STACKPOINTER

1-1988. Organ för Datorföreningen STACKEN, KTH.

Datorföreningen STACKEN

STACKEN är datorföreningen på KTH. Vi har en mängd intressanta verksamheter på gång. Dock hänger det ytterst på den enskilde medlemmen att avgöra vad han eller hon vill göra för föreningen. STACKEN är en ideell förening, där intresse för datorer är den gemensamma faktorn. Sedan föreningen grundades 1978 har vi (bland annat) åstadkommit:

- samköp av mikrodatorbyggsatser
- kurser, föredrag och studiebesök
- AMIS, den portabla EMACS-kompatibla editorn för TOPS-10, VMS, PRIME, NORD, ...
- STACKPOINTER, vår tidning
- en egen datorhall för vår DEC-10 och våra andra stordatorer

Sedan ett och ett halvt år har vi en egen DEC-10, som vi installerat, felsökt och kör på. Det är en gammal modell med KA10-processor. Vi kallar henne KATIA. Hon står i vår maskinhall "B30", på Brinellvägen 30 (V-sektionen) på gaveln mot Lill-Jansskogen (där vi har en egen ingång). En våning ovanför finns en hörsal (V4), där vi håller till vid större möten.

Ordinarie möten är första torsdag i varje månad, kl 19, vid datorhallen eller i hörsalen. Är Du intresserad av föreningen, är Du välkommen till något av våra möten. Vill Du sedan bli medlem, lämnar Du en skriftlig ansökan till styrelsen eller skickar den till vår postadress.

STACKPOINTER

STACKPOINTER är organ för Datorföreningen STACKEN på KTH. Den utkommer när material i tillräcklig mängd finns, förhoppningsvis 4–5 gånger per år. Återgivande av delar av innehållet är tillåtet när källan anges.

Redaktör: Hans Nordström
I redaktionen: Jan Michael Rynning och Henrik Björkman
Fotograf: Ulf Asplund
Ansvarig utgivare: Mats O Jansson

Färdigställd: 1988-01-27

I detta nummer

Adresser m m

POST till föreningen skickas till nedanstående adress eller läggs i postfacket på NADA.

Datorföreningen STACKEN
c/o NADA
KTH
100 44 STOCKHOLM

Medlemsavgift: 88 kronor för 1988

Ordf: Stellan Lagerström {08-790 78 14 (arb) 08-63 68 25 (hem)
Sekr: Olle Betzén {0758-318 48 (hem)
Kass: Mats O Jansson {08-34 93 20 (arb) 08-711 70 37 (hem)
Red: Hans Nordström {08-797 80 13 (arb)
Hexm: Henrik Björkman {08-30 89 01 (hem)
Suppl: Henning Croona {08-799 82 80 (arb) 08-739 17 40 (hem)
Suppl: Thord Nilson {08-712 00 90 (arb) 08-749 21 92 (hem)

Postgiro: 433 01 15-9
Bankgiro: 344-3595

Klubblokal: Lindstedtsvägen 5 (1 trappa ner)
Datorhall: Brinellvägen 32 (på gaveln mot Lill-Jansskogen)

UUCP: stacken@stacken.kth.se (...{uunet,mcvax}!enea!ttds!stacken)
ARPA: stacken@stacken.kth.se (stacken%stacken.kth.se@uunet.uu.net)
SUNET DECnet: KICKI::STACKEN (STACKEN@KICKI)
PSI: PSI%0240200101905::KICKI::STACKEN

Höstmötesprotokoll

ROTOKOLL fört vid Datorföreningen STACKENs höstmöte, avhållet torsdagen 1987-12-03 klockan 19.00, i sal E7 på KTH.

Närvarande i alfabetisk ordning:

Eva Albetsson, Bo Arnoldson, Olle Betzén, Henrik Björkman, Anna-Christina Borstedt, Henning Croona, Sven Erik Enblom, Harald Eneroth, Jonny Eriksson, Per Eriksson, Sten Thure Eriksson, Mikael Hübsch, Carl-Arne Johannesson, Christer Johansson, Stellan Lagerström, Erik Levin, Jan Lien (från mitten §15), Per Lindberg (från §16), Lars S Ljungdahl, Peter Löthberg, Kurt Minnberg, Thord Nilson, Hans Nordström, Thomas Nyström, Dan Pettersson (från mitten §15), Jan Michael Rynning, Peter Röstin, Nils Segerdahl, Per Sjöholm, Bernhard Stockman, Erik Svensson, Anders Westerberg, Klaus Zeuge och Bengt Åhlin.

§1. **Mötets öppnande.**
Stellan Lagerström öppnade mötet.

§2. **Val av justeringsmän.**
Henrik Björkman och Henning Croona valdes till justeringsmän.

§3. **Val av mötesordförande.**
Stellan Lagerström valdes till mötesordförande.

§4. **Val av mötessekreterare.**
Olle Betzén valdes till mötessekreterare.

§5. **Tillkännagivande av röstlängd.**
Då röstlängden ännu inte hade anlänt, föredrog Stellan Lagerström om STACKENs nya lokaler på Lindstedtsvägen 5, 1 trappa ner. Röstlängen anlände och de närvarande prickades av.

§6. **Frågan om mötet ansåg sig stadgeenligt utlyst.**
Mötet ansåg sig stadgeenligt utlyst.

§7. **Frågan om dagordningens godkännande.**
Mötet godkännde dagordningen.

§8. **Val av styrelse.**
Valberedningen föreslog följande:

Ordförande:	Stellan Lagerström
Sekreterare:	Olle Betzén
Kassör:	Henning Croona
Hexmästare:	Henrik Björkman
Redaktör:	Hans Nordström
Suppleant:	Mats O Jansson
Suppleant:	Thord Nilsson

Mötet ansåg detta förslag tillfyllest och valde styrelse i enlighet med förslaget.

§9. **Fastställande av firmatecknare.**
Stellan Lagerström och Henning Croona valdes till firmatecknare.

§10. **Val av revisorer.**
Peter Löthberg och Lars S Ljungdahl valdes till revisorer.

§11. **Val av valberedning.**
Nils Segerdahl och Thord Nilsson valdes till valberedare.

§12. **Fastställande av budget.**
Medlemsavgifterna beslutades skulle användas till föreningens tidning och utskick av densamma. Vidare bestämdes att extra inkomster ska gå till extra utgifter. Extra inkomster härrör från försäljningen av AMIS, extra utgifter beräknas uppstå i samband med transporter av datautrustning samt upprustning av det hus som STACKEN förhoppningsvis ska få tillgång till i början av 1988.

§13. **Fastställande av årsavgift.**
Tre förslag förelåg: 88 kr, 100 kr och 1000 kr. Majoriteten ansåg 88 kr vara det bästa förslaget, två föredrog 100 kr och ingen röstade för 1000 kr.

§14. **Fastställande av mötesdagar.**
Torsdagar valdes att som tidigare vara mötesdagar. Vårmöte 1988 skall

avhållas torsdagen 1988-02-25. Höstmöte 1988 skall avhållas torsdagen 1988-12-01. Månadsmöten avhålls den första torsdagen varje månad.

§15. **Bekräftande av uteslutningen av Sten Thure Eriksson.**
Sten Ture Eriksson redogjorde själv för vad han gjort. Det rör sig om oansvarigt uppträdande, som hotat att allvarligt skada föreningens relationer till KTH, samt skadegörelse på föreningens egendom.

Jan Michael Rynning begärde sluten omröstning, vilken resulterade i 19 röster för och 10 röster emot uteslutning. Tre röster var blanka och en var ogiltig.

§16. **Motioner.**

Motion om rökning.
Mötet anser med stark majoritet att rökning på möten och i föreningens lokaler inte skall tillåtas. (Om huruvida rökning av datorer skall tillåtas beslutades inget.)

Motion om medlemsavgifter.
Beslut fattades om att tillämpa de regler som gällde fram till 1982: *"Vid beviljad ansökan om nytt medlemsskap under kalenderårets tre sista månader gäller inbetald årsavgift även för efterföljande kalenderår."*

Motion om ersättare.
Motionen antogs en första gång. Stadgarna förenklas genom att begreppet ersättare avskaffas. Första meningen i §16 ändras till *"Styrelsen skall bestå av minst fem ledamöter."*, uttrycket *"eller ersättare"* stryks i §18 och ordet *"styrelsemedlem"* ändras till *"styrelseledamot"* i §10 och §19. De valda ersättarna blir styrelseledamöter när ändringen träder i kraft.

Motion om fastställande av tidpunkt för höst- och vårmöte.
Motionen antogs en första gång. Stadgarna förenklas genom att bestämmelsen avskaffas. Meningen *"Tidpunkt för höst/vårmöte fastställs av föreningsmöte."* stryks i §14/15.

Motion on deltagande i NUCC.
Motionen antogs. Föreningens deltagande i NUCC (Nordic University Computer Clubs) bekräftades formellt.

§17. Övriga frågor.

Klaus Zeuge undrar vem som äger de datorer som föreningen använder. Peter Löthberg klargör att av de maskiner som står i CCCC är bara Kicki hans, och att om någon vill hålla med ström och kyla till någon av STACKENs maskiner som står där, så går detta bra.

KTHNET har installerat modem, genom vilka man kan nå STACKENs datorer:

08-21 53 45: 300/300 och 1200/75 baud
08-21 57 45: 1200/1200 baud

§18. Mötets avslutande.

Stellan Lagerström avslutar mötet efter 1 timme och 45 minuter. Invigning av de nya lokalerna påbörjades.

Vid protokollet

Olle Betzén

Olle Betzén, mötessekreterare

Justeras

Henrik Björkman

Henrik Björkman, justeringsman

Justeras

Stellan Lagerström

Stellan Lagerström, mötesordförande

Justeras

Henning Croona

Henning Croona, justeringsman

Beskrivning av den fantastiska och enda SuperMAILER

av Thomas Nyström

PÅ KICKI finns ett ihoptejpat MAIL-system, som kan skicka och ta emot MAIL, både lokalt och till/från olika nät. Till båda delarna används kommandot MAIL.

Hur skickar man?

Vill man skicka ett brev lokalt skriver man t.ex.:

.MAIL *username*

Programmet MAIL kommer då att startas och frågar efter 'Subject:', trycker man bara <CR> på denna fråga kommer det inte att läggas in någon ärenderad i brevet. Man skriver därefter sin lilla (eller stora text) och trycker sedan Ctrl-Z. När brevet har skickats iväg kommer en bekräftelse på detta att skrivas ut. Om man ville skicka ett brev utanför KICKI skriver man:

.MAIL *username@address*

address är då den adress där användaren finns, antingen en domänadress eller namnet på en nod på DECnet. Det går inte att ange någon generell form hur man skriver adresser eftersom hela systemet med brev mellan olika nät är ett stort lapptäcke...

Medan man skriver sitt brev i MAIL kan man stoppa in en fil i brevet, detta görs genom att trycka Ctrl-G, man får då en fråga om filnamn. Om man ska skicka ett brev till flera mottagare går det bra att ange flera mottagare skilda åt med komma, fungerar både lokalt och ut från datorn.

Hur tar man emot?

Man läser sina brev genom att ge kommandot MAIL utan några argument, har man inte några brev kommer MAIL tala om det, om man har brev kommer programmet MSGH att startas för att man ska kunna läsa sina brev. När MSGH har startat kommer en lista

på de brev man har i sin brevlåda att skrivas ut samt ärenderaden (om det finns en sådan). Därefter promptar MSGH med en '*', man kan nu ge olika kommandon, de grundläggande är: Ctrl-J, ESCAPE, TYPE, DELETE, MAIL, ANSWER och EXIT. Här följer en kort beskrivning av dem:

Ctrl-J: Skriv ut nästa brev.

ESCAPE: (Tangenten ESCAPE) Skriver ut föregående brev.

TYPE #: Skriv ut brev, # är ordningsnummret bland dina brev, om man utelämnar ordningsnummret skrivs samma brev ut som man läste sist eller första brevet om man inte har läst något ännu.

DELETE #: Tag bort ett brev, # anger vilket eller om man utelämnar det så tas sist lästa brev bort.

MAIL *namn*: Används för att skicka ett brev medan man läser sina egna brev.

ANSWER: Används för att skicka ett svar till avsändaren av det brev man just har läst.

EXIT: Avslutar MSGH, först nu kommer de brev som man har tagit bort att försvinna, lämnar man programmet med Ctrl-C kommer inget brev att tas bort.

Dessutom finns ytterligare ett antal kommandon, man kan ge kommandot HELP till MSGH, så får man lite allmän info eller HELP * så listas en kort beskrivning av alla kommandon. Om man har några brev lagrade i sin mailfil kommer ett meddelande om detta att skrivas ut då man loggar in.

Hur får jag ett username?

För att man ska kunna använda MAIL måste man ha ett username. Det får man genom att köra programmet USERS (.R USERS) och svara på frågorna. Man kan ha flera olika username, men denna möjlighet finns inte i MS (mer om detta längre fram) om vi kommer att gå över till detta system. Det första username som man har är det namn som används då man skickar brev.

Vad har jag för adress?

När man fått ett username kan man ta emot brev under adressen:

username@stacken.kth.se

Från en VMS-maskin på DECnet som inte klarar domänadresser skriver man till:

KTH::"*username*@stacken.kth.se"

eller till:

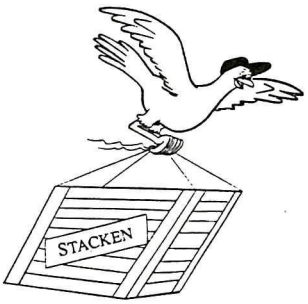
KICKI::*username*

Hur funkar det?

Det finns till alla TOPS-10-datorer ett 'riktigt' MAIL-system kallat MS, mig veterligen fungerar det inte på någon TOPS-10-maskin än... På KICKI består MAIL-systemet av sex olika program: tre demoner och nyss beskrivna MAIL, MSGH och USERS. Alla brev finns på [3,5] under namnet programmerarnummer.MAI, när man skickar ett brev lokalt kommer MAIL själv att skriva in brevet i rätt fil, skickar man däremot ett brev ut på nätet kommer brevet att läggas i en fil på [3,5] som heter nummer.POS, nummer är bara ett löpnummer för att skilja på breven. Denna fil läses av demonen SMTSND som skickar brevet vidare till VERA, VERA kommer sedan att skicka det till önskad dator. Brev som kommer från andra datorer tas emot av demonen M11SRV, brevet läggs i en fil på [3,5] med extension .REC. Denna fil läses av demonen MAIRTR, MAIRTR kommer sedan att skicka brevet till rätt brevlåda. Om VERA inte accepterar ett brev så kommer SMTSND skriva en rapport om detta i en fil med extension .EBB på [3,5], även denna fil läses av MAIRTR och skickas till avsändaren av det ursprungliga brevet samt till POSTMASTER (som är undertecknad). Då ett brev har processats av SMTSND kommer extension på filen med brevet att ändras till .SNT, MAIRTR gör på samma sätt men ändrar till .OK om brevet gick att skicka vidare eller till .REJ om det inte gick, i det senare fallet skickas även en rapport till POSTMASTER.

Vi har "flyttat"

av Jan Michael Rynning



STACKEN har bytt electronic mail-adress till föreningen, nu när Thomas Nyström implementerat mailinglistor i SuperMAILER. Den nya electronic mail-adressen är:

stacken@stacken.kth.se

Computer Science Facilities Group
Rutgers University, New Jersey

Charles L. Hedrick
3 July 1987

Introduction to the Internet Protocols

THIS is an introduction to the Internet networking protocols (TCP/IP). It includes a summary of the facilities available and brief descriptions of the major protocols in the family.

