Internett

Das Internet für NEXTSTEP gezähmt – Teil 2

von Uli Zappe

Nachdem wir in der ersten Folge den Anschluß ans Internet eher grundsätzlich besprochen haben, geht es nun ans Eingemachte. Dieser Teil ist sehr ausführlich gehalten, so daß auch diejenigen, die sich bislang noch nie an UNIX getraut haben, damit nicht nur zurechtkommen sollten, sondern (hoffentlich) auch verstehen können, was sie da tun. Alle erforderlichen Programme gibt es auf der neuen *Peanuts*-CD und dem *Peanuts*-FTP-Server; auf letzterem liegt auch ein Dokument mit sämtlichen hier verwendeten Konfigurationsdateien, so daß man sich mühsames Abtippen ersparen kann.

Doch zunächst noch drei Korrekturen zum ersten Teil. Kaum war der verfaßt, hat NeXT es tatsächlich doch noch geschafft, einen funktionsfähigen Treiber für die serielle Schnittstelle von Intel-Computern auf die Beine zu stellen, der sogar einige Vorzüge gegenüber Mux aufzuweisen hat. Daher werden wir entgegen der Besprechung im ersten Teil für die Installation den neuen Treiber von NeXT verwenden.

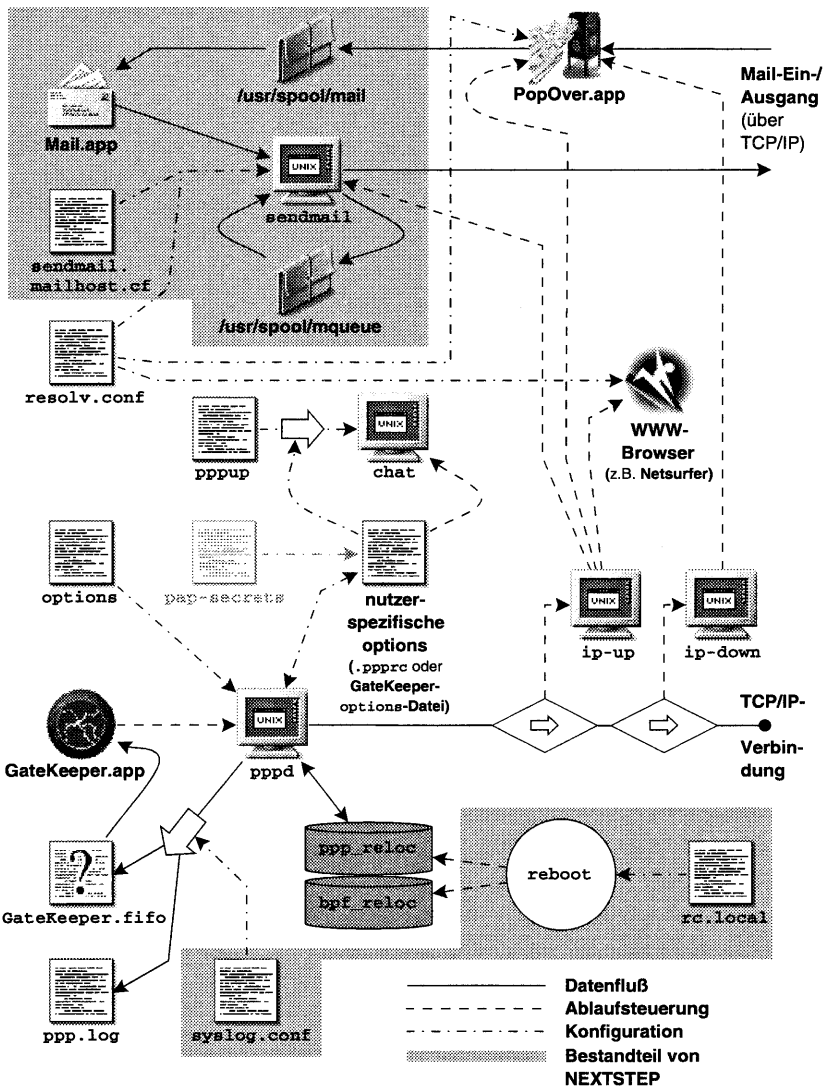
Desweiteren ist bei der Darstellung der Funktion der Netzmaske leider etwas durcheinander geraten; sie dient nicht zur Einordnung von *ankommender* Post, sondern ganz im Gegenteil von *abgesandter*: Alle Post, deren Adresse in den von der Netzmaske nicht ausgeblendeten Stellen (und das sind ja die, die im gesamten lokalen Netz gleich sind und seine Identität ausmachen) mit der Adresse des eigenen Rechners – und also der Identität des lokalen Netzes – *nicht* übereinstimmt, kann folglich nicht lokal zugestellt, sondern muß ins Internet geschickt werden. Sie wird also an den Router gesandt, der somit die **Default-Route** des lokalen Netzes darstellt: an ihn geht alle Post, deren Adresse nicht anderweitig näher bestimmbar ist.

