs; and 6) maintaining and operating the Satellite IMPs.

The following paragraphs give a few examples (in chronological order) of the kinds of effort we have expended in this area during the past quarter.

In October we finished moving Satellite IMP program assembly from our PDP-1 to our TENEX computer. From October into early November we spent a great deal of time chasing down problems with Goonhilly's SPADE equipment, incompatabilities with new IMP system releases, and some Satellite IMP problems. In the

Satellite IMP program we also implemented generalized I/O code which auto-selects the modem interface which has the satellite on it at each site, does a number of useful error recovery functions, and allows the NCC to cross-patch the on-line or backup modem interface to isolate problems.

In November we made packet error rate measurements. A bit error rate of one in ten to the sixth (10**-6) was found. These results were presented informally at a working group meeting in London in November. We made a large number of comments on the draft plan for the two-year experimental period being prepared. We continued development on the Satellite IMP program.

In December we (together with UCLA) studied potential solutions to the following immediate problems:

- What is a good algorithm for slotted Aloha with respect to blocking and gating?
- What are good fixes (short and long range) which will allow measurements of RFNM-less traffic?

Finally, by the end of the last quarter, a major new version of the Satellite IMP software was ready for release, and this is certain to happen very early in the next quarter. The new version will have the changes described below. (The rest of this section assumes detailed understanding of the Satellite IMP algorithms and software.)

(1) If a packet has a bad length, software checksum, or Satellite IMP header, it is sent to the NCC Host for inspection. This should help in diagnosing hardware and software problems.

(2) If a Satellite IMP reaches 200 retransmissions for a packet, it does a 30-second reset. During these 30 seconds, the Satellite IMP discards satellite input and ceases output. After 16 of these seconds, the IMP declares the line down and lets Satellite IMP garbage-collect run. Satellite IMP garbage-collect resubmits outstanding packets to the IMP's "task" routine. After the 30 seconds, the Satellite IMP attains slot sync and resumes normal operation.

This change lets the IMP declare the Satellite line down much as it declares land lines down, and thus ensures comparable flow control for the satellite line.

This change also lets experimenters recover on their own from channel saturation.

(3) Up to now, there have been 43 Satellite IMP buffers (one was always tied up on the input interface, and one was for routing copyup). From now on, there will be 80 (one will

always be tied up on the input interface, and two will be for routing copyup). This change gives enough buffers to fill all 60 data slots in an ack frame, and still have well above the 10 on the free list required for a copyup. Thus, buffers will be in short supply only if the retransmission queue grows large. The second routing buffer guarantees that the most recent routing information will be sent to a neighbor Satellite IMP.

(4) A TDMA frame can now be defined as two or more slots instead of three or more. The assembled default will be two; a future Satellite IMP version will allow this to be set by a parameter message. Each routing frame now contains two routing slots instead of three.

(5) All Satellite IMP queues, plus the Satellite IMP ack table, have counters; the copy-down queue also has a maximum count. These should help in debugging and simplify some future code.

(6) After attaining slot sync, a Satellite IMP now waits until the beginning of an ack frame before it starts transmitting data packets (at most a wait of 64 slots). This is because, before a new frame starts, the Satellite IMP does

not know which TDMA position belongs to which Satellite IMP.

(7) Etam now sends routing before Goonhilly, to get around SPADE channel problems.

(8) Some old Satellite IMP DDT code was permanently installed, to help with debugging. This code allows up to four counters to be associated with four arbitrary program locations, counting each execution of the locations without disrupting real-time program operation.

(9) When the Satellite IMP code is turned off, outstanding packets are garbage collected. This change improves network reliability.

(10) New code was installed which collects histograms of how many buffers are in use.

(11) The slotted Aloha protocol code has been modified as follows:

a. When one or more packets are on the retransmit queue, new packets are blocked.

b. When a total of two or more packets are on the new packet queues, a second R.N. gate is used to decide

whether to send a new packet (when the retransmit queue is empty). The default for this second gate is currently a probability of 1.0, making it identical to the existing version which was earlier patched to block as in (a).

c. The retransmission gate value has been changed from a probability of 0.1 to 0.5 to correspond to the fact that there are only two Satellite IMPs. The 0.1 value was defined for a large population of Satellite IMPs, whereas theoretical studies have indicated that a value of 1/N is more appropriate for small N. It should be emphasized that this is simply a best guess default value--a future Satellite IMP version will allow this and the second gate value to be set by a parameter message, allowing experimental determination of the optimal values.