Table of Contents

This document is a brief introduction to TCP/IP, followed by advice on what to read for more information. This is not intended to be a complete description. It can give you a reasonable idea of the capabilities of the protocols. But if you need to know any details of the technology, you will want to read the standards yourself. Throughout the text, you will find references to the standards, in the form of "RFC" or "IEN" numbers. These are document numbers. The final section of this document tells you how to get copies of those standards.

1. What is TCP/IP?

TCP/IP is a set of protocols developed to allow cooperating computers to share resources across a network. It was developed by a community of researchers centered around the ARPAnet. Certainly the ARPAnet is the best-known TCP/IP network. However as of June, 87, at least 130 different vendors had products that support TCP/IP, and thousands of networks of all kinds use it.

First some basic definitions. The most accurate name for the set of protocols we are describing is the "Internet protocol suite". TCP and IP are two of the protocols in this suite. (They will be described below.) Because TCP and IP are the best known of the protocols, it has become common to use the term TCP/IP or IP/TCP to refer to the whole family. It is probably not worth fighting this habit. However this can lead to some oddities. For example, I find myself talking about NFS as being based on TCP/IP, even though it doesn't use TCP at all. (It does use IP. But it uses an alternative protocol, UDP, instead of TCP. All of this alphabet soup will be unscrambled in the following pages.)

The Internet is a collection of networks, including the Arpanet, NSFnet, regional networks such as NYsernet, local networks at a number of University and research institutions, and a number of military networks. The term "Internet" applies to this entire set of networks. The subset of them that is managed by the Department of Defense is referred to as the "DDN" (Defense Data Network). This includes some research-oriented networks, such as the Arpanet, as well as more strictly military ones. (Because much of the funding for Internet protocol developments is done via the DDN organization, the terms Internet and DDN can sometimes seem equivalent.) All of these networks are connected to each other. Users can send messages from any of them to any other, except where there are security or other policy restrictions on access. Officially speaking, the Internet protocol documents are simply standards adopted by the Internet community for its own use. More recently, the Department of Defense issued a MILSPEC definition of TCP/IP. This was intended to be a more formal definition, appropriate

for use in purchasing specifications. However most of the TCP/IP community continues to use the Internet standards. The MILSPEC version is intended to be consistent with it.

Whatever it is called, TCP/IP is a family of protocols. A few provide "low-level" functions needed for many applications. These include IP, TCP, and UDP. (These will be described in a bit more detail later.) Others are protocols for doing specific tasks, e.g. transferring files between computers, sending mail, or finding out who is logged in on another computer. Initially TCP/IP was used mostly between minicomputers or mainframes. These machines had their own disks, and generally were self-contained. Thus the most important "traditional" TCP/IP services are:

- file transfer. The file transfer protocol (FTP) allows a user on any computer to get files from another computer, or to send files to another computer. Security is handled by requiring the user to specify a user name and password for the other computer. Provisions are made for handling file transfer between machines with different character set, end of line conventions, etc. This is not quite the same thing as more recent "network file system" or "netbios" protocols, which will be described below. Rather, FTP is a utility that you run any time you want to access a file on another system. You use it to copy the file to your own system. You then work with the local copy. (See RFC 959 for specifications for FTP.)

- remote login. The network terminal protocol (TELNET) allows a user to log in on any other computer on the network. You start a remote session by specifying a computer to connect to. From that time until you finish the session, anything you type is sent to the other computer. Note that you are really still talking to your own computer. But the telnet program effectively makes your computer invisible while it is running. Every character you type is sent directly to the other system. Generally, the connection to the remote computer behaves much like a dialup connection. That is, the remote system will ask you to log in and give a password, in whatever manner it would normally ask a user who had just dialed it up. When you log off of the other computer, the telnet program exits, and you will find yourself talking to your own computer. Microcomputer implementations of telnet generally include a terminal emulator for some common type of terminal. (See RFC's 854 and 855 for specifications for telnet. By the way, the telnet protocol should not be confused with Telenet, a vendor of commercial network services.)

- computer mail. This allows you to send messages to users on other computers. Originally, people tended to use only one or two specific computers. They would maintain "mail files" on those machines. The computer mail system is simply a way for you to add a message to another user's mail file. There are some problems with this in an environment where microcomputers are used. The most serious is that a micro is not well suited to receive computer mail. When you send mail, the mail software expects to be able to open a connection to the addressee's computer, in order to send the mail. If this is a microcomputer, it may be turned off, or it may be running an application other than the mail system. For this reason, mail is normally handled by a larger system, where it is practical to have a mail server running all the time. Microcomputer mail software then becomes a user interface that retrieves mail from the mail server. (See RFC 821 and 822 for specifications for computer mail. See RFC 937 for a protocol designed for microcomputers to use in reading mail from a mail server.)

These services should be present in any implementation of TCP/IP, except that micro-oriented implementations may not support computer mail. These traditional applications still play a very important role in TCP/IP-based networks. However more recently, the way in which networks are used has been changing. The older model of a number of large, self-sufficient computers is beginning to change. Now many installations have several kinds of computers, including microcomputers, workstations, minicomputers, and mainframes. These computers are likely to be configured to perform specialized tasks. Although people are still likely to work with one specific computer, that computer will call on other systems on the net for specialized services. This has led to the "server/client" model of network services. A server is a system that provides a specific service for the rest of the network. A client is another system that uses that service. (Note that the server and client need not be on different computers. They could be different programs running on the same computer.) Here are the kinds of servers typically present in a modern computer setup. Note that these computer services can all be provided within the framework of TCP/IP.

- network file systems. This allows a system to access files on another computer in a somewhat more closely integrated fashion than FTP. A network file system provides the illusion that disks or other devices from one system are directly connected to other systems. There is no need to use a special network utility to access a file on another system. Your computer simply thinks it has some extra disk drives. These extra "virtual" drives refer to the other system's disks. This capability is useful for several different purposes. It lets you put

large disks on a few computers, but still give others access to the disk space. Aside from the obvious economic benefits, this allows people working on several computers to share common files. It makes system maintenance and backup easier, because you don't have to worry about updating and backing up copies on lots of different machines. A number of vendors now offer high-performance diskless computers. These computers have no disk drives at all. They are entirely dependent upon disks attached to common "file servers". (See RFC's 1001 and 1002 for a description of PC-oriented NetBIOS over TCP. In the workstation and minicomputer area, Sun's Network File System is more likely to be used. Protocol specifications for it are available from Sun Microsystems.)

- remote printing. This allows you to access printers on other computers as if they were directly attached to yours. (The most commonly used protocol is the remote lineprinter protocol from Berkeley Unix. Unfortunately, there is no protocol document for this. However the C code is easily obtained from Berkeley, so implementations are common.)

- remote execution. This allows you to request that a particular program be run on a different computer. This is useful when you can do most of your work on a small computer, but a few tasks require the resources of a larger system. There are a number of different kinds of remote execution. Some operate on a command by command basis. That is, you request that a specific command or set of commands should run on some specific computer. (More sophisticated versions will choose a system that happens to be free.) However there are also "remote procedure call" systems that allow a program to call a subroutine that will run on another computer. (There are many protocols of this sort. Berkeley Unix contains two servers to execute commands remotely: rsh and rexec. The man pages describe the protocols that they use. The user-contributed software with Berkeley 4.3 contains a "distributed shell" that will distribute tasks among a set of systems, depending upon load. Remote procedure call mechanisms have been a topic for research for a number of years, so many organizations have implementations of such facilities. The most widespread commercially-supported remote procedure call protocols seem to be Xerox's Courier and Sun's RPC. Protocol documents are available from Xerox and Sun. There is a public implementation of Courier over TCP as part of the user-contributed software with Berkeley 4.3. An implementation of RPC was posted to Usenet by Sun, and also appears as part of the user-contributed software with Berkeley 4.3.)

- name servers. In large installations, there are a number of different collections of names that have to be managed. This includes users and their passwords, names and network addresses for computers, and accounts. It becomes very tedious to keep this data up to date on all of the computers. Thus the databases are kept on a small number of systems. Other systems access the data over the network. (RFC 822 and 823 describe the name server protocol used to keep track of host names and Internet addresses on the Internet. This is now a required part of any TCP/IP implementation. IEN 116 describes an older name server protocol that is used by a few terminal servers and other products to look up host names. Sun's Yellow Pages system is designed as a general mechanism to handle user names, file sharing groups, and other databases commonly used by Unix systems. It is widely available commercially. Its protocol definition is available from Sun.)

- terminal servers. Many installations no longer connect terminals directly to computers. Instead they connect them to terminal servers. A terminal server is simply a small computer that only knows how to run telnet (or some other protocol to do remote login). If your terminal is connected to one of these, you simply type the name of a computer, and you are connected to it. Generally it is possible to have active connections to more than one computer at the same time. The terminal server will have provisions to switch between connections rapidly, and to notify you when output is waiting for another connection. (Terminal servers use the telnet protocol, already mentioned. However any real terminal server will also have to support name service and a number of other protocols.)

- network-oriented window systems. Until recently, high-performance graphics programs had to execute on a computer that had a bit-mapped graphics screen directly attached to it. Network window systems allow a program to use a display on a different computer. Full-scale network window systems provide an interface that lets you distribute jobs to the systems that are best suited to handle them, but still give you a single graphically-based user interface. (The most widely-implemented window system is X. A protocol description is available from MIT's Project Athena. A reference implementation is publically available from MIT. A number of vendors are also supporting NeWS, a window system defined by Sun. Both of these systems are designed to use TCP/IP.)

Note that some of the protocols described above were designed by Berkeley, Sun, or other organizations. Thus they are not officially part of the Internet pro-

tocol suite. However they are implemented using TCP/IP, just as normal TCP/IP application protocols are. Since the protocol definitions are not considered proprietary, and since commercially-support implementations are widely available, it is reasonable to think of these protocols as being effectively part of the Internet suite. Note that the list above is simply a sample of the sort of services available through TCP/IP. However it does contain the majority of the "major" applications. The other commonly-used protocols tend to be specialized facilities for getting information of various kinds, such as who is logged in, the time of day, etc. However if you need a facility that is not listed here, we encourage you to look through the current edition of Internet Protocols (currently RFC 1011), which lists all of the available protocols, and also to look at some of the major TCP/IP implementations to see what various vendors have added.

2. General description of the TCP/IP protocols

TCP/IP is a layered set of protocols. In order to understand what this means, it is useful to look at an example. A typical situation is sending mail. First, there is a protocol for mail. This defines a set of commands which one machine sends to another, e.g. commands to specify who the sender of the message is, who it is being sent to, and then the text of the message. However this protocol assumes that there is a way to communicate reliably between the two computers. Mail, like other application protocols, simply defines a set of commands and messages to be sent. It is designed to be used together with TCP and IP. TCP is responsible for making sure that the commands get through to the other end. It keeps track of what is sent, and retransmitts anything that did not get through. If any message is too large for one datagram, e.g. the text of the mail, TCP will split it up into several datagrams, and make sure that they all arrive correctly. Since these functions are needed for many applications, they are put together into a separate protocol, rather than being part of the specifications for sending mail. You can think of TCP as forming a library of routines that applications can use when they need reliable network communications with another computer. Similarly, TCP calls on the services of IP. Although the services that TCP supplies are needed by many applications, there are still some kinds of applications that don't need them. However there are some services that every application needs. So these services are put together into IP. As with TCP, you can think of IP as a library of routines that TCP calls on, but which is also available to applications that don't use TCP. This strategy of building several levels of protocol is called "layering". We think of the applications programs such as mail, TCP, and IP, as being separate "layers", each of which calls on the services of the layer below it. Generally, TCP/IP applications use 4 layers:

- an application protocol such as mail
- a protocol such as TCP that provides services need by many applications
- IP, which provides the basic service of getting datagrams to their destination
- the protocols needed to manage a specific physical medium, such as Ethernet or a point to point line.

TCP/IP is based on the "catenet model". (This is described in more detail in IEN 48.) This model assumes that there are a large number of independent networks connected together by gateways. The user should be able to access computers or other resources on any of these networks. Datagrams will often pass through a dozen different networks before getting to their final destination. The routing needed to accomplish this should be completely invisible to the user. As far as the user is concerned, all he needs to know in order to access another system is an "Internet address". This is an address that looks like 128.6.4.194. It is actually a 32-bit number. However it is normally written as 4 decimal numbers, each representing 8 bits of the address. (The term "octet" is used by Internet documentation for such 8-bit chunks. The term "byte" is not used, because TCP/IP is supported by some computers that have byte sizes other than 8 bits.) Generally the structure of the address gives you some information about how to get to the system. For example, 128.6 is a network number assigned by a central authority to Rutgers University. Rutgers uses the next octet to indicate which of the campus Ethernets is involved. 128.6.4 happens to be an Ethernet used by the Computer Science Department. The last octet allows for up to 254 systems on each Ethernet. (It is 254 because 0 and 255 are not allowed, for reasons that will be discussed later.) Note that 128.6.4.194 and 128.6.5.194 would be different systems. The structure of an Internet address is described in a bit more detail later.

Of course we normally refer to systems by name, rather than by Internet address. When we specify a name, the network software looks it up in a database, and comes up with the corresponding Internet address. Most of the network software deals strictly in terms of the address. (RFC 882 describes the name server technology used to handle this lookup.)

TCP/IP is built on "connectionless" technology. Information is transfered as a sequence of "datagrams". A datagram is a collection of data that is sent as a single message. Each of these datagrams is sent through the network individu-

ally. There are provisions to open connections (i.e. to start a conversation that will continue for some time). However at some level, information from those connections is broken up into datagrams, and those datagrams are treated by the network as completely separate. For example, suppose you want to transfer a 15000 octet file. Most networks can't handle a 15000 octet datagram. So the protocols will break this up into something like 30 500-octet datagrams. Each of these datagrams will be sent to the other end. At that point, they will be put back together into the 15000-octet file. However while those datagrams are in transit, the network doesn't know that there is any connection between them. It is perfectly possible that datagram 14 will actually arrive before datagram 13. It is also possible that somewhere in the network, an error will occur, and some datagram won't get through at all. In that case, that datagram has to be sent again.

Note by the way that the terms "datagram" and "packet" often seem to be nearly interchangable. Technically, datagram is the right word to use when describing TCP/IP. A datagram is a unit of data, which is what the protocols deal with. A packet is a physical thing, appearing on an Ethernet or some wire. In most cases a packet simply contains a datagram, so there is very little difference. However they can differ. When TCP/IP is used on top of X.25, the X.25 interface breaks the datagrams up into 128-byte packets. This is invisible to IP, because the packets are put back together into a single datagram at the other end before being processed by TCP/IP. So in this case, one IP datagram would be carried by several packets. However with most media, there are efficiency advantages to sending one datagram per packet, and so the distinction tends to vanish.

2.1. The TCP level

Two separate protocols are involved in handling TCP/IP datagrams. TCP (the "transmission control protocol") is responsible for breaking up the message into datagrams, reassembling them at the other end, resending anything that gets lost, and putting things back in the right order. IP (the "internet protocol") is responsible for routing individual datagrams. It may seem like TCP is doing all the work. And in small networks that is true. However in the Internet, simply getting a datagram to its destination can be a complex job. A connection may require the datagram to go through several networks at Rutgers, a serial line to the John von Neuman Supercomputer Center, a couple of Ethernets there, a series of 56Kbaud phone lines to another NSFnet site, and more Ethernets on another campus. Keeping track of the routes to all of the destinations and handling incompatibilities among different transport media turns out to be a complex job.

