A UNITED STATES
DEPARTMENT OF
COMMERCE
PUBLICATION

NBS TECHNICAL NOTE 800

Computer Networking: Approaches to Quality Service Assurance

U.S.
PARTMENT
OF
OMMERCE
National
Bureau
of
Standards

NATIONAL BUREAU OF STANDARDS

The National Bureau of Standards [1] was established by an act of Congress March 3, 1901. The Bureau's overall goal is to strengthen and advance the Nation's science and technology and facilitate their effective application for public benefit. To this end, the Bureau conducts research and provides: (1) a basis for the Nation's physical measurement system, (2) scientific and technological services for industry and government, (3) a technical basis for equity in trade, and (4) technical services to promote public safety. The Bureau consists of the Institute for Basic Standards, the Institute for Materials Research, the Institute for Applied Technology, the Institute for Computer Sciences and Technology, and the Office for Information Programs.

THE INSTITUTE FOR BASIC STANDARDS provides the central basis within the United States of a complete and consistent system of physical measurement; coordinates that system with measurement systems of other nations; and furnishes essential services leading to accurate and uniform physical measurements throughout the Nation's scientific community, industry, and commerce. The Institute consists of a Center for Radiation Research, an Office of Measurement Services and the following divisions:

Applied Mathematics — Electricity — Mechanics — Heat — Optical Physics — Nuclear Sciences [2] — Applied Radiation [2] — Quantum Electronics [3] — Electromagnetics [3] — Time and Frequency [3] — Laboratory Astrophysics [3] — Cryogenics [3].

THE INSTITUTE FOR MATERIALS RESEARCH conducts materials research leading to improved methods of measurement, standards, and data on the properties of well-characterized materials needed by industry, commerce, educational institutions, and Government; provides advisory and research services to other Government agencies; and develops, produces, and distributes standard reference materials. The Institute consists of the Office of Standard Reference Materials and the following divisions:

Analytical Chemistry — Polymers — Metallurgy — Inorganic Materials — Reactor Radiation — Physical Chemistry.

THE INSTITUTE FOR APPLIED TECHNOLOGY provides technical services to promote the use of available technology and to facilitate technological innovation in industry and Government; cooperates with public and private organizations leading to the development of technological standards (including mandatory safety standards), codes and methods of test; and provides technical advice and services to Government agencies upon request. The Institute consists of a Center for Building Technology and the following divisions and offices:

Engineering and Product Standards — Weights and Measures — Invention and Innovation — Product Evaluation Technology — Electronic Technology — Technical Analysis — Measurement Engineering — Structures, Materials, and Life Safety [4] — Building Environment [4] — Technical Evaluation and Application [4] — Fire Technology.

THE INSTITUTE FOR COMPUTER SCIENCES AND TECHNOLOGY conducts research and provides technical services designed to aid Government agencies in improving cost effectiveness in the conduct of their programs through the selection, acquisition, and effective utilization of automatic data processing equipment; and serves as the principal focus within the executive branch for the development of Federal standards for automatic data processing equipment, techniques, and computer languages. The Institute consists of the following divisions:

Computer Services — Systems and Software — Computer Systems Engineering — Information Technology.

THE OFFICE FOR INFORMATION PROGRAMS promotes optimum dissemination and accessibility of scientific information generated within NBS and other agencies of the Federal Government; promotes the development of the National Standard Reference Data System and a system of information analysis centers dealing with the broader aspects of the National Measurement System; provides appropriate services to ensure that the NBS staff has optimum accessibility to the scientific information of the world. The Office consists of the following organizational units:

Office of Standard Reference Data — Office of Information Activities — Office of Technical Publications — Library — Office of International Relations.

---

[1] Headquarters and Laboratories at Gaithersburg, Maryland, unless otherwise noted; mailing address Washington, D.C. 20234.
[2] Part of the Center for Radiation Research.
[3] Located at Boulder, Colorado 80302.
[4] Part of the Center for Building Technology.

Computer Networking: Approaches to Quality Service Assurance

Rona B. Stillman

Institute for Computer Sciences and Technology
Systems Development Division
National Bureau of Standards
Washington, D.C. 20234

Sponsored by

The National Science Foundation
18th and G Street, N W.
Washington, D.C. 20550

U.S. DEPARTMENT OF COMMERCE, Frederick B. Dent, *Secretary*
NATIONAL BUREAU OF STANDARDS, Richard W. Roberts, *Director*

Issued January 1974

National Bureau of Standards Technical Note 800

Nat. Bur. Stand. (U.S.), Tech. Note 800, 26 pages (Jan. 1974)

CODEN: NBTNAE

U.S. GOVERNMENT PRINTING OFFICE
WASHINGTON: 1974

For sale by the Superintendent of Documents, U.S. Government Printing Office, Washington, D.C. 20402
(Order by SD Catalog No. C13.46:811). Price 60 cents.

TABLE OF CONTENTS

FOREWORD

This report is one of a series of publications produced by the Institute for Computer Sciences and Technology, National Bureau of Standards, under Grant AG-350 from the National Science Foundation.

This grant supports a broad program of research into the foundations of computer networking in support of scientific efforts.

A listing of completed and planned publications produced by the Institute under this grant follows:

1. Primary Issues in User Needs
   D. W. Fife
   Chapter 10 in Networks for Research and Education: Sharing of Computer and Information Resources Nationwide
   MIT Press, Cambridge, Mass.
   Expected Publication October 1973

2. Some Technical Considerations for Improved Service to Computer Users
   T. N. Pyke, Jr.
   COMPCON, 1973
   Seventh Annual IEEE Computer Society International Conference

3. Computer Networking Technology - A State-of-the-Art Review
   R. P. Blanc and T. N. Pyke, Jr.
   COMPUTER Magazine
   Computer Society of the IEEE

4. Review of Network Management Problems and Issues
   A. J. Neumann
   NBS Technical Note 795
   October 1973

5. Annotated Bibliography of the Literature on Resource Sharing Computer Networks
   R. P. Blanc, I. W. Cotton, T. N. Pyke, Jr., and S. W. Watkins
   NBS Special Publication 384
   September 1973

