;login:
The USENIX Association Newsletter

Volume 11, Number 1       January/February 1986

CONTENTS

To the Members from the Executive Director .......................................................... 3
USENIX Winter '86 Conference .................................................................................. 4
  Schedule of Events ................................................................................................. 4
  Registration Fees .................................................................................................. 5
  Pre-Registration Mailing ....................................................................................... 5
  Technical Session A – Window Environments and UNIX ........................................ 6
  Wednesday’s Tutorials ......................................................................................... 7
  Technical Session B – UNIX on Big Iron ............................................................... 9
  Thursday’s Tutorials ........................................................................................... 10
  Technical Session C – Ada and the UNIX System ............................................... 12
  Friday’s Tutorials .............................................................................................. 13
Future Meetings ..................................................................................................... 15
Third Printing of USENIX 4.2BSD Manuals ............................................................... 16
Local User Groups .................................................................................................. 17
Publications Available ............................................................................................. 18
Ease: A Configuration Language for Sendmail ....................................................... 19
  James S. Schone
Load Balancing With Maitre d’ ............................................................................. 32
  Brian Bershad
A Perspective on the USENET .............................................................................. 46
  Erik E. Fair
Release Form for 1986 USENIX Distribution Tapes ........................................... 53
4.2BSD Manual Reproduction Authorization and Order Form ............................ 55

The closing date for submissions for the next issue of ;login: is January 31, 1986
NOTICE

;login: is the official newsletter of the USENIX Association, and is sent free of charge to all members of the Association.

The USENIX Association is an organization of AT&T licensees, sub-licensees, and other persons formed for the purpose of exchanging information and ideas about UNIX and UNIX-like operating systems and the C programming language. It is a non-profit corporation incorporated under the laws of the State of Delaware. The officers of the Association are:

President Alan G. Nemeth
Vice-President Deborah K. Scherrrer
Secretary Lewis A. Law
Treasurer Waldo M. Wedel
Directors Thomas Ferrin
Steve C. Johnson
Lou Katz
Michael D. Tilson
Executive Director James E. Ferguson

The editorial staff of ;login: is:
Publisher James E. Ferguson
Copy Editor Michelle Peetz
Managing Editor Tom Strong
Editorial Director Lou Katz

Other staff members are:
Betty J. Madden Office Manager
Connie I. Howard Membership Secretary
Erik E. Fair Technical Consultant

USENIX Association Office
P.O. Box 7
El Cerrito, CA 94530
(415) 528-8649
{ucbvax,decvax}@usenixoffice

Judy DesHarnais Conference Coordinator
USENIX Conference Office
P.O. Box 385
Sunset Beach, CA 90742
(213) 592-3243 or 592-1381
{ucbvax,decvax}@usenixjudy

John L. Donnelly Exhibit Manager
USENIX Exhibit Office
Oak Bay Building
4750 Table Mesa Drive
Boulder, CO 80303
(303) 499-2600
{ucbvax,decvax}@usenixjohnd

Contributions Solicited

Members of the UNIX community are heartily encouraged to contribute articles and suggestions for ;login:. Your contributions may be sent to the editors electronically at the addresses above or through the U.S. mail to the Association office. The USENIX Association reserves the right to edit submitted material.

;login: is produced on UNIX systems using troff and a variation of the -me macros. We appreciate receiving your contributions in n/troff input format, using any macro package. If you contribute hardcopy articles please leave left and right margins of 1" and a top margin of 1½" and a bottom margin of 1¼". Hardcopy output from a line printer or most dot-matrix printers is not reproducible.

Acknowledgments

The Association uses a VAX† 11/730 donated by the Digital Equipment Corporation for support of office and membership functions, preparation of ;login:, and other association activities. It runs 4.2BSD, which was contributed, installed, and is maintained by mt XINU. The VAX uses a sixteen line VMZ-32 terminal multiplexor donated by Able Computer of Irvine, California.

Connected to the VAX is a QMS Lasergrafix* 800 Printer System donated by Quality Micro Systems of Mobile, Alabama. It is used for general printing and draft production of ;login: with ditroff software provided by mt XINU.

This newsletter may contain information covered by one or more licenses, copyrights, and/or non-disclosure agreements. No reproduction of this newsletter in its entirety or in part may be made without written permission of the Association.

†UNIX is a trademark of AT&T Bell Laboratories.

‡VAX is a trademark of Digital Equipment Corporation.

*Lasergrafix is a trademark of Quality Micro Systems.
To the Members from the Executive Director

Call for Papers

We are trying to make ;login: more interesting for the members by including more technical articles. However, we need your help to do so. We seek your help in submitting to us technical papers which are appropriate for publication in ;login:. We are not planning to make ;login: a refereed journal. The primary standards on which papers will be evaluated are:

1. Interest to a significant number of USENIX members
2. Technically sound

If you have a paper and are willing to share it with your fellow USENIX members, please send it to us – either electronically or hardcopy – at the Association office.

1986 Dues Billings and Membership Survey

USENIX is doing something different for its 1986 membership dues.

First, we are sending everyone a billing notice, rather than merely reminding you about your dues through ;login:. Second, we are showing you what information we have on file about you in our membership database, and asking you to correct or update the information, if needed. Third, we are asking you to take just a few minutes to answer the questions on a membership survey so USENIX may better serve the needs and interests of its members.

By the time you read this, you should have already received your 1986 Membership Renewal Invoice and the Membership Survey – provided the weather and Christmas mail do not slow it down and make U. S. Mail stand for Unpredictable Snail Mail.

We would like to review and discuss the results of the Membership Survey at the Board of Directors meeting during the Denver Conference, January 15-17. Naturally we prefer that you mail your 1986 dues and completed Membership Survey at the same time. However, if you are a little short $$$wise now, because of Christmas or whatever, will you please complete and return the Membership Survey NOW, and then mail your 1986 dues when it is a little more convenient. Thanks.

1986 Winter Conference in Denver

From all of the early registrations and other indications, it appears the Denver Conference will be a much greater success than anticipated. As one member pointed out to us, “3 separate tutorials, in 3 days, for only $300 – is the best deal USENIX or anyone else has offered in a long, long time.” Interest in all three of the 1-day technical sessions is likewise pleasantly exceeding our original expectations.

If you have not already decided to attend the Denver Conference, we encourage you to reconsider and do so. It will be well worth your time and money.

Best Wishes

The Board and Staff of USENIX extend their best wishes that 1986 will be an even better year for you than 1985 was.

Jim Ferguson
Executive Director
USENIX Winter '86 Conference
January 15-16-17, 1986
Marriott Hotel – City Center
Denver, Colorado

The USENIX Winter '86 Conference will consist of three workshop-oriented technical sessions accompanied by fourteen full-day tutorials. Each of the technical sessions will be a full day devoted to only one topic. For each topic area, there will be related tutorials on adjacent days, concurrent with the other technical sessions. There will also be several tutorials of general interest. The topics of the technical sessions are:

- Window Environments and UNIX – Wednesday, January 15
- UNIX on Big Iron – Thursday, January 16
- Ada® and the UNIX System – Friday, January 17

You may attend the conference for one, two, or all three days, but only one technical session or tutorial per day. A full description of each technical session and tutorial appears below.

The Conference Host for this meeting is the Computer Science Department of the University of Colorado at Boulder. Evi Nemeth is the local coordinator. All events are being held at the Marriott Hotel – City Center in Denver.

There is excellent downhill and cross-country skiing available near Denver. Public transportation is available to most areas. Arrangements are being considered for both day and weekend trips, both before and after the meeting. Further information is being discussed in net.usenix and net.ski and will be available at the meeting.

Schedule of Events

Wednesday, January 15, 1986, 9am–5pm

Technical Session A: Window Environments and UNIX
- Tutorial #1: Design Considerations for SNA Communications Under UNIX
- Tutorial #2: UNIX Device Driver Design (4.2BSD)
- Tutorial #3: System V Interprocess Communication Application Programming
- Tutorial #4: Ada – From The Top; An Introduction
- Tutorial #5: UNIX System V Internals

Thursday, January 16, 1986, 9am–5pm

Technical Session B: UNIX on Big Iron
- Tutorial #6: Introduction to 4.2BSD Internals
- Tutorial #7: Windowing System Implementations
- Tutorial #8: Language Construction Tools on the UNIX System
- Tutorial #9: Advanced C Programming

Friday, January 17, 1986, 9am–5pm

Technical Session C: Ada and the UNIX System
- Tutorial #10: Advanced Topics on 4.3BSD Internals
- Tutorial #11: UNIX Networking
- Tutorial #12: Managing a Local Area Network
- Tutorial #13: Introduction to UNIX Systems Administration
- Tutorial #14: Writing Portable C Programs
Registration Fees

Registration fees are based on a per day rate. You may attend the conference for one, two, or three days, choosing either a technical session or a tutorial class on each day. It is not possible to attend more than one topic per day. A copy of the conference proceedings will be given to each person registering for at least one technical session. The proceedings will also be for sale at the conference.

<table>
<thead>
<tr>
<th>Membership Status</th>
<th>Number of Days Attending Conference</th>
<th>one day</th>
<th>two days</th>
<th>three days</th>
</tr>
</thead>
<tbody>
<tr>
<td>USENIX Member</td>
<td>Pre-registered – postmarked no later than 12/27/85</td>
<td>$150</td>
<td>$250</td>
<td>$300</td>
</tr>
<tr>
<td></td>
<td>On-site or postmarked after 12/27/85</td>
<td>$200</td>
<td>$300</td>
<td>$350</td>
</tr>
<tr>
<td>Non-Member</td>
<td>Pre-registered – postmarked no later than 12/27/85</td>
<td>$180</td>
<td>$280</td>
<td>$330</td>
</tr>
<tr>
<td></td>
<td>On-site or postmarked after 12/27/85</td>
<td>$230</td>
<td>$330</td>
<td>$380</td>
</tr>
<tr>
<td>Student</td>
<td>Pre-registered or on-site (Must enclose photocopy of current student ID card with registration)</td>
<td>$75</td>
<td>$125</td>
<td>$150</td>
</tr>
</tbody>
</table>

For pre-registration rates, registration forms must be received with full payment and postmarked no later than December 27, 1985. Visa, Mastercard, and American Express are accepted for registration.

Pre-Registration Mailing

Pre-registration materials containing detailed conference information along with registration and hotel reservation forms were mailed in the middle of November to USENIX members and previous conference attendees. If you need further information or a registration packet, please contact:

USENIX Conference Office  
P.O. Box 385  
Sunset Beach, CA 90742  
(213) 592-3243 or 592-1381

Hotel Registration Deadline – December 24, 1985  
Pre-Registration Deadline – December 27, 1985
Winter '86 Conference Schedule – Wednesday, January 15

Technical Session A – Window Environments and UNIX

Program Committee: Sam Leffler, Lucasfilm, Chair
                    Mike Hawley, The Droid Works, Co-Chair
                    Jim Gettys, Digital Equipment Corp.
                    James Gosling, Sun Microsystems
                    Rob Pike, Bell Laboratories

This meeting will explore the design and integration of UNIX-based window systems and their applications. Three sessions and a panel discussion have been organized. The following presentations are scheduled.

8:30 - 10:10  Hardware and Hardware Issues:

Opening Remarks
Conference Organizers and USENIX Board
Galadriel: A Display List-Based Window Manager
Bob Lewis, Tektronix
Next Generation Hardware for Windowed Displays
Steven McGeady
Real-Time Resource Sharing for Graphics Workstations
Mark Grossman & Glen E. Williams, Silicon Graphics, Inc.

10:30 - 12:00  Applications:

GLO – A Tool for Developing Window-Based Programs
Thomas Neuendorffer, Carnegie-Mellon University

A Workstation-Based In-patient Clinical System in the Johns Hopkins Hospital
Stephen N. Kahane, et al, Johns Hopkins University Hospital

The Feel of Pi
T. A. Cargill, AT&T Bell Laboratories

1:30 - 3:30  Systems and System Issues:

Flamigo: Object-Oriented Abstractions for User Interface Management

A Proposal for Interwindow Communication and Translation Facilities
Daniel P. Gill, Exxon Research and Engineering

Problems Implementing Window Systems in UNIX
Jim Gettys, Massachusetts Institute of Technology

SUNDEW: A Distributed and Extensible Window System
James Gosling, Sun Microsystems

4:00 - 5:00  Panel Discussion:

Color? Do we need it? How can we use it? How do we deal with it? ...
Wednesday’s Tutorials

Tutorial # 1: Design Considerations for SNA Communications Under UNIX
Instructor: Daniel Fisher
System Strategies, Inc.

One of the major design considerations a developer must address when integrating SNA or other communications protocols into a UNIX-based system is the distribution of software system components. This presentation will define the three basic options open to the developer for placement of software components: utilization of user space; integration at the kernel level; and board level integration. The relative advantages and disadvantages of each alternative will be discussed in detail. Sample implementations of a SNA/3270 emulation and a X.25 emulation on UNIX-based systems will then be presented, followed by a discussion of the concept of streams implementation currently under development at AT&T Information Systems and its future implications as they relate to UNIX users.

System Strategies, Inc., has developed several major communication packages that provide compatibility with IBM’s standard network protocols, SNA and BSC. It provides portable 3270 SNA and BSC packages under UNIX. Daniel Fisher is a Senior Software Engineer in the product development group at Systems Strategies. He has developed and implemented several UNIX-based communications systems.

Tutorial # 2: UNIX Device Driver Design (4.2BSD)
Instructor: Daniel Klein
Consultant

4.2BSD SOURCE LICENSE REQUIRED FOR THIS TUTORIAL

This course is designed for people who wish to become familiar with the fundamentals of designing UNIX device drivers. A knowledge of the major structures and internals of 4.2BSD UNIX is a desirable prerequisite to this tutorial, although a specific knowledge of the finer details is not required. This seminar will cover the major aspects of driver design, implementation, and device integration. Both DMA and programmed I/O device drivers will be covered, as well as block and character (buffered and unbuffered) interfaces. We will outline the design and implementation of structured I/O devices (i.e. disk drives), and semi-structured devices (i.e. tape drives and serial communication links). This course will also discuss all aspects of adding a new device to the kernel (i.e. autoconfiguration, special files, device tables, and debugging). The intended audience for this course is systems programmers who will be actively engaged in the maintenance or design and implementation of UNIX device drivers. Although this course will be geared towards 4.2BSD, a comparison between the Berkeley and Bell Labs approaches will be offered. Users of System III or System V will therefore also find this course to be informative.

Daniel Klein has been involved with UNIX since the original university distribution of Version 6 in 1976, including writing device drivers, utility programs, applications systems, and enhancements to the kernel. A graduate of Carnegie-Mellon University, Mr. Klein was manager of software systems at Mellon Institute for six years. He is presently engaged in teaching UNIX internals and developing an on-line educational system for UNIX, as well as developing a multi-processor simulation system.
Winter ’86 Conference Schedule – Wednesday, January 15  (Continued)

Tutorial # 3:  System V Interprocess Communication Application Programming
Instructor:  Jon H. LaBadie
AUXCO

This tutorial will be targeted toward application programmers who wish to know how and why to use the interprocess communication (IPC) facilities described in the UNIX System V Interface Specification. Syntax and examples of the use of the seven types of IPC facilities will be discussed. The facilities to be covered include: semaphores, message queues, shared memory, named and unnamed pipes, signals, and process waiting and exit status.

Jon LaBadie’s UNIX involvement spans the past seven years during which time he has designed and implemented a number of graphics and database applications using UNIX and C. For the past three years, he has served as a consultant to the UNIX training group at AT&T where he has developed and taught numerous UNIX and C courses. He holds a B.S. degree in biology from Drexel University, and a Ph.D. in Biochemistry from Pennsylvania State University.