Note that the interface between TCP and IP is fairly simple. TCP simply hands IP a datagram with a destination. IP doesn't know how this datagram relates to any datagram before it or after it.

It may have occurred to you that something is missing here. We have talked about Internet addresses, but not about how you keep track of multiple connections to a given system. Clearly it isn't enough to get a datagram to the right destination. TCP has to know which connection this datagram is part of. This task is referred to as "demultiplexing". In fact, there are several levels of demultiplexing going on in TCP/IP. The information needed to do this demultiplexing is contained in a series of "headers". A header is simply a few extra octets tacked onto the beginning of a datagram by some protocol in order to keep track of it. It's a lot like putting a letter into an envelope and putting an address on the outside of the envelope. Except with modern networks it happens several times. It's like you put the letter into a little envelope, your secretary puts that into a somewhat bigger envelope, the campus mail center puts that envelope into a still bigger one, etc. Here is an overview of the headers that get stuck on a message that passes through a typical TCP/IP network:

We start with a single data stream, say a file you are trying to send to some other computer:

.....

TCP breaks it up into manageable chunks. (In order to do this, TCP has to know how large a datagram your network can handle. Actually, the TCP's at each end say how big a datagram they can handle, and then they pick the smallest size.)

....

TCP puts a header at the front of each datagram. This header actually contains at least 20 octets, but the most important ones are a source and destination "port number" and a "sequence number". The port numbers are used to keep track of different conversations. Suppose 3 different people are transferring files. Your TCP might allocate port numbers 1000, 1001, and 1002 to these transfers. When you are sending a datagram, this becomes the "source" port number, since you are the source of the datagram. Of course the TCP at the other end has assigned a port number of its own for the conversation. Your TCP has to know the port number used by the other end as well. (It finds out when the connection starts, as we will explain below.) It puts this in the "destination" port field. Of course if the other

end sends a datagram back to you, the source and destination port numbers will be reversed, since then it will be the source and you will be the destination. Each datagram has a sequence number. This is used so that the other end can make sure that it gets the datagrams in the right order, and that it hasn't missed any. (See the TCP specification for details.) TCP doesn't number the datagrams, but the octets. So if there are 500 octets of data in each datagram, the first datagram might be numbered 0, the second 500, the next 1000, the next 1500, etc. Finally, I will mention the Checksum. This is a number that is computed by adding up all the octets in the datagram (more or less—see the TCP spec). The result is put in the header. TCP at the other end computes the checksum again. If they disagree, then something bad happened to the datagram in transmission, and it is thrown away. So here's what the datagram looks like now.

<table>
<tr><td colspan="4">Source Port</td><td colspan="4">Destination Port</td></tr>
<tr><td colspan="8">Sequence Number</td></tr>
<tr><td colspan="8">Acknowledgment Number</td></tr>
<tr><td>Data Offset</td><td colspan="2">Reserved</td><td>URG</td><td>ACK</td><td>PSH</td><td>RST</td><td>SYN</td><td>FIN</td><td colspan="2">Window</td></tr>
<tr><td colspan="5">Checksum</td><td colspan="3">Urgent Pointer</td></tr>
<tr><td colspan="8">your data—next 500 octets
...</td></tr>
</table>

If we abbreviate the TCP header as "T", the whole file now looks like this:

T.... T.... T.... T.... T.... T....

You will note that there are items in the header that I have not described above. They are generally involved with managing the connection. In order to make sure the datagram has arrived at its destination, the recipient has to send back an "acknowledgement". This is a datagram whose "Acknowledgement number" field is filled in. For example, sending a packet with an acknowledgement of 1500 indicates that you have received all the data up to octet number 1500. If the sender doesn't get an acknowledgement within a reasonable amount of time, it sends the data again. The window is used to control how much data can be in transit at any one time. It is not practical to wait for each datagram to be acknowledged before sending the next one. That would slow things down too much. On the other hand, you can't just keep sending, or a fast computer might overrun the capacity of a slow one to absorb data. Thus each end indicates how much new data it is currently prepared to absorb by putting the number of octets

in its "Window" field. As the computer receives data, the amount of space left in its window decreases. When it goes to zero, the sender has to stop. As the receiver processes the data, it increases its window, indicating that it is ready to accept more data. Often the same datagram can be used to acknowledge receipt of a set of data and to give permission for additional new data (by an updated window). The "Urgent" field allows one end to tell the other to skip ahead in its processing to a particular octet. This is often useful for handling asynchronous events, for example when you type a control character or other command that interrupts output. The other fields are beyond the scope of this document.

2.2. The IP level

TCP sends each of these datagrams to IP. Of course it has to tell IP the Internet address of the computer at the other end. Note that this is all IP is concerned about. It doesn't care about what is in the datagram, or even in the TCP header. IP's job is simply to find a route for the datagram and get it to the other end. In order to allow gateways or other intermediate systems to forward the datagram, it adds its own header. The main things in this header are the source and destination Internet address (32-bit addresses, like 128.6.4.194), the protocol number, and another checksum. The source Internet address is simply the address of your machine. (This is necessary so the other end knows where the datagram came from.) The destination Internet address is the address of the other machine. (This is necessary so any gateways in the middle know where you want the datagram to go.) The protocol number tells IP at the other end to send the datagram to TCP. Although most IP traffic uses TCP, there are other protocols that can use IP, so you have to tell IP which protocol to send the datagram to. Finally, the checksum allows IP at the other end to verify that the header wasn't damaged in transit. Note that TCP and IP have separate checksums. IP needs to be able to verify that the header didn't get damaged in transit, or it could send a message to the wrong place. For reasons not worth discussing here, it is both more efficient and safer to have TCP compute a separate checksum for the TCP header and data. Once IP has tacked on its header, here's what the message looks like:

<table>
<tr><td>Version</td><td>IHL</td><td>Type of Service</td><td colspan="2">Total Length</td></tr>
<tr><td colspan="3">Identification</td><td>Flags</td><td>Fragment Offset</td></tr>
<tr><td colspan="2">Time to Live</td><td>Protocol</td><td colspan="2">Header Checksum</td></tr>
<tr><td colspan="5">Source Address</td></tr>
<tr><td colspan="5">Destination Address</td></tr>
<tr><td colspan="5">TCP header, then your data
...</td></tr>
</table>

If we represent the IP header by an "I", your file now looks like this:

IT.... IT.... IT.... IT.... IT.... IT....

Again, the header contains some additional fields that have not been discussed. Most of them are beyond the scope of this document. The flags and fragment offset are used to keep track of the pieces when a datagram has to be split up. This can happen when datagrams are forwarded through a network for which they are too big. (This will be discussed a bit more below.) The time to live is a number that is decremented whenever the datagram passes through a system. When it goes to zero, the datagram is discarded. This is done in case a loop develops in the system somehow. Of course this should be impossible, but well-designed networks are built to cope with "impossible" conditions.

At this point, it's possible that no more headers are needed. If your computer happens to have a direct phone line connecting it to the destination computer, or to a gateway, it may simply send the datagrams out on the line (though likely a synchronous protocol such as HDLC would be used, and it would add at least a few octets at the beginning and end).

2.3. The Ethernet level

However most of our networks these days use Ethernet. So now we have to describe Ethernet's headers. Unfortunately, Ethernet has its own addresses. The people who designed Ethernet wanted to make sure that no two machines would end up with the same Ethernet address. Furthermore, they didn't want the user to have to worry about assigning addresses. So each Ethernet controller comes with an address builtin from the factory. In order to make sure that they would never have to reuse addresses, the Ethernet designers allocated 48 bits for the Ethernet address. People who make Ethernet equipment have to register with a central authority, to make sure that the numbers they assign don't overlap any other manufacturer. Ethernet is a "broadcast medium". That is, it is in effect like an old party line telephone. When you send a packet out on the Ethernet, every machine on the network sees the packet. So something is needed to make sure that the right machine gets it. As you might guess, this involves the Ethernet header. Every Ethernet packet has a 14-octet header that includes the source and destination Ethernet address, and a type code. Each machine is supposed to pay attention only to packets with its own Ethernet address in the destination field. (It's perfectly possible to cheat, which is one reason that Ethernet communications are not terribly secure.) Note that there is no connection between the

Ethernet address and the Internet address. Each machine has to have a table of what Ethernet address corresponds to what Internet address. (We will describe how this table is constructed a bit later.) In addition to the addresses, the header contains a type code. The type code is to allow for several different protocol families to be used on the same network. So you can use TCP/IP, DECnet, Xerox NS, etc. at the same time. Each of them will put a different value in the type field. Finally, there is a checksum. The Ethernet controller computes a checksum of the entire packet. When the other end receives the packet, it recomputes the checksum, and throws the packet away if the answer disagrees with the original. The checksum is put on the end of the packet, not in the header. The final result is that your message looks like this:

<table>
<tr><td colspan="2">Ethernet Destination Address (first 32 bits)</td></tr>
<tr><td>Ethernet Destination (last 16 bits)</td><td>Ethernet Source (first 16 bits)</td></tr>
<tr><td colspan="2">Ethernet Source Address (last 32 bits)</td></tr>
<tr><td>Type Code</td><td></td></tr>
<tr><td colspan="2">IP header, then TCP header, then your data
...
end of your data</td></tr>
<tr><td colspan="2">Ethernet Checksum</td></tr>
</table>

If we represent the Ethernet header with "E", and the Ethernet checksum with "C", your file now looks like this:

EIT....C EIT....C EIT....C EIT....C EIT....C EIT....C

When these packets are received by the other end, of course all the headers are removed. The Ethernet interface removes the Ethernet header and the checksum. It looks at the type code. Since the type code is the one assigned to IP, the Ethernet device driver passes the datagram up to IP. IP removes the IP header. It looks at the IP protocol field. Since the protocol type is TCP, it passes the datagram up to TCP. TCP now looks at the sequence number. It uses the sequence numbers and other information to combine all the datagrams into the original file.

The ends our initial summary of TCP/IP. There are still some crucial concepts we haven't gotten to, so we'll now go back and add details in several areas. (For detailed descriptions of the items discussed here see, RFC 793 for TCP, RFC 791 for IP, and RFC's 894 and 826 for sending IP over Ethernet.)

3. Well-known sockets and the applications layer

So far, we have described how a stream of data is broken up into datagrams, sent to another computer, and put back together. However something more is needed in order to accomplish anything useful. There has to be a way for you to open a connection to a specified computer, log into it, tell it what file you want, and control the transmission of the file. (If you have a different application in mind, e.g. computer mail, some analogous protocol is needed.) This is done by "application protocols". The application protocols run "on top" of TCP/IP. That is, when they want to send a message, they give the message to TCP. TCP makes sure it gets delivered to the other end. Because TCP and IP take care of all the networking details, the applications protocols can treat a network connection as if it were a simple byte stream, like a terminal or phone line.

Before going into more details about applications programs, we have to describe how you find an application. Suppose you want to send a file to a computer whose Internet address is 128.6.4.7. To start the process, you need more than just the Internet address. You have to connect to the FTP server at the other end. In general, network programs are specialized for a specific set of tasks. Most systems have separate programs to handle file transfers, remote terminal logins, mail, etc. When you connect to 128.6.4.7, you have to specify that you want to talk to the FTP server. This is done by having "well-known sockets" for each server. Recall that TCP uses port numbers to keep track of individual conversations. User programs normally use more or less random port numbers. However specific port numbers are assigned to the programs that sit waiting for requests. For example, if you want to send a file, you will start a program called "ftp". It will open a connection using some random number, say 1234, for the port number on its end. However it will specify port number 21 for the other end. This is the official port number for the FTP server. Note that there are two different programs involved. You run ftp on your side. This is a program designed to accept commands from your terminal and pass them on to the other end. The program that you talk to on the other machine is the FTP server. It is designed to accept commands from the network connection, rather than an interactive terminal. There is no need for your program to use a well-known socket number for itself. Nobody is trying to find it. However the servers have to have well-known numbers, so that people can open connections to them and start sending them commands. The official port numbers for each program are given in "Assigned Numbers".

Note that a connection is actually described by a set of 4 numbers: the Internet

address at each end, and the TCP port number at each end. Every datagram has all four of those numbers in it. (The Internet addresses are in the IP header, and the TCP port numbers are in the TCP header.) In order to keep things straight, no two connections can have the same set of numbers. However it is enough for any one number to be different. For example, it is perfectly possible for two different users on a machine to be sending files to the same other machine. This could result in connections with the following parameters:

	Internet addresses	TCP ports
Connection 1	128.6.4.194, 128.6.4.7	1234, 21
Connection 2	128.6.4.194, 128.6.4.7	1235, 21

Since the same machines are involved, the Internet addresses are the same. Since they are both doing file transfers, one end of the connection involves the well-known port number for FTP. The only thing that differs is the port number for the program that the users are running. That's enough of a difference. Generally, at least one end of the connection asks the network software to assign it a port number that is guaranteed to be unique. Normally, it's the user's end, since the server has to use a well-known number.

Now that we know how to open connections, let's get back to the applications programs. As mentioned earlier, once TCP has opened a connection, we have something that might as well be a simple wire. All the hard parts are handled by TCP and IP. However we still need some agreement as to what we send over this connection. In effect this is simply an agreement on what set of commands the application will understand, and the format in which they are to be sent. Generally, what is sent is a combination of commands and data. They use context to differentiate. For example, the mail protocol works like this: Your mail program opens a connection to the mail server at the other end. Your program gives it your machine's name, the sender of the message, and the recipients you want it sent to. It then sends a command saying that it is starting the message. At that point, the other end stops treating what it sees as commands, and starts accepting the message. Your end then starts sending the text of the message. At the end of the message, a special mark is sent (a dot in the first column). After that, both ends understand that your program is again sending commands. This is the simplest way to do things, and the one that most applications use.

File transfer is somewhat more complex. The file transfer protocol involves two different connections. It starts out just like mail. The user's program sends commands like "log me in as this user", "here is my password", "send me the

file with this name". However once the command to send data is sent, a second connection is opened for the data itself. It would certainly be possible to send the data on the same connection, as mail does. However file transfers often take a long time. The designers of the file transfer protocol wanted to allow the user to continue issuing commands while the transfer is going on. For example, the user might make an inquiry, or he might abort the transfer. Thus the designers felt it was best to use a separate connection for the data and leave the original command connection for commands. (It is also possible to open command connections to two different computers, and tell them to send a file from one to the other. In that case, the data couldn't go over the command connection.)

Remote terminal connections use another mechanism still. For remote logins, there is just one connection. It normally sends data. When it is necessary to send a command (e.g. to set the terminal type or to change some mode), a special character is used to indicate that the next character is a command. If the user happens to type that special character as data, two of them are sent.

We are not going to describe the application protocols in detail in this document. It's better to read the RFC's yourself. However there are a couple of common conventions used by applications that will be described here. First, the common network representation: TCP/IP is intended to be usable on any computer. Unfortunately, not all computers agree on how data is represented. There are differences in character codes (ASCII vs. EBCDIC), in end of line conventions (carriage return, line feed, or a representation using counts), and in whether terminals expect characters to be sent individually or a line at a time. In order to allow computers of different kinds to communicate, each applications protocol defines a standard representation. Note that TCP and IP do not care about the representation. TCP simply sends octets. However the programs at both ends have to agree on how the octets are to be interpreted. The RFC for each application specifies the standard representation for that application. Normally it is "net ASCII". This uses ASCII characters, with end of line denoted by a carriage return followed by a line feed. For remote login, there is also a definition of a "standard terminal", which turns out to be a half-duplex terminal with echoing happening on the local machine. Most applications also make provisions for the two computers to agree on other representations that they may find more convenient. For example, PDP-10's have 36-bit words. There is a way that two PDP-10's can agree to send a 36-bit binary file. Similarly, two systems that prefer full-duplex terminal conversations can agree on that. However each application has a standard representation, which every machine must support.