6. Network Management Survey
   I. W. Cotton
   NBS Technical Note
   Fall 1973

7. User Procedures Standardization for Network Access
   A. J. Neumann
   NBS Technical Note 799
   October 1973

8. Review of Computer Networking Technology
   R. P. Blanc
   NBS Technical Note
   Fall 1973

9. Microeconomics and the Market for Computer Services
   I. W. Cotton
   Submitted to Computing Surveys

10. Cost Analyses for Computer Communications
    R. P. Blanc
    NBS Technical Note
    Fall 1973

11. Network User Information Support
    A. J. Neumann
    NBS Technical Note
    Fall 1973

12. Computer Networking: Approaches to Quality Service Assurance
    R. B. Stillman
    NBS Technical Note
    Fall 1973

13. A Guide to Networking Terminology
    A. J. Neumann
    NBS Technical Note
    Fall 1973

14. Research Considerations in Computer Networking
    D. W. Fife
    NBS Technical Note
    Fall 1973

COMPUTER NETWORKING: APPROACHES TO QUALITY SERVICE ASSURANCE

Rona B. Stillman

The problem of quality service assurance in a (generalized) computer networking environment is addressed. In the absence of any direct, well-defined, quantitative measure of service quality and reliability, error collection and analysis is the only basis for service quality control. Therefore, mechanisms are described which facilitate reporting of operational errors, documentation of error corrections, and collection of system performance data. Since techniques for hardware quality control are well known, these mechanisms focus on collecting data which can be used to assess and control software quality. Finally, specific network facilities are described which support research in the area of software quality, and potential areas of new research using the network are identified.

Key words: Compiler, computer network, documentation, dynamic software analysis, interpreter, quality control, software testing, software verification, static software analysis, structured programming, system errors, system performance, theorem-proving.

1. INTRODUCTION

Although the goal of reliable, fail-soft network service at reasonable cost suggests certain concepts in network design (e.g., acentric rather than star network, process rather than processor orientation, etc.), this report will, as far as is possible, be independent of the details of any particular network philosophy or topology. We do assume, however, that all resource providers concur in the belief that user satisfaction is the primary goal of the network, and will, therefore:

(1) require that programmers document the results of their work.

(2) assign responsibility for assuring the quality of user service to a designated group of experts. In particular, a responsible individual must be identified for each software module.

(3) exhibit complete honesty in acknowledging failures and tracking down their causes (although information obtained from users and independently collected by the network will provide a check on this).

(4) abide by network rules and conventions, which are designed to minimize the effects of system failures on users and to encourage stable, reliable service.

A primary factor in the success of any network is user satisfaction. Two important causes of user dissatisfaction are: isolation, i.e., lack of access to consultation and application expertise, no established channel through which to report system errors to the responsible engineers, and unavailability of data on network performance in general, and individual subsystem performance in particular; and poor quality service, e.g., chaotic service, frequent system crashes, unstable data and programs, inaccurate and/or inadequate documentation, bug-laden and perfunctorily maintained system software. At the same time, resource providers find it difficult to correct and maintain their systems without sufficient feedback information from users.

To alleviate these problems in networking, we suggest establishing "network central," a technical-administrative office of the network. (Note that, in fact, network central need not be a single organization in the network operations management structure. For reasons of convenience, and because we are concerned with the functions rather than the implementation of network central, we refer to it as a single entity. Further information on network management structure and implementation can be found in [12].) Network central will serve as an information center to users, providing guidance and consultation on specific problems. It will serve as the point of contact between the user and the network. In particular, network central will channel complaints from users to the appropriate host nodes, and will convey reports of system modifications from host nodes to users. It will also be responsible for maintaining records on system performance and service quality, and will establish and enforce network quality control procedures. Within this context, we will describe generalized mechanisms for:

(1) constructing performance profiles for individual subsystems in the network, and for the network as a whole;
(2) defining and implementing quality control procedures for network systems;
(3) using the unique environment provided by the network to experiment with and evaluate new techniques for increasing software reliability. Specific network facilities to support software research will be described, and potential areas of research using the network will be identified.

2. SYSTEM PERFORMANCE MEASUREMENT

In order to assist users and network managers in analyzing the performance of a distributed network or of individual network subsystems over time, and to permit the performance of different subsystems to be compared, network central must maintain a file of system performance profiles. The file would consist of periodic (e.g., weekly, monthly, quarterly) performance profiles for each subsystem, provided by the host node, as well as performance profiles for the network as a whole provided by network central (see Figure 1).

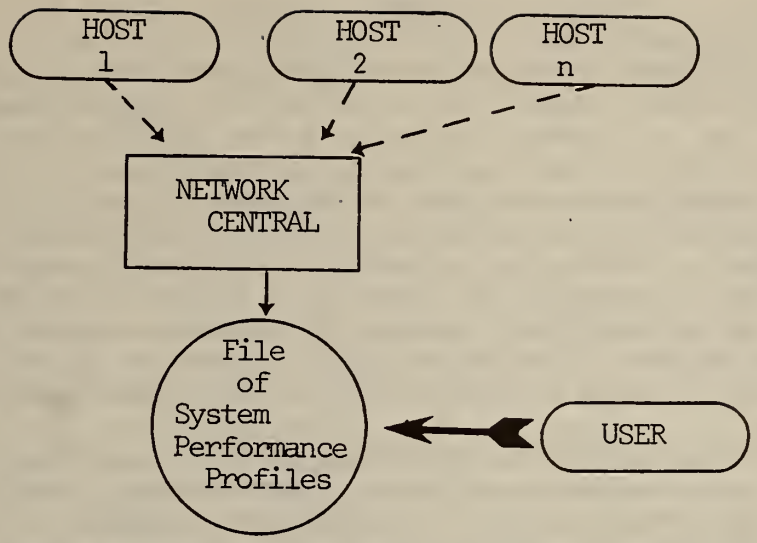

Figure 1: Maintenance of File of System Performance Profiles

Legend: → Read/Write
⇒ *Read only*
- - → Communications Path (e.g., written records, on-line files, etc.)

A System Performance Profile should include:

SYSTEM IDENTIFICATION
REPORTING PERIOD
TIME SCHEDULED
TOTAL DOWN TIME
PERCENT OF TIME DOWN
INTERRUPTIONS TO SERVICE (NUMBER)
MEAN TIME BETWEEN FAILURES (MTBF)
MEAN TIME TO REPAIR (MTTR)
ARRAY DESCRIBING THE INTERRUPTIONS TO SERVICE:

TYPE	NUMBER	TIME LOST	MTBF	MTTR
HARDWARE				
SOFTWARE				
COMMUNICATIONS				
ENVIRONMENT				
HUMAN				
UNCLASSIFIED				
UNKNOWN				

MISCELLANEOUS PROBLEMS

An interruption to service is any type of system failure which aborts execution of programs being run by a majority of the system's current users, or which causes a suspension of system operation for more than X (e.g., five) minutes regardless of whether any jobs are aborted. For example, loss of electrical power at a host node is clearly an interruption to that system's service. Alternatively, any catastrophic error which necessitates a complete system dump followed by a restart constitutes an interruption to service. Moreover, if a disk pack is being moved from a faulty disk unit onto a replacement, and if all system activity is suspended for more than X minutes while the exchange takes place, an interruption to service is recorded. In any event, all problems, whether they cause interruptions to service or not, are recorded (as miscellaneous problems) in the system's performance records.