__________

Tutorial # 4:  Ada – From The Top; An Introduction
Instructor:  Putnam P. Texel
Texel & Company

This tutorial is designed for those individuals familiar with a procedure oriented high order language, but do not have much familiarity with Ada. The level of the tutorial is applicable for managers who need to understand Ada concepts and software engineers taking their first look at this most powerful and expressive language.

Texel & Company specializes in Ada consulting, Ada education, and Ada R&D with clients in government and industry. Ms. Texel has presented tutorials at both SIGAda and AdaJUG conferences, has taught Ada at the graduate level at Monmouth College, and has been an invited panelist on several panels dealing with Ada education. She is Chairperson of the Education Committee of ACM SIGAda and Chairperson of the ACM Princeton Chapter Local SIGAda.

__________

Tutorial # 5:  UNIX System V Internals
Instructor:  Maury Bach & Steve Buroff
AT&T Information Systems

SYSTEM V SOURCE LICENSE REQUIRED FOR THIS TUTORIAL

This tutorial is a survey of the internal structure of AT&T’s UNIX System V, and it is intended for people who maintain, modify, or port UNIX systems. The tutorial will discuss existing UNIX kernel concepts such as I/O system, file system, and process and memory management, as well as new features to be included in the next release of System V, such as the file system switch, streams, remote file sharing, and shared libraries. Attendees should have a good working knowledge of the UNIX system; basic kernel knowledge is recommended.

Maury Bach and Steve Buroff are members of the development staff at AT&T Information Systems. Maury Bach has worked on multi-processor UNIX development, and Steve Buroff has worked on paging virtual memory implementation. Bach also has taught a multi-week UNIX internals course within Bell Labs in order to train Bell Labs systems programmers. This one day tutorial draws upon this work.
Technical Session B – UNIX on Big Iron

Program Committee: Peter Capek, IBM Research, Chair
Jim Lipkis, New York University, Courant Institute
Eugene Miya, NASA Ames Research Center

During this session the use of UNIX on two new classes of systems – very large single processor machines such as the Cray-1 and systems with many processors such as the Alliant – will be discussed. The following presentations are scheduled.

8:30 - 10:00  Applications and Requirements
Opening Remarks
Peter Capek, IBM Research
User Requirements for Future-nix
Eugene Miya, NASA Ames Research Center
Experience with Large Applications on UNIX
Bob Bilyeu, McNeill-Schwindler, Inc.
UNIX Scheduling for Large Systems
Jeffrey H. Straathof, Ashok K. Thareja, & Ashok K. Agrawalal,
University of Maryland

10:30 - 12:00  Real Systems I
A Straightforward Implementation of 4.2BSD
on a High Performance Multiprocessor
Dave Probert, Culler Scientific Systems Corporation
Porting UNIX to the System/370 Extended Architecture
Joseph R. Eykholt, Amdahl Corporation
Full Duplex Support for Mainframes
Don Sterk, Amdahl Corporation

1:30 - 3:10  Real Systems II
Concentrix – A UNIX for the Alliant Multiprocessor
Jack Test, Alliant Computer Corporation
A User-Tunable Multiprocessor Scheduler
Herb Jacobs, Alliant Computer Corporation
High Performance Enhancements of C-1 UNIX
Rob Kolstad, Convex Computer Corporation

3:30 - 5:30  Real Systems III
Considerations for Massively Parallel UNIX Systems on the NYU Ultracomputer
and the IBM RP3
Jan Edler, Allan Gottlieb & Jim Lipkis,
New York University – Courant Institute
UNIX or CTSS for the Cray-1, Cray X-MP and Cray-2 Supercomputers
Karl Auerbach, ZERO-ONE Systems and NASA Ames Research Center
Robin O’Neill, National Magnetic Fusion Energy Computer Center
Experience Porting System V to the Cray-2
Tim Hoel, Cray Research
Thursday's Tutorials

Tutorial # 6:  Introduction to 4.2BSD Internals
Instructor:  Thomas W. Doeppner, Jr.
Brown University

4.2BSD SOURCE LICENSE REQUIRED FOR THIS TUTORIAL

This tutorial is geared to the programmer with a good knowledge of UNIX programming in C, but with little or no experience with UNIX internals. The course will cover process management, high-level I/O (including the file system), low-level I/O (i.e., device drivers), virtual memory, interprocess communication, and networking. After taking the tutorial, the individual will have a basic knowledge of the structure of 4.2BSD and should be able make his or her way through kernel code.

Thomas W. Doeppner, Jr. received his Ph.D. in Computer Science from Princeton University in 1977 and has been on the faculty at Brown University since 1976. He has lectured extensively on UNIX internals over the past two years for the Institute for Advanced Professional Studies.

Tutorial # 7:  Windowing System Implementations
Instructor:  David Rosenthal
Sun Microsystems, Inc.

The tutorial is intended for developers of UNIX window manager applications. It will survey the range of current window systems, concentrating on the programmer's interface, the imaging model, and their support for interaction techniques. Familiarity with C and the UNIX programming environment will be assumed.

David Rosenthal has been researching interactive graphics and user interfaces since 1968. He co-chaired the technical review of GKS, and until recently was Associate Director of the Information Technology Center at Carnegie-Mellon University. He was one of the developers of ITC's Andrew portable window system.

Tutorial # 8:  Language Construction Tools on the UNIX System
Instructor:  Stephen C. Johnson
AT&T Bell Laboratories

This tutorial is intended for C programmers who want to become familiar with the language development tools available on the UNIX system. The course will be directed towards application designers who may wish to use these tools to make front ends for their applications, rather than towards "traditional" compiler writing. Specific topics covered include: designing a language recognizer, the lex and yacc programs, symbol table issues, error reporting and recovery, strong type checking, and testing. Several in-class exercises will be given to lead the students through the construction of a simple front end.

Steve Johnson received his Ph.D. degree in pure mathematics from Columbia University in 1968. In 1967, he joined Bell Laboratories, Murray Hill, N.J., where he worked in psychometrics, computer music, and the computation center before joining the Computer Science Research Department. As a researcher, he worked on computer algebra, wrote the yacc parser generator, contributed to complexity theory and the theory of code generation and parsing, wrote the Portable C Compiler, and, for several years, was involved in experimental VLSI design and silicon compilation. He is now head of the Language Development Department in AT&T Information Systems.
Winter '86 Conference Schedule – Thursday, January 16  (Continued)

Tutorial # 9:  Advanced C Programming  
Instructor:  William C. Steward  
AUXCO

This seminar will be directed toward applications programmers with at least six months experience in the C language. Its focus is on the theories behind C language syntax, which will be illustrated with programming examples. The topics for discussion include: multi-dimensional arrays, pointers to arrays, structures and pointers to structures, pointers to functions, and dynamic memory allocation and linked lists.

William Steward has developed and taught introductory and advanced topics in UNIX and C programming. Formerly with a major research institute, Mr. Steward has extensive experience in presenting UNIX related seminars.
Technical Session C – Ada and the UNIX System

Program Committee: Charles Wetherell, AT&T Information Systems, Chair
                  Donn Milton, Verdim Corp.
                  Tucker Taft, Intermetrics
                  Larry Yelowitz, Ford Aerospace

The Ada and UNIX session will educate and inform UNIX users about the new programming language Ada, and Ada mavens about the UNIX system. The preliminary agenda includes an introduction to the world of Ada with projections about Ada's future growth, a demonstration of a large real-time Ada application running under UNIX, and a variety of technical papers on the use of Ada with UNIX systems. No prior experience with Ada is required; talks and presentations will help you understand Ada itself and its relation to UNIX programming and environments.

The technical papers will include talks comparing C and Ada under UNIX systems, approaches to providing the standard Ada system interfaces to UNIX systems, Ada runtime systems implemented in UNIX environments, revision control and library management systems specially adjusted for the problems of Ada programs, and several common UNIX facilities re-implemented in Ada. Both European and American authors will be represented. Papers will stress the problems and opportunities UNIX systems provide for Ada environments and may help you see some UNIX capabilities.

The following presentations are scheduled.

9:00 - 10:30  Introductory Remarks
               Ada and UNIX
                  Robert Firth, Tartan Labs
               UNIX, C and Ada
                  Herman Fischer, Mark V Business Systems

11:00 - 12:00 Revision Control Tools and the Ada Program Library
                  Dick Schefstrom, TeleLOGIC AB
               Managing Separate Compilation in the AT&T Ada Translator System
                  G. W. Elsesser, M. S. Safran & T. Tiesler, AT&T Information Systems

1:30 - 3:00  Targeting Ada to 68000/UNIX
               Mitchell Gart, Alysys Inc.
               A Comparison of UNIX and CAIS Systems Facilities
                  Helen Gill, Rebecca Bowerman & Chuck Howell, Mitre Corp.
               SVID as an Interim Basis for CAIS
                  Herman Fischer, Mark V Business Systems

3:30 - 4:30  An Overview of the Ada Shell
               Lisa Campbell & Mark Campbell, NCR Corp.
               Implemented Curses in Ada
                  Karl Nyberg, Verdim Corp.
Friday’s Tutorials

Tutorial # 10: Advanced Topics on 4.3BSD Internals
Instructor: Mike Karels & Marshall Kirk McKusick
University of California, Berkeley

4.2BSD SOURCE LICENSE REQUIRED FOR THIS TUTORIAL

This tutorial is directed to systems programmers who have taken a course on 4.2BSD internals or who have had at least a year of experience working on the 4.2BSD kernel. The tutorial will cover the performance work done for 4.3BSD and will also discuss recent and planned changes to the kernel interfaces and facilities. The intent of the tutorial is to present a wide variety of material at a descriptive level. Presentations will emphasize code organization, data structures, and algorithms.

Mike Karels received his B.S. in Microbiology at the University of Notre Dame. While a graduate student at the University of California, he was the major contributor to the 2.9BSD release of the Berkeley Software Distribution for PDP-11’s. He currently is the Principal Programmer at the Berkeley Computer Systems Research Group, continuing the development of future versions of Berkeley UNIX.

Kirk McKusick got his undergraduate degree in Electrical Engineering from Cornell University. His graduate work was done at the University of California, where he received Masters degrees in Computer Science and Business Administration, and a Ph.D. in the area of programming languages. While at Berkeley he implemented the 4.2BSD fast file system and was involved in implementing the Berkeley Pascal system. He currently is the Research Computer Scientist at the Berkeley Computer Systems Research Group, continuing the development of future versions of Berkeley UNIX.

Tutorial # 11: UNIX Networking
Instructor: Bruce Borden
Silicon Graphics, Inc.

Local area networks (LANs) are slowly finding mass acceptance with the entry of IBM into the competition. In October or November, industry analysts expect IBM to announce yet another LAN, probably based on the 802.5 standard but with unknown protocol software. This tutorial will cover the 802.X standards and the most common protocols in use today and expected for the future. Sun’s NFS and AT&T’s distributed file systems will be compared. Berkeley’s Socket implementation will be compared with AT&T’s Streams. This tutorial is directed at experienced UNIX users and developers with knowledge of Ethernet (802.3) and IP/TCP. It is not an introductory tutorial, and will not teach attendees how to program Berkeley 4.X networks.

Bruce Borden is Director of Engineering for Silicon Graphics, Inc., developing very high performance 3-D color graphics workstations running UNIX System V. Prior to SGI, Bruce was a founder of 3Com, developed the Excelan TCP/IP front-end protocol package, and authored the Rand MH mail handling System.

Tutorial # 12: Managing a Local Area Network
Instructor: Evi Nemeth & Andy Rudoff
University of Colorado, Boulder

This tutorial is a summary of all the things we (and many others) have learned over the past couple of years in managing a growing local area network. It is intended for system administrators and others involved in planning, configuring, installing, and maintaining a networked UNIX facility. The tutorial emphasizes 4.2/4.3BSD networks, yet includes issues that are global to all networks.
Topics to be covered are: building the network including hardware/software installation; global management schemes including source code management, login management, resource management; distributed tools to make these chores easier; security; accounting; heterogeneous hardware (IBMs and others); other protocols (non TCP/IP).

Evi Nemeth is on the Computer Science faculty at the University of Colorado and has led the growth of the University's Engineering Research Computing Facility from a single VAX 11/780 to its present complement of 20 machines that include six different manufacturers' hardware.

Andy Rudoff is a Computer Science student who has been involved with the systems work concerning this network's growth (hoeing, harvesting, weeding, etc.) for the past four years.

Tutorial # 13: Introduction to UNIX Systems Administration
Instructor: Ed Gould  
mt Xinu

The basics of administering a UNIX system will be covered. The tutorial will be oriented mainly towards Berkeley VAX UNIX systems, but the principles, and some of the examples, will apply to System V, 2.9BSD, and other systems. Topics covered will include system startup and shutdown, resource management, performance and tuning, the UNIX file system, and security, as well as others. The tutorial is designed for system administrators, not for systems programmers. A rudimentary knowledge of UNIX is assumed.

Ed Gould has been working with UNIX since 1976. At the Computer Center at the University of California in Berkeley he was involved with the management and administration of several systems that were used for general purpose timesharing for the campus. In 1983, along with Vance Vaughan and Bob Kridle, he founded mt Xinu, a company dedicated to the support and enhancement of technically advanced UNIX systems.

Tutorial # 14: Writing Portable C Programs
Instructor: Tom Plum  
Plum Hall, Inc.

Today, the C programming language is widely used to implement portable applications programs. But there are many pitfalls for the unwary, some obvious, but some very subtle. If you are not aware of the issues, it is easy to write programs that will not operate correctly on another hardware architecture, or another UNIX version, or another version of the C compiler. It then becomes expensive to move the application to a new machine. This course will teach you to recognize the trouble spots and avoid these pitfalls. You will learn to write truly machine – and system – independent code, and to protect yourself when this is not possible. This course is intended for experienced C application developers. If you are involved in the development of software which is to be used or distributed on a variety of systems, you should take this course.

Tom Plum is chairman of Plum Hall, Inc., a publishing and training firm specializing in the C language. He is the author of two textbooks on C. He is also vice-chair of the ANSI X3J11 C Language Standards Committee.
Future Meetings

USENIX Meeting – January 15-17, 1986, Denver, Colorado

The emphasis will be on a series of intensive workshop seminars and tutorials. Please see the “USENIX Winter ’86 Conference” article in this issue of ;login: for more information.

To avoid conflicting with the 1986 UniForum in Anaheim, there will be no formal vendor exhibition.

UniForum – February 4-7, 1986, Anaheim, California

The 1986 UniForum trade show and conference will be held in Anaheim on February 4-7, 1986. One day will be devoted to 15 day-long tutorials on various topics for UNIX users, programmers, managers, and sales and marketing people. Subsequent days will have general panel sessions oriented towards marketers and managers, and technical sessions with paper presentations. The exhibitor trade show will run for three days. For further information, please contact the /usr/group office at 408-986-8840.

In the last issue of ;login:, USENIX members were provided a $25 discount coupon good towards the registration fee. If you have misplaced your coupon, please contact the USENIX office for a replacement.

AUUG Meeting – February 10-11, 1986, Perth, Australia

The 1986 Summer Meeting of the Australian UNIX systems Users' Group will be held on the 10th and 11th of February at the University of Western Australia, Perth, Western Australia.

Kirk McKusick, from the University of California at Berkeley, will present the keynote address on the “History and Future of Berkeley UNIX.” Papers on subjects related to the UNIX system and UNIX-like systems will be presented. There will also be a display of hardware and software.

The registration fee is $50, with a $10 discount for AUUG members and a further $10 discount if the registration payment is received by January 17.