3.1. An example application: SMTP

In order to give a bit better idea what is involved in the application protocols, I'm going to show an example of SMTP, which is the mail protocol. (SMTP is "simple mail transfer protocol".) We assume that a computer called TOPAZ.RUTGERS.EDU wants to send the following message.

```
Date: Sat, 27 Jun 87 13:26:31 EDT
From: hedrick@topaz.rutgers.edu
To: levy@red.rutgers.edu
Subject: meeting

Let's get together Monday at 1pm.
```

First, note that the format of the message itself is described by an Internet standard (RFC 822). The standard specifies the fact that the message must be transmitted as net ASCII (i.e. it must be ASCII, with carriage return/linefeed to delimit lines). It also describes the general structure, as a group of header lines, then a blank line, and then the body of the message. Finally, it describes the syntax of the header lines in detail. Generally they consist of a keyword and then a value.

Note that the addressee is indicated as LEVY@RED.RUTGERS.EDU. Initially, addresses were simply "person at machine". However recent standards have made things more flexible. There are now provisions for systems to handle other systems' mail. This can allow automatic forwarding on behalf of computers not connected to the Internet. It can be used to direct mail for a number of systems to one central mail server. Indeed there is no requirement that an actual computer by the name of RED.RUTGERS.EDU even exist. The name servers could be set up so that you mail to department names, and each department's mail is routed automatically to an appropriate computer. It is also possible that the part before the @ is something other than a user name. It is possible for programs to be set up to process mail. There are also provisions to handle mailing lists, and generic names such as "postmaster" or "operator".

The way the message is to be sent to another system is described by RFC's 821 and 974. The program that is going to be doing the sending asks the name server several queries to determine where to route the message. The first query is to find out which machines handle mail for the name RED.RUTGERS.EDU. In this case, the server replies that RED.RUTGERS.EDU handles its own mail. The

program then asks for the address of RED.RUTGERS.EDU, which is 128.6.4.2. Then the mail program opens a TCP connection to port 25 on 128.6.4.2. Port 25 is the well-known socket used for receiving mail. Once this connection is established, the mail program starts sending commands. Here is a typical conversation. Each line is labelled as to whether it is from TOPAZ or RED. Note that TOPAZ initiated the connection:

```
RED     220 RED.RUTGERS.EDU SMTP Service at 29 Jun 87 05:17:18 EDT
TOPAZ   HELO topaz.rutgers.edu
RED     250 RED.RUTGERS.EDU - Hello, TOPAZ.RUTGERS.EDU
TOPAZ   MAIL From:<hedrick@topaz.rutgers.edu>
RED     250 MAIL accepted
TOPAZ   RCPT To:<levy@red.rutgers.edu>
RED     250 Recipient accepted
TOPAZ   DATA
RED     354 Start mail input; end with <CRLF>.<CRLF>
TOPAZ   Date: Sat, 27 Jun 87 13:26:31 EDT
TOPAZ   From: hedrick@topaz.rutgers.edu
TOPAZ   To: levy@red.rutgers.edu
TOPAZ   Subject: meeting
TOPAZ
TOPAZ   Let's get together Monday at 1pm.
TOPAZ   .
RED     250 OK
TOPAZ   QUIT
RED     221 RED.RUTGERS.EDU Service closing transmission channel
```

First, note that commands all use normal text. This is typical of the Internet standards. Many of the protocols use standard ASCII commands. This makes it easy to watch what is going on and to diagnose problems. For example, the mail program keeps a log of each conversation. If something goes wrong, the log file can simply be mailed to the postmaster. Since it is normal text, he can see what was going on. It also allows a human to interact directly with the mail server, for testing. (Some newer protocols are complex enough that this is not practical. The commands would have to have a syntax that would require a significant parser. Thus there is a tendency for newer protocols to use binary formats. Generally they are structured like C or Pascal record structures.) Second, note that the responses all begin with numbers. This is also typical of Internet protocols. The allowable responses are defined in the protocol. The numbers allow the user program to respond unambiguously. The rest of the response is text, which is

normally for use by any human who may be watching or looking at a log. It has no effect on the operation of the programs. (However there is one point at which the protocol uses part of the text of the response.) The commands themselves simply allow the mail program on one end to tell the mail server the information it needs to know in order to deliver the message. In this case, the mail server could get the information by looking at the message itself. But for more complex cases, that would not be safe. Every session must begin with a HELO, which gives the name of the system that initiated the connection. Then the sender and recipients are specified. (There can be more than one RCPT command, if there are several recipients.) Finally the data itself is sent. Note that the text of the message is terminated by a line containing just a period. (If such a line appears in the message, the period is doubled.) After the message is accepted, the sender can send another message, or terminate the session as in the example above.

Generally, there is a pattern to the response numbers. The protocol defines the specific set of responses that can be sent as answers to any given command. However programs that don't want to analyze them in detail can just look at the first digit. In general, responses that begin with a 2 indicate success. Those that begin with 3 indicate that some further action is needed, as shown above. 4 and 5 indicate errors. 4 is a "temporary" error, such as a disk filling. The message should be saved, and tried again later. 5 is a permanent error, such as a non-existent recipient. The message should be returned to the sender with an error message.

(For more details about the protocols mentioned in this section, see RFC's 821/822 for mail, RFC 959 for file transfer, and RFC's 854/855 for remote logins. For the well-known port numbers, see the current edition of Assigned Numbers, and possibly RFC 814.)

4. Protocols other than TCP: UDP and ICMP

So far, we have described only connections that use TCP. Recall that TCP is responsible for breaking up messages into datagrams, and reassembling them properly. However in many applications, we have messages that will always fit in a single datagram. An example is name lookup. When a user attempts to make a connection to another system, he will generally specify the system by name, rather than Internet address. His system has to translate that name to an address before it can do anything. Generally, only a few systems have the database used to translate names to addresses. So the user's system will want to send a query to one of the systems that has the database. This query is going to be very short.

It will certainly fit in one datagram. So will the answer. Thus it seems silly to use TCP. Of course TCP does more than just break things up into datagrams. It also makes sure that the data arrives, resending datagrams where necessary. But for a question that fits in a single datagram, we don't need all the complexity of TCP to do this. If we don't get an answer after a few seconds, we can just ask again. For applications like this, there are alternatives to TCP.

The most common alternative is UDP ("user datagram protocol"). UDP is designed for applications where you don't need to put sequences of datagrams together. It fits into the system much like TCP. There is a UDP header. The network software puts the UDP header on the front of your data, just as it would put a TCP header on the front of your data. Then UDP sends the data to IP, which adds the IP header, putting UDP's protocol number in the protocol field instead of TCP's protocol number. However UDP doesn't do as much as TCP does. It doesn't split data into multiple datagrams. It doesn't keep track of what it has sent so it can resend if necessary. About all that UDP provides is port numbers, so that several programs can use UDP at once. UDP port numbers are used just like TCP port numbers. There are well-known port numbers for servers that use UDP. Note that the UDP header is shorter than a TCP header. It still has source and destination port numbers, and a checksum, but that's about it. No sequence number, since it is not needed. UDP is used by the protocols that handle name lookups (see IEN 116, RFC 882, and RFC 883), and a number of similar protocols.

Another alternative protocol is ICMP ("Internet control message protocol"). ICMP is used for error messages, and other messages intended for the TCP/IP software itself, rather than any particular user program. For example, if you attempt to connect to a host, your system may get back an ICMP message saying "host unreachable". ICMP can also be used to find out some information about the network. See RFC 792 for details of ICMP. ICMP is similar to UDP, in that it handles messages that fit in one datagram. However it is even simpler than UDP. It doesn't even have port numbers in its header. Since all ICMP messages are interpreted by the network software itself, no port numbers are needed to say where a ICMP message is supposed to go.

5. Keeping track of names and information: the domain system

As we indicated earlier, the network software generally needs a 32-bit Internet address in order to open a connection or send a datagram. However users prefer to deal with computer names rather than numbers. Thus there is a database

that allows the software to look up a name and find the corresponding number. When the Internet was small, this was easy. Each system would have a file that listed all of the other systems, giving both their name and number. There are now too many computers for this approach to be practical. Thus these files have been replaced by a set of name servers that keep track of host names and the corresponding Internet addresses. (In fact these servers are somewhat more general than that. This is just one kind of information stored in the domain system.) Note that a set of interlocking servers are used, rather than a single central one. There are now so many different institutions connected to the Internet that it would be impractical for them to notify a central authority whenever they installed or moved a computer. Thus naming authority is delegated to individual institutions. The name servers form a tree, corresponding to institutional structure. The names themselves follow a similar structure. A typical example is the name BORAX.LCS.MIT.EDU. This is a computer at the Laboratory for Computer Science (LCS) at MIT. In order to find its Internet address, you might potentially have to consult 4 different servers. First, you would ask a central server (called the root) where the EDU server is. EDU is a server that keeps track of educational institutions. The root server would give you the names and Internet addresses of several servers for EDU. (There are several servers at each level, to allow for the possibly that one might be down.) You would then ask EDU where the server for MIT is. Again, it would give you names and Internet addresses of several servers for MIT. Generally, not all of those servers would be at MIT, to allow for the possibility of a general power failure at MIT. Then you would ask MIT where the server for LCS is, and finally you would ask one of the LCS servers about BORAX. The final result would be the Internet address for BORAX.LCS.MIT.EDU. Each of these levels is referred to as a "domain". The entire name, BORAX.LCS.MIT.EDU, is called a "domain name". (So are the names of the higher-level domains, such as LCS.MIT.EDU, MIT.EDU, and EDU.)

Fortunately, you don't really have to go through all of this most of the time. First of all, the root name servers also happen to be the name servers for the top-level domains such as EDU. Thus a single query to a root server will get you to MIT. Second, software generally remembers answers that it got before. So once we look up a name at LCS.MIT.EDU, our software remembers where to find servers for LCS.MIT.EDU, MIT.EDU, and EDU. It also remembers the translation of BORAX.LCS.MIT.EDU. Each of these pieces of information has a "time to live" associated with it. Typically this is a few days. After that, the information expires and has to be looked up again. This allows institutions to change things.

The domain system is not limited to finding out Internet addresses. Each domain name is a node in a database. The node can have records that define a number of different properties. Examples are Internet address, computer type, and a list of services provided by a computer. A program can ask for a specific piece of information, or all information about a given name. It is possible for a node in the database to be marked as an "alias" (or nickname) for another node. It is also possible to use the domain system to store information about users, mailing lists, or other objects.

There is an Internet standard defining the operation of these databases, as well as the protocols used to make queries of them. Every network utility has to be able to make such queries, since this is now the official way to evaluate host names. Generally utilities will talk to a server on their own system. This server will take care of contacting the other servers for them. This keeps down the amount of code that has to be in each application program.

The domain system is particularly important for handling computer mail. There are entry types to define what computer handles mail for a given name, to specify where an individual is to receive mail, and to define mailing lists.

(See RFC's 882, 883, and 973 for specifications of the domain system. RFC 974 defines the use of the domain system in sending mail.)

6. Routing

The description above indicated that the IP implementation is responsible for getting datagrams to the destination indicated by the destination address, but little was said about how this would be done. The task of finding how to get a datagram to its destination is referred to as "routing". In fact many of the details depend upon the particular implementation. However some general things can be said.

First, it is necessary to understand the model on which IP is based. IP assumes that a system is attached to some local network. We assume that the system can send datagrams to any other system on its own network. (In the case of Ethernet, it simply finds the Ethernet address of the destination system, and puts the datagram out on the Ethernet.) The problem comes when a system is asked to send a datagram to a system on a different network. This problem is handled by gateways. A gateway is a system that connects a network with one or more other networks. Gateways are often normal computers that happen to have more than

one network interface. For example, we have a Unix machine that has two different Ethernet interfaces. Thus it is connected to networks 128.6.4 and 128.6.3. This machine can act as a gateway between those two networks. The software on that machine must be set up so that it will forward datagrams from one network to the other. That is, if a machine on network 128.6.4 sends a datagram to the gateway, and the datagram is addressed to a machine on network 128.6.3, the gateway will forward the datagram to the destination. Major communications centers often have gateways that connect a number of different networks. (In many cases, special-purpose gateway systems provide better performance or reliability than general-purpose systems acting as gateways. A number of vendors sell such systems.)

Routing in IP is based entirely upon the network number of the destination address. Each computer has a table of network numbers. For each network number, a gateway is listed. This is the gateway to be used to get to that network. Note that the gateway doesn't have to connect directly to the network. It just has to be the best place to go to get there. For example at Rutgers, our interface to NSFnet is at the John von Neuman Supercomputer Center (JvNC). Our connection to JvNC is via a high-speed serial line connected to a gateway whose address is 128.6.3.12. Systems on net 128.6.3 will list 128.6.3.12 as the gateway for many off-campus networks. However systems on net 128.6.4 will list 128.6.4.1 as the gateway to those same off-campus networks. 128.6.4.1 is the gateway between networks 128.6.4 and 128.6.3, so it is the first step in getting to JvNC.

When a computer wants to send a datagram, it first checks to see if the destination address is on the system's own local network. If so, the datagram can be sent directly. Otherwise, the system expects to find an entry for the network that the destination address is on. The datagram is sent to the gateway listed in that entry. This table can get quite big. For example, the Internet now includes several hundred individual networks. Thus various strategies have been developed to reduce the size of the routing table. One strategy is to depend upon "default routes". Often, there is only one gateway out of a network. This gateway might connect a local Ethernet to a campus-wide backbone network. In that case, we don't need to have a separate entry for every network in the world. We simply define that gateway as a "default". When no specific route is found for a datagram, the datagram is sent to the default gateway. A default gateway can even be used when there are several gateways on a network. There are provisions for gateways to send a message saying "I'm not the best gateway—use this one instead." (The message is sent via ICMP. See RFC 792.) Most network software is designed to use these messages to add entries to their routing tables. Sup-

pose network 128.6.4 has two gateways, 128.6.4.59 and 128.6.4.1. 128.6.4.59 leads to several other internal Rutgers networks. 128.6.4.1 leads indirectly to the NSFnet. Suppose we set 128.6.4.59 as a default gateway, and have no other routing table entries. Now what happens when we need to send a datagram to MIT? MIT is network 18. Since we have no entry for network 18, the datagram will be sent to the default, 128.6.4.59. As it happens, this gateway is the wrong one. So it will forward the datagram to 128.6.4.1. But it will also send back an error saying in effect: "to get to network 18, use 128.6.4.1". Our software will then add an entry to the routing table. Any future datagrams to MIT will then go directly to 128.6.4.1. (The error message is sent using the ICMP protocol. The message type is called "ICMP redirect".)

Most IP experts recommend that individual computers should not try to keep track of the entire network. Instead, they should start with default gateways, and let the gateways tell them the routes, as just described. However this doesn't say how the gateways should find out about the routes. The gateways can't depend upon this strategy. They have to have fairly complete routing tables. For this, some sort of routing protocol is needed. A routing protocol is simply a technique for the gateways to find each other, and keep up to date about the best way to get to every network. RFC 1009 contains a review of gateway design and routing. However rip.doc is probably a better introduction to the subject. It contains some tutorial material, and a detailed description of the most commonly-used routing protocol.