Interruptions to service are further identified, whenever possible, as to cause, e.g., hardware, software, communications. Environmental failures include power failures, air conditioning equipment failures, etc. Human failures are usually procedural errors made during operation, e.g., the operator pushing the wrong button at a critical moment. Unclassified failures are those for which the immediate symptom is known (e.g., every other word in memory dropped bit 15), but the underlying cause (hardware or software) is not. Unknown failures are those which, despite considerable analysis, defy explanation. The system performance profile is essentially identical to summaries which have been used successfully on the Dartmouth Time Sharing System [11] and on Bell's No. 1 Electronic Switching System [1].

Clearly, the host node is responsible for the accuracy, completeness, and timeliness of his system's profiles. As in accounting, it becomes very difficult to compare results between one time period and the next, or between one subsystem and another, if the recording rules vary. Therefore, it is essential that the recording rules be strictly and uniformly obeyed throughout the network.

The resource supplier, however, should not be the sole source of information on his system's performance. Independent sources can provide complementary and, to some extent, redundant information, and thereby serve to "keep the profiles honest." One such source is the network itself. By polling the activity of the subsystems at regular intervals, the network can derive its own gross profile of subsystem performance (e.g., up time vs. down time). A more important source of information is the formal mechanism provided by the network (and described in the next section) through which users can report the details of any operational difficulties they encounter.

3. QUALITY CONTROL PROCEDURES

3.1 Trouble Reporting Procedures: The Operational Errors File and the Corrections File

The ultimate source of information about the quality of network service is the user. Therefore, it is important to establish a mechanism which assures that system problems discovered by users are properly documented and routed to the responsible engineers. We suggest that this mechanism consist of a set of trouble reporting procedures and two files maintained by network central called the Operational Errors File and the Corrections File. When users experience operating difficulties with the network, they describe the problem (over the phone) to a group at network central that is responsible for user support. If the problem cannot be identified as procedural, or cannot otherwise be determined to be user caused, and if the problem is not a duplicate of one that has been reported previously, an Operational Error Report is generated by network central and entered into the Operational Errors File. Every Operational Error Report is assigned a unique identification number, and includes:

ERROR IDENTIFICATION NUMBER
DATE OF ERROR REPORT
COMPLAINANT IDENTIFICATION
HOST NODE IDENTIFICATION
PARTICULAR SERVICES INVOLVED
PROBLEM DESCRIPTION:
    DATE, TIME OF DIFFICULTY
    WHAT WAS DONE
    RESULTS EXPECTED
    RESULTS OBTAINED
    COPY OF PROGRAM, DATA (when appropriate)
LIST OF SUBSEQUENT COMPLAINANTS, DATES

Network central then alerts the appropriate specialists at the host node (a list of services provided and engineers responsible for them is maintained at network central) to the relevant Operational Error Report. When a solution has been found, the host specialists generate a Corrections Report, which includes:

CORRECTION IDENTIFICATION NUMBER
DATE OF CORRECTION REPORT
HOST NODE IDENTIFICATION
OPERATIONAL ERROR(S) ADDRESSED (i.e., list of Error ID's, if any)
SOURCE OF ERROR (Hardware, Software, etc.)
ERROR DESCRIPTION (e.g., module in which Software error occurred, statements involved, etc.)
DATE OF CORRECTION
CORRECTION DESCRIPTION
DESCRIPTION OF REGRESSION TESTS PERFORMED (type and extent)
DOCUMENTATION CHANGES (if appropriate).

The correction description is a detailed account of what was done, where, and why it was done, e.g., in the case of a software correction, which statements in which modules were added, deleted, or altered, and an explanation of why this method of correction was deemed appropriate. All efforts to verify that the correction has not introduced new errors are detailed in the description of regression tests performed, e.g., satisfactory execution of a standard set of tests, use of software testing tools to construct new tests to exercise the software, etc. Network central enters the Correction Report into the Corrections File, and maintains tables cross referencing the Operational Errors File and the Corrections File. Data on user-caused problems (e.g., misinterpreting the documentation, not having documentation) may be recorded separately by network central, for use in later analysis on the effectiveness of training sessions, the quality and availability of documentation, etc.

This system of reporting complaints and repairs is particularly appropriate in a national network environment, where the user would otherwise have no direct contact with system designers and engineers. Moreover, the files reveal not only how well the network and its component subsystems are behaving, but also how well they are being maintained (see Figure 2).

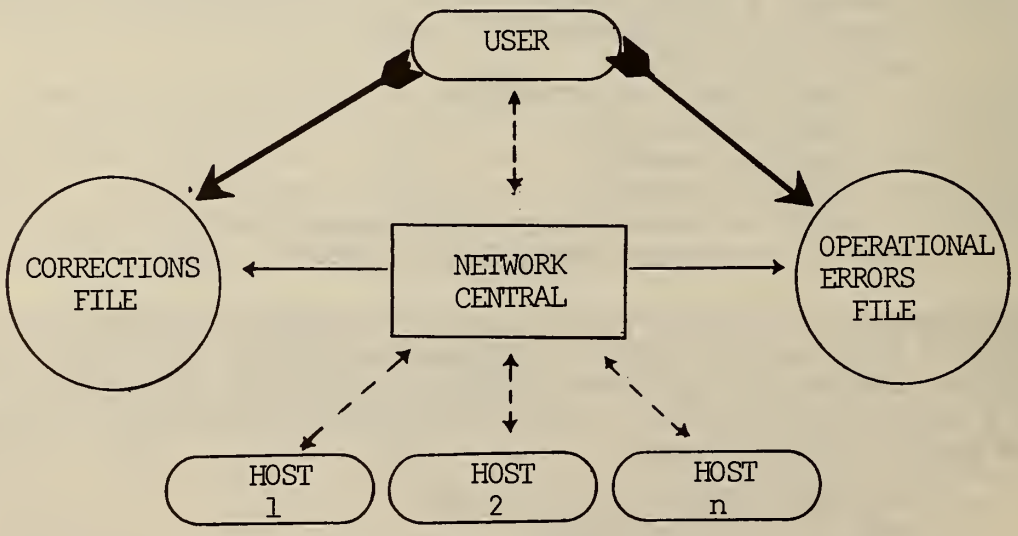

Figure 2: Maintenance of Operational Errors and Corrections Files

Legend:
- ——→ Read/Write
- ▬——→ *Read only*
- - - - - → Communications Path (e.g., phone, written records, on-line files, etc.).