4. PLURIBUS TECHNOLOGY

4.1 Recent Developments

Late in the third quarter, we installed the first Pluribus IMP at the Seismic Data Analysis Center (SDAC) in Alexandria, Virginia. The justification for the addition of the Pluribus at this site was that the newly-installed Seismic Data Network, which makes much use of the SDAC site, would need better response and reliability than the existing 316 IMPs could provide.

As of October 1, the Pluribus was supporting three network lines, one of which enabled the NSA site, which had been isolated, to gain access to the network. During October, the various Hosts at SDAC were connected on a trial basis, and by November 1, one of them, a timesharing service Host, was using the Pluribus regularly. During November, we installed the special EIA interfaces required for the line to Norway, and that line was moved to the Pluribus. Subsequently, the Seismic network Command and Control Processor (CCP), which is also a Pluribus, became the second Host to use the Pluribus IMP regularly.

On December 5, the 316 IMP at SDAC was finally removed from the network, and all lines and Hosts at SDAC connected to the Pluribus. The 316 was reconnected once, for one day, when

problems with the Pluribus interface threatened to isolate the European IMPs. From December 12 onward, the Pluribus has run steadily, having been taken down once for five minutes to release new software.

The Pluribus IMP software, which was basically operational in September, has become quite dependable under the pressure of day-to-day use. One major change during the fourth quarter reorganized the buffer accounting system to prevent buffer lockups, to make better use of the large numbers of buffers the Pluribus can support, and to facilitate extensions that require use of the IMP buffers, such as VDH or TIP code. The SDAC configuration, five modems and four Hosts, is the maximum that the current NCC program conventions and network addressing will allow, and more than a single 316 IMP can support. The Pluribus, with 48K of shared memory, has 140 IMP buffers, and space for reassembly of 13 messages at once.

Meanwhile, we have been using the remaining 13-processor prototype in the BBN TENEX facility on a test basis. The Pluribus there has supported a variety of Hosts at different times, including TENEX and several PDP-11's. We have also proved, using that machine, the feasibility of dividing the resources of the Pluribus logically, so that a diagnostic program may be run

using some of the hardware, while the balance of the machine continues to run the operational system. We hope to automate this procedure in the future so that machine repair and maintenance need never prevent the IMP program from running.

Besides the effort concerned with the operation of Pluribus IMPs in the ARPANET, we did a little residual Pluribus development work this past quarter, tying up loose ends and completing work that had been started in previous quarters. In particular, the Pluribus documentation effort has been much more difficult than we expected. The material has been largely written for some time, but the process of pulling it all together into a coherent package has been very slow. As the quarter drew to a close, two more Pluribus documents were on the verge of publication, and another two are getting quite near being ready for publication. Unfortunately, two more are not particularly near being ready for publication and we are considering not finishing these rather than having an indefinitely long tail on the Pluribus development effort.

As the ARPA-sponsored Pluribus development effort winds down, it seems appropriate to consider in more general terms what has been significant about the Pluribus development.

4.2 The Major Accomplishment

We have built a resilient computer which keeps running even when we actively inflict trouble by pulling cables or breaking components. It is an interesting computer for three reasons. First, it is a true multiprocessor, capable of being configured with large numbers of processors and common memories, all processors sharing access to the common memories and all processors participating equally in the performance of system functions. Second, the machine is not just an experimental prototype; more than half a dozen have been built and others are currently in production. Third, the reliability functions have been implemented in the software with virtually no special hardware beyond that necessary to make a multiprocessor. The idea of using software for the reliability functions may at first seem strange, since software is traditionally considered the most fragile part of a computer system, but we will attempt to describe below why we have come to believe that the software can be made solid enough to be entrusted with this critical job.

As it becomes harder to build faster and faster computers, and easier to make cheap, small, slow computers, the notion of multiprocessing becomes more and more attractive. We envision a future in which computer systems are routinely built from small,

simple processors, the performance of the system being determined by the number of processors chosen. All other resources (memories, I/O, power, interconnections, etc.) would also be modular. The Pluribus system we have designed is such a "multi-resource" system.

As multiprocessors become a reality, the designer of fault tolerant systems will find himself working with machines wnich already have redundant hardware, and the emphasis will switch to efficient use of that existing redundancy. For example, consider a task which can be performed equally well by a single powerful machine or a 10-processor multiprocessor. Either implementation can be made to survive a single failure by the addition of one more processor, but this doubles the cost in the first case and adds only 10% in the second. Thus multiprocessors seem attractive not only to those seeking cost-effective computing power, but also to designers of fault tolerant systems.