7. Details about Internet addresses: subnets and broadcasting

As indicated earlier, Internet addresses are 32-bit numbers, normally written as 4 octets (in decimal), e.g. 128.6.4.7. There are actually 3 different types of address. The problem is that the address has to indicate both the network and the host within the network. It was felt that eventually there would be lots of networks. Many of them would be small, but probably 24 bits would be needed to represent all the IP networks. It was also felt that some very big networks might need 24 bits to represent all of their hosts. This would seem to lead to 48 bit addresses. But the designers really wanted to use 32 bit addresses. So they adopted a kludge. The assumption is that most of the networks will be small. So they set up three different ranges of address. Addresses beginning with 1 to 126 use only the first octet for the network number. The other three octets are available for the host number. Thus 24 bits are available for hosts. These numbers are used for large networks. But there can only be 126 of these very big networks. The Arpanet is one, and there are a few large commercial

networks. But few normal organizations get one of these "class A" addresses. For normal large organizations, "class B" addresses are used. Class B addresses use the first two octets for the network number. Thus network numbers are 128.1 through 191.254. (We avoid 0 and 255, for reasons that we see below. We also avoid addresses beginning with 127, because that is used by some systems for special purposes.) The last two octets are available for host addesses, giving 16 bits of host address. This allows for 64516 computers, which should be enough for most organizations. (It is possible to get more than one class B address, if you run out.) Finally, class C addresses use three octets, in the range 192.1.1 to 223.254.254. These allow only 254 hosts on each network, but there can be lots of these networks. Addresses above 223 are reserved for future use, as class D and E (which are currently not defined).

Many large organizations find it convenient to divide their network number into "subnets". For example, Rutgers has been assigned a class B address, 128.6. We find it convenient to use the third octet of the address to indicate which Ethernet a host is on. This division has no significance outside of Rutgers. A computer at another institution would treat all datagrams addressed to 128.6 the same way. They would not look at the third octet of the address. Thus computers outside Rutgers would not have different routes for 128.6.4 or 128.6.5. But inside Rutgers, we treat 128.6.4 and 128.6.5 as separate networks. In effect, gateways inside Rutgers have separate entries for each Rutgers subnet, whereas gateways outside Rutgers just have one entry for 128.6. Note that we could do exactly the same thing by using a separate class C address for each Ethernet. As far as Rutgers is concerned, it would be just as convenient for us to have a number of class C addresses. However using class C addresses would make things inconvenient for the rest of the world. Every institution that wanted to talk to us would have to have a separate entry for each one of our networks. If every institution did this, there would be far too many networks for any reasonable gateway to keep track of. By subdividing a class B network, we hide our internal structure from everyone else, and save them trouble. This subnet strategy requires special provisions in the network software. It is described in RFC 950.

0 and 255 have special meanings. 0 is reserved for machines that don't know their address. In certain circumstances it is possible for a machine not to know the number of the network it is on, or even its own host address. For example, 0.0.0.23 would be a machine that knew it was host number 23, but didn't know on what network.

255 is used for "broadcast". A broadcast is a message that you want every system

on the network to see. Broadcasts are used in some situations where you don't know who to talk to. For example, suppose you need to look up a host name and get its Internet address. Sometimes you don't know the address of the nearest name server. In that case, you might send the request as a broadcast. There are also cases where a number of systems are interested in information. It is then less expensive to send a single broadcast than to send datagrams individually to each host that is interested in the information. In order to send a broadcast, you use an address that is made by using your network address, with all ones in the part of the address where the host number goes. For example, if you are on network 128.6.4, you would use 128.6.4.255 for broadcasts. How this is actually implemented depends upon the medium. It is not possible to send broadcasts on the Arpanet, or on point to point lines. However it is possible on an Ethernet. If you use an Ethernet address with all its bits on (all ones), every machine on the Ethernet is supposed to look at that datagram.

Although the official broadcast address for network 128.6.4 is now 128.6.4.255, there are some other addresses that may be treated as broadcasts by certain implementations. For convenience, the standard also allows 255.255.255.255 to be used. This refers to all hosts on the local network. It is often simpler to use 255.255.255.255 instead of finding out the network number for the local network and forming a broadcast address such as 128.6.4.255. In addition, certain older implementations may use 0 instead of 255 to form the broadcast address. Such implementations would use 128.6.4.0 instead of 128.6.4.255 as the broadcast address on network 128.6.4. Finally, certain older implementations may not understand about subnets. Thus they consider the network number to be 128.6. In that case, they will assume a broadcast address of 128.6.255.255 or 128.6.0.0. Until support for broadcasts is implemented properly, it can be a somewhat dangerous feature to use.

Because 0 and 255 are used for unknown and broadcast addresses, normal hosts should never be given addresses containing 0 or 255. Addresses should never begin with 0, 127, or any number above 223. Addresses violating these rules are sometimes referred to as "Martians", because of rumors that the Central University of Mars is using network 225.

8. Datagram fragmentation and reassembly

TCP/IP is designed for use with many different kinds of network. Unfortunately, network designers do not agree about how big packets can be. Ethernet packets can be 1500 octets long. Arpanet packets have a maximum of around 1000

octets. Some very fast networks have much larger packet sizes. At first, you might think that IP should simply settle on the smallest possible size. Unfortunately, this would cause serious performance problems. When transferring large files, big packets are far more efficient than small ones. So we want to be able to use the largest packet size possible. But we also want to be able to handle networks with small limits. There are two provisions for this. First, TCP has the ability to "negotiate" about datagram size. When a TCP connection first opens, both ends can send the maximum datagram size they can handle. The smaller of these numbers is used for the rest of the connection. This allows two implementations that can handle big datagrams to use them, but also lets them talk to implementations that can't handle them. However this doesn't completely solve the problem. The most serious problem is that the two ends don't necessarily know about all of the steps in between. For example, when sending data between Rutgers and Berkeley, it is likely that both computers will be on Ethernets. Thus they will both be prepared to handle 1500-octet datagrams. However the connection will at some point end up going over the Arpanet. It can't handle packets of that size. For this reason, there are provisions to split datagrams up into pieces. (This is referred to as "fragmentation".) The IP header contains fields indicating the a datagram has been split, and enough information to let the pieces be put back together. If a gateway connects an Ethernet to the Arpanet, it must be prepared to take 1500-octet Ethernet packets and split them into pieces that will fit on the Arpanet. Furthermore, every host implementation of TCP/IP must be prepared to accept pieces and put them back together. This is referred to as "reassembly".

TCP/IP implementations differ in the approach they take to deciding on datagram size. It is fairly common for implementations to use 576-byte datagrams whenever they can't verify that the entire path is able to handle larger packets. This rather conservative strategy is used because of the number of implementations with bugs in the code to reassemble fragments. Implementors often try to avoid ever having fragmentation occur. Different implementors take different approaches to deciding when it is safe to use large datagrams. Some use them only for the local network. Others will use them for any network on the same campus. 576 bytes is a "safe" size, which every implementation must support.

9. Ethernet encapsulation: ARP

There was a brief discussion earlier about what IP datagrams look like on an Ethernet. The discussion showed the Ethernet header and checksum. However it left one hole: It didn't say how to figure out what Ethernet address to use

when you want to talk to a given Internet address. In fact, there is a separate protocol for this, called ARP ("address resolution protocol"). (Note by the way that ARP is not an IP protocol. That is, the ARP datagrams do not have IP headers.) Suppose you are on system 128.6.4.194 and you want to connect to system 128.6.4.7. Your system will first verify that 128.6.4.7 is on the same network, so it can talk directly via Ethernet. Then it will look up 128.6.4.7 in its ARP table, to see if it already knows the Ethernet address. If so, it will stick on an Ethernet header, and send the packet. But suppose this system is not in the ARP table. There is no way to send the packet, because you need the Ethernet address. So it uses the ARP protocol to send an ARP request. Essentially an ARP request says "I need the Ethernet address for 128.6.4.7". Every system listens to ARP requests. When a system sees an ARP request for itself, it is required to respond. So 128.6.4.7 will see the request, and will respond with an ARP reply saying in effect "128.6.4.7 is 8:0:20:1:56:34". (Recall that Ethernet addresses are 48 bits. This is 6 octets. Ethernet addresses are conventionally shown in hex, using the punctuation shown.) Your system will save this information in its ARP table, so future packets will go directly. Most systems treat the ARP table as a cache, and clear entries in it if they have not been used in a certain period of time.

Note by the way that ARP requests must be sent as "broadcasts". There is no way that an ARP request can be sent directly to the right system. After all, the whole reason for sending an ARP request is that you don't know the Ethernet address. So an Ethernet address of all ones is used, i.e. ff:ff:ff:ff:ff:ff. By convention, every machine on the Ethernet is required to pay attention to packets with this as an address. So every system sees every ARP requests. They all look to see whether the request is for their own address. If so, they respond. If not, they could just ignore it. (Some hosts will use ARP requests to update their knowledge about other hosts on the network, even if the request isn't for them.) Note that packets whose IP address indicates broadcast (e.g. 255.255.255.255 or 128.6.4.255) are also sent with an Ethernet address that is all ones.

10. Getting more information

This directory contains documents describing the major protocols. There are literally hundreds of documents, so we have chosen the ones that seem most important. Internet standards are called RFC's. RFC stands for Request for Comment. A proposed standard is initially issued as a proposal, and given an RFC number. When it is finally accepted, it is added to Official Internet Protocols, but it is still referred to by the RFC number. We have also included two IEN's. (IEN's used to be a separate classification for more informal documents. This classification

no longer exists—RFC's are now used for all official Internet documents, and a mailing list is used for more informal reports.) The convention is that whenever an RFC is revised, the revised version gets a new number. This is fine for most purposes, but it causes problems with two documents: Assigned Numbers and Official Internet Protocols. These documents are being revised all the time, so the RFC number keeps changing. You will have to look in rfc-index.txt to find the number of the latest edition. Anyone who is seriously interested in TCP/IP should read the RFC describing IP (791). RFC 1009 is also useful. It is a specification for gateways to be used by NSFnet. As such, it contains an overview of a lot of the TCP/IP technology. You should probably also read the description of at least one of the application protocols, just to get a feel for the way things work. Mail is probably a good one (821/822). TCP (793) is of course a very basic specification. However the spec is fairly complex, so you should only read this when you have the time and patience to think about it carefully. Fortunately, the author of the major RFC's (Jon Postel) is a very good writer. The TCP RFC is far easier to read than you would expect, given the complexity of what it is describing. You can look at the other RFC's as you become curious about their subject matter.

Here is a list of the documents you are more likely to want:

rfc-index	list of all RFC's
rfc1012	somewhat fuller list of all RFC's
rfc1011	Official Protocols. It's useful to scan this to see what tasks protocols have been built for. This defines which RFC's are actual standards, as opposed to requests for comments.
rfc1010	Assigned Numbers. If you are working with TCP/IP, you will probably want a hardcopy of this as a reference. It's not very exciting to read. It lists all the offically defined well-known ports and lots of other things.
rfc1009	NSFnet gateway specifications. A good overview of IP routing and gateway technology.
rfc1001/2	netBIOS: networking for PC's
rfc973	update on domains
rfc959	FTP (file transfer)
rfc950	subnets
rfc937	POP2: protocol for reading mail on PC's
rfc894	how IP is to be put on Ethernet, see also rfc825

rfc882/3	domains (the database used to go from host names to Internet address and back—also used to handle UUCP these days). See also rfc973
rfc854/5	telnet—protocol for remote logins
rfc826	ARP—protocol for finding out Ethernet addresses
rfc821/2	mail
rfc814	names and ports—general concepts behind well-known ports
rfc793	TCP
rfc792	ICMP
rfc791	IP
rfc768	UDP
rip.doc	details of the most commonly-used routing protocol
ien-116	old name server (still needed by several kinds of system)
ien-48	the Catenet model, general description of the philosophy behind TCP/IP

The following documents are somewhat more specialized.

rfc813	window and acknowledgement strategies in TCP
rfc815	datagram reassembly techniques
rfc816	fault isolation and resolution techniques
rfc817	modularity and efficiency in implementation
rfc879	the maximum segment size option in TCP
rfc896	congestion control
rfc827,888,904,975,985	EGP and related issues

To those of you who may be reading this document remotely instead of at Rutgers: The most important RFC's have been collected into a three-volume set, the DDN Protocol Handbook. It is available from the DDN Network Information Center, SRI International, 333 Ravenswood Avenue, Menlo Park, California 94025 (telephone: 800-235-3155). You should be able to get them via anonymous FTP from sri-nic.arpa. File names are:

RFC's:
  rfc:rfc-index.txt
  rfc:rfcxxx.txt
IEN's:
  ien:ien-index.txt

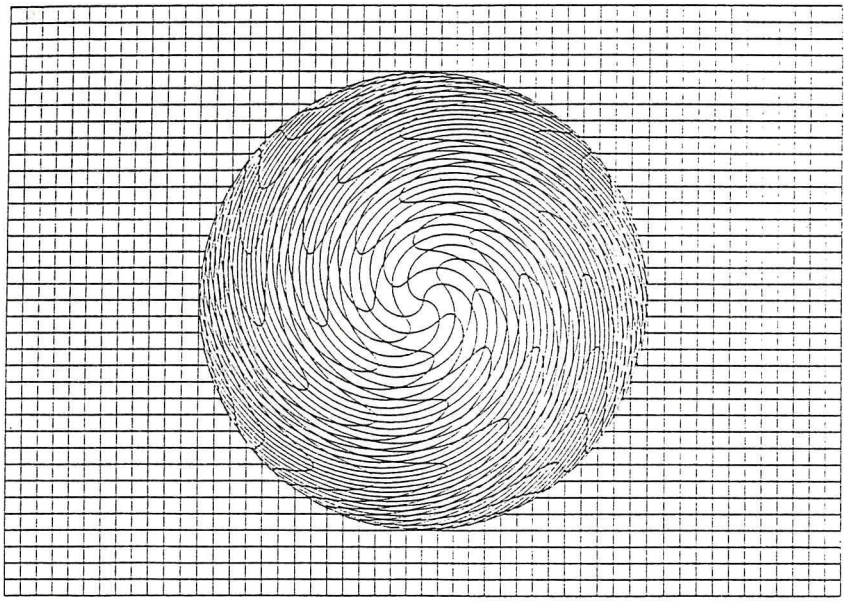
ien:ien-xxx.txt

rip.doc is available by anonymous FTP from topaz.rutgers.edu, as /pub/tcp-ip-docs/rip.doc.

Sites with access to UUCP but not FTP may be able to retreivè them via UUCP from UUCP host rutgers. The file names would be

RFC's:
/topaz/pub/pub/tcp-ip-docs/rfc-index.txt
/topaz/pub/pub/tcp-ip-docs/rfcxxx.txt
IEN's:
/topaz/pub/pub/tcp-ip-docs/ien-index.txt
/topaz/pub/pub/tcp-ip-docs/ien-xxx.txt
/topaz/pub/pub/tcp-ip-docs/rip.doc

Note that SRI-NIC has the entire set of RFC's and IEN's, but rutgers and topaz have only those specifically mentioned above.



Framsidan

av Jan Michael Rynning

NÄR man hittar någonting roligt man vill fotografera, då går man så lång bort man någonsin kan och trycker av.
— Det där gula är en skylt jag och Uffe hittade i San Fransisco, berättar Stellan, och vi bara *måste* fotografera den.
— Vad står det på den?
— Ja, det är frågan...
— Gick det inte att komma längre bort?
— Den blev kanske lite liten...

Den *måste* med i STACKPOINTER. Iväg till Bild- och grafiklabbet!
— Hej Matti! Jag har en bild som skulle behöva en del bildbehandling...