3.2 System Change Procedures

In order to try out modifications to a software module without undue risk and to test corrections to existing problems, both official and experimental versions of the module should be offered on the network. Users of the experimental versions would be warned that they do so at their own risk, and would, therefore, be expected to protect their data and programs before proceeding. Modules that cannot be debugged as experimental versions - either because they require complete control over the hardware or because of some other characteristic that militates against more than one such module operating on the network at any time - and changes in hardware can be checked out during regularly scheduled experimental periods, during which service to the user community is not guaranteed (Sundays, holidays, and third shift hours are prime candidates for experimental periods).

System Performance Profiles, Operational Error Reports, and Correction Reports should be generated during all experimental periods, and no change should be adopted until it has been shown to operate successfully under experimental conditions. Furthermore, changes to the documentation should occur concurrently with any system changes.

4. EXPERIMENTAL RESEARCH ON THE NETWORK

The purpose of a national computer-based network is to promote the sharing of resources of all types: hardware facilities, software facilities, data bases, and human experience. As such, it fosters a climate that is conducive to research in general, and to research in the area of software development, testing, and validation in particular. The benefits of providing new software services on the network as opposed to doing so at an isolated installation include:

(1) rapid exposure of the service to a large group of users who can be expected to exercise it thoroughly. Because of the diversity of their interests, experiences, and biases, their reactions (e.g., the Operational Errors File, special interest group communications) should provide more reliable data more quickly than is obtainable otherwise.

(2) sharing the cost of developing new services among many users, which permits a wide variety of services to be offered, and which encourages innovation.

(3) discouraging the "not invented here" syndrome, by involving a larger portion of the data processing community earlier in the development of software projects.

(4) encouraging and facilitating meaningful communications within the research community.

4.1 Specific Network Facilities to Support Software Research

Many of the tools which facilitate effective software production already exist, but not in a common environment where they can be effectively utilized by a broad spectrum of programmers. A distributed network could provide the framework for a software production laboratory, which would include a software library, compatible interpreters and compilers offering sophisticated debugging and optimization features, program execution monitors and test data generators, verification condition generators for a variety of languages, and theorem-proving programs. The software production laboratory would serve, therefore, both to facilitate further software research and as an environment in which users could produce better programs more efficiently.

4.1.1 The Software Library

A software library is a collection of programs and data which are of general interest and utility. The library, of course, need not reside at a single installation, but may be distributed over the network. What distinguishes the software library from other programs shared over the network is that their reliability and conformance with explicitly defined standards is, in some real sense, guaranteed by the network. That is, network central will establish stringent requirements for entering a program into the library (e.g., that it has run under experimental conditions for a given amount of time with an acceptably low frequency of Operational Errors) and for maintaining it once it belongs to the library (e.g., a daily resolution of the complaints in the Operational Errors File). By taking programs from the software library, network users can avoid duplicating each other's efforts in writing and debugging commonly needed routines.

The following types of programs are prime candidates for inclusion in a network software library:

(1) Mathematical Function Routines: These are routines which compute the trigonometric functions and other commonly used functions (e.g., Bessel functions, Ackermann's function, etc.) with some prescribed degree of accuracy, and which perform customary mathematical operations (e.g., linear regression analysis and the like). Because these programs have been thoroughly tested and are vigilantly maintained, they are useful also as a standard against which the user may compare his own work.

(2) Functional Test Routines for Compilers (of Widely Used Languages): These are sets of programs which determine whether or not a subject compiler provides specific capabilities, and, in particular, whether a certain "standard subset" of the language is compiled in an acceptable way.

Well structured and maintained functional test routines can constitute the basis for de-facto language standards, and as such are especially important and interesting. Functional test routines currently exist for COBOL and FORTRAN compilers, the former being a part of the Government's definition of standard COBOL. It would be both appropriate and convenient to provide these functional test sets over a distributed network.

It is unsettling, however, that the vitally important problem of establishing (minimum) standards for compiler diagnostics has been hitherto ignored. Since the efficacy of a compiler is directly proportional to the quality of its diagnostics, i.e., to the amount of information the compiler supplies concerning the nature and location of unacceptable code, it would be worthwhile to develop, in addition to the set of functional test routines, a set of standards for compiler diagnostics. Within this context, the diagnostic standards could take the form of a set of programs with specific errors in them. To meet all standards, then, a compiler would have to process both the functional test routines and the deliberately incorrect programs in an acceptable manner.

4.1.2 Fully Compatible Interpreters and Compilers

Much of the programming activity on a distributed network will be done in an interactive conversational mode. It is important, therefore, to provide tools which support interactive program production, debugging, modification, and testing. In particular, it is convenient to compose a program at a terminal using an interpreter which can field breaks or errors within a computation, evaluate arbitrary expressions during breaks or at the top level, provide a trace of the values of specified variables from the breakpoint back through the computation, and permit the programmer to modify or cancel the effects of the current command, thus recovering an earlier state. When the code has been debugged, however, it may be desirable to compile it, perhaps using an optimizing compiler (e.g., if it is a production program which will be executed frequently, or if, as in a theorem-proving program, even a single execution is expected to be very time consuming). By offering fully compatible interpreters and compilers, then, the network can provide its users with a rich and flexible environment for programming.

4.1.3 Automated Software Testing Tools

The tasks of debugging, modifying, testing, documenting, and, in general, understanding the logical structure of a program are greatly facilitated by the use of software testing tools. There are two main categories of analysis: static analysis, which is performed without

executing the software, and dynamic analysis, which is dependent upon information collected while the software is in execution.

4.1.3.1 Static Analyzers: These software tools accept a subject program as input and produce the following type of information as output:

(1) a display of the program structure and logic flow;

(2) a description of the global data, i.e., the data which is shared among the subroutines;

(3) a subroutine/global variables referenced listing;

(4) a global variable/subroutines where referenced listing;

(5) a subroutine/subroutines referenced listing;

(6) a subroutine/subroutines where referenced listing;

(7) an entry point/subroutine listing;

(8) a subroutine/entry points listing;

(9) a description of the disconnected portions of code, i.e., code which cannot be reached from the 'start' state;

(10) a description of the blocked portions of code, i.e., code from which an 'exit' state cannot be reached.