For further information, please contact:

Registration & general:
Glenn Huxtable
glenn@wacsvax.oz (09) 380 2878

Program:
Chris McDonald
chris@wacsvax.oz (09) 380 2878

Display:
Frank O'Connor
frank@wacsvax.oz (09) 380 2639

Accommodation:
Damian Meyer
damian@wacsvax.oz (09) 380 2282

Mail format:
ACSNet: name@wacsvax.oz
CSNET: name@wacsvax.oz
UUCP: seismo!munnari!wacsvax.oz!name
ARPA: name%wacsvax.oz@seismo.css.gov

or, by mail,

Glenn Huxtable
Department of Computer Science
University of Western Australia
Nedlands, Western Australia, 6009 AUSTRALIA
EUUG Meeting – April 21-24, 1986, Florence, Italy

The next EUUG Conference and Exhibition will be held in Florence, Italy on April 21-24. It will have technical conferences, tutorials, industrial sessions, and an exhibition.

The technical conference will run for three days, from April 22-24. The main theme will be real applications of the UNIX system. This is a direct follow on from the X/OPEN group announcement made at the Copenhagen Conference. Notwithstanding this, papers on all subjects relating to the UNIX system will be presented.

Two days of tutorials will be presented: one addressing advanced features of UNIX, and the second day on introductory material.

There will be three days of industrial sessions, beginning on April 21. These talks are intended to be commercial and not necessarily technical in content.

The EUUG has decided to run two conferences per year, one of which will host the major European UNIX exhibition. At the Florence conference, there will be a three-day exhibition opening on April 22.

If you would like to receive booking information when it is available, please contact the EUUG immediately at the address below. Pre-conference bookings will be offered at a discount.

Mrs. Helen Gibbons +44 763 73039
EUUG mcvax!ukclinset!euug
Owles Hall
Buntingford, Herts. SG9 9PL, United Kingdom

The Programme Chair is:

Nigel Martin +44 1 482 2525
The Instruction Set mcvax!ukclinset!nigel
152–156 Kentish Town Road
London, NW1 9QB, United Kingdom

USENIX Meeting – June 10-13, 1986, Atlanta, Georgia

The USENIX Summer ’86 Conference will be held in Atlanta on June 10-13, 1986. There will be a conference, tutorials, and vendor exhibits.

USENIX Meeting – June 9-12, 1987, Phoenix, Arizona

The USENIX Summer ’87 Conference will be held in Phoenix on June 9-12, 1987. There will be a conference, tutorials, and vendor exhibits.

Third Printing of USENIX 4.2BSD Manuals

Since inventories of the second USENIX-sponsored printing of 4.2BSD manuals were “sold out” as of July 19, 1985, and since the office has continued to receive orders since that date, USENIX decided to make a third manual printing. Manual shipments began in late October.

The third printing is much the same as the previous two. Prices remain unchanged, and procedures for ordering the manuals remain as before. A Manual Authorization and Order Form is contained near the end of this newsletter. It is important to fill out this form carefully, include a check or valid purchase order, and have an authorized person sign the form. As before, only current USENIX Corporate, Institutional and Supporting members can order manuals. Failure to follow the procedures detailed on the order form has been the cause of delays in many members’ previous orders, so be sure to include all the information asked for.
Local User Groups

The USENIX Association will support local user groups in the United States and Canada in the following ways:

- Assisting the formation of a local user group by doing an initial mailing for the group. This mailing may consist of a list supplied by the group, or may be derived from the USENIX membership list for the geographical area involved. At least one member of the organizing group must be a current member of the USENIX Association. Membership in the group must be open to the public.

- Publishing information on local user groups in :login: giving the name, address, phone number, net address, time and location of meetings, etc. Announcements of special events are welcome; send them to the editor at the USENIX office.

Please contact the USENIX office if you need assistance in either of the above matters. Our current list of local groups follows.

In the Boulder, Colorado area a group meets about every two months at different sites for informal discussions.
Front Range Users Group
N.B.I., Inc.
P.O. Box 9001
Boulder, CO 80301
Steve Gaede (303) 444-5710
haolbisresgaede

Dallas/Fort Worth UNIX User’s Group
Advanced Computer Seminars
2915 L.B.J. Freeway, Suite 161
Dallas, TX 75234
Irv Wardlow (214) 484-UNIX

In the Washington, D.C., area there is an umbrella organization called Capitol Shell. It consists of commercial, government, educational, and individual UNIX enthusiasts. For information and a newsletter write:
Capitol Shell
8375 Leesburg Pike, #277
Vienna, Virginia 22180
seismolcap-shell

In the New York City area there is a non-profit organization for users and vendors of products and services for UNIX systems.
Unigrooup of New York
G.P.O. Box 1931
New York, NY 10116

In Minnesota a group meets on the first Wednesday of each month. For information contact:
UNIX Users of Minnesota
Carolyn Downey (612) 934-1199

In the Atlanta area there is a group for people with interest in UNIX or UNIX-like systems:
Atlanta UNIX Users Group
P.O. Box 12241
Atlanta, GA 30355-2241
Marc Merlin (404) 255-2848
Mark Landry (404) 874 6037

In the Seattle area there is a group with over 150 members, a monthly newsletter and meetings the fourth Tuesday of each month.
Seattle/UNIX Group
P.O. 58852
Seattle, WA 98188
Irene Pasternack (206) FOR-UNIX
uw-beaver!tikallssc!slug

An informal group is starting in the St. Louis area:
St. Louis UNIX Users Group
Plus Five Computer Services
765 Westwood, 10A
Clayton, MO 63105
Eric Kiebler (314) 725-9492
ihnp4lplus5@sluug
In the northern New England area is a group that meets monthly at different sites. Contact one of the following for information:

Emily Bryant  (603) 646-2999
Kiewit Computation Center
Dartmouth College
Hanover, NH  03755
dcvx!dartvax!emilyb

David Marston  (603) 883-3556
Daniel Webster College
University Drive
Nashua, NH  03063

A UNIX/C language users group has been formed in Tulsa. For current information on meetings, etc. contact:

Pete Rourke
$USR
7340 East 25th Place
Tulsa, OK  74129

Publications Available

The following publications are available from the Association Office or the source indicated. Prices and overseas postage charges are per copy. California residents please add applicable sales tax. Payments must be enclosed with the order and must be in US dollars payable on a US bank.

USENIX Conference Proceedings

<table>
<thead>
<tr>
<th>Meeting</th>
<th>Location</th>
<th>Date</th>
<th>Price</th>
<th>Overseas Mail</th>
<th>Source</th>
</tr>
</thead>
<tbody>
<tr>
<td>USENIX</td>
<td>Portland</td>
<td>Summer '85</td>
<td>$25</td>
<td>$25</td>
<td>$5</td>
</tr>
<tr>
<td>USENIX</td>
<td>Dallas</td>
<td>Winter '85</td>
<td>$20</td>
<td>$25</td>
<td>$5</td>
</tr>
<tr>
<td>USENIX</td>
<td>Salt Lake</td>
<td>Summer '84</td>
<td>$25</td>
<td>$25</td>
<td>$5</td>
</tr>
<tr>
<td>UniForum</td>
<td>Wash. DC</td>
<td>Winter '84</td>
<td>$30</td>
<td>$20</td>
<td>/usr/group</td>
</tr>
<tr>
<td>UNICOM</td>
<td>San Diego</td>
<td>Winter '83</td>
<td>$15</td>
<td>$25</td>
<td>$5</td>
</tr>
</tbody>
</table>

USENIX Association
P.O. Box 7
El Cerrito, CA  94530

/usr/group
4655 Old Ironsides Dr., #200
Santa Clara, CA  95050

EUUG Publications

The following EUUG publications may be ordered from the USENIX Association office.

The EUUG Newsletter, which is published four times a year, is available for $4 per copy or $16 for a full-year subscription. The earliest issue available is Volume 3, Number 4 (Winter 1983).

The July 1983 edition of the EUUG Micros Catalog is available for $8 per copy.
Ease: A Configuration Language for Sendmail

James S. Schoner
Purdue University Computing Center
West Lafayette, Indiana 47907

ABSTRACT

The rapid expansion of computer networks and ensuing variation among mailing address formats have made address interpretation an increasingly complex task. In the UNIX* 4.2BSD operating system, a program named sendmail was introduced which provided a general internetwork mail routing facility. This facility has significantly diminished the complexity of handling address interpretation.

sendmail's address interpretation is based on a rewriting system composed of a number of rewriting rules (or productions) arranged as part of a configuration file. Unfortunately, the syntactical format of a configuration file for sendmail is both terse and rigid, making it rather difficult to modify. The standard format certainly serves its purpose, but, as the need to change these configurations increases in frequency, a more readable format (i.e., one that is similar to the format of modern programming languages) is required to permit reasonably quick modifications to the configuration. As a solution to this problem, Ease provides a level of abstraction which eliminates most of the current syntactic hindrances faced by programmers who must reconfigure sendmail's address parsing scheme.

As a high-level specification format, Ease is proving to be an excellent alternative to sendmail's cryptic configuration file syntax. The syntactic structures of Ease are patterned after modern language constructs, making the language easy to learn and easy to remember. The format of the address rewriting rule is perhaps the most significant syntactical improvement. It was undoubtedly the most needed improvement. Nevertheless, every element of a configuration file is structurally enhanced through the use of Ease.

Introduction

The Ease language is a high-level specification format for sendmail's configuration file. The motivation for its development was to fulfill a goal of providing a readable and easily modifiable sendmail configuration file format. Ease fulfills this goal by shielding the programmer from the cryptic configuration specification required by sendmail and providing a high-level language with which the programmer may specify all modifications to a configuration file. The development of Ease coincided with the development of an Ease translator, et, which translates a configuration file written in Ease to an equivalent file of the standard format accepted by sendmail.

* UNIX is a trademark of AT&T Bell Laboratories.

This article is © 1985 James S. Schoner – All Rights Reserved. It may be reproduced only by USENIX Association members for personal or in-house, non-commercial use. It may not be reproduced or distributed in any form for any other use without permission of the author.
Ease in Profile

As will be seen in the next section, the syntax of Ease is quite readable and easy to learn. In order to acquire a relevant perspective on this issue, the reader is advised to examine a raw configuration file for sendmail (the output of the Ease translator, et, will do nicely). The raw syntax, while quite fitting for quick translation, can prove to be a programmer's nightmare.

Undoubtedly, one of the more prominent features of Ease is the ability to attach names to address fields. When address field names are well-chosen, a distinct, self-documenting quality becomes a visible part of the address rewriting rules. Ostensibly, address field names provide a new level of semantic abstraction. A brief comparison of the formats can be accomplished by examining the following equivalent representations of an address pattern:

user_path@host_name  (Ease format)
$+@$-
      (raw format)

In the above, “user_path” represents a field of one or more address tokens, and “host_name” represents one address token exactly. These token fields are represented by “$+” and “$-” in the raw format. Clearly, the Ease format is preferable, not only for increased readability, but structural comprehension as well.

Other features of Ease include ruleset naming, long identifiers for macros and classes, flow-of-control structures, and free formatting. In addition, the C language preprocessor (cpp) can be used for file inclusion and conditionally defined code constructs. The next section describes the Ease language in complete detail.

Ease Syntax*

At its highest level, Ease can be viewed as a collection of block-structures, where each block begins with a keyword and is followed by zero or more related definitions and/or declarations. There are ten distinct block types. The following is a list containing all ten block keywords and the block type it denotes.

- bind - ruleset identifier bindings
- macro - macro definitions
- class - class definitions
- options - sendmail option definitions
- precedence - precedence definitions
- trusted - trusted users
- header - mail header definitions
- mailer - mailer definitions
- field - address field definitions
- ruleset - address rewriting rules

In general,

- Letters are distinguished by case,
- An Ease identifier is defined to be a letter, followed by zero or more letters, digits, underscores (_), or dashes (-),
- A literal newline or double quotation (") character may be included in any quoted string by preceding the character with a backslash (\), and
- Ease source is preprocessed by the C language preprocessor (cpp), thus source comments (i.e., text enclosed by "//" and "/*"/*)) may appear anywhere as part of Ease whitespace.

* No attempt is made to describe the complete semantic meaning associated with all of the constructs of a sendmail configuration file. Items not covered in this document include the semantic distinction among rulesets, the special uses of pre-defined macros, and the method of building configuration files. To obtain this information, the reader is advised to refer to the "Installation and Operation Guide for Sendmail" (UNIX Programmer's Manual, Volume 2c), by Eric Allman.
For notational convenience, this document specifies all reserved words of the Ease language in italics. In addition, quantities enclosed in angle brackets (<>..>) represent arbitrary identifiers, strings, or numbers.

### Ruleset Identifier Bindings

A ruleset (a set of rewriting rules) is identified solely by an integer in sendmail's configuration file. Ease, however, allows each ruleset to be named with a meaningful identifier. Since a special numeric association for each ruleset is required by the address parsing scheme of sendmail, a bind block must be present in any Ease file which defines one or more rulesets. A bind block consists of the keyword bind, followed by zero or more statements of the form:

```plaintext
<ruleset-id> = ruleset <ruleset-number> ;
```

The following example,

```plaintext
bind

FINAL_RW = ruleset 4;
```

specifies that "FINAL_RW," the final rewriting ruleset, is sendmail's ruleset number 4.

### Macro Definitions

A macro is an identifier which, when referenced in the text of a program, is replaced by its value, a string of zero or more characters. The value of a macro may include references to other macros, but not itself. sendmail allows a maximum of 26 user-declared macros in its configuration file. In addition, there are a number of pre-declared macros which have special meaning to sendmail (see Appendix A). Ease macros are defined in macro blocks. Ease allows any macro to be declared (which is equivalent to simply referencing it) before it is defined. A macro identifier is replaced by its value when it is preceded by the character '$'. In addition, a macro reference inside a quoted string must always include braces ((...) around the macro identifier (for delimiting purposes).

A macro block consists of the keyword macro, followed by zero or more statements taking either of the following forms:

```plaintext
<macro-identifier> = "<macro-value>" ;
<macro-identifier> = conditional-expression ;
```

The conditional-expression format will be discussed later.

The following example,

```plaintext
macro

first_name = "James";
last_name = "Schoner";
whole_name = "$(first_name) $(second_name)";
```

defines the macros "first_name," "last_name," and "whole_name," where "whole_name" is the string, "James Schoner."

### Class Definitions

A class is denoted by an identifier representing a logical grouping of zero or more names. Classes are used to represent the range of values a token may assume in the pattern matching of an address. Further discussion on the use of classes will be deferred until address fields are described.

One identifier may be used to distinctly represent both a macro and class (i.e., the set of macro identifiers and the set of class identifiers may form a non-empty intersection). A name, or class element, may be an identifier or any quoted word.

A class block consists of the keyword class, followed by zero or more statements taking any of the following forms:
; login:

```plaintext
<class-identifier> = { <name1>, <name2>, <name3>, ... } ;
<class-identifier> = readclass ( "<file-name>" ) ;
<class-identifier> = readclass ( "<file-name">", "<read-format>" ) ;
```

The second and third forms cause `sendmail` to read the names of the class from the named file. The third form contains a read format, which should be a `scanf(3)` pattern yielding a single string.

The following example,

```plaintext
class
campus_hosts = { statistics, engineering, chemistry, physics, physics-2 } ;
versions = { "1.0", "1.1", "4.0", "4.2", latest-and-greatest } ;
phone_hosts = readclass ( "/tmp/phonenet.list" ) ;
```

defines the classes “campus_hosts,” “versions,” and “phone_hosts.”

Sendmail Option Definitions

A number of options to the `sendmail` program may be specified in an `options` block. For a description of the various `sendmail` options and their values, see Appendix B.