Scanna in den på lägsta upplösning, 128 × 128 pixels, för att prova ut lämpliga parametrar. Titta på gråvåhistogram efter varje försök, tills spridningen räcker för efterbehandling.

Scanna in den med rätt parametrar och högsta upplösning, 1024 × 1024, 1 Mbyte data, tur man har riktig dator. Skylten är inte ens 1% av bildytan.

Försök justera gråvåerna, så texten går att läsa. Nu är resten av bilden antingen vit eller svart och bokstäverna fortfarande suddiga.

Skärp alla kanter i bilden, inklusive bokstäverna som blir mindre suddiga.

Ta reda på koordinaterna för hörnen på skylten. Utnyttja att alla punkter i en konvex polygon ligger på samma sida om alla kantlinjer (till vänster om man går motsols runt kanten, till höger om man går medsols). Skriv ett LISP-program som går igenom den utskurna rektangeln och drar isär gråvåerna inuti skylten. Ut med den på laserskrivaren.

Från början.

Efter kantskärpning.

Isärdragna gråvåer.

San Fransisco

av Stellan Lagerström

Våren 1987.

Det har börjat samlas tillräckligt med prylar i San Francisco för att det skall vara värt en containertransport. Bland annat två stycken KA10-CPU:er, en BBN-pager, som ger KA10 virtuellt minne, samt förmodligen en SA10, en burk som firma System Concepts säljer, som gör att man kan ha I*M-kringutrustning på en DEC-10.

I det längsta försöker Peter övertyga Mike Levith på SC, hos vilka prylarna står, att fixa lastningen åt oss. Han vill dock ha hjälp, och kan vara lika envis som Peter, så någon måste åka dit.

En ordförande får aldrig tveka, så jag ställer upp. Min kompis Ulf Asplund (numera STACKEN-medlem) har pratat om att åka till USA, så jag frågar listigt om han har lust att hjälpa mig packa några hemdatorer. Det tycker han låter kul...

Onsdag 8 april.

Mail från Peter. Jag måste åka inom en vecka. Mike har insett att hyran på lokalen där prylarna står bara är betald en vecka till (SC har flyttat). 2 DEC-2020-datorer har tillkommit, så vi kan bara inte missa den här lasten.

Torsdag morgon.

Ringer Ulf.
— Vad gör du i påskveckan?
— Ingenting.
— San Francisco?
— OK.
En timme senare, Ulf ringer tillbaka:
— Påsk, det är ju nästa vecka!!!!!

En vecka med forcerande av flygbiljetter, lektion i containerpackning etc. Jag får reda på att 2020:orna inte står hos Mike, utan någon annanstans i SF.

Onsdag 15.

Upp ur sängen, ner på golvet.
Ner i hissen, ner i tunnelbanan.
Upp till bussen, ut till Arlanda.
Upp från Arlanda, ner i Köpenhamn.
Upp från Kastrup, ner i Amsterdam.
Upp från Schipol, ner i New York.
Bära väskor genom tullen.
Upp från JFK, ner i San Francisco.
(Allt medan Stellan sliter hårt, har Ulf en behaglig resa STO-CPH-LAX-SFO.)

Hitta väskor. Där är Mike Levith, Stewart Nelson och Uffe med en 5 meter lång van.
— Hej, hej, säger Mike. Har ni mitt DECnetprogram med er?
— Va?
— Ja, det som Peter lovade skicka med er!
— Mummel, det kommer säkert med posten...

Vi blir medsläpade till en kineskrog för middag. På vägen dit diskuterar Mike och Stewart vilken av cirka 37 kineskrogar som jag och Uffe gillar bäst. Mike berättar att containern ska komma på fredag morgon och bli hämtad onsdag lunch. Efter 20 timmar i luften och middagen är vi lite lätt trötta, så vi åker hem till Mike, där vi ska bo.

Han har en mysig 4:a med utsikt över diverse broar, centralt (Marina District). Det är några år sen han flyttade in. Alla prylar står fortfarande i lådor i matrummet. Vi får ett rum alldeles för oss själva. Alldeles, inga möbler eller annat tjafs. Rättelse: En *stor* TV finns det.
— Det finns en matta i det andra rummet ni kan rulla ut och sova på.
— Matta? Peter sa att det skulle räcka med lakan?
— Oh no, then he's screwed up again...

Torsdag.

Vaknar. Mike borta. Frukost? Inget i kylen. Knallar till SC, köper Cola och kex på vägen. Orienterar oss. Ordnar nycklar till gamla SC, magikort till nya SC, nycklar till Mikes lägenhet. Tar en tur med SC:s van:

- Tittar på eländet. (Tänk er en fotbollsplan med tak och pelare, på fjärde våningen, och i ett hörn en version av B30 efter kollision med schaktmaskin. 2×KA10 + 4×MF10 + pager + diverse datorer av okänt ursprung.)
- Köper virke, verktyg, lås.
- Lastar av på nya SC, Mike behöver bilen genast...

Samlar ihop RP06 shipping locks. Knallar till gamla Adapt. Här finns 2 st 2020 samt 4 st RP06 (samt en TU45, men den fick vi inte...). RP06:orna har motorer för 110 V/60 Hz, men en liten röst viskar "reservdelar" och Mike går så småningom med på att vi tar 2 och han 2. Kopplar isär och packar alltihop.

På kvällen en ny tur med skåpbilen: Bokar plats för containern genom att ställa bilen där.

Fredag.

Containern kommer. Taket går att ta av! Konstigt! Det verkar dock sitta hyfsat fast och tätt, så vafan...

Hjälper flyttgubbar rensa hos Adapt och forsla till Mike, nya Adapt och sist

containern. Mike trodde det skulle ta en timme, drygt. Tre timmar blir det innan vi är klara. 2020:orna, RP06:orna lastas direkt från baklyften i containern. Skruvar isär första KA:n.

Lördag.

Skruvar isär resten av KA-CPU:erna. Det är många sladdar mellan skåpen... Lunchen äts på ett lågt bord. Bord förresten? Det är ju en PDP-8! Den tar vi. Kanske. Om vi får plats. Uffe vill ha den som bordskalkylator.

Söndag.

Flyttar ner till markplanet. Jag bockar 1″ stålbeslag i ramen på ett 19″-skåp, genom att ta en backe (inomhus) i bästa rally-stil. Tyngdpunkten ligger *högt* i SC:s minnesskåp...

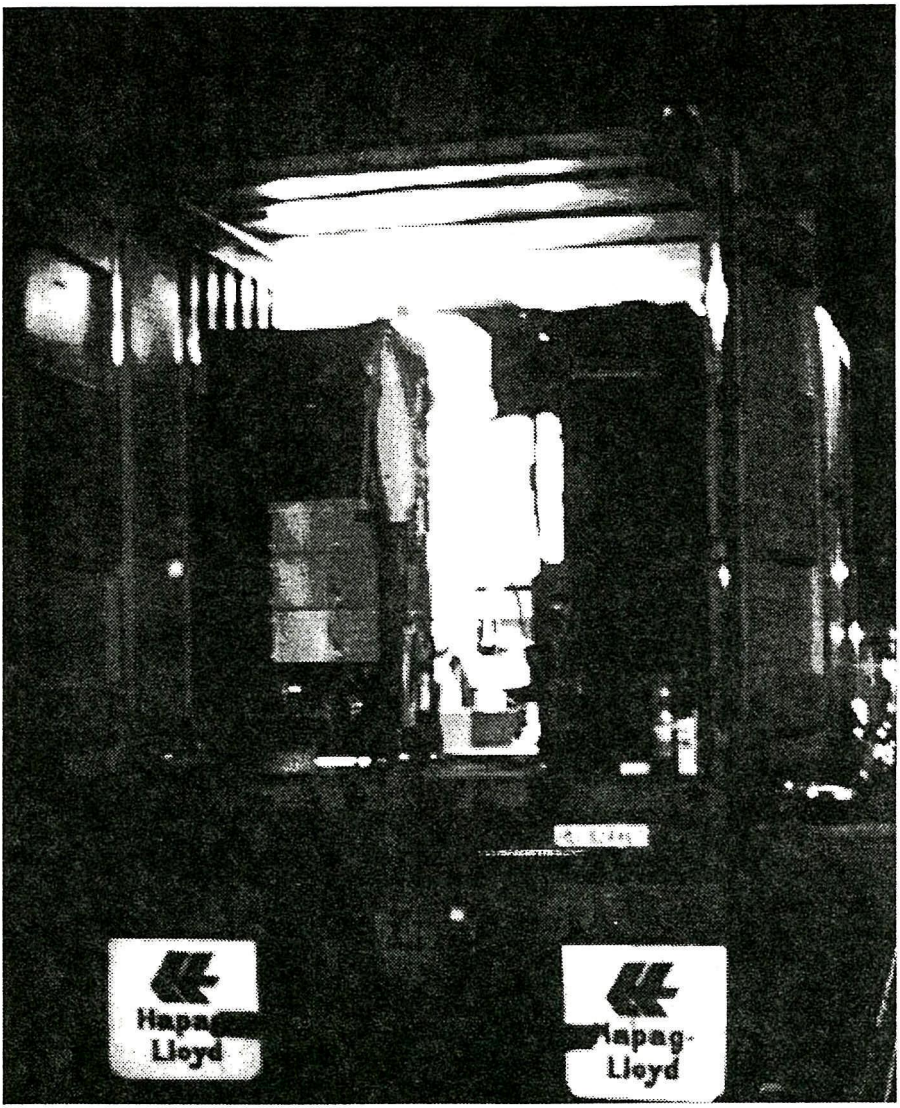
Måndag.

Gaffeltruck inklusive virtuos lånas på närbelägen plåtfirma. Mike känner folk. Lyfter in KA10:orna samt några småskåp. Klär containerns insida med well-

Stellan skissar packningsplaner på lunchbordet/PDP-8:n.

papp och börjar packa in KA:rna och 2020:orna. Börjar ställa tre breda skåp innerst enligt beprövad Peter-metod. Hmmmm. Får det inte plats? Containern ska vara 8 fot, sa han. Nähä. *Den här* containern är 4 tum smalare...

Svartjobb utförs lämpligen nattetid.



Snabb revision av alla förpackningsplaner. Förslagen haglar. 17 skisser och några meter virke senare är 2×KA + 2×2020 på plats. 30° i skuggan. ≫30° i containern i solen... Varmt. Köper långa skruvar för att transportsäkra skåpen. Åker och hämtar de diskpackar, ben och sladdar vi inte fick med oss från Adapt.

Återgår till nya SC på kvällen. Mike inser att SA10:an vi ska få (den enda byggd för 220 V) inte är komplett, samt borde testas innan den åker. Genom att norpa PROM:ar från ett annat ex, samt sätta mig på att leta igenom lokalerna efter bussanpassningskort (sen flyttningen vet *ingen* var allt finns), och sedan koppla in den i hans KI-dubbelprocessor DEC-10:a för testprogram, får vi den till slut leveransklar. Den ser lite konstig ut, dock, byggd i ett standard DEC-skåp, men ingen frontpanel, bara ett hål. Mike mumlar nåt om att den brukar se ut så.

Tisdag.

Mike har insett att hans van-försäkring kanske inte gäller om föraren är under 30 år, men viker sig efter *viss* övertalning. Vi köper en såg till. Saker försvinner på SF:s bakgator,det räcker ibland med att vända ryggen till. Det är dock skillnad på "sakletarna". En av de trevligare vi träffar heter John King och har en hel del kul att berätta. Jobbar morgon. Varmare, trots inkopplad fläkt. Eftermiddag: för varmt för att jobba, tar ledigt och besöker Games of Berkely för inhandlande av simulerade livsfarliga vapen. Lånar gaffeltrucken igen och lyfter in det sista. Jobbar hela natten.

Onsdag.

Containern skall hämtas kl 12. Mike har insett att vi behöver kablar till vår SA10, samt att han inte har några. Alltså sätter han personalen på att snabbtillverka. Vi fortsätter packa in de sista skåpen. John dyker upp vid elvatiden och visar sig ovärderlig. Bär och lyfter, samt försäkrar oss att den säkert inte blir hämtad före 3, om den var beställd till 12... Kablarna anländer med kurir. Dörrarna lastas, och sista väggen byggs. Containern åker slutligen kl 15! Vi åker också — hem och sover.

Nu kommer överaskning del 2. Vi skall vara dom som skall vara Mikes sopgubbar, dvs flytta allt hans skräp till olika ställen.

De prylar vi inte ville ha (c:a 20 m^3) indelas i två kategorier: Sparas, som stoppas in i ett rum Mike har kvar. Resten, som lastas i skrothandlarens container.

Torsdag.

Efter en veckas finlir är det en viss känsla att vräka in en diskdrive i en

container med spett... Ytterligare en het dag med massor av blod, svett och Cola.

Fredag.

Mera städa. Uffe smiter iväg till Seattle.

Lördag.

Ringer Marc Crispin och undrar var/när vi kan träffas. Han föreslår Japantown. Han ska gå på Cherry Blossom Festival där. Det är han som fixat BBN-pagern åt oss och jag får ritningar, manualer etc till den. Följer sedan med honom hem, och får träffa Panda och Unix. (Den första är en dator, den andra ett djur.) Det är han som har en 2020 hemma samt en hund som skulle hetat något helt annat, men bara lystrat till namnet Unix...

Söndag.

Se SF på dagen.

Måndag.

Planet hem går 8.15

"I'm the plumber about the burst pipe, are you the carpenter about the jammed door?"

Mer om musikutrustning

av Christer Lindström

JAHA, det rör sig verkligen i den digitala musikvärlden. Tre nya intressanta produkter har dykt upp. Dels en ny digital efterklangsenhet, dels ett nytt mixerbord med inbyggd bandspelare, dels en ny MIDI-programvara. Det här med MIDI och programvara är lite för stort att ta upp här, så det får bli en hel artikelserie istället så småningom.

Alesis MICROVERB

Från företaget Alesis har tidigare kommit en mängd intressanta produkter, senast då 'Midiverb' och 'Midifex'. Dessa två efterklangsenheter med möjlighet till styrning från MIDI och 63 olika program, har snabbt blivit populära på grund av sitt låga pris (4.000–5.000:-) och goda ljudkvalitet.

Nu har Alesis släppt en ännu enklare version med endast 9 program. Man har helt enkelt valt ut de 9 mest använda ljuden, och förbättrat dem. Resultatet är en otroligt billig efterklangsenhet med mycket bra ljud. Inga grova tendenser till 'svirr-ljud' som alltför ofta förekommer i digitala enheter. Prisläge troligtvis runt 2.500:-.

Akai MG1214

Akai har sen något år tillbaks satsat på hemmamusikmarknaden. Man släppte några syntar, tillbehör och en kombinerad bandspelare/mixer på 12 kanaler. Tyvärr var den lite väl dyr (55–60.000:-) och blev inte någon större succe. Dessutom var koppling till SMPTE-utrustning inte möjlig, vilket gjorde att många användningområden inom t.ex. video försvann. (SMPTE är en standard för kodning av tid m.m.).

Nu har man släppt en nästan identisk maskin med lite nya möjligheter. SMPTE-kod är möjlig, och man kan t.ex. koppla ihop två maskiner med varandra och på så sätt få 24 audiokanaler! Dessutom har priset sänkts en bit (ca. 45.000:-) vilket gör maskinen betydligt attraktivare. I stort sett består den av två delar, en mixer och en bandspelardel.