Other tools have been suggested which analyze the possible execution paths of a program, and output a (hopefully minimal) subset of paths which exercise every statement and/or branch option in the program ([3], [6]). These potential path analyzers can also identify execution paths which include a particular instruction or sequence of instructions. This information is extremely valuable to a programmer in constructing a set of test cases that will thoroughly exercise his code. The major challenge in developing potential path analyzers is finding some appropriate way to deal with the enormous number of possible execution paths of even relatively simple programs. Perhaps modularly designed structured programs offer some promise in this regard: if each module is analyzed independent of the others, and then the flow from module to module is considered, the combinatorial problem will be eased.

4.1.3.2 Dynamic Analyzers: There are software tools which, by inserting traps in the subject program, cause the following types of information to be produced in addition to the program's normal output:

(1) the number of times each statement in the program has been executed in a single run or series of runs;

(2) the number of times each transfer in the program has been executed in a single run or series of runs;

(3) the number of times each subroutine in the program has been entered during a single run or series of runs;

(4) the amount of time spent in each subroutine during a single run or series of runs;

(5) for each statement assigning a new value to a specified variable, the maximum, minimum, first, and last value assigned during the computation.

The operation of a dynamic analyzer is shown schematically in Figure 3.

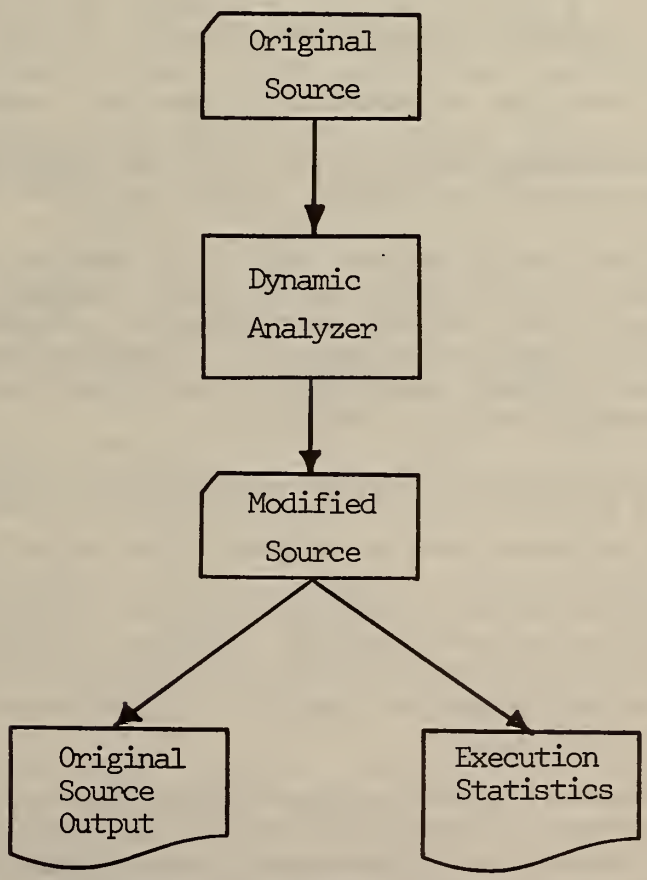

Figure 3: Dynamic Analyzer

Because the programmer can now accurately analyze the effectiveness of his test cases, i.e., he knows how many times each statement (transfer, subroutine) has been exercised, and, in particular, he knows which statements (transfers, subroutines) have never been exercised, he can construct a set of test cases that is both thorough and minimally redundant. Regression testing, required during validation and maintenance phases, is simplified as well. When a portion of code has been altered (corrected, improved, etc.), those test runs involving the changed code, i.e., the set of tests that must be re-evaluated, is readily identified.

Although the traps inserted by the dynamic analyzer will usually be removed before the program begins 'normal' operation (the traps introduce considerable overhead in both space and time), it may sometimes be desirable to leave them intact for a while. For example, if a program is to be optimized, it is extremely important to know which portions of code are repeatedly executed during normal operation. Small improvements in these will result in a significantly more efficient program. Conversely, if a portion of code is executed only rarely, it might not be worthwhile to bother optimizing it at all. In a similar vein, a precise description of the normally running program in terms of the types of instructions executed, number of calls made to specific system routines, time spent performing certain functions, average running time, etc., is essential if an accurate model of the program is to be built.

It should be noted that static and dynamic analyzers accept a program written in some (higher level) language A as input, and output a detailed program description, or another (augmented) program in language A, respectively. Theoretically, then, a single set of these tools could be useful for language A programs running anywhere on the network.

4.1.4 Verification Condition Generators and Theorem Provers

For some programs, such as programs which deploy nuclear weapons, handle air traffic control, or control access to ultra-sensitive files, testing is not sufficient. Testing a program thoroughly serves to increase confidence in its reliability. However, no set of test cases (short of an exhaustive list of all possible inputs) will ever guarantee correctness in any mathematical sense. A rigorous proof consists of two separate but related tasks:

(1) Given the subject program together with certain additional information (assertions over the program variables) provided by the programmer, generate a set of potential theorems, the proof of which ensures the correctness of the program. The potential theorems are called verification conditions.

(2) Prove each of the verification conditions.

The overall process of proving that a program is correct is depicted in Figure 4.