4.3 Advantages of the Software Approach

Assuming that it is possible to implement the reliability functions in the software of a multiprocessor, there are substantial advantages beyond cost savings. For example, one can implement more complex functions than would be practical in hardware, and one can reprogram to modify or augment the

reliability algorithms without changing the hardware. The price one pays for the generality of a software approach is, as usual, a penalty in speed. The system will be slow (fractions of a second) at detecting malfunctions and slower still at isolating failed hardware and returning to normal operation.

It is in the area of fault detection that the flexibility of a software approach is so attractive. At the very least one can mimic more traditional techniques; for example, one can run a program segment on several machines and vote on the answer, one can compare the outputs of two machines until they disagree, or one can spread the checking out in time by repeating a computation. The real gain, however, lies in the ability to introduce high-level checking which would be prohibitively expensive in hardware. Such checking may be tailored to suit the particular algorithm being checked. As a trivial example of this technique one might check a matrix inversion by verifying that the product of the original matrix and its inverse equals the identity. This test is far less expensive than a redo of the original computation, and is equally effective if performed on a different machine. In fact, high level consistency checks offer substantial additional benefits in that they not only catch hardware failures but also tend to catch software bugs and hardware design errors. Our experience indicates that such

problems constitute a significant source of system failures, despite years of system operation and debugging.

We would like to single out a particular approach to fault detection, which is the one we implemented, because we see it as satisfying the requirements of a large class of applications. The goal of this approach is guaranteed system availability, not perfect operation. Our particular application is that of a communications processor in a network and, much as in the telephone system, occasional disruptions of single communications are acceptable (so long as they happen infrequently), but interruption of service for any significant time is catastrophic. Relating this goal to a fault detection strategy, we built into the software a comprehensive set of consistency checks on system control functions and key data structures, including a periodic verification that we can indeed send and receive messages. With this approach, it is unnecessary to verify independently every detailed computation, or even to check the control structure more often than once or twice a second. A pleasant result of this more relaxed fault detection is that only 1% of our processor capability is explicitly spent in error detection. In order to simplify checking we have also sacrificed some efficiency in the code, which makes the total effective cost be more like 5%.

4.4 Insuring Software Operation

To do fault correction in the software, the following three things must be provided:

1) an intact set of hardware: memory, processor, vital I/O, etc.;
2) a useful state for that hardware: memory correctly loaded with code and processor running the code;
3) freedom from interference by failed components.

The working hardware occurs quite naturally on a multi-resource type of machine. One simply builds a system large enough so that it contains at least two instances of every component. Of course one must take care that the copies of a resource are isolated from one another and not dependent upon some other common resource. This means that elements such as power supplies, cooling modules, etc., must all be replicated.

It is easy to guarantee that the hardware is in a useful state. One way to do this is to provide an unstoppable periodic interrupt to some reliability code held in ROM. To avoid the inflexibility of ROM, however, we chose a slightly more complex approach. We provide a mechanism which periodically attempts to reload and restart the processors. This action is normally held

off by regular indications from each processor that it has successfully performed a set of internal consistency checks (including, for example, checksum of the code itself). This is a specific instance of a philosophy which pervades our design: we are willing to trust the fate of the system to any process which can repeatedly generate such an indication. We believe that the indication process can be made sufficiently complex that the probability of spurious generation is reduced to that of multiple hardware failures (we are not concerned with "malicious" processes). For example, one can intersperse pieces of the indication process with checking so that simple control failures cannot mimic correct behavior. Nevertheless, some logic must ultimately make a decision whether to reload or not, and this logic is potentially vulnerable. We will explain our solution to this below, following the discussion of isolation.

The third function necessary for recovery is isolation of the system from actively failing components, i.e., we must provide for those cases in which continued presence in the system of the failed part interferes with proper operation. We achieve this isolation by providing a disabling (amputation) mechanism in all of the cables which join the major units of the machine together. This mechanism is activated under program control by use of a password. If the system can tell which unit has failed

it can thus amputate that unit from the system. If the system is unable to single out an individual unit, there is a straightforward, if tedious, way to isolate the failure: amputate components one after another until the symptom goes away. (This is equivalent to experimental card replacement, performed by technicians in the course of debugging problems.)