Mixerdelen: 12 kanaler med 3 svepbara filter, 4 tappningar till eko m.m., ingångsväljare, nivåväljare, lysdiodstapel på varje kanal. Dessutom 2 extraingångar stereo. Tyvärr kan man inte styra ingångskanalen till valfri bandspelarkanal, utan mixerkanal 1 går alltid till bandkanal 1 o.s.v. Detta ger

lite problem vid nermixningar av flera instrument på en bandkanal.

Bandspelardelen: Som band använder sig Akai av en egen konstruktion som påmminer ganska mycket om en video-kasett. På en ½" bredd ligger 14 kanaler varav 12 är för audiobruk. De övriga 2 är för intern datakod (möjlighet till inläggning av brytpunkter m.m.) och SMPTE-kod. Två olika hastigheter finns tillgängliga, 9.5 och 19 cm/sek. vilket ger en speltid av 20 resp. 10 minuter per kasett.

I stort är Akai-systemet mycket välljudande och användbart, främst till demo-inspelningar och små videostudios. Använder man sig av ordentlig kringutrustning (mikrofoner, högtalare m.m) kan även en professionell inspelning göras för skivgravering.

Ja, det var alles. Återkommer!

CHRISTMA EXEC

by Malcolm Dickinson

HERE is a self-propagating "virus" exec that has recently been released by a grumpy scrooge onto BITNET. It is called "CHRISTMA EXEC".

This virus is a sneaky one. You may find it in your reader, sent to you by a friend of yours in BITNET. If you load it from your reader and then type CHRISTMAS, it will draw a little christmas tree on your screen, then *send a copy of itself to every userid in your* * NAMES file.

This may not seem to be so malicious, until one considers the amount of file traffic such an exec can create. If you have 20 people in your NAMES file, and 15 of them run the exec that gets sent to them, and each of them has 20 people in their NAMES file, the resulting file traffic on the network will be 320 file transfers. In the next generation it will be 6400 files, and in the next generation, 128,000 files. If the exec is spread thoroughly without a concerted effort to stop it, a few million copies will be flying around the net within a week, slowing file traffic immeasurably and costing thousands of dollars in CPU time.

Modema

av Harald Eneroth

Med en hem- eller persondator, ett modem och ett terminalprogram får man tillgång till 100-tals hobbydatabaser. De flesta är gratis att använda. Detta gäller t.ex. BBS-baserna, som ofta drivs av privatpersoner på egna smådatorer. Även några större är gratis.

Här finns ett otal möten för diskussion av alla möjliga ämnen. Det rör sig om både lättsamma och seriösa inlägg. Exempel på mötesnamn är:

Allmänna	Fritt Forum
	Biorecensioner
	Pryltorget
	Senaste nyheter
	Musik
Datalogi	Modem
	Kommunikation
	Pascal
	Databaser
	Olika datorer

Kostnaden för telefonen är i 08-området bara någon krona per timme.

Modem

75/1200 bps (V23) split speed. Sänder med 75 bps, vilket räcker för att skriva på tangentbordet. Tar emot med 1200, vilket är något för snabbt för att hinna läsa. Bra för att skumma, och sedan stoppa med Ctrl-S när något intressant kommer.

300 bps (V21 alt Bell 103) Texten kommer enerverande långsamt.

1200 bps (V22 alt Bell 212A)

2400 bps Får köpas fritt efter 1/3-88, men kan abonneras redan nu. Ger en snabb filöverföring.

Man kan börja med ett modem med bara en hastighet, men det klarar då inte alla baser. I Gula tidningen finns begagnade modem för 500–1000:-. Nya kostar c:a 1000–3000:- och uppåt. De dyrare har finesser som flera hastigheter, interspeeder och autouppringning m.m. Inbyggnadsmodem är något billigare.

Leverantörer: Compusciense (Valhallav. 180), IMP-data, Elfa, Selic m.fl.

Tre anslutningskablar behövs: till dator, till telefon och till elström.

Terminalprogram

Det finns olika program som är fria d.v.s. gratis. Man betalar bara expeditionsavgift på några tior. De fås från t.ex. QZ,

Stockholm Computer Club, Goto data i Umeå, eller lättast från föreningen för det aktuella datormärket. Varje datortyp har sin version av program.

Med programmet kan man också föra över filer och fria program från databaser, med en inställning av både värd och mottagare på ett visst överföringsprotokoll ex: Kermit, X-modem, Y-modem.

Men alla program klarar inte alla hastigheter och alla protokoll.

Allt måste stämma för att överföringen ska fungera. Bägge datorerna bör alltså vara inställda på samma parametra: 8, 1, parity None eller 7, 1, parity Even

Många Kermit-program klarar 75/1200 men inte X-modem. Procomm-programmet klarar inte 75/1200 men däremot X-modem.

Exempel på att starta Kermit för PC-datorer:

Koppla in kablarna till dator, modem och telefon.

Kermit-MS>set parity none	(väljer 8,1,none)
Kermit-MS>set baud 75/1200	(välj hastighet)
Kermit-MS>set terminal VT102	(välj terminalemulering)
Kermit-MS>status	(ger information)
Kermit-MS>?	(visar kommandon)
Kermit-MS>c	(connect, anslut)

Ring upp telefonnumret. När datasignalen hörs: tryck på modemets dataknapp. Vid 300 bps vänta på signal nummer två. Lägg sedan på luren.

<return>	
<return>	
<return>	Kontakt bör vara uppnådd.

Hela startproceduren kan läggas i en .BAT-fil för automatisk start. Kontrollera också initieringsfilen (.INI).

Ctrl-]?	Ger hjälp.
Ctrl-F	Lägger skärminnehållet på fil.
Ctrl-C	Close, stänger anslutningen.
Ctrl-P	Till DOS.

Sedan kan kontakten återupptas.

Databaser

		Hastighet/bps				Parametrar
Dataforum	08-608116	300	75/1200	1200	2400	7 1 E
För ingenjörer. Bl.a. platsannonser.						
Computext	08-7740070	300	75/1200		2400	8 1 N
Livlig debatt. Liknar KOM.						
Yellow PC	08-7603312	300				8 1 N
Har många filer. Av FIDO-typ.						
Alley Cat	08-7495826	300	75/1200	1200	2400	8 1 N
Har många filer. Av OPUS-typ.						
EDKX1	08-7195789	300	75/1200	1200	2400	8 1 N
Nacka BBS	08-7153172	300		1200	2400	8 1 N
Annonsmarknaden	08-339200		75/1200			7 1 E
Videotext.						
Permobas	0764-20110	300	75/1200			7 1 E
Livlig debatt. Liknar KOM.						

Man brukar få uppge namn och välja ett lösenord. Sedan kan man få vänta någon dag på att detta registreras.

Det är ofta upptaget, spec. kvällstid, varför nummer till många alternativa baser kan behövas. Listor finns i:

- Tidningen Print Out (Stockholm Computer Club). Brukar finnas hos Databiten, Sveavägen och USR-data, Tegnérgatan.
- Tidningen Datormagazin (12:- i Pressbyrån, kommer varje månad).
- Utländska datortidskrifter, t.ex. Personal Computer

Ytterligare information om t.ex. videotext och avgiftsbelagda baser finns hos Televerket, Kungsgatan.

Slaget på Berzeliterassen

av xt @ CD (Carl Hallén)

EN AV DAGARNA som Peter Löth och JMR var i Göteborg gick åt till att valla Peter mellan div företag, medan JMR slet på Medicindata. Klockan 20 var vi trötta och utsvultna, så vi (Peter, Bengt Farre och jag) hämtade JMR och åkte till Berzeliterassen för att äta, på Löthbergs rekommendation. Till en början verkade det bra, men sedan...

Det kom in en gäst till och satte sig vid bordet intill. Det var en lätt påstruken golvläggare. Som vanligt gick käften på Löthberg hela tiden. Detta störde golvläggaren som med rop försökte få honom att låta oss andra få en syl i vädret. Vi (Löthberg) ignorerade uppmaningen, och snart hade vi mannen hos oss. Han vidhöll att visst kakhål skulle stängas.

Vi hade ännu inte fått vår mat förrän vår granne kom för andra gången. Denna gång klart förbannad över att "den jävla stockholmaren" inte klarade att hålla "brödfittan" stängd. Löthberg reste sig och sedan var det i full gång...

Så fort Löthberg fick se polisbilen reste han sig och sprang ut från restaurangen. Syndaren var på väg ner för gatan men blev snabbt upphunnen av polisen. Även JMR tog sig ut på gatan. Från våra platser på terassen kunde vi se polisen (nu sex eller sju) stå och diskutera med sitt kap. Efter en stund lämnade den stackars golvläggaren polisen och våra Stackare. Sedan var det bara för Peter & JMR att sätta sig polisbilen och bli bortkörda till polishuset.

Kvar satt bara Bengt och jag, som just fått våra pepparstekar, och samtalde utan att behöva överrösta nån "jävla stockholmare".

(Ansvaret för text inom " " är en autentisk delmängd av ordförrådet hos en göteborgsk golvläggare.)

[Golvläggaren dömdes sedermera till 2000 kronor i dagsböter för olaga hot. "Det är dyrt att inte låta mig prata så mycket jag vill", var Peters kommentar. Red:s anm.]

Laserutskrift

av Hans Nordström

Anledning av en utvärdering i höstas blev jag intresserad av laserskrivare. Vad kostar det egentligen att skriva ut en sida? Det har först och främst gällt Apples LaserWriter Plus. Priserna är de som gällde efter prissänkningen i september 1987.

Skrivaren är konstruerad för att klara av totalt 180 000 utskrifter. Varje månad skall den kunna orka med 3 000 kopior. Det låter kanske inte så mycket. Men under arbetstid är det ungefär en utskrift var tredje minut. Livslängden bör bli 5 år (180k/3k=60 mån). Det stämmer med uppgifter jag fått från annat håll.

En tonerkassett sägs klara mellan 3 till 6 000 kopior. Vilket det blir, hänger nära samman med hur mycket stora svarta ytor som finns på utskrifterna. Det är framförallt grafik och bilder som slukar kolpulver.

Många av posterna i kalkylen var redan givna av leverantörerna. Det blev genast svårare när det gällde hållbarheten. Men en notis i en amerikansk tidskrift kom till min hjälp. Där uppges att laserskrivverket Canon LBP-CX i LW Plus har ett MIBF (Mean Impressions Between Failures) på 30 000 utskrifter. Snabbt ser jag då att skrivaren kommer att behöva reparation 5 gånger under sin livstid. I artikeln nämns också att en totalkatastrof är mycket sällsynt på Canon LBP-CX. Istället är det så att största orsaken till stoppen är, just det, smuts. Genomsnittsreparationen kostar SEK 1 800.

Den här kalkylen skall inte tas som den definitiva beräkningen. De ovälkomna utgifterna är ju bara ett genomsnitt. Att datorn skulle gå sönder under de här 5 åren, finns heller inte med.

Jag skulle uppskatta om jag fick ta del av andras erfarenheter. Kanske kan jag återkomma med en till verkligheten bättre anpassad kalkyl.

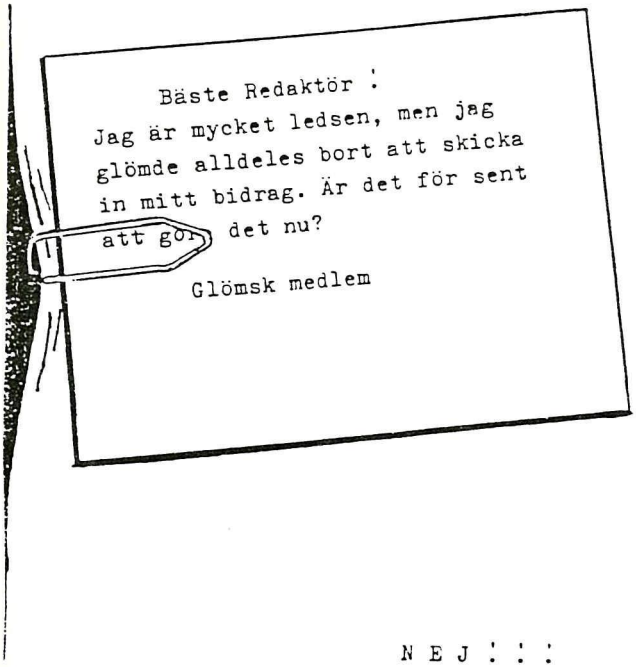
De ingående delarna för att få en utskrift på laserskrivare.

Macintosh Plus (dator)	15 900
AppleTalk-kablar	1 000
Program för utskrift (t.ex. MacWrite)	1 250
LaserWriter Plus	44 900
Servicekontrakt per skrivbord och år	3 600
Reparationsarbete för LW Plus under ett år	1 800
Toner Kassett (färgpulver)	980
Kopieringspapper, 1000 st, av bra kvalitet	33,50
El per kWh	0,27

Posternas olika bidrag i öre, till varje kopia. Med olika synsätt.

	Optimistisk	Pessimistisk
Macintosh Plus	9	9
AppleTalk-kablar	1	1
Program	1	1
LaserWriter Plus	25	25
Servicekontrakt		8
Reparationsarbeten på lasern		4
Toner Kassett 30st på 5 år	16	
Toner Kassett 60st på 5 år		33
Papper	3	3
El, 260 arbetsdagar/år i 5 år	2	
El, aldrig avstängd i 5 år		7
Kopiepriset vid 180k utskrifter under 5 år	57	94 öre

På detta priset tillkommer mvs.



HyperCard — hypertext eller hypermedia?

av Kjell Krona

AUGUSTI 1987 släppte Apple Computer ett nytt program, som i fortsättningen skall levereras med alla nya Macintosh. Programmets namn är HyperCard, och har kommit att väcka en hel del uppmärksamhet, inte minst ibland de som forskar om hypertext.

En av anledningarna till denna uppståndelse är troligen att HyperCard inte går att passa in i de traditionella programkategorierna. HyperCard innehåller element av både hypertextsystem, databas-system, programmerings-system, och av "paint-program" — samtidigt som det har många unika egenskaper. En annan anledning torde vara att inte heller användarkategorierna är lätta att avgränsa; HyperCard uppfattas olika av olika personer. Eller för att citera ur Apples press-release: "HyperCard is a personal toolkit that give users the power to use, customize, and create new information using... text, graphics, video, music, and animation."

HyperCard-systemet består av själva programmet, HyperCard, och ett antal filer eller, med HyperCards terminologi, *stackar*. Förutom de stackar som följer med, kan användare bygga egna, eller köpa från mjukvarufirmor. För dessa färdiga stackar har Apple myntat (och registrerat) begreppet *"stackware"*. En komplicerad stack kan ses som ett "miniprogram", för vilket HyperCard utgör operativsystemet; Apple lanserar också HyperCard som systemprogramvara.

Varje stack består sedan av ett antal *kort*, som lånar utav "NoteCards-metaforen". Korten i varje stack är strukturerade i vad som synes vara en cirkulär, dubbellänkad lista; det finns ett första och ett sista kort, och varje kort har ett föregående och ett efterföljande kort (första kortets "föregående" är det sista kortet). Förutom denna linjära ordning inom stacken finns det emellertid också möjlighet att länka ihop kort i eller mellan stackar i godtycklig ordning. Detta är möjligt då varje kort har ett unikt idnummer och, om så önskas, också kan ges ett namn.

Varje kort är visuellt lika stort som den ursprungliga Macintoshens bildskärm; bara ett kort kan vara synligt åt gången (vid större bildskärmar presenteras kortet i ett fönster; det finns inga möjligheter att få upp mer än ett fönster). Varje kort består av två lager, en *bakgrund* som normalt delas med andra kort i stacken, och en *förgrund* som är unik för kortet. Dessa lager kan innehålla bitmappade bilder, textfält, och *"tryckknappar"*. "Tryckknappar" utgör länkar till andra kort i eller utom stacken; genom ett "musklick" förflyttas man till det länkade kortet.