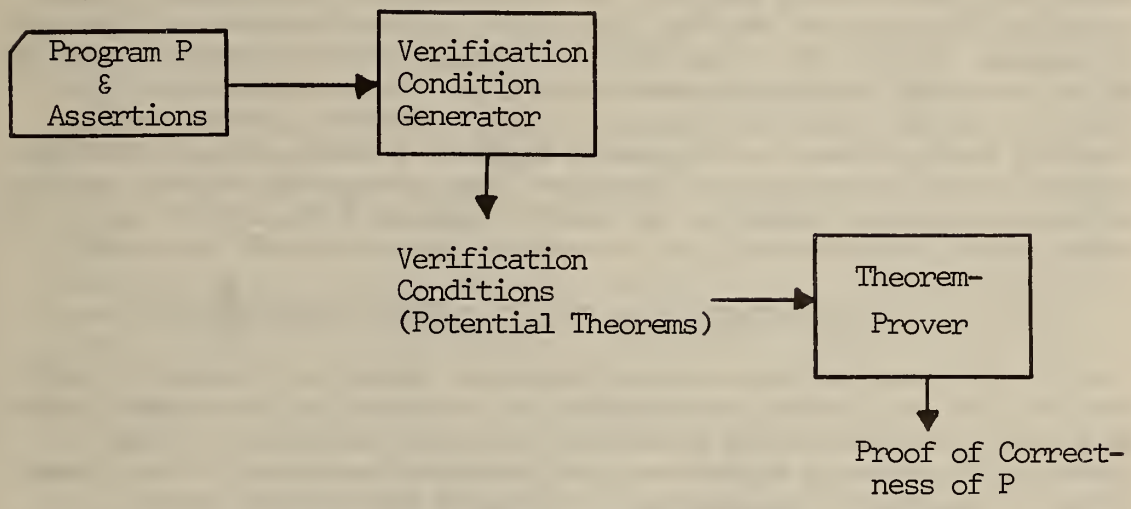

Figure 4: Proof of Program Correctness

The verification condition generator accepts the program and the assertions as input, and, using a semantic definition of the programming language, generates verification conditions. Each verification condition (many, but not all of which are trivially simple to prove) is proven manually, i.e., by a human, or automatically, i.e., by a theorem-proving program. Some important research (which has implications in the areas of programming language design, and overall system design) in the area of hand-generated proofs has been done by R. L. London [9], [10], C.A.R. Hoare [5], and others. Since these proofs can be lengthy and tedious, however, they are subject to error in much the same way as the original program was. For this reason, and because proving the correctness of large programs involves proving many verification conditions, the concept of machine-generated proofs is appealing.

The principal obstacle to proving programs correct automatically lies in the fact that all current theorem-provers are inefficient. Most of the inferences they generate turn out either to be irrelevant to the proof which is eventually produced, or to provide less information than an inference generated previously. For most interesting problems, all available resources (i.e., time and space) are exhausted before a proof can be constructed. Various strategies--some involving the interactive intervention of human intelligence at key points in the proof process--have been devised in an attempt to increase the efficiency and effectiveness of theorem-provers. It would be a major contribution to this research if a "program-proving facility" were made available over the network. Since neither verification condition

generators nor theorem-provers are available at most installations, this facility would include modules embodying verification condition generators (for each of several programming languages), basic inference generators, and a broad range of proof strategies. Additional modules would be incorporated into the facility by interested users as they are developed. Given such a facility, new program-proving systems could be built by re-configuring the various modules. Since experts in this field are widely separated geographically, a network program-proving facility would serve also to promote rapid up-to-date communication and software sharing among them. Moreover, the network can offer a variety of hardware not available at any single research installation (e.g., the associative processor which may be available over the ARPA network).

4.1.5 On-Line Documentation and Interactive Help Routines

On-line documentation capabilities and extensive interactive help routines are particularly appropriate in a networking environment, where many continuously changing facilities are being shared by a broad spectrum (in terms of experience and interests) of users. The documentation aids serve to familiarize a user with the software and to identify recent changes made to it; help routines assist him in diagnosing and correcting problems he encounters in using the software. To maximize its utility, programmers should be able to determine (e.g., by choosing among several options) the quantity, level of detail, and, wherever appropriate, the format of the documentation he requests. Help routines should be flexible as well. For example, the "standard" prelude should be omitted at the user's request.

In systems relying on hard-copy communication of changes (e.g., newsletters, updates to systems manuals), documentation lag is an inherent and unavoidable characteristic. An especially valuable feature of on-line documentation, therefore, is that it can always be kept current (provided system changes are made together with, and not prior to, changes in the on-line documentation). Within this context, then, the user should be able to request information concerning the current status of the system, for example,

(1) a list of all changes made to a particular system during the last week (month, day, etc.);

(2) a list of all corrections made in response to complaints initiated by the user;

(3) a list of all corrections made to a particular module within a system, etc.

Note that this information is readily obtainable from the Operational Errors File and the Corrections File.

4.2 Potential Areas of Research Using the Network

In the absence of any direct and objective measure, software quality and reliability (or the lack thereof) can be gauged only in terms of malfunctions that occur, e.g., mean time between failures, mean time to repair. The Operational Errors File and the Corrections File, therefore, constitute a formal mechanism by which to measure the reliability of network software and assess the value of new software features or approaches. That is, by using new techniques to build some network software package, and then carefully analyzing the Operational Errors and Corrections Files (to determine where and how they differ from the Files associated with similar network software packages that do not involve the new technique), it might be possible to assess the impact and efficacy of the new technique. We caution that factors which are either difficult to measure or are unknown or both (e.g., capability of the programmer) will be reflected in the Files, and that therefore, any conclusions drawn from the Files will have to be based on very gross changes in such things as frequency of errors, etc. With this reservation in mind, we now describe experiments involving the concepts of structured programming and systematic testing of programs (using software testing tools), both for their inherent scientific value and also as illustrations of the type of research that might be performed using the network and its Files.

4.2.1 Structured Programming

Software which is built from a very limited and well-defined set of control structures is thought to be more reliable than conventionally written (i.e., unstructured) code [4]. The argument is that programmers use their unrestricted GO-TO rights to construct an intricate maze of arbitrary transfers, directing control helter-skelter through the program and thereby obscuring the underlying logic. The execution characteristics of such a program are extremely difficult to analyze and the programmer is unlikely to know exactly what is going on. On the other hand, if the program is built as a hierarchy of modules, and if strict rules are enforced governing the transfer of control within and between modules, the logic will be much more explicit, and the program should be simpler to understand, document, debug, test, and maintain. The computational completeness of certain restricted classes of control structures has been proven by Böhm and Jacopini [2], and Kosaraju [3].

By analyzing the Operational Errors File and the Corrections File of a "structured" compiler offered on the network, and comparing them to the Files of conventionally written compilers, we might be able to address the following types of questions.

(a) Is structured programming worthwhile with respect to reliability, e.g., are there fewer, less serious errors in structured programs?

(b) Are the types of errors that occur in structured programs different from those occurring in unstructured programs, and can they be more easily detected and/or avoided?

(c) Are structured programs easier to maintain, and are they less sensitive to modification, i.e., after program modification, do fewer or less serious errors occur in structured programs?

4.2.2 Systematic Testing of Software

Until it becomes practical to prove correctness for large programs in a mathematically rigorous way, testing will be an important phase of software development. However, since even simple software packages may have an infinite input domain and an extraordinarily large number of execution paths, it is impossible to test a program under all conceivable running conditions. Current practice is to design and implement a system, and then to test it for some arbitrary subset of possible input values and environmental conditions. The program is accepted when it executes these test cases correctly. However, there are usually a significant number of residual errors. The user uncovers these errors in the course of operation, when the software fails to run for certain inputs, when the computed results are clearly incorrect, or when the software reacts with its environment in unexpected and undesirable ways. The cost to the user is substantial.