These reliability algorithms rely upon having some agent to perform them. This agent must be unaffected by the fault. It may be either one of the processors in the system or something external. By carefully isolating the processors from one another, we make it extremely unlikely that a single failure can attack all of them at once. We increase the isolation between them by providing each with a small private memory. By operating out of its private memory, a processor can retreat into a relatively invulnerable position, thus enhancing its chances of survival, both as an individual and as a participant for joint decisions. The net result of this mechanism is that the "fragile" code and machine state will survive virtually any failure, without any external assistance.

An external reset/reload mechanism should thus be required rarely. The nature of such a mechanism will depend on the application. In the case of the network, we are able to provide

a multiple-source mechanism, operating from any of several other network nodes. Other applications might typically use some form of ROM, diskette, etc. This mechanism, in whatever form, is responsible for placing the key checking and bootstrap algorithms into all private memories and starting each processor operating in an initial self-protective manner. The cables that join the processors into the remainder of the system are initialized to a state in which local signals can get out but external signals cannot get in. This insures that upon startup each processor is insulated from active failures elsewhere in the system. From this state, the processors will test the environment, begin operating the application programs, and join the other processors for shared decisions.

4.5 Transient Behavior

An interesting sidelight to this approach to reliability is that we expect such a machine to be significantly more reliable than a corresponding conventional machine even when the multi-processor is configured in its minimal form--one processor and no other redundant resources. This expectation is based on our observation that a large fraction of system failures do not require replacement of a failed component; that is, they fall into the categories of hardware transients and low probability

software bugs. Since we have hardened up the "fragile" code and control, our minimal machine should survive most such failures.

Unclassified

SECURITY CLASSIFICATION OF THIS PAGE (When Data Entered)

REPORT DOCUMENTATION PAGE	READ INSTRUCTIONS BEFORE COMPLETING FORM
1. REPORT NUMBER: Quarterly technical rept. no. 4, 1 Oct-31 Dec 75	2. GOVT ACCESSION NO. / 3. RECIPIENT'S CATALOG NUMBER
4. TITLE (and Subtitle): Interface Message Processors for the ARPA Network - QTR No.4	5. TYPE OF REPORT & PERIOD COVERED: Quarterly Technical 1/10/75 - 31/12/75
	6. PERFORMING ORG. REPORT NUMBER: BBN-3236
7. AUTHOR(s): F E Heart	8. CONTRACT OR GRANT NUMBER(s): F08606-75-C-0032
9. PERFORMING ORGANIZATION NAME AND ADDRESS: Bolt Beranek and Newman Inc. 50 Moulton Street Cambridge, MA 02138	10. PROGRAM ELEMENT, PROJECT, TASK AREA & WORK UNIT NUMBERS: ARPA Order No-2351 Program Element Codes 62301E,62706E,62708E
11. CONTROLLING OFFICE NAME AND ADDRESS: Advanced Research Projects Agency 1400 Wilson Boulevard Arlington, VA 22209	12. REPORT DATE: January 1976 / 13. NUMBER OF PAGES: 27 (30 p.)
14. MONITORING AGENCY NAME & ADDRESS (if different from Controlling Office): Range Measurements Laboratory Building 981 Patrick Air Force Base Florida 32925	15. SECURITY CLASS. (of this report): Unclassified / 15a. DECLASSIFICATION/DOWNGRADING SCHEDULE
16. DISTRIBUTION STATEMENT (of this Report): Distribution Unlimited	
17. DISTRIBUTION STATEMENT (of the abstract entered in Block 20, if different from Report)	
18. SUPPLEMENTARY NOTES	
19. KEY WORDS (Continue on reverse side if necessary and identify by block number): Computers and Communication, Store and Forward Communication, ARPA Computer Network, Packets, Packet-switching, Interface Message Processor, IMP, Terminal IMP, TIP, Pluribus Satellite IMP, Private Line Interface, PLI, Broadcast Communications	

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

The ARPA Computer Network is a packet-switching store-and-forward communications system designed for use by computers and computer terminals. This Quarterly Technical Report briefly describes various aspects of network operation and maintenance, including IMP software modifications to permit more than 63 IMPs on the net and more than 4 Host computers on an IMP, and shipment of the first Private Line Interface to the field; and discusses in some detail the new TIP software to be released shortly, the Packet Satellite Demonstration and Satellite IMP activities, and recent

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

Unclassified

SECURITY CLASSIFICATION OF THIS PAGE (When Data Entered)

060100

20. Abstract

developments in Pluribus technology, plus a summary of accomplishments to date in the latter area.