An `options` block consists of the keyword `options`, followed by zero or more statements taking any of the following forms:

```plaintext
<option-identifier> = "<option-value>" ;
'o_delivery' = special-value ;
'o_handling' = special-value ;
```

All but two options (`o_delivery` and `o_handling`) use the first form. To specify an option without a value, simply assign to it the null string (""). The `special-value` field of the second and third form refers to special values (non-quoted) which are specified in Appendix B.

The following example,

```plaintext
options
  o_alias = "/usr/lib/alias" ;
  o_tmode = "0600" ;
  o_delivery = d_background ;
```

sets the options `o_alias`, `o_tmode`, and `o_delivery`.

Precedence Definitions

Message headers may contain a “Precedence:” field describing the precedence of the message class. Identifiers which may appear in the precedence field of a message are given precedence values in a configuration file `precedence` definition. This association will be illustrated below in an example.

A `precedence` block consists of the keyword `precedence`, followed by zero or more statements of the form:

```plaintext
<precedence-identifier> = <precedence-integer> ;
```

The following example,

```plaintext
precedence
  special-delivery = 100;
  junk = -100;
```

defines the precedence level for the names “special-delivery” and “junk.” Thus, whenever the name “junk” appears in a “Precedence:” field, the corresponding message class will be set to -100.
;login:

Trusted Users

/sendmail\'s -f flag allows trusted users to override the sender's machine address. Trusted users are listed in trusted blocks. A trusted block consists of the keyword trusted, followed by zero or more sets of users taking the form:

\{ <user1>, <user2>, ... \}

The following example,

trusted
\{ root, uucp, network \}
\{ acu, kcs, jss \}

specifies that the users root, uucp, network, acu, kcs, and jss can be trusted to use /sendmail\'s -f flag.

Mail Header Definitions

The format of the message headers inserted by /sendmail\ is defined in one or more header blocks in the configuration file. A header block consists of the keyword header, followed by zero or more statements taking any of the following forms:

\[
\begin{align*}
\text{for ( <mailer-flag1>, <mailer-flag2>, ... )} & \quad \text{\textbf{define} ( "<header-title>" , header-value ) ;} \\
\text{for ( <mailer-flag1>, <mailer-flag2>, ... )} & \quad \text{\textbf{define} ( "<header-title>" , header-value ) ;} \\
& \quad \text{\textbf{define} ( "<header-title>" , header-value ) ;} \\
& \quad \ldots \\
& \quad \text{\textbf{define} ( "<header-title>" , header-value ) ;}
\end{align*}
\]

The first form is used to define one header for one or more mailer flags. The second form differs from the first in that more than one header may be defined for a given set of flags. The third form is used to define a header, regardless of mailer flags. Refer to Appendix C for a list of \textit{Ease} identifiers representing mailer flags. The header title is a simple string of characters (no macro references), whereas the header-value can be either a string of characters (possibly containing macro references) or a \textit{conditional-expression} (discussed later).

The following example,

header

\textbf{define} ( "Subject:", "" ) ;
\textbf{for} ( f\_return )
\textbf{define} ( "Return-Path:", "\$(m\_sreladdr)" ) ;
\textbf{for} ( f\_date )
\textbf{define} ( "Resent-Date:", "\$(m\_odate)" ) ;
\textbf{define} ( "Date:", "\$(m\_odate)" ) ;

defines a "Subject" field for all mailers, regardless of their flags, a "Return-Path" field for mailers whose definition specifies the flag \textit{f\_return}, and the headers "Resent-Date" and "Date" for mailers whose definition specifies the flag \textit{f\_date}. 
;login:

Mailer Definitions

`sendmail`'s definition of a mailer (or an interface to one) occurs in a mailer block. A mailer block consists of the keyword mailer, followed by zero or more statements of the form:

```
<mailer-identifier> { mailer-spec } ;
```

The field, mailer-spec, is a list of zero or more of the following attribute assignments (where successive assignment statements are separated by commas):

- `Path` = `string-attribute`
- `Argv` = `string-attribute`
- `Eol` = `string-attribute`
- `Maxsize` = `string-attribute`
- `Flags` = `{ <mailer-flag1>, <mailer-flag2>, ... }
- `Sender` = `<sender-ruleset-id>
- `Recipient` = `<recipient-ruleset-id>

The string-attribute value can take the form of a quoted string (possibly containing macro references) or a conditional-expression (discussed later).

The following example,

```
mailer
  local {
    Path = "/bin/mail",
    Flags = { f_from, f_locm },
    Sender = Sender_RW,
    Recipient = Recipi_RW,
    Argv = 'mail -d $m_ruser",
    Maxsize = "200000"
  };
```

defines a mailer named "local."

Address Field Definitions

`sendmail`'s address parsing scheme treats an address as a group of tokens (an address token is precisely defined in the Arpanet protocol RFC822). In general, `sendmail` divides an address into tokens based on a list of characters assigned as a string to the special macro `m_addrops`. These characters will individually be considered as tokens and will separate tokens when parsing is performed.

For the Ease language, there is a distinct set of address tokens (defined below) which are used in combination to represent generic forms of addresses. In addition to literal address tokens, the pattern to be matched in a rewriting rule (often referred to as the LHS) may include field identifiers which match one of five possibilities:

- zero or more tokens
- one or more tokens
- one token exactly
- one token which is an element of an arbitrary class X
- one token which is not an element of an arbitrary class X

A particular field type may be assigned to one or more identifiers. Each field identifier is associated with (or defined to be) a field type in a field declarations block. A field declarations block consists of the keyword field, followed by zero or more field definitions of the form:

```
field-id-list : field-type ;
```
A field-list is a list of one or more identifiers, each separated by a comma. A field-type, on the other hand, is a representation of one of the five fields described above. The syntax for each of the five forms follows:

- match ( 0* )
- match ( 1* )
- match ( 1 )
- match ( 1 ) in <class-X>
- match ( 0 ) in <class-X>

The star in the first two forms means "or more." Thus, the first form would read: "match zero or more tokens." The fourth form describes a field where one token is matched from an arbitrary class (class-X), whereas the fifth form describes a field where one token is matched if it is not of the given class (class-X).

The following example,

```plaintext
field
  anypath: match ( 0* );
  recipient_host: match ( 1 );
  local_site: match ( 1 ) in m_sitename;
  remote_site: match ( 0 ) in m_sitename;
```

de\'annes the fields "anypath," "recipient_host," "local_site," and "remote_site."

Address Rewriting Rules

Address rewriting rules are grouped according to the function they perform. For example, it is desirable to form a distinct group for those rewriting rules which perform transformations on recipient addresses.

Sets of rewriting rules are defined in ruleset blocks. A ruleset block consists of the keyword ruleset, followed by zero or more ruleset definitions of the form:

```
<ruleset-id> { <rewriting-rule1> <rewriting-rule2> ... }
```

The ruleset identifier, ruleset-id, must be defined in a bind block, as described earlier. The rewriting rules have the form:

```
if( <match-pattern> )
  <match-action> ( <rewriting-pattern> ) ;
```

where "match-pattern," "rewriting-pattern," and "match-action" are described below.

Match-patterns

A match-pattern is a sequence of Ease address elements representing an address format. If the address being rewritten matches the pattern "match-pattern," then the address is reformatted using the pattern "rewriting-pattern," and the corresponding action ("match-action") is performed. The five distinct Ease address elements which may constitute a match-pattern are as follows:

1. Field Identifiers (refer to previous section)
2. Non-alphanumeric characters (the exception is the case for literal double quotes, which must be preceded by a backslash (\))
3. Macro references
4. Quoted strings ("...")
5. Conditional-expressions (discussed later)
;login:

Below are two sample match-patterns, each describing the same address format:

user-id @ hostname . $arpa_suffix
user-id @ hostname ":ARPA"

where user-id and hostname are field identifiers, and arpa_suffix is a user-defined macro with the value "ARPA."

Rewriting-patterns

A rewriting-pattern specifies the form in which to rewrite a matched address. The seven distinct elements which may be used to form a rewriting-pattern are as follows:

1. Non-alphanumeric characters (the exception is the case for literal double quotes, left parentheses, or right parentheses, each of which must be preceded by a backslash (\)).
2. A call to another ruleset. This is used to perform rewrites on a suffix of the rewriting-pattern. The proper use of this feature will be demonstrated by example below.
3. Quoted strings ("...").
5. A macro reference.
6. A positional reference in the matched address. A positional reference takes the form: \$<integer-position>. For example, \$3 references the value of the third Ease address element in the matched address.
7. Canonicalized host names of the form canon (<id-token>), where "id-token" is a regular identifier, a quoted identifier (with double quotes), a macro reference yielding an identifier, or a positional reference in the matched address. The canonicalization of a host name is simply a mapping to its canonical (or official) form.

Below are two sample rewriting-patterns:

\$1 % $2 @ $3 $.ARPA >
OLDSTYLE_RW ( $1 )

The first form specifies an address such as "a%b<@c.ARPA>,” where 'a', 'b', and 'c' represent matched identifiers or paths. The second form specifies a call to the ruleset "OLDSTYLE_RW,” for old-style rewriting on the parameter $1, which probably references the entire matched address. This will become clear in later examples.

Match-actions

When a ruleset is called, the address to be rewritten is compared (or matched) sequentially against the match-address of each rewriting rule. When a match-address describes the address sendmail is attempting to rewrite, the address is rewritten (or reformatted) using the rule’s rewriting-pattern. Following this rewrite, the corresponding match-action is performed. There are four match-actions:

retry a standard action which causes the rewritten address to be again compared to the match-address of the current rule.

next an action which causes the rewritten address to be compared to the match-address of the next rewriting rule of the current ruleset. If the end of the list is reached, the ruleset returns the rewritten address.

return an action which causes an immediate return of the ruleset with the current rewritten address.

resolve an action which specifies that the address has been completely resolved (i.e., no further rewriting is necessary). The resolve action is described in more detail below.
The match-action, `resolve`, is special in that it terminates the address rewriting altogether. The semantic structure of `sendmail`'s rewriting scheme requires that a `resolve` action appear only in the ruleset whose numerical binding is to the number zero. The `resolve` action must specify three parameters: `mailer`, `host`, and `user`. If the `mailer` is local, the `host` parameter may be omitted. Both `mailer` and `host` must be specified as a single word or an expression which expands to a single word (i.e., a macro reference or a canonicalization). The `user` specification is a rewriting-pattern, as described above.

In general, the format of a `resolve` action will be as follows:

```
resolve ( mailer ( <mailer-name> ),
        host  ( <host-name> ),
        user  ( <user-address> ) );
```

Examples of the match-action statement are shown below:

```
fld
    anypath   : match (0*);
    usr, path : match (1*);
    hostname  : match (1);
    phone_host: match (1) in phonehosts;
```

```
ruleset EXAMPLE_RW {
    if ( anypath < path > anypath ) /* basic RFC821/822 parse */
        retry ( $2 );
    if ( usr " at " path )        /* "at" => "@" */
        next ( $1 @ $2 );
    if ( @path: usr )
        return ( LOCAL_RW ( < @$1 > : $2 ) );
    if ( anypath < @phone_host'ARPA' > anypath )
        resolve ( mailer ( tcp ),
                  host ( csnet-relay ),
                  user ( $1 % $2 < @'csnet-relay' > $3 ) );
}
```

The example above defines the ruleset “EXAMPLE_RW,” which contains four rewriting rules. The first rewriting rule discards all tokens of an address which lie on either side of a pair of angle brackets (< >), thereby rewriting the address as the sequence of tokens contained within the angle brackets ($2$). Following the address rewrite, the rule is applied again (`retry`). When the first rule fails to match the address being rewritten, the second rule is applied.

The second rule simply replaces the word “at” by the symbol ‘@’. The “next” action specifies that if a match is made, a rewrite is performed and matching continues at the next (or following) rule.

The third rule illustrates the use of the “return” action, which is executed if the pattern “@path: usr” describes the current format of the address being rewritten. In this example, the `return` action returns the result of a call to ruleset “LOCAL_RW,” which rewrites the address “<@$1>:@$2,” where $1$ and $2$ are substituted with the token(s) matched respectively by “path” and “usr.”

The fourth (and final) rule signals a resolution (and termination) of the rewriting process if the given pattern is matched. The resolution specifies that the `mailer` “tcp” will be used to deliver the message to the host “csnet-relay.” The `user` parameter specifies the final form of the address which `sendmail` has just resolved.
The **Ease** construct which remains to be examined is the **conditional-expression**. The **conditional-expression** provides a method for constructing strings based on the condition that some test macro is (or is not) set. The general form begins with the concatenation of a string and a **string-conditional**:

```plaintext
concat ( <quoted-string>, string-conditional )
concat ( string-conditional, <quoted-string> )
```

A **string-conditional** assumes either of the following forms:

```plaintext
ifset ( <macro-name>, <ifset-string> )
ifset ( <macro-name>, <ifset-string>, <notset-string> )
```

A **string-conditional** of the first form evaluates to "ifset-string" if the macro "macro-name" has been assigned a value; otherwise it evaluates to the null string. The second form behaves similarly, except that the **string-conditional** evaluates to "notset-string," instead of the null string, if the macro "macro-name" has no value.

The following **conditional-expression**,

```plaintext
concat ( "New", ifset ( city, "York", "Jersey" ) )
```
evaluates to the string "New York," if the macro "city" is set. Otherwise, the **conditional-expression** evaluates to the string "New Jersey."

**Ease Translation**

It is important to note that **Ease** is translated by a stand-alone translator to the raw configuration file format. No modifications were made to the **sendmail** program itself. As a result, syntactical verification of a configuration file can be performed without invoking **sendmail**.

The **Ease** language is translated by invoking the **C** language preprocessor (**cpp**) with **Ease** source as input, then piping the output as input to the **Ease** translator (**et**). The **Ease** translator may be invoked on the command line in one of four ways:

```plaintext
et <input-file> <output-file>  [read from a file, write to a file]
et <input-file>             [read from a file, write to standard output]
et - <output-file>         [read from standard input, write to a file]
et                     [read from standard input, write to standard output]
```

**Conclusion**

**Ease** is currently in use at the Purdue University Computing Center. Source code for the **Ease** translator (**et**) may be obtained on request by writing to:

**U.S. Mail:**
James S. Schoner  
Mathematical Sciences Building, Office 204  
Purdue University  
West Lafayette, Indiana 47907

**Electronic Mail:** jss@purdue-asc.ARPA

It may also be obtained in a **tar** file via anonymous FTP to **purdue-asc** (asc.purdue.edu). Use the login "anonymous" with any non-null password. You will need to get the file called **ease.tar** in the **pub** directory.