The high error content of developed and tested software is not due to poor workmanship on the part of the developers and testers, but rather to the lack of techniques for dealing adequately with the complexity of large computer programs. In particular,

(1) the developer cannot accurately measure the effectiveness of a particular test;

(2) the developer cannot determine whether his set of test cases has thoroughly exercised the software. Moreover, he cannot specify particular paths in the software which have never been exercised;

(3) current software packages are so complex that a thorough manual analysis of the test space is not feasible.

Dynamic analyzers (as described in Section 4.1.3.2) have been proposed as a means of dealing more effectively with complex software logic. The information provided by dynamic analyzers, e.g., which statements are executed, which branches are taken, which subroutines are entered and in what order, forms a basis for defining and constructing a set of test cases which thoroughly tests a program. Several definitions of a "thorough set of test cases" come to mind, for example, a set which exercises every statement in the program at least

once, or a set which causes the execution of every branch in the program. Having tested a network program "thoroughly," we can compare it to similar programs tested ad-hoc and attempt to answer the following types of questions:

(1) Are "thoroughly" tested programs more reliable, i.e., do they have fewer, less serious errors?

(2) How is testing thoroughness related to reliability, i.e., are there degrees of thoroughness in testing, and are these indicative of the reliability of the program?

(3) Is regression testing (performed after modifying the software) easier (i.e., faster, cheaper) when testing tools are used, and are the re-tested programs more reliable than modified programs tested ad-hoc?

(4) Are certain programs, for example, structured programs, more "testable" than others, i.e., does it take fewer test cases to thoroughly test them, or are the test cases more easily constructed?

4.2.3 Further Analysis of the Operational Errors and Corrections Files

We have already suggested how the Operational Errors and Corrections Files can be used to measure the reliability of network software, the diligence with which network software is being maintained, and the effectiveness of new software techniques. We now suggest that the data collected in these files is useful in itself. One major obstacle to software quality research has been the lack of hard data concerning errors, e.g., what causes them, which type of errors occur most (least) frequently, which cause the most (least) serious malfunctions, etc. Given this data--which is precisely the data collected in the Operational Errors and Corrections Files--it might be possible to categorize software errors, and to determine how each class of errors could have been avoided, for example:

(1) by using a modified version of the programming language, or a different language altogether;

(2) by writing structured programs, or abandoning the concept;

(3) by using "standard" library versions of frequently needed routines, rather than re-inventing (and re-debugging) them each time;

(4) by employing mathematical proof concepts on a broader scale;

(5) by systematic testing using automated software testing tools;

(6) by using more dynamic range and data bound checks, and optionally compilable assertions (i.e., run-time tests), or by distributing them differently.

5. SUMMARY AND RECOMMENDATIONS

This report has described mechanisms for error collection and analysis -- the System Performance Profile, and the Operational Errors and Corrections Files -- with two goals in mind. The first is to provide a basis for measuring network reliability (in terms of the frequency of errors, or the frequency of user complaints, or the amount of system downtime, etc.) and maintenance quality (in terms of the delay between error reports and implemented corrections, or the number of new errors introduced in the course of modifying the system, etc.). Network standards for system reliability and maintenance might, therefore, be established and enforced. The second goal of the Files is to facilitate research in the area of software quality, which has to date been crippled by a paucity of hard data concerning the nature of software errors in large systems. No one has yet been able to analyze and categorize software errors, determine their frequency of occurrence, and then suggest ways to identify and/or avoid them. The data collected in the Files of a large distributed network should be valuable in this type of endeavor.

Moreover, by carefully monitoring and analyzing changes in the Operational Errors and Corrections Files, the efficacy of new software techniques might be assessed. Experiments were suggested to evaluate the utility of structured programming and of systematic software testing. Finally, the possibility of creating a "software production laboratory" on the network was addressed, and specific facilities for supporting such a concept were suggested. These included compatible interpreters and compilers, a wide range of verification condition generators and theorem-provers, automated software testing tools, and extensive on-line documentation and interactive help routines.

6. REFERENCES

1. Bloom, S., McPheters, M. J., Tsiang, S. H., "Software Quality Control," 1973 IEEE Symposium on Computer Software Reliability, May 1973.

2. Böhm, C. and Jacopini, G., "Flow Diagrams, Turing Machines, and Languages with Only Two Formation Rules," C.ACM 9, 5, May 1966.

3. Brown, J. R., DeSalvio, A. J., Heine, D. E., Purdy, J. G., "Automated Software Quality Assurance: A Case Study of Three Systems," ACM SIGPLAN Symposium on Computer Program Test Methods, June 1972.

4. Dijkstra, E. W., "Notes on Structured Programming," Technische Hogeschool, Eindhoven, August 1969.

5. Hoare, C. A. R., "An Axiomatic Basis for Computer Programming," C.ACM 12, 10, October 1969.

6. Hoffman, R. H., "Automated Verification System User's Guide," TRW Note No. 72-FMT-891, Project Apollo, January 1972.

7. Knuth, D. E., "An Empirical Study of FORTRAN Programs," Stanford Artificial Intelligence Project, Computer Science Department, Stanford University, Report No. STAN-CS-186.

8. Kosaraju, R., "Analysis of Structured Programs," Quality Software Technical Report, Electrical Engineering Department, Johns Hopkins University, November 1972.

9. London, R. L., "Computer Programs Can be Proved Correct," Proceedings of the Fourth Systems Symposium--Formal Systems and Non-Numerical Problem Solving by Computers, Case Western Reserve University, November 1968.

10. London, R. L., "Software Reliability Through Proving Programs Correct," Publ. 71C6-C, IEEE Computer Society, New York, March 1971.

11. McGeachie, J. S., "Reliability of the Dartmough Time Sharing System," 1973 IEEE Symposium on Computer Software Reliability, May 1973.

12. Neumann, A. J., "Review of Network Management Problems and Issues," NBS Technical Note 795, October 1973.

13. Stucki, L. G., "Automatic Generation of Self-Metric Software," 1973 IEEE Symposium on Computer Software Reliability, May 1973.

FORM NBS-114A (1-71)

U.S. DEPT. OF COMM. BIBLIOGRAPHIC DATA SHEET	1. PUBLICATION OR REPORT NO. NBS TN-800	2. Gov't Accession No.	3. Recipient's Accession No.