Textfälten kan innehålla upp till 32K text, kan placeras var som helst på kortet, och kan förses med "scroll bars". Bilder kan importeras via klippboken från andra program, eller ritas med de inbyggda ritmöjligheterna, som motsvarar ett vidareutvecklat MacPaint; HyperCard är för övrigt skrivet av samma person som utvecklade det ursprungliga MacPaint, Bill Atkinson.

En ny länk till ett annat kort byggs upp genom att en ny tryckknapp skapas, placeras på ett lämpligt ställe på kortet, och eventuellt modifieras till sitt visuella uttryck; ge kommandot "link to...", förflytta sig till det kort man vill länka knappen till, och konfirmera "link to this card".

Det första man ser möts av när man startar HyperCard är den s.k. "Home Stack", som utgör en ikon-baserad karta eller katalog över de stackar man normalt brukar använda. Till varje ikon hör en tryckknapp, som är länkad till motsvarande stack. Genom att klicka på ikonen förs man alltså till den stack man önskar nå. I "Home Stack" lagras också en del parametrar för HyperCards funktion. De stackar som inte är inlagda på "Home Card" kan nås genom det vanliga "open..."-kommandot. För att installera en ny stack behöver man alltså bara skapa en ny knapp, ge den ett lämpligt utseende, gå till den nya stacken, och konfirmera länken.

Användarnivåer

En stack kan, sedan den framställts, låsas från att manipuleras av obehöriga. HyperCard har inte mindre än 5 olika tillträdes-nivåer. Nivå 1, "Browsing", ger bara möjlighet att följa länkarna och titta på innehållet i stacken. Nivå 2, "Typing" ger också möjlighet att lägga till och ta bort text i texfält, liksom att lägga till nya kort. Nivå 3, "Painting", ger möjlighet att rita och sudda på korten. Först nivå 4, "Authoring", ger möjlighet att lägga till och ta bort tryckknappar, och att ändra på länkarna. Nivå 5, slutligen, kallas för "Scripting", och ger tillträde till ett inbyggt objektorienterat programmeringsspråk, *HyperTalk*. De länkar som skapas på nivå 4 är i själva verket enkla exempel på "script" i HyperTalk; när en tryckknapp länkas till ett kort, skapas automatiskt ett litet script som lyder (om kortet haft ett namn, hade detta stått i stället för id-numret):

```
on mouseup
  go to card id xxxxx
end mouseup
```

Detta är ett exempel på en "message handler", som exekveras när ett objekt får motsvarande "message". Finns ingen passande handler, skickas meddelandet vidare i systemets hierearki: button eller field → card → stack → home stack → HyperCard. Förutom "message handlers" kan man skriva också funktioner med parametrar. I övrigt har HyperTalk samma kommandon och kontrollstrukturer som traditionella programmeringsspråk, plus en del som är specifika för HyperCard. Ett exemepl är "visual effect", som bestämmer det visuella intrycket när ett kort byts mot ett annat; ett annat är "play", som kan ta en sekvens av noter och spela upp dem med ljud ifrån ett "samplat" instrument; ex:

```
play "Sax" tempo 200 c c c e
  d d d f e e d d c
```

Förutom inbyggda kommadon och funktioner, och de man kan skriva i HyperTalk, finns det dessutom möjlighet att lägga till nya sådana, genom att skriva egna funktioner i t.ex. C elller Pascal, kompilera dem, och lägga in dem som resurser i HyperCard, eller i en specifik stack. Ett exempel på en sådan funktion som cirkulerat på datornäten är "speak", som läser upp en textsträng med hjälp av MacinTalk.

Slutligen finns det också förberett för att kunna nå ut på de vanliga kommunikations-portarna till yttre enheter, t.ex. modem eller videospelare. Framförallt kan det finnas möjlighet att ta tillvara de nya möjligheter som kommer genom CD-ROM med stora mängder av data i både text-, bild-, och ljud-form.

HyperCard och hypertext

Hur relaterar sig då HyperCard till hypertext-system i övrigt, som kanske är den typ av system som ligger närmast i framtoning? Jämfört med dessa betonar HyperCard tungt användningen av bilder, ljud, och kopplingar till andra media; å andra sidan ter sig HyperCard betydligt svagare dels i möjligheten att hantera text, dels i möjligheten att få en överblick över systemets länkar, vilka kort som finns o.s.v. Det finns t.ex. för närvarande ingen möjlighet (även om det har talats om det som en möjlighet till nästa version) att länka från eller till en viss punkt i en text, utan bara från en tryckknapp (som inte följer en scrollande text) till ett kort som helhet. Detta är på sätt och vis nog för att diskvalificera HyperCard från att kallas hypertext, åtminstone enligt Jeff Conklins definitioner (Jeff Conklin, A Survey of Hypertext, MCC 1986; s. 31). Att inte kunna få en "översiktskarta" över korten i en stack är också problematiskt, framförallt när man sätter samman stacken och ännu inte har den slutliga strukturen klar. Det finns dock möjlighet att få en

översiktskarta över de senast besökta 32 korten.

Det system som ligger närmast är kanske NoteCards från Xerox, i varje fall i den metafor de bägge systemen tillgriper. Den största skillnaden är att NoteCards kan ha ett antal kort framme samtidigt, och att länkarna är betydligt mer flexibla; t.ex. kan de i NoteCards vara av olika typ; inget stöd för typade länkar existerar i HyperCard (å andra sidan kan man naturligtvis själv definiera egna typer, eftersom de "normala" länkarna bara är automatiskt genererade script, och kan ersättas med vilket HyperTalk-script som helst).

Vi kan se lite närmare på hur Conklin skulle ha definierat HyperCard. Om vi betraktar ett kort som en nod, och en knapp som en punkt, kan vi först konstatera att länkar alltid går till en nod, antingen från en punkt eller (ovanligt) från en annan nod. Denna punkt kan dock inte, som vi konstaterade ovan, följa med texten; inte heller följer de med när texten kopieras eller klippes ut, utan är helt oberoende av denna.

Dessa länkar är i sig själva bara envägs, men HyperCard lagrar kedjan av aktiverade kort så att den aktuella vägen lätt kan reverseras. Länkarna har i sig själva varken någon typ eller något namn, utan utgörs s.a.s. bara av aktiverad kod. En nod refereras alltid till genom sitt id-nummer eller sitt namn, och av sin stack (som identifieras av motsvarande filnamn. Detta gör, som Conklin också påpekar, länkarna känsliga för namnbyte och ommöbleringar i filstrukturen.

Medan de flesta hypertext-system normalt har en hierkisk struktur, som eventuellt kan förses med icke-hierarkiska länkar, utgörs HyperCard snarare av en lineär struktur, som kan förses med icke-hierarkiska länkar. Då en ny stack normalt innehåller en bakgrund med trycknappar för att manövrera framåt och bakåt i denna strktur, är det lätt att denna tar överhanden (detta speciellt när författarna inte besvärar sig med att eliminera uppenbart felaktiga lineära länkar, som bryter ned ett annars omsorgsfullt uppbyggt intryck av en icke-hierarkisk struktur).

För författande ter sig HyperCard i sin nuvarande utformning mindre lämpat. Det är dock möjligt att att det går att bygga upp en "författarmiljö" genom programmering i HyperTalk. Funktioner som att få en tryckknapp (alltså en länk) att följa med i texten är dock svåra att skapa.

Användning

Vad är då HyperCard lämpat för? Vi har redan nämnt den framtida kopplingen till CD-ROM, som kan få stå som symbol för ett helt spektrum av tillämpningar: alla som kräver någon form av relationsdatabas, utan att för den skull behöva utnyttja avancer-

ade metoder för data-verifiering och data-säkerhet. Detta kan sträckas sig ifrån persoliga anteckningar, kalendrar, o.dyl., till komplicerade informationssystem och animerade handböcker. En annan användning kommer troligen att visa sig vara prototyparbete för programmering; program som skall utföra liknande typer av uppgifter kan snabbt provas ut och demonstreras i HyperCard, och om nödvändigt för effektivitetens skull sedan översättas till traditionella programmeringsspråk.

I övrigt kan man kanske säga att HyperCard kan användas till allt som någon annars skulle ha använt BASIC till; för att göra små, ibland triviala program som utför en för stunden speciell uppgift, och som inte motiverar användning av traditionell programmering.

Problem

Finns det då inga problem med HyperCard? Förutom de som har berörts tidigare, i form av avsaknad av automatisk karta och omöjligheten att referera fråm en viss punkt i en text, är det största problemet med HyperCard att det i flera avseenden avviker från det numera vedertagna sätt som Macintosh-program arbetar på. Detta görs utan att öka funktionaliteten eller begripligheten i användar-gränssnittet; förhoppningsvis kommer dessa skönhetsfläckar att elimineras i framtiden. Ett annat, på sikt mindre problem, är att HyperCard i praktiken kräver ett fast skivminne, och arbetar bättre med tillgång till >1MB internminne.

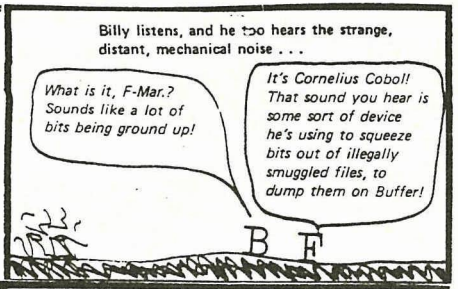
Sammanfattning

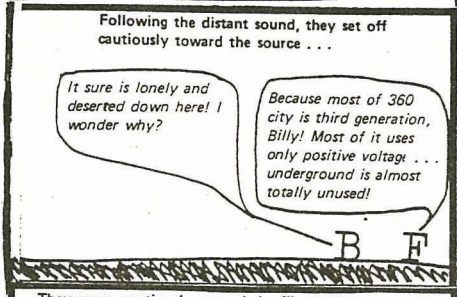
HyperCard utgör tveklöst ett av de intressantaste programkoncepten under senare hälften av 1980-talet. Framförallt genom att sudda ut gränsen mellan programanvändning, applikationsgenerering (om vi kan kalla länkandet av olika kort så) och fullödig programmering utgör HyperCard en milstolpe.Genom att ge möjlighet åt snart sagt varje person att själv bygga upp sin privata informationshantering, utvecklas datorn ytterligare mot ett redskap för den personliga kreativiteten. Och genom sin möjlighet att distribuera kommersiellt framtagen information, bland annat genom CD-ROM, kommer HyperCard att spela en viktig roll i utvecklingen mot nya medietyper. Därför är det kanske rimligare att benämna HyperCard "hypermedia", snarare än "hypertext".

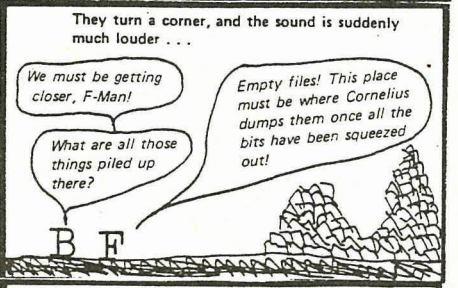
FORTRAN MAN

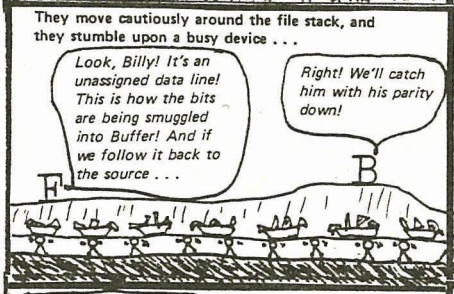
Lee Schneider

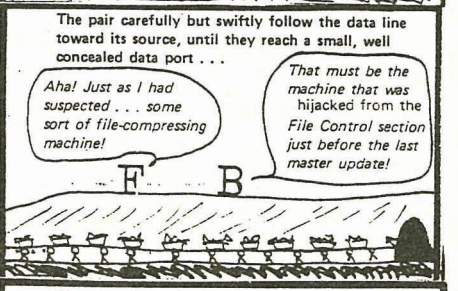
Todd Voros

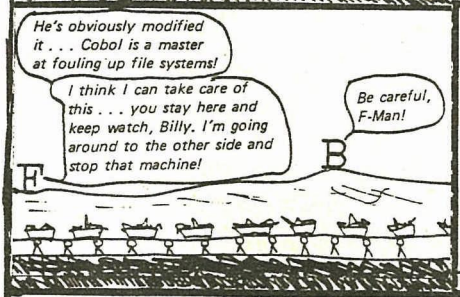


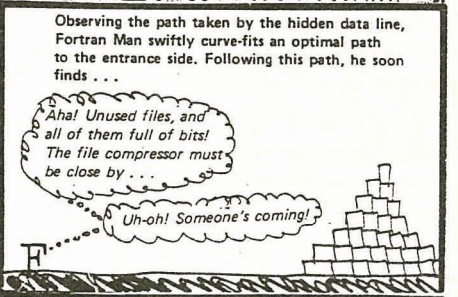


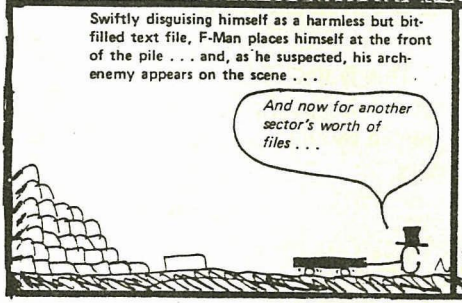


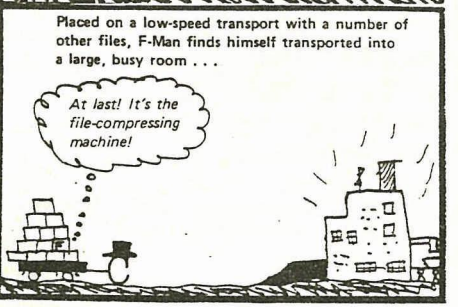












Virus Invades Lehigh University

by Kenneth R. van Wyk

LAST week, some of our student consultants discovered a virus program that's been spreading rapidly throughout Lehigh University. I thought I'd take a few minutes and warn as many of you as possible about this program since it has the chance of spreading much farther than just our University. We have no idea where the virus started, but some users have told me that other universities have recently had similar problems.

The virus: the virus itself is contained in the stack space of COMMAND.COM. When a pc is booted from an infected disk, all a user need do to spread the virus is to access another disk via TYPE, COPY, DIR, etc. If the other disk contains COMMAND.COM, the virus code is copied to the other disk. Then, a counter is incremented on the parent. When this counter reaches a value of 4, any and every disk in the PC is erased thoroughly. The boot tracks are nulled, as are the FAT tables, etc. All Norton's horses couldn't put it back together again... :-) This affects both floppy and hard disks. Meanwhile, the four children that were created go on to tell four friends, and then they tell four friends, and so on, and so on.

Detection: while this virus appears to be very well written, the author did leave behind a couple footprints. First, the write date of the command.com changes. Second, if there's a write protect tab on an uninfected disk, you will get a WRITE PROTECT ERROR... So, boot up from a suspected virus'd disk and access a write protected disk—if an error comes up, then you're sure. Note that the length of COMMAND.COM does not get altered.

I urge anyone who comes in contact with publicly accessible (sp?) disks to periodically check their own disks. Also, exercise safe computing—always wear a write protect tab. :-)

This is not a joke. A large percentage of our public site disks has been gonged by this virus in the last couple days.