Much of the success of this project is attributable to the constant support and insight offered by Mark Shoemaker. To him, I owe a debt of gratitude. In addition, I would like to thank Kevin Smallwood and Paul Albitz for their many notable suggestions and valuable insight.
### Appendix A – Pre-Declared Macros

<table>
<thead>
<tr>
<th>Ease Macro</th>
<th>Raw Equivalent</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>m_odate</td>
<td>a</td>
<td>The origination date in Arpanet format</td>
</tr>
<tr>
<td>m_adate</td>
<td>b</td>
<td>The current date in Arpanet format</td>
</tr>
<tr>
<td>m_hops</td>
<td>c</td>
<td>The hop count</td>
</tr>
<tr>
<td>m_udate</td>
<td>d</td>
<td>The date in UNIX (ctime) format</td>
</tr>
<tr>
<td>m_smtp</td>
<td>e</td>
<td>The SMTP entry message</td>
</tr>
<tr>
<td>m_saddr</td>
<td>f</td>
<td>The sender (from) address</td>
</tr>
<tr>
<td>m_sreladdr</td>
<td>g</td>
<td>The sender address relative to the recipient</td>
</tr>
<tr>
<td>m_rhost</td>
<td>h</td>
<td>The recipient host</td>
</tr>
<tr>
<td>m_qid</td>
<td>i</td>
<td>The queue id</td>
</tr>
<tr>
<td>m_oname</td>
<td>j</td>
<td>The official domain name for this site</td>
</tr>
<tr>
<td>m_ufrom</td>
<td>l</td>
<td>The format of the UNIX from line</td>
</tr>
<tr>
<td>m_daemon</td>
<td>n</td>
<td>The name of the daemon (for error messages)</td>
</tr>
<tr>
<td>m_addrops</td>
<td>o</td>
<td>The set of “operators” in addresses</td>
</tr>
<tr>
<td>m_pid</td>
<td>p</td>
<td>sendmail’s pid</td>
</tr>
<tr>
<td>m_defaddr</td>
<td>q</td>
<td>The default format of sender address</td>
</tr>
<tr>
<td>m_protocol</td>
<td>r</td>
<td>Protocol used</td>
</tr>
<tr>
<td>m_hostname</td>
<td>s</td>
<td>Sender’s host name</td>
</tr>
<tr>
<td>m_ctime</td>
<td>t</td>
<td>A numeric representation of the current time</td>
</tr>
<tr>
<td>m_ruser</td>
<td>u</td>
<td>The recipient user</td>
</tr>
<tr>
<td>m_version</td>
<td>v</td>
<td>The version number of sendmail</td>
</tr>
<tr>
<td>m_sitename</td>
<td>w</td>
<td>The hostname of this site</td>
</tr>
<tr>
<td>m_sname</td>
<td>x</td>
<td>The full name of the sender</td>
</tr>
<tr>
<td>m_stty</td>
<td>y</td>
<td>The id of the sender’s tty</td>
</tr>
<tr>
<td>m_rhdir</td>
<td>z</td>
<td>The home directory of the recipient</td>
</tr>
</tbody>
</table>

;login:

Appendix B – Sendmail Options


<table>
<thead>
<tr>
<th>Sendmail Option (Ease)</th>
<th>Sendmail Option (Raw)</th>
</tr>
</thead>
<tbody>
<tr>
<td>o_alias</td>
<td>A</td>
</tr>
<tr>
<td>o_ewait</td>
<td>a</td>
</tr>
<tr>
<td>o_bsub</td>
<td>B</td>
</tr>
<tr>
<td>o_qwait</td>
<td>c</td>
</tr>
<tr>
<td>o_delivery (special values below)</td>
<td>d</td>
</tr>
<tr>
<td>d_interactive</td>
<td>i</td>
</tr>
<tr>
<td>d_background</td>
<td>b</td>
</tr>
<tr>
<td>d_queue</td>
<td>q</td>
</tr>
<tr>
<td>o_rebuild</td>
<td>D</td>
</tr>
<tr>
<td>o_handling (special values below)</td>
<td>e</td>
</tr>
<tr>
<td>h_print</td>
<td>p</td>
</tr>
<tr>
<td>h_exit</td>
<td>q</td>
</tr>
<tr>
<td>h_mail</td>
<td>m</td>
</tr>
<tr>
<td>h_write</td>
<td>w</td>
</tr>
<tr>
<td>h_mailz</td>
<td>e</td>
</tr>
<tr>
<td>o_tmode</td>
<td>F</td>
</tr>
<tr>
<td>o_usave</td>
<td>f</td>
</tr>
<tr>
<td>o_gid</td>
<td>g</td>
</tr>
<tr>
<td>o_fsmtp</td>
<td>H</td>
</tr>
<tr>
<td>o_skipd</td>
<td>i</td>
</tr>
<tr>
<td>o_slog</td>
<td>L</td>
</tr>
<tr>
<td>o_rsend</td>
<td>m</td>
</tr>
<tr>
<td>o_dnet</td>
<td>N</td>
</tr>
<tr>
<td>o_hformat</td>
<td>o</td>
</tr>
<tr>
<td>o_qdir</td>
<td>Q</td>
</tr>
<tr>
<td>o_tread</td>
<td>r</td>
</tr>
<tr>
<td>o_flog</td>
<td>S</td>
</tr>
<tr>
<td>o_safe</td>
<td>T</td>
</tr>
<tr>
<td>o_timeout</td>
<td>t</td>
</tr>
<tr>
<td>o_timezone</td>
<td>u</td>
</tr>
<tr>
<td>o_dmuid</td>
<td>v</td>
</tr>
<tr>
<td>o_verbosity</td>
<td>W</td>
</tr>
<tr>
<td>o_wizpass</td>
<td>x</td>
</tr>
<tr>
<td>o_loadq</td>
<td>X</td>
</tr>
<tr>
<td>o_loadnc</td>
<td></td>
</tr>
</tbody>
</table>
Appendix C – Mailer Flags


<table>
<thead>
<tr>
<th>Mailer Flag (Ease)</th>
<th>Mailer Flag (Raw)</th>
</tr>
</thead>
<tbody>
<tr>
<td>_f_from</td>
<td>f</td>
</tr>
<tr>
<td>_f_tfrom</td>
<td>t</td>
</tr>
<tr>
<td>_f_noreset</td>
<td>S</td>
</tr>
<tr>
<td>_f_noufrom</td>
<td>n</td>
</tr>
<tr>
<td>_f_locm</td>
<td>l</td>
</tr>
<tr>
<td>_f_strip</td>
<td>s</td>
</tr>
<tr>
<td>_f_mult</td>
<td>m</td>
</tr>
<tr>
<td>_f_from</td>
<td>F</td>
</tr>
<tr>
<td>_f_date</td>
<td>D</td>
</tr>
<tr>
<td>_f_msg</td>
<td>M</td>
</tr>
<tr>
<td>_f_full</td>
<td>X</td>
</tr>
<tr>
<td>_f_return</td>
<td>P</td>
</tr>
<tr>
<td>_f_uupperu</td>
<td>U</td>
</tr>
<tr>
<td>_f_uupperh</td>
<td>H</td>
</tr>
<tr>
<td>_f_arpa</td>
<td>A</td>
</tr>
<tr>
<td>_f_ufrom</td>
<td>E</td>
</tr>
<tr>
<td>_f_expensive</td>
<td>X</td>
</tr>
<tr>
<td>_f_dot</td>
<td>L</td>
</tr>
<tr>
<td>_f_limit</td>
<td>p</td>
</tr>
<tr>
<td>_f_heitsmtp</td>
<td>I</td>
</tr>
<tr>
<td>_f_addrw</td>
<td>C</td>
</tr>
</tbody>
</table>
Load Balancing With Maitre d'

Brian Bershad
Computer Science Division
Department of Electrical Engineering and Computer Science
University of California, Berkeley
Berkeley, California 94720

December 9, 1985

ABSTRACT
As the number of machines in a computer installation increases, the likelihood that they are all being equally used is very small. We have implemented a load-balancing system to increase the overall utilization and throughput of a network of computers. With this system, a busy machine will locate an underutilized one and attempt to process certain types of CPU intensive jobs there. We present here a complete functional description of the system and an analysis of its performance.

1. Introduction

In any multi-machine computing installation, there is a need to evenly distribute the workload over the participating machines. Otherwise, some machines will remain idle while others become overloaded. The Berkeley UNIX environment, with its networking capabilities, provides for the inter-connection of powerful processors. The busier machines may move tasks to the less busy machines, offering a more even distribution of workload across the entire system, and a decrease in overall command response time.

This paper describes one load-balancing system called Maitre d'\(^1\) that is currently in use in the EECS Department of the University of California at Berkeley. For a given class of relatively expensive jobs, Maitre d' will attempt to locate a lightly loaded machine and process the job at that machine. In this way, imbalances in processor demand across machines can be smoothed through an automatic redistribution of the load.

This paper is broken into several sections: background, functional description, operational considerations, implementation notes and performance analysis.

2. Background

Maitre d' was originally proposed by several students at Berkeley to relieve peak usage demands on the machines used by the Computer Science Division of EECS, particularly those dedicated to instruction. These machines had always been VAX 11/750s and 11/780s, characterized by extremely uneven workloads. Research and administrative machines alike suffered from high daytime loads, while being relatively idle at night. Instructional machines would go unused for long periods of time, but would become so loaded in the days prior to an assignment being due that they would become almost unusable.

In December 1984 the University received a gift of six VAX 11/750's from Digital Equipment Corporation[3] as part of a grant earmarked for undergraduate research and instruction. These machines were not ready for assignment to formal classes when the school semester began, but they were accessible through a 10 megabit Ethernet [1,6] connection with the rest of campus. Consequently, the new VAXes were designated as remote process servers for overloaded instructional computers being rented by the Department, with Maitre d' acting as the agent.

\(^1\) Maitre d' is French for host or server.
2.1. Functional Description

*Maître d'* operates around a modified state-broadcast algorithm. Every potential client maintains a list of known server machines. Associated with each server is a binary value representing that server’s availability, as determined and advertised by the server. When a user invokes an application modified to run under *Maître d’*, a decision is made as to whether the job should be performed remotely or on the user’s machine by comparing the UNIX five minute load average against a minimum load threshold.² If it is determined that a remote machine should be used (local load average > sendoff threshold), the list of known servers is consulted, and the least recently used available server in the list is chosen to perform the job. That prospective server is contacted, informed of its selection and then told to perform the service.

The process of selection and contact can be made entirely transparent to the user. This is especially necessary in an instructional environment, where most users are naive and unable to handle exceptional conditions (such as failure). They care only that their job gets done, and not where.

*Maître d’* can be used to off-load almost any type of non-interactive job. The initial version of *Maître d’* exported Pascal and C compiles. It has since been expanded to include typesetting (*troff*), circuit simulation (*spice*), a true Pascal compiler (*pc*), and others (see details in Appendix A). New applications are being added as need and opportunity arise.

3. How It Works

This section describes the operational details of *Maître d’*. Readers uninterested in the mechanics of the system should skip to section four.

3.1. Establishing Connections

Before considering the total operation of *Maître d’*, it is important to review some of the basics of interprocess communication. This section describes the establishment of connections and the relationship between the client and server machines. It assumes no familiarity with UNIX Interprocess Communications (*IPC*). We present a brief review of *IPC* under UNIX in the following paragraph. The reader unfamiliar with UNIX *IPC* may wish to consult either [5] or [7] for a more thorough description.

Separate processes may communicate with one another using any of several alternative methods. We describe here only the user-level protocol for a connected, bi-directional stream communication capable of crossing machine boundaries. The terms *client* and *server* are used to describe the parties during the connection phase. The server is generally referred to as a *daemon*, or a process which runs indefinitely, usually blocked while waiting for an event. The server daemon listens at some well-known address³ waiting for requests for service. The client calls the server via some high-speed communications medium (in this case, the Ethernet) and requests a connection. Implicit in this client’s call is the requester’s originating address. The server accepts by completing the connection with the client. Once the connection has been established, general practice is to have the server create a duplicate server process which interacts only with the initiating client, leaving the original server free to continue listening for further requests. Once the connection is established, both the client and server may read from and write to their common connection as though it were a standard UNIX file. This makes data transfer between the two processes extremely simple.

---

² The UNIX five minute load average is defined as the average number of jobs in the run queue exponentially smoothed over the past five minutes. It is a limited metric, but very cheap to obtain. For a more complete evaluation of load metrics, see [2].
³ An address is the combination of the machine’s internet address and a local port number.
3.2. System Components

Load balancing under Maitre d' comprises four distinct components:

(1) maitrd
   All machines which are to be able to off-load jobs to other processors run a maitrd daemon. This process maintains the list of all server machines, including status information as to whether or not they are currently willing to accept jobs. For the remainder of this paper, the terms maitrd and client may be used interchangeably.

(2) garcon
   This is the server daemon running on each machine which is to be a compute server. It has two functions: maintaining status connections with client machines, and accepting jobs from application programs. Garcon and server may be used interchangeably.

(3) application programs
   The maitrd and garcon components provide only a control environment through which the execution of programs may actually be distributed among many processors. The software that provides the interface between the user's requested task and Maitre d' is referred to as the application.

(4) miscellaneous
   This includes the black-box library routines used to interface with maitrd (described later), and a dynamic control program called mdc used to tune parameters while the system is running.

When a maitrd process is started, it tries to create control channels with the garcon daemons running at a pre-designated set of servers. This set is given in a startup configuration file (Appendix A) kept at the maitrd's machine. Through these control channels, the maitrd process can determine which machines are up and accepting jobs, which are up and not accepting, and which are down.

The counterpart to a maitrd process is garcon. This daemon listens at its well-known control port for requests. When a connection from a maitrd is accepted, the garcon daemon lets the maitrd know whether the garcon host is willing to accept jobs. A server declares itself ready if its UNIX five-minute load average is less than some threshold and there are fewer than a given number of active users logged in to the server machine. Both of these thresholds can be set from a configuration file. After the connection is made, garcon checks its own status every 30 seconds, informing each of its connected clients (i.e., multicasting) whenever availability changes.

The typical configuration for Maitre d' is to have a set of machines in which each runs both the maitrd and garcon daemons. Each machine in the cooperative would look for help from others whenever it became too busy, and in return would be willing to take on jobs from its peers when it would otherwise be idle. One alternative to this is to introduce dedicated servers into the system (machines running only garcon), which accept jobs, but never send them out. A second alternative involves running a clearinghouse maitrd and is discussed below.

For each client/server pair, there is an open stream connection maintained for the duration of the relationship. This imposes a limit on the number of servers that may be kept track of by a single maitrd process. Although not particularly cumbersome for a small number of machines, the total number of status connections for N clients and M servers grows as NM, and is order N^2 when all N machines are accommodating both garcon and maitrd. To slow this growth, Maitre 'd has the capability for clearinghouse clients. Instead of running a maitrd on every machine that is to offload, applications can request an available machine from a remote maitrd. This is most effective in clusters of diskless workstations. It is sufficient to have a single maitrd running on the file server handling requests from all of the file server's clients. Any reliability gained from having redundant maitrd's would be pointless, as the workstations aren't very useful when the file server is down.

---

4 Maitre d' (with a capital M and spelled correctly) is the name of the system. One of the component programs is called maitrd (with a lower case m and modified spelling).

5 In 4.2BSD, this limit was set by the operating system at 20. The newer 4.3 allows 64.
Certainty of state is the main reason why open connections are used for the control channels. A status update from a \textit{garcon} is guaranteed to arrive at each connected \textit{maitr\textsc{d}}. If an update is undeliverable, \textit{garcon} assumes the intended \textit{maitr\textsc{d}} no longer exists and removes it from its multicast list. Similarly, if a \textit{garcon} disappears, UNIX IPC enables each connected \textit{maitr\textsc{d}} daemon to recognize this \textit{immediately} and mark the associated server machine as unavailable. The \textit{maitr\textsc{d}} daemon attempts to re-establish connections to downed servers every five minutes.

3.3. Selecting A Machine

The \textit{maitr\textsc{d}/garcon} connection performs no real work. Its only purpose is to give the local \textit{maitr\textsc{d}} a pool of processors from which it may choose a server. In addition to maintaining connections with remote servers, \textit{maitr\textsc{d}} also listens in on a second, local socket for requests from an application program, which is any program that has been modified to run under \textit{Maitre d'}. These applications first connect to the local \textit{maitr\textsc{d}} process, asking for an available machine. If the local load is less than the sendoff threshold, or no remote servers are presently available, the application is told to perform the job locally. Otherwise, \textit{maitr\textsc{d}} does a round-robin traversal of its list until it finds a machine where the \textit{garcon} has advertised a willingness to accept jobs. It passes back to the application program the internet address of this server and terminates the local connection with the application.