4. TITLE AND SUBTITLE

COMPUTER NETWORKING: APPROACHES TO QUALITY SERVICE ASSURANCE

5. Publication Date January 1974

6. Performing Organization Code

7. AUTHOR(S) Rona B. Stillman

8. Performing Organization

9. PERFORMING ORGANIZATION NAME AND ADDRESS

NATIONAL BUREAU OF STANDARDS
DEPARTMENT OF COMMERCE
WASHINGTON, D.C. 20234

10. Project/Task/Work Unit No. 640-2415

11. Contract/Grant No.

12. Sponsoring Organization Name and Address

Office of Computing Activities
National Science Foundation
18th & G Street, NW
Washington, D. C. 20550

13. Type of Report & Period Covered Final

14. Sponsoring Agency Code

15. SUPPLEMENTARY NOTES

16. ABSTRACT (A 200-word or less factual summary of most significant information. If document includes a significant bibliography or literature survey, mention it here.)

The problem of quality service assurance in a (generalized) computer networking environment is addressed. In the absence of any direct, well-defined, quantitative measure of service quality and reliability, error collection and analysis is the only basis for service quality control. Therefore, mechanisms are described which facilitate reporting of operational errors, documentation of error corrections, and collection of system performance data. Since techniques for hardware quality control are well known, these mechanisms focus on collecting data which can be used to assess and control software quality. Finally, specific network facilities are described which support research in the area of software quality, and potential areas of new research using the network are identified.

17. KEY WORDS (Alphabetical order, separated by semicolons) Compiler; computer network; documentation; dynamic software analysis; interpreter; quality control; software testing; software verification; static software analysis; structured programming; system errors; system performance; theorem-proving.

[X] UNLIMITED.

[ ] FOR OFFICIAL DISTRIBUTION. DO NOT RELEASE TO NTIS.

19. SECURITY CLASS (THIS REPORT)	21. NO. OF PAGES
UNCLASSIFIED	26
20. SECURITY CLASS (THIS PAGE)	**22. Price**
UNCLASSIFIED	.60

USCOMM-DC 66244-P71

NBS TECHNICAL PUBLICATIONS

PERIODICALS

JOURNAL OF RESEARCH reports National Bureau of Standards research and development in physics, mathematics, and chemistry. Comprehensive scientific papers give complete details of the work, including laboratory data, experimental procedures, and theoretical and mathematical analyses. Illustrated with photographs, drawings, and charts. Includes listings of other NBS papers as issued.

Published in two sections, available separately:

- **Physics and Chemistry (Section A)**

Papers of interest primarily to scientists working in these fields. This section covers a broad range of physical and chemical research, with major emphasis on standards of physical measurement, fundamental constants, and properties of matter. Issued six times a year. Annual subscription: Domestic, $17.00; Foreign, $21.25.

- **Mathematical Sciences (Section B)**

Studies and compilations designed mainly for the mathematician and theoretical physicist. Topics in mathematical statistics, theory of experiment design, numerical analysis, theoretical physics and chemistry, logical design and programming of computers and computer systems. Short numerical tables. Issued quarterly. Annual subscription: Domestic, $9.00; Foreign, $11.25.

DIMENSIONS, NBS

The best single source of information concerning the Bureau's measurement, research, developmental, cooperative, and publication activities, this monthly publication is designed for the layman and also for the industry-oriented individual whose daily work involves intimate contact with science and technology —*for engineers, chemists, physicists, research managers, product-development managers, and company executives.* Annual subscription: Domestic, $6.50; Foreign, $8.25.

NONPERIODICALS

Applied Mathematics Series. Mathematical tables, manuals, and studies.

Building Science Series. Research results, test methods, and performance criteria of building materials, components, systems, and structures.

Handbooks. Recommended codes of engineering and industrial practice (including safety codes) developed in cooperation with interested industries, professional organizations, and regulatory bodies.

Special Publications. Proceedings of NBS conferences, bibliographies, annual reports, wall charts, pamphlets, etc.

Monographs. Major contributions to the technical literature on various subjects related to the Bureau's scientific and technical activities.

National Standard Reference Data Series. NSRDS provides quantitative data on the physical and chemical properties of materials, compiled from the world's literature and critically evaluated.

Product Standards. Provide requirements for sizes, types, quality, and methods for testing various industrial products. These standards are developed cooperatively with interested Government and industry groups and provide the basis for common understanding of product characteristics for both buyers and sellers. Their use is voluntary.

Technical Notes. This series consists of communications and reports (covering both other-agency and NBS-sponsored work) of limited or transitory interest.

Federal Information Processing Standards Publications. This series is the official publication within the Federal Government for information on standards adopted and promulgated under the Public Law 89–306, and Bureau of the Budget Circular A–86 entitled, Standardization of Data Elements and Codes in Data Systems.

Consumer Information Series. Practical information, based on NBS research and experience, covering areas of interest to the consumer. Easily understandable language and illustrations provide useful background knowledge for shopping in today's technological marketplace.

BIBLIOGRAPHIC SUBSCRIPTION SERVICES

The following current-awareness and literature-survey bibliographies are issued periodically by the Bureau:

Cryogenic Data Center Current Awareness Service (Publications and Reports of Interest in Cryogenics). A literature survey issued weekly. Annual subscription: Domestic, $20.00; foreign, $25.00.

Liquefied Natural Gas. A literature survey issued quarterly. Annual subscription: $20.00.

Superconducting Devices and Materials. A literature survey issued quarterly. Annual subscription: $20.00. Send subscription orders and remittances for the preceding bibliographic services to the U.S. Department of Commerce, National Technical Information Service, Springfield, Va. 22151.

Electromagnetic Metrology Current Awareness Service (Abstracts of Selected Articles on Measurement Techniques and Standards of Electromagnetic Quantities from D-C to Millimeter-Wave Frequencies). Issued monthly. Annual subscription: $100.00 (Special rates for multi-subscriptions). Send subscription order and remittance to the Electromagnetic Metrology Information Center, Electromagnetics Division, National Bureau of Standards, Boulder, Colo. 80302.

Order NBS publications (except Bibliographic Subscription Services) from: Superintendent of Documents, Government Printing Office, Washington, D.C. 20402.

U.S. DEPARTMENT OF COMMERCE
National Bureau of Standards
Washington, D.C. 20234

OFFICIAL BUSINESS

Penalty for Private Use, $300

POSTAGE AND FEES PAID
U.S. DEPARTMENT OF COMMERCE
COM-215