3.4. Program Execution

In addition to listening on its control port, \textit{garcon} also listens on a data or service port. It is this address that \textit{maitr\textsc{d}} gives to the application, and it is the responsibility of the application to create the connection. Once the connection is created, the \textit{garcon} process creates a copy of itself. This copy communicates with the local application and executes the requested task on the remote machine, leaving the original \textit{garcon} free to handle further requests from other applications. All communication between the application and the server is done through this data port. Commands and data are passed from the application process to the remote machine; results and an exit status are passed back from the server to the application process.

Figure I shows a typical \textit{Maitre d'} interaction occurring in three stages. Permanent communication streams are shown in thick black; temporary ones in thin. In stage I the user invokes some application (compiler, formatter, etc.) which connects to the local \textit{maitr\textsc{d}} and requests an available server. This \textit{maitr\textsc{d}} has been maintaining status connections with many \textit{garcons} (only one shown in the picture). The client daemon passes a server address out of its \textit{maitr\textsc{d}} port (A) to the application program and terminates its connection with the application. Stage II shows the application connecting to the \textit{garcon} port (C) that was returned by the local \textit{maitr\textsc{d}} and requesting a service. If the application and request are valid (see Security), the server creates a copy of itself (called forking in UNIX). Note that the service port is passed down to the newly created dedicated child process (C'). The dashed lines leading into the \textit{garcon} port indicate that the parent stops attending to the application on the other end of the channel once the child \textit{garcon} has taken over. Stage III has the dedicated \textit{garcon} creating the process to perform the requested task and acting as a data buffer between the application program and that task. When the requested process finishes, its exit status is returned to the application at the originating machine, and the child \textit{garcon} terminates.

3.5. I/O Handling

The C-shell provides every process with three default file descriptors or data channels: standard input (\textit{stdin}), standard output (\textit{stdout}) and standard error (\textit{stderr}). \textit{Stdout} and \textit{stderr} are both output channels: \textit{stdin} is the only input channel. Many UNIX programs are capable of taking their input from \textit{stdin}, and placing their output on \textit{stdout}. Convention has error messages going to \textit{stderr}. All processes return an 8 bit status value upon termination.

\textit{Stdin} to a \textit{garcon} task is passed directly from the originating machine. But, on the return path of the service channel, \textit{Maitre d'} provides only one data stream. The extra bandwidth needed to handle \textit{stdout}, \textit{stderr} and the exit status is obtained by returning all data from the server in finite
packets, and prepending each packet with a four byte header. The first byte indicates the type of data being returned: stdout, stderr or exit status. If the header indicates an exit status, the second byte indicates how the process exited, and the third byte is the exit status. Otherwise, the final three bytes in the header give the size of the packet to be sent. The dedicated garcon process, acting as a buffer between the application and the remote task, handles the data encoding.

3.5.1. The Black Box

Although the sequence of connection, selection, and output decoding required by every application program appears complicated, there is a function called RemoteRun which does it all:

```c
RemoteRun(inFD, outFD, cmdp);
int inFD;
int outFD;
struct pipepiece *cmdp;
```

where
struct pipepiece {
    char *pp_name;
    char **pp_args;
}

and inFD and outFD are the input and output file descriptors that should be used for input to and output from the remote task. There are no provisions for redirecting stderr. Note that cmdp is a pointer to struct pipepiece. A list of these structs indicates a sequence of piped commands, allowing multiple tasks to be piped together on the remote end.

4. Operational Considerations

4.1. Error Handling

Maitre d' itself does not have any provision for handling errors other than reporting them to the application. Some common error situations are:

- lost connection with remote host
- remote garcon prematurely exited
- remote machine could not execute process

Whenever possible, an application should recover from the error and accommodate the user's task as quietly and politely as possible. This may be done either by finding another host, or by processing the job on the user's machine. All of our applications attempt to run the job locally if there is a remote failure.

Because of redirection facilities and pipes in UNIX, a difficult situation arises if the remote server is the recipient of a local process's output, and one of the errors listed above occurs. The process cannot be restarted in any way. For example:

tbl file.me | matfront nroff -me > file.out

where matfront is a generic front end that attempts to process its arguments on a remote machine using stdin as input. Once data from tbl is passed to matfront, it cannot be retrieved if the remote nroff terminates due to some error related to Maitre d'. Even if matfront kept a copy of tbl's output (which would be prohibitively expensive), output already generated by the remote nroff would have gone to file.out. Any attempt to restart the job might cause duplicate output to appear in file.out. Because Maitre d' operates at the user level, above the C-shell and UNIX kernel, no simple solution exists for this problem. Consequently, if an error occurs after a remote job has started executing, and the job is receiving its input from a pipe, the user's task terminates with an error message. Processes not using redirection do not have this problem, and can be restarted.

4.2. Security

When a machine services users on other machines, various security problems arise. Some of the concerns are:

Unauthorized Use
Administrative barriers must be respected.

Unauthorized Access
Use of spare cycles should not allow access to restricted files.

Unauthorized Execution
Not all machines should have to provide all services to all clients.
Maitre d' solves these security problems with client verification, task verification, non-privileged users, and logfiles.

When a connection is requested to either of garcon's two ports, the originating host is checked against a list of authorized hostnames contained in the startup configuration file. The request is ignored if the host is not authorized and the illegal access attempt is recorded in a log file.
;login:

If a request for service arrives, *garcon* guards against unscrupulous applications by checking to see that it has actually advertised itself as *available*. If not, the request is ignored. The request is then checked against a list of *reasonable* services that *garcon* has been told about in the configuration file. If it is not in the list, *garcon* informs the application that it doesn’t know about the service. It is then up to the application to either choose another server or process the job locally.

Associated with each process under UNIX is a user name governing that process’ access rights. When a task runs under *garcon*, the privileges are first set to those of some named user given in the configuration file. In Berkeley’s implementation, all service tasks run as *nobody*, which is an actual entry in the password file originally created for system administration functions. *Nobody* has no password, home directory or shell and can only read public files. If process accounting is being run on the server machines, it would be worthwhile to create a dummy account used only by *garcon*. In this way, the standard accounting software could determine the percentage of resources which are being used on behalf of remote requests.

Once a server has begun a remote job, and if its load has risen above the acceptance threshold, it is possible to have this job’s priority lowered or re-*niced* during the period that the server is not accepting new jobs. Active jobs from other machines will then not impinge upon jobs coming from a server machine’s own users. The priority is raised again once the server’s load falls below the threshold.

5. Implementation Notes

All programs that are to operate under the *Maitre d’* system require a front end (using *RemoteRun*) on the client machine to decode the returned data. Those applications that use *stdin*, *stdout*, and *stderr* for I/O do not need a front end on the remote machine, and may simply use *matfront* as an interface. Since *Maitre d’* supports only these three channels of data transfer, a few programs did not easily integrate into the system.

5.1. The Compilers

5.1.1. C

Creating the application to handle C compiles was relatively trivial due to the structure of the compiler, which is broken down into four distinct parts: pre-processing, compilation, assembly and loading. Pre-processing and loading are always performed on the local machine. The compilation and assembly, which comprise almost 70% of the total cycles, are done on the remote machine.

5.1.2. Pascal

A good example of a package that did not lend itself well to running under *Maitre d’* was the Berkeley Pascal interpreter (*pi*). There were problems on both the client and server end.

*Server*

Berkeley Pascal (*pi*) does not use *stdin* for anything, but instead requires that its input come from a file (or several files). The executable image produced by *pi* goes directly to a file and cannot be directed elsewhere (such as *stdout*). Things are further complicated by the fact that *pi* uses *stderr* for only one error message. All other error messages go to *stdout*. This causes problems for *garcon*, which expects remote tasks to communicate back to the application via *stdout* and *stderr*.

*Client*

Because of the way *include* files are handled in *pi*, it is semantically legal to concatenate all of the files and compile them as a single stream. Originally, we ran a very simple Pascal preprocessor over all of the user’s source, scanning for *include* directives and including them as we found them, sending over all of the files as one large program. Unfortunately, all of the

---

6 When the interpreter encounters a line of the form “*#include filename*” in the source file, it reads the named file into its input stream as though it were coded in-line by the user[4].

38 January/February 1986 Volume 11, Number 1
debugging and diagnostic errors produced by Berkeley Pascal have line numbers relative to the beginning of each source file, so this was not a satisfactory solution. Students were being told that they had a syntax error on line 1237, when their largest file contained only 250 lines.

These problems were solved by building a simple file transfer protocol on top of the three data channels already provided for by Maitre d'. This extra level required another layer of pre-processors on both the client and the server machines. A user runs Rpi on the local machine which reads the user's program, looking for #include directives. Instead of starting up pi on the server machine, Rpi instead requests that Spi (server pi) be executed. Data going from Rpi to Spi includes a header giving the size of the file, the length of the file name, the file name, and finally the file. This is done for every file that is needed in the compilation. Spi reconstructs the original source code in a temporary directory on the server machine. When all files have been received, Spi executes pi on those files in the temporary directory and the compilation is performed. Spi takes care of transferring pi's stdout to stderr. If the compilation completes successfully, Spi reads the executable image left behind by pi and sends it to stdout. This is picked up by garcon and sent to Rpi, which knows that anything coming to stdout from the remote machine belongs in an executable file on the local machine. The relationships between the processes, machines, and files are shown in figure II.

Maitre d' provides only a primitive connection mechanism through which a task on a remote machine may communicate with its calling process on the local machine. For tasks flexible enough to operate using only three data channels, the mechanism is sufficient. The point of this discussion on the Pascal application is to show that those tasks requiring more complicated file access can be accommodated by building on top of the interface provided. It may seem cumbersome and expensive to transfer all of a user's source files from client to server for each application's invocation. But, until a reliable remote file system becomes widely available, this type of explicit data transfer is necessary.7

6. Evaluation & Performance

6.1. Design Criteria

Alonso[2] describes six points for choosing a good load-balancing strategy: stability, implementability, cost, autonomy, transparency, and tunability. Maitre d' meets all of these criteria.

(1) Stability. A server machine could become inundated with requests from clients if it had recently announced its availability. As instantaneous state information is very hard to come by, use of an algorithm that avoids processor flooding is very important. Maitre d' relies on a round-robin selection mechanism of available hosts to minimize the possibility that any one client might overload a server. Although it is possible for several clients to all simultaneously request the same server, this type of selection synchronization is highly unlikely.

(2) Implementability. Maitre d' runs at the user level and requires no modifications to the UNIX kernel. It is running on VAX 750s, 780s, 785s, MicroVAX IIs, and a Sequent parallel processor. The basic package is about 4000 lines of well-commented C code (approximately 50K object) and is portable to any machine running 4.2 or 4.3 BSD UNIX.8

(3) Cost. The overhead in running Maitre d' is minimal. There are three considerations for cost: client overhead, network traffic, and server overhead. In the worst case, when no servers are available, the user's process usually requires less than .5 seconds of real time to determine that the job must be performed locally. Since the class of jobs running under Maitre d' are typically long-lived, this time is unnoticeable (but consistent). Traces show that it is rare for jobs to require more than a few hundred kilobytes of data to be transferred between client and server machines. On a 3 or 10 megabit Ethernet, the impact of the extra traffic is minimal. Since servers only broadcast changes in state information and spend most of their time blocked waiting on requests, the cost of running the server is also low. Using acceptance load thresholds of 5

---

7 Even with a remote file system, the data would still need to be moved between machines. The transfer would just be completely transparent.

8 There were some problems initially with the Sequent, but it turned out to be a bug in their 4.2 release and not in Maitre d'.

Volume 11, Number 1 January/February 1986
and 2 on our VAX 785’s and 750’s respectively, each server was averaging about 40 state changes a day.

(4) Autonomy. The decision to accept or reject jobs is completely up to the server, so no machine can be forced to take on work from outside clients. In addition, the types of jobs a server will accept, and the client machines from which those jobs may arrive is definable by the system administrator in a configuration file. The server machine may also impose a preemption policy on remote jobs, always giving priority to its own users.

Experience has shown these loads to be comfortable operating points. Below these values, the machines tend toward idleness. Above them, response time becomes intolerable.
(5) **Transparency.** The fact that a process is being executed on a remote machine can be made completely transparent to the user. Users need never know that their tasks are being performed elsewhere.

(6) **Tunability.** System parameters can be set and reset through a configuration file and a dynamic control program (mdc). Thus, *Maitre d* can be run in various environments without the need for recompilation.

### 6.2. Performance

Several factors contribute to the success of *Maitre d*. First, the decision to export only high-cost, CPU intensive jobs allows a machine, regardless of how busy it might be, to provide swift response time for those jobs. As the less expensive, non-ported jobs are no longer competing with the more costly ones for resources, they too enjoy an improvement in response time. Table I shows comparative statistics for April 1984 and April 1985 taken on a VAX 11/780 (ucbcory). In 1984, the machine ran without *Maitre d*. In April 1985, the only jobs being offloaded were Pascal and C compiles. About 3000 compiles per week were being exported to two VAX 750's operating as dedicated remote servers. Given are the UNIX five minute load averages; the times to start up the editor on a trivial file; and the times (in seconds) to compile and execute locally the following short CPU intensive program:

```c
main()
{
    int i, j, m;
    for (i=0; i<1000; i++)
        for (j=0; j<1000; j++)
            m = m+1;
}
```

The demands on the machine from instructional coursework were identical for the two months being compared. The increase in performance can be seen across the board in all the statistics, with an average improvement factor of over 2. The increase in perceived machine performance is even more dramatic considering that the VAX 780 was running with 16 megabytes of memory in 1984, but only half that during the 1985 sampling period. There were no other configuration changes. The decrease in variance for all the figures demonstrates that a balanced system has the added feature of offering predictable response times.

One metric used to gauge the relative utilization of machines over a long period of time is the mean load average. This is simply the average value of a machine's load average when sampled at regular intervals. From this value we also found:

(a) that those machines whose mean load average before *Maitre d*' was less than *garcon*'s acceptance threshold value tended to have their mean increase after load sharing was in place, and

(b) that those machines whose mean load average was above *maître*'s sendoff threshold saw a decrease in their mean load average, as well as a marked decrease in the variance of the load average.

This simply means that those machines most in need of assistance will benefit the most from load sharing, and machines which were idle will find themselves more busy. This is exactly what should be happening and is not surprising.
Table 1
VAX 11/780 Performance Comparisons

<table>
<thead>
<tr>
<th>ucberory</th>
<th>April 84 (w/o Maitre d’)</th>
<th>April 85 (w/ Maitre d’)</th>
<th>Improvement Factor</th>
</tr>
</thead>
<tbody>
<tr>
<td># samples</td>
<td>2140</td>
<td>2134</td>
<td>-</td>
</tr>
<tr>
<td>mean users</td>
<td>20.01</td>
<td>19.31</td>
<td>-</td>
</tr>
<tr>
<td>median users</td>
<td>20</td>
<td>19.74</td>
<td>-</td>
</tr>
<tr>
<td>variance users</td>
<td>97</td>
<td>100</td>
<td>-</td>
</tr>
<tr>
<td>mean load</td>
<td>6.12</td>
<td>2.52</td>
<td>2.42</td>
</tr>
<tr>
<td>median load</td>
<td>4.6</td>
<td>1.8</td>
<td>-</td>
</tr>
<tr>
<td>variance load</td>
<td>23.95</td>
<td>6.07</td>
<td>-</td>
</tr>
<tr>
<td>mean editor (secs)</td>
<td>1.46</td>
<td>0.82</td>
<td>1.78</td>
</tr>
<tr>
<td>median editor</td>
<td>1</td>
<td>1</td>
<td>-</td>
</tr>
<tr>
<td>variance editor</td>
<td>1.87</td>
<td>0.362</td>
<td>-</td>
</tr>
<tr>
<td>mean compile (secs)</td>
<td>11.90</td>
<td>7.04</td>
<td>1.69</td>
</tr>
<tr>
<td>median compile</td>
<td>8</td>
<td>6</td>
<td>-</td>
</tr>
<tr>
<td>variance compile</td>
<td>94.6</td>
<td>20.42</td>
<td>-</td>
</tr>
<tr>
<td>mean execution (secs)</td>
<td>63</td>
<td>26.7</td>
<td>2.35</td>
</tr>
<tr>
<td>median execution</td>
<td>30</td>
<td>14</td>
<td>-</td>
</tr>
<tr>
<td>variance execution</td>
<td>5564</td>
<td>1142</td>
<td>-</td>
</tr>
</tbody>
</table>

7. Conclusions

We have implemented an effective load-balancing system and demonstrated its utility. Further applications can be easily introduced to operate within the package’s environment. Performance metrics indicate that this system is highly successful in creating a more responsive and pleasant working environment for users.

Acknowledgments

This work was made possible in part by a donation of equipment from Digital Equipment Corporation and the support of the State of California MICRO Program[3]. The original version of Maitre d’ was initially proposed and authored by undergraduate students Chris Williams and Chris Guthrie[8]. Professors Richard Fateman and Lou Katz should be credited for their patience in seeing this project through to the end. Finally, special thanks to Doug Cooper for his tireless advice on this paper’s organization and presentation.

Sites interested in obtaining the Maitre d’ load balancing software should contact the author in care of the Computer Systems Support Group (CSSG) at UC Berkeley.
References

(1) G. Almes and E. Lazowska, *The Behavior of Ethernet-Like Computer Communications Networks*.


(7) S. Sechrest, *Tutorial Examples of Interprocess Communication In Berkeley UNIX 4.2 BSD*, Computer Science Research Group, Department of Electrical Engineering and Computer Science, University of California, Berkeley.

Appendix A
Sample Configuration File

# UCBSIM
#
# Maitrd Load Balancing Configuration File
#
# Instructions available to outside world
# I<command name command path proc uid>
# Possible Client
# C<ucbxyzzy>
# People Servers for me
# S<ucbxyzzy> ... no entries means all
# Entry both a client and a server
# B<ucbxyzzy>
# Garcon configuration option
# G<option> <value>
# Clearing House Machine
# m<ucbxyzzy>
#
# These machines are allowed in:
Cucbear
Cucbeast
Cucbdali
Cucbarpa
# These machines should be willing to take jobs from me:
Sucfranny
Sucbzzoey
Sucbsim
Sucbbuddy
# This guy takes and receives:
Bucbernie
#
# Request from (localhost == ourselves)
nlocalhost
#

#############################################################
# Commands We Can Run
#
#
# Anything here can be run by people on the outside
# First column = name by request
# Second column = path of associated program
# Third column = user name for task
#
# lcsh /bin/csh root would be a bad entry
#
# lcom /lib/ccom nobody
lnroff /usr/bin/nroff nobody
lcat /bin/cat nobody
ldate /bin/date nobody
lecho /bin/echo nobody
lwhoami /usr/ucb/whoami nobody
;login:

<table>
<thead>
<tr>
<th>Command</th>
<th>Directory</th>
<th>User</th>
</tr>
</thead>
<tbody>
<tr>
<td>lhost</td>
<td>/bin/hostname</td>
<td>nobody</td>
</tr>
<tr>
<td>lspi</td>
<td>/usr/local/spi</td>
<td>nobody</td>
</tr>
<tr>
<td>lscce</td>
<td>/usr/local/scce</td>
<td>nobody</td>
</tr>
<tr>
<td>ltroff</td>
<td>/usr/bin/troff</td>
<td>nobody</td>
</tr>
<tr>
<td>ltroff_p</td>
<td>/usr/local/troff_p</td>
<td>nobody</td>
</tr>
<tr>
<td>lrwho</td>
<td>/usr/ucb/rwho</td>
<td>nobody</td>
</tr>
<tr>
<td>lspice</td>
<td>/cad/bin/spice</td>
<td>nobody</td>
</tr>
</tbody>
</table>

#

# Server runtime options....

# Load threshold above which jobs are not accepted
Gload 3.0
# Number of non-idle users above which jobs are not accepted
Gusers 20
# Maximum number of clients allowed in
Gclients 15
# When the load exceeds threshold, raise priority (renice) of
# active jobs to argument. In unix, this is analogous to preemption.
Gnice 4
A Perspective on the USENET

Erik E. Fair
Technical Consultant
USENIX Association
P. O. Box 7
El Cerrito, CA 94530

Introduction

This section will be a quick description of the USENET. If you are already familiar with the basic structure of it, you can skip to the next section.

The USENET is a large, distributed computer conferencing network. The primary difference between the USENET and traditional computer conferencing (or bulletin board) systems is that the USENET is a network of computer systems, and the information is automatically transmitted to the user's computer system from other peer systems, rather than having the user call up a central conferencing system at some remote location in order to participate. The network consists of some 2000 medium to large computer systems, most of which run the UNIX operating system. These systems primarily exchange messages of English text, although foreign language text, program source code, and even executable binaries have been sent.

The software that implements this network is called netnews. The traffic consists of individual messages, called articles, which are organized into topics, called newsgroups. The majority of newsgroups are distributed network-wide (i.e. worldwide), but newsgroups can have a restricted distribution region (e.g. "ca.general" goes only to sites in the state of California).

In terms of the UNIX file system, articles are stored as individual files, newsgroups are directories (organized hierarchically, of course), and articles may appear in more than one newsgroup at a time without extra cost in transmission or disk space, courtesy of file links. Dots in the newsgroup name are translated to slashes to make a directory name (e.g. net.news.group → net/news/group).

When a user posts an article to a selected newsgroup (or newsgroups; posting to multiple newsgroups at once is called cross-posting), the netnews software causes the article to be transmitted to the site's immediate network neighbors, who, in turn, transmit it to their neighbors, ad infinitum. Thus the effect of broadcasting is achieved by this "store and forward" network. Although the netnews software can initiate article transmission over almost any medium, articles are transferred between USENET sites primarily over 1200 baud dial-up lines with the UNIX to UNIX Copy (UUCP) communication software.

When articles arrive on a system, they are put into the news spool directory in the appropriate newsgroup directories, under sequentially numbered files (e.g. /usr/spool/news/net/general/105). When users on that system invoke any of the user interface programs, the program will present to the user those articles on the system that he or she has not already read. After reading each article, the user can respond to the item either by electronic mail directly to the author, or by posting an article to the network for the entire network community to read. Articles remain on a system for some period of time (default is two weeks), after which they are removed by the expire program, to free up disk space for more incoming articles.

The USENET has been built up over the last five years by individual system administrators making agreements with each other to exchange netnews articles. There has never been any central authority or control over the network as a whole, so when network-wide decisions need to be made, the USENET operates by consensus with a dash of anarchy thrown in.
Successes

If you already have a UNIX system, the network is cheap to join: the cost is a modem and the phone bills to call the nearest USENET site. The netnews software is in the public domain, and is freely available from any site that already has it. It runs on all the versions of UNIX that we know about, and it is maintained by a cadre of very dedicated volunteers.

Because of the low cost of joining, the USENET has grown explosively during its five year history. The earliest network map I have is dated April, 1981, and it shows 35 sites in the United States and Canada. Presently the best guess is that the USENET is somewhere between 2000 and 2500 sites in size, covering North America, Europe, Australia and the Far East.

The varieties of human interest have also found expression in the nearly 300 newsgroups covering a very diverse set of topics from the excruciatingly technical (e.g. netunix-wizards and net.lang.c) to the mundane (e.g. net.cooks and net.movies).

In addition to the worldwide and regional newsgroups, some corporations have found the netnews software useful for organizing and distributing their internal corporate communications in internally distributed newsgroups (e.g. AT&T, Tektronix).

The bottom line is that there is a wealth of information that is easily and cheaply available to anyone with a UNIX system and the time to set things up.

So What’s Wrong?

Because of the unchecked growth of the USENET, coupled with a high user turnover rate, most of the users of the network did not live through the events during which the network’s conventions and standards of etiquette were established. New users have also shown a remarkable unwillingness to educate themselves, and the majority of system administrators, on whom the network community depends to educate the users, have not made sufficient efforts at doing so.

The network community also depends upon system administrators to run the latest release of the software so that bugs can be quickly squashed, and improvements can be used as soon as implemented. Unfortunately, system administrators often can’t or won’t do an update of the software each time it is released, so the software maintainers must keep backward compatibility in mind when they make changes.

It is widely perceived that a large fraction of the articles flowing through the network are junk. Unfortunately, the definition of “junk” varies sufficiently from individual to individual, making network-wide decisions about content a very charged political issue. The anarchic consensus process that is used to make decisions at the network-wide level compounds this problem considerably.

The network also has its share of self-righteous and/or malicious users who cause problems for the network community.

Being a site on the USENET is an embarrassment of riches, in that there are too many articles flowing through the network for any one person to plow through all of them. Average traffic for a site that gets all the newsgroups is approximately one megabyte per day, and while “newsgroups” are a reasonable system for general separation of articles into topics of interest, there is no organization or structure of the articles below that level. To compound things, there are no filtering mechanisms provided by the user interfaces, other than refusing individual articles (which takes too long because there are so many articles) or entire newsgroups (which is too coarse a level of cutoff).

Lastly, most of the effort in improving the software has been directed at making transmission, processing, and storage more efficient, with comparatively little development of either existing or new user interfaces. While the former is indeed a worthy goal (particularly if we wish the software to continue to handle the increasing volume of traffic gracefully), most of the network’s problems result from the behavior of individual users, and we need to change the user interfaces in order to influence that behavior toward fewer articles of higher quality.
Thinking About Solutions

Since neither users nor system administrators are behaving as we would like, we should change the software to do as much of the education as possible, and make administration of netnews as "turnkey" as possible, since we can expect that both users and system administrators will (on average) put in minimum effort, and will expect maximum return on their investment of time.

In order to do this, there are some questions to think about:

1. How can we best organize articles in a newsgroup for presentation to a user?
2. What sort of article filtering tools should be available to the overwhelmed user?
3. Since the majority of articles are responses, rather than original articles, what is the ideal behavior of a user responding to an article, and how can we encourage that behavior in the software?
4. What netnews administration tasks can be partially or entirely automated so that administration of netnews takes a minimum of effort?

Article Presentation

Ideally, one should be able to read all the articles in a newsgroup in the order in which they were posted. However, the current presentation programs present the articles in the order in which the arrived on the local system, which is frequently not the right order. Also, there is usually more than one discussion thread in a newsgroup, so that straight chronological order isn’t right, either.

To create structure in the way that articles are presented, we can begin by grouping articles whose ‘Subject:’ header lines match. This gives us the thread of a discussion.

Unfortunately, articles can and frequently do arrive on a system out of order. To get the ordering right within a discussion we need to sort each discussion by the date of submission of each article in the discussion. This information is in the ‘Date:’ header line of every article, expressed in Greenwich Mean Time, so that the time stamp is absolute. This operation will give us a discussion in the chronological order in which it occurred.

However, much like conversation at a cocktail party, discussion topics tend to wander away from the original subject, but on the USENET the ‘Subject:’ header line is usually not changed by a user composing a response to an article. The positive aspect of this is that it is easy to find the same discussion you’ve been following for some weeks. Unfortunately, some poor innocent might get tricked into reading a debate about something very different from what the subject line indicated, which would be a waste of time, if it were not serendipitous.

Fortunately, there is even more information in the header that we can use to order articles into a discussion more accurately than with ‘Subject:’ and ‘Date:.’ Each article has a network-wide unique message identifier string assigned to it, usually consisting of a sequence number (or some number generated from the date and time), and the name of the host that the article originated from. When a response to an article is composed, the netnews software puts the message-id into another header line called ‘References:’, so that the response has a pointer back to the article to which it is a response.

Presently, the only use that any of the user interfaces make of this field is for finding the “parent” article of the current article (i.e. the article to which the current article is a response). However, if the article processing program (rnws) built the doubly linked list implied by this single back pointer, and constructed a database of discussions, the following things are possible:

1. Accurate ordering and presentation of the discussions that take place on the network.
2. When the content of a message is no longer reflected by the ‘Subject:’ line, the users can feel free to change it without worrying that the user interface will lose the thread of discussion, because the references will still cause the response with the different subject to be grouped with the discussion.
3. Since the users will be free to change the subject lines as the subject changes, we can easily differentiate between the various sub-branches of the tree of a discussion (e.g. one branch might go off discussing “foo” from “foobar,” while another discusses “bar” from “foobar”).
4. With the notion of discussions firmly entrenched into the user interfaces, it will not be necessary to include the text of the article to which one is responding, because that article can be easily fetched for display.

Article Filtering

Unfortunately, even with wonderful presentation methods, there's still too much material to read. As the volume of messages in the newsgroups increases, there are two steps that a user can take to be more efficient:

1. The user can arrange to read netnews at a higher baud rate (instead of 1200 baud, how about 9600 or 19200?). This will allow making article selections faster, and hopefully more articles per unit time can be handled than at the lower speed. Ideally, an instantaneous display system would eliminate all waiting for the computer in the article selection process.

2. The user can prioritize the his or her list of newsgroups and remove some newsgroups from the bottom of the list, until the volume presented is manageable again. Unfortunately, unsubscribing to a newsgroup that one is interested in tends to leave the feeling that one might miss something really terrific.

However, these traditional mechanisms for limiting time spent reading netnews are no longer sufficient, because they're not specific enough, or too specific. What we need now is a set of automatic filtering mechanisms for articles.

Consider the following information which is contained in (or can be derived from) a netnews article header that might be useful to filter by:

<table>
<thead>
<tr>
<th>Field</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>author</td>
<td>(also known as the 'bozo' filter)</td>
</tr>
<tr>
<td>site</td>
<td>(they're all bozos on at that site)</td>
</tr>
<tr>
<td>date</td>
<td>(e.g. refuse articles that are 'n' days old)</td>
</tr>
<tr>
<td>transit-time</td>
<td>(e.g. refuse articles that took more than 'n' days to get here)</td>
</tr>
<tr>
<td>length</td>
<td>(e.g. refuse anything too small or too big)</td>
</tr>
<tr>
<td>newsgroups</td>
<td>(e.g. in a multiple group posting, refuse if 'net.flame' is one of the other groups)</td>
</tr>
</tbody>
</table>

It should be noted that in addition to refusing articles by these criteria, we can also seek them (e.g. show me all the articles by <x@y.UUCP> in net.unix-wizards).

Finally, one more: moderators are a filtering mechanism, in that they select appropriate articles to broadcast to the network. In our electronic publishing medium, they are editors.

While the whole concept of moderation has generated controversy over perceived censorship on the USENET, the ARPA Internet community has found it most useful for keeping mailing lists from degenerating into prolonged "more heat than light" debates, and that community trusts their moderators implicitly. I have observed that a good moderator can do wonders to keep a discussion on track and productive, and I think that this is one case where the USENET community really has a lot to learn from its forebears on the ARPA Internet.

Responses and User Behavior

Since the vast majority of the articles on the network are responses, rather than original postings, if we want to cut down on traffic while increasing the information content of what is posted, we should consider carefully the behavior of a user when he sees something that he wants to respond to. It is this behavior that we should attempt to affect by making changes in the user interface software.

In particular, this is the time that the user should be checked and double checked as far as the software can discern. This is the time to ask, "Are you sure that's what you want to do?" to discourage spur of the moment postings.

One of the most chronic problems of both the USENET and of electronic mailing lists are hundreds of responses to a trivial query. A user will post a message asking a question, to which everyone
knows the answer and *everyone* immediately posts a response as they see the query. The network is deluged with different copies of the same answer to this trivial query, and no one is happy plowing through the resulting traffic. This occurs because both reading electronic mail and reading netnews are essentially sequential processes, with each message being handled discretely.

However, given that our hypothetical user interface is aware of "discussions" as described in the previous section, and given that there are more articles on the local system which refer to the query (one presumes that they would be answers), the user interface could hold the follow-up article until the user has read all the articles in the discussion. After the entire discussion has been read, the user interface could ask whether the submitted response is still appropriate to post network-wide, or mail to the author. Alternately, the user interface could inform the user that there are more articles to read on that particular subject, and suggest that they all be read before responding.

This could even be implemented simply by providing a mechanism in the user interface to "mark" articles that the user might want to respond to, and recapitulating these articles when all the articles in the newsgroup have been read.

While this technique is limited by the propagation speed of the network (it does no good if the responses aren't there yet), not everyone is hanging on the edge of their modem, waiting eagerly for the very next article to arrive, and that being the case, either of these two measures should cause the incidence of this problem to drop significantly, resulting in a decrease in total network traffic. Also, the propagation time of a given article should go down over the long run as the speed of transmission of articles over the network increases because of modem and network technology improvements.

The other serious problem is too many response articles posted network wide, when all parties concerned would have been better served by the respondent sending electronic mail to the author of the article that inspired the response. Right now, all the user interfaces ask what to do with a response before it has been composed by the user. This interrupts the flow from decision to respond, to composition of response, and does not allow consideration of the content of the response in the decision of how to send, because the response has not been composed yet.

The question of "How to send?" is best asked after the response has been composed, so that the user can go straight from the decision to respond, to composition of the response, and then to the decision as to whether the response should be electronic mail to the author of the article or posted to the entire network.

Asking after composition allows the user to consider the content of his response in the decision of how to send it, and also allows the composition period to be a "cool down" period, if the decision to respond was made passionately. It would also allow the user interface implementors to reduce the number of commands for responding.

**Site Administration and Network Administration**

Speaking as the lazy administrator of two USENET sites, I think that netnews administration takes more effort than it should. However, there are some things that can be done to help this situation:

1. Control messages that require administrator intervention could be handled as part of the user interface, when the administrator reads the *control* newsgroup. Then control messages from the wrong people or wrong sites could be filtered with the same mechanisms that normal articles are. It would be even better if they could be aggregated so that when the administrator got to the end of the *control* newsgroup, all of the actions could be executed at once (and hopefully those actions that canceled each other out could be culled before execution).

   Alternately, it might be reasonable to let all control messages be executed after a grace period for administrator intervention (i.e. if the administrator does nothing, let the control messages be executed).

2. More of the myriad files that netnews needs could be maintained automatically from a central network point (or from several points). The files that come to mind are: *aliases*, *moderators*, and
distributions. In particular, software to automatically generate mail paths for the *moderators* file would be helpful.

3. A *rename* control message for newsgroups so that newsgroup name space management would be a little easier. This control message should do all the work that is done by hand now (change the active file, add to the aliases file, etc.).

In short, some special handling in the user interfaces for netnews administrators would make their job easier.

**A Word About *rn* and *notesfiles***

At this point I'll bet that users from both the *rn* and *notesfiles* camps are bursting to tell me that their interfaces do a lot of what I have been writing about. I disagree.

Both *rn* and *notesfiles* are still trying to follow discussions by matching subjects. While this is a good start, that method has limitations, and there is (as I have described) a vastly better method for doing so available. On the other hand, only *rn* has a good article filtering mechanism.

I think that *rn* would be a terrific base to develop the things described herein, and *notesfiles* has the subject menu done right (sequence number, number of articles, subject; such that only one line per discussion appears in the menu). Neither of them is quite what we need to cope with the great flood of information that we're currently wading through.

**Some Blue Sky**

Here's an incomplete wish list for whizzy new features to add to the netnews software. Fortunately for us, the header of the netnews article is easily extensible.

1. **Article types.** Suppose I have a graphics terminal, and I want to send a pretty picture. Given I could reduce the picture to some intermediate code (NAPLPS, gremlin, etc.), it would all work better if the article could be tagged such that the body of the article is passed through a program (or a pipeline) before being displayed on a terminal. Possible types or tags include: *nroff*, *naplps*, *gremlin*, *face*.

2. **Voting and survey support.** Imagine being able to format a survey as a list of questions and possible responses for a program that would ask the questions, collect the specific responses from the user, and mail them off to the surveyor (as opposed to posting the results network-wide, as frequently happens now). This program would be invoked by the user interface for each user that read the article, with the body of the article as *stdin*. This can be accomplished with the article type described above (the formal input could be tagged *survey* with the survey program invoked to provide the special processing).

   This might even be able to aid in network-wide decision-making with proper surveys of network opinion.

3. **Article accolades.** In many offices, different people read different publications, and if any one of them finds a particularly interesting article, they usually photocopy it for the entire office. Imagine if you will, a control message or command to mark articles that you thought were good or socially redeeming in some way, so that other people at the same site could dig into a newsgroup that they rarely read, and only read those articles marked with some number of accolade marks or with accolade marks from certain local users.

   While it might be interesting to have this information distributed network-wide, I expect that the extra traffic required would be prohibitive, so I don't recommend it.

   *(Thanks to Chuq Von Rospach for the name "accolades.")*

4. **Education in software.** Imagine a beginner's interface to netnews, much like the editor *edit* is to *ex*, or as the *learn* program was intended to teach the basic concepts of UNIX, that would put forth the basic concepts (what's an article, a newsgroup, a response; how distributions are used) and network etiquette.
Better help systems in the standard user interfaces might also accomplish this just as effectively.

5. **Posting to a region outside your own.** It is presently not possible to post a message to nj.general if you're at a site in California, for example. At least, not without the collusion of a friend in New Jersey.

6. **Authentication with encryption.** The network currently has no security from forgeries. It is nearly impossible for someone to be sure that article 'x' was in fact written by the person claimed in the header. If RSA style digital signatures were used, you could be sure beyond reasonable doubt. This might find best use in authenticating control messages for article cancellation, and other network-wide management.

**Summary**

It should be fairly obvious at this point that I believe in “better networking through software technology.” This is primarily because I know how to program computers vastly better than I can program people, and there is really only one set of software that needs to change to effect the desired changes in the network, versus the thousands (or tens of thousands) of people that participate in the USENET. If the programming were properly done, it could produce a consistency of education in the conventions and etiquette of the network that would benefit the USENET community a great deal.

I think that Larry Wall, author of the *nn* user interface, has done a marvelous job at designing the first interface with real power to deal with the great volumes of articles that the USENET has been circulating in recent years, and that program is the right base for the “ideal” user interface that I'm looking for.

However, the volume of traffic keeps going up, and unless we do something more to limit the number of articles (particularly if we can distinguish and limit the number of “junk” articles), the traffic will become more than our current methods of transmission can handle or afford.

**Bibliography**

Erik E. Fair, “Information Overload and What We Can Do About It,” *USENET message-id <10381@ucbvx.ARPA>*, September 14, 1985.


Douglas E. Comer, Purdue University, Larry L. Peterson, University of Arizona, “Conversation-Based Mail,” DRAFT TR, August 26, 1985.
Release Form for 1986 USENIX Distribution Tapes

CORPORATE, INSTITUTIONAL, AND SUPPORTING MEMBERS ONLY

As a Corporate, Institutional, or Supporting Member of the USENIX Association, I wish to receive the 1986 USENIX Distribution Tape(s) (hereafter referred to as the TAPES). By virtue of this signed release form, I agree to the following Terms and Conditions governing the use of any and all TAPES sent to me. Neither the USENIX Association, its officers, employees, and agents, nor the authors nor submitters of the materials on the TAPES (hereafter referred to as the ABOVE NAMED) make any representation or warranty, expressed or implied, as to the accuracy, completeness, or usefulness, of the TAPES or the contents thereof for any purpose whatsoever.

(1) To the extent permitted by law, I hereby explicitly indemnify and hold harmless the ABOVE NAMED from any claims, costs, damages and liability whatsoever with respect to the TAPES or the contents thereof.

(2) No property rights with respect to the TAPES or the contents thereof shall transfer to me. The TAPES shall remain the property of the USENIX Association and the contents thereof shall remain the property of the person or persons who possessed that property before the TAPES were written.

(3) The licensing of my use of the materials on the TAPES by the USENIX Association is limited to those materials for which the USENIX Association is authorized to do so. The USENIX Association has attempted to verify that it has the right to distribute all of the materials on the TAPE, but it cannot guarantee that it has such right. In the event that any dispute arises as to the distributed materials, I agree that the USENIX Association may direct me to erase materials from the TAPES and that I will remove said materials from the TAPES, destroy all copies thereof, and make no claim against the ABOVE NAMED for any action arising out of the use or the destruction of the materials.

(4) I certify that I have the UNIX™ licenses or sub-licenses indicated on the next page and that I shall use the materials on the TAPES so as not to violate said licenses. In particular, I agree that if there should be material on the TAPES that I am not licensed under our UNIX license(s) to receive or use, I shall neither use nor make copies of said materials.

(5) I agree that this license grants me no rights to redistribute any or all of the materials on the TAPES, except for use at the address designated on the next page or on my membership form. This license grants me no rights to incorporate any of the materials on the TAPES in a commercial product, and I agree that, if such materials are not in the public domain, I will secure such rights from the owner of the property before incorporating any of the materials on the tape in a commercial product.

In consideration of the mutual promises contained herein, I release and discharge the ABOVE NAMED, their officers, employees and agents, of and from any claims, causes of action, costs or demands of whatever nature, whether anticipated or unanticipated, and whether known or unknown, which I have, may claim to have, or at any time heretofore have had against the above mentioned individuals and entities, or against any of them, including but not limited to, any and all claims, demands or causes of action which are contained in or may arise out of the circumstances set forth in this agreement, above.

The parties hereto acknowledge that they are familiar with the provisions of Section 1542 of the California Civil Code and expressly agree that the Release set forth above constitutes a waiver and release of any rights or benefits that they may have under California Civil Code Section 1542, which provides that:

"A general Release does not extend to claims which the creditor does not know or suspect to exist in his favor at the time of executing the release, which known by him must have materially affected his settlement with the debtor."

Subject to the above conditions, the USENIX Association grants and I hereby accept a limited license to use the materials on the TAPES at the address given on the next page.
The contributions are generally separated into distribution tapes containing material covered by different source licenses. Tapes will include all material from earlier versions of source licenses which the current license specifically allows (e.g., the System III tape will include material licensed for V6, V7, etc. as well as no-disclosure material). **Binary license holders may receive ONLY the tape with no-disclosure material.**

As a Corporate, Institutional or Supporting Member, you are entitled to one tape per distribution. (However, additional different versions, for which you have appropriate source licenses, may be purchased. Contact the USENIX Office for more information.) Since there is no guarantee that material will be submitted for all license categories, please indicate the UNIX source licenses you own and your preference for receiving tapes (1 = first choice, 2 = second, etc.):

<table>
<thead>
<tr>
<th>System V</th>
<th>Source License</th>
<th>Tape Preference</th>
<th>Source License</th>
<th>Tape Preference</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>[ ]</td>
<td></td>
<td>Phototypesetter</td>
<td>[ ]</td>
</tr>
<tr>
<td>System III</td>
<td>[ ]</td>
<td></td>
<td>Version 6</td>
<td>[ ]</td>
</tr>
<tr>
<td>UCB 4.x BSD</td>
<td>[ ]</td>
<td></td>
<td>Mini-UNIX</td>
<td>[ ]</td>
</tr>
<tr>
<td>Version/32V</td>
<td>[ ]</td>
<td></td>
<td>UNIX/TSS</td>
<td>[ ]</td>
</tr>
<tr>
<td>UCB 2.x BSD</td>
<td>[ ]</td>
<td></td>
<td>UNIX/UNIVAC</td>
<td>[ ]</td>
</tr>
<tr>
<td>Version 7</td>
<td>[ ]</td>
<td></td>
<td>Other:</td>
<td>[ ]</td>
</tr>
<tr>
<td>PWB/UNIX Version 1.0</td>
<td>[ ]</td>
<td>No-Disclosure tape</td>
<td>(none)</td>
<td>[ ]</td>
</tr>
</tbody>
</table>

Tape density (specify only if you cannot accept either density):

[ ] 800 bpi  [ ] 1600 bpi

Address to which your TAPES should be sent and where TAPES will be used, if different from address on membership form:

Company or Institution Name: ____________________________ Member Number: _____

Member Representative Name: ____________________________

Member Representative Signature: ________________________ Date: __________

USENIX Representative Signature: ________________________ Date: __________

Please sign and return two copies of both pages of this form to:

USENIX Association Office
P.O. Box 7
El Cerrito, CA 94530

One copy will be signed by a USENIX representative and returned to you.

Copies of your relevant UNIX license(s) must be on file at the USENIX Office for you to receive tapes. It is your responsibility to notify the USENIX Office should there be any change in your licensing categories, tape preference, or tape mailing address.

---

For Office Use

date _________ lic _________ how _________ by _________
type sent _________ date _________ by _________ db _________
type sent _________ date _________ by _________ db _________
4.2BSD Manual Reproduction Authorization and Order Form

USENIX Member No.: ____________
Purchase Order No.: ____________
Date: ____________

As the representative of a USENIX Association Institutional or Supporting Member in good standing, and as a bona fide license holder of both a 4.2BSD software license from the Regents of the University of California and a UNIX™/32V or System III or System V license or sublicense from Western Electric Company, and pursuant to the copyright notice as found on the rear of the cover page of the UNIX/32V Programmer’s Manual stating that

"Holders of a UNIX™/32V software license are permitted to copy this document, or any portion of it, as necessary for licensed use of the software, provided this copyright notice and statement of permission are included”,

I hereby appoint the USENIX Association as my agent, to act on my behalf to duplicate and provide me with such copies of the Berkeley 4.2BSD Manuals as I may request.

Signed: ____________________________________________
Institution: __________________________________________

Ship to:
Name: ____________________________________________
__________________________ ____________________________
__________________________ ____________________________
__________________________ ____________________________
__________________________ ____________________________
Phone: ____________________________

Billing address, if different:
Name: ____________________________________________
__________________________ ____________________________
__________________________ ____________________________
__________________________ ____________________________
Phone: ____________________________

The prices below do not include shipping and handling charges or state or local taxes. All payments must be in US dollars drawn on a US bank.

4.2BSD User’s Manual ______ at $18.50 each = $________
4.2BSD Programmer’s Manual ______ at $17.00 each = $________
4.2BSD System Manager’s Manual ______ at $ 9.50 each = $________
Total ______

[ ] Purchase order enclosed; invoice required
(Purchase orders must be enclosed with this order form.)

[ ] Check enclosed for the manuals: $________
(Howard Press will send an invoice for the shipping and handling charges and applicable taxes.)

Make your check or purchase order out to Howard Press and mail it with this order form to:

Howard Press
c/o USENIX Association
P.O. Box 7
El Cerrito, CA 94530

for office use: l.v.: ____________ check no.: ____________ amt. rec’d: ____________
Winter ’86 Conference & Tutorials

4.2BSD Manual Printing

Ease: A Configuration Language for Sendmail

Load Balancing With Maitre d’

A Perspective on the Usenet