Schließlich hat sich herausgestellt, daß der **Reply-To**-Funktion beim Postversand zwar das eine oder andere Unbill, *nicht* aber das Versagen der automatischen Belegfunktion von NeXTMail zuzurechnen ist. Dafür ist vielmehr ein anderer Fehler verantwortlich, der unten bei der Konfiguration von **sendmail** genau besprochen wird.

Wie funktioniert's?

Vor der eigentlichen Installation soll im Folgenden das Zusammenspiel der zu installierenden und konfigurierenden Programme noch etwas genauer betrachtet werden (Der upzudatende serielle Treiber ist hierbei nicht berücksichtigt).

Übersicht:



Zunächst fällt auf, daß es außer den UNIX-Programmdateien recht viele Textdateien gibt. Dabei handelt es sich ausnahmslos um *Konfigurationsdateien*, die es erlauben, das Verhalten der Programme genau den eigenen Bedingungen anzupassen: wenn die UNIX-Programme ablaufen, lesen sie diese Dateien und berücksichtigen die darin enthaltenen Informationen. Das ist unter UNIX der Standardmechanismus, sich das System wunschgemäß einzurichten.

Mail.app bedient sich wie gesagt des UNIX-Programms `sendmail`, um geschriebene Post zu versenden (das u.U. seinerseits noch weitere Programme zu Hilfe nimmt, was uns aber hier nicht kümmern muß). `sendmail` schickt die Post wenn möglich sofort auf die Reise; wenn aber die Verbindung zum Internet unterbrochen ist, lagert es alle Post, die dann nicht zugestellt werden kann, in **mqueue**. Um zu wissen, nach welchen Regeln genau es bei seiner Tätigkeit mit Absender- und Empfängeradresse umzugehen hat, berücksichtigt `sendmail` die Konfigurationsdatei `sendmail.mailhost.cf` (der Teufel steckt hier im Detail, weswegen diese Datei gräßlich kompliziert ist). Schließlich muß `sendmail` in der Lage sein, die logischen IP-Adressen in numerische zu wandeln (da letztlich eben nur die von Rechnern verstanden werden) und wissen, welcher Domain der eigene Rechner angehört, um die korrekte Absenderadresse bilden zu können. Würde Post nur innerhalb eines NEXTSTEP-Netzwerkes verschickt, bräuchten wir uns darum nicht zu kümmern, da NEXTSTEP bei der Netzwerkkonfiguration alle erforderlichen Informationen in einer sogenannten **netinfo**-Datenbank abspeichert. Wollen wir aber den Weg ins Internet öffnen, so müssen wir `sendmail` den Namen der übergeordneten Domain unseres Providers (der wir dann ja auch angehören, und die `sendmail` dem lokalen **netinfo** natürlich nicht entnehmen kann) und die Adresse eines UNIX-üblichen **Nameservers** mitteilen, der die notwendigen Übersetzungstabellen von logischen in numerische Adressen bereithält. Das geschieht in der Konfigurationsdatei `resolv.conf` (von *to resolve* = die logische Adresse in Nummern *auflösen*). Die Adressen der Nameserver in `resolve.conf` sind in diesem einzigen Fall *unbedingt numerisch* anzugeben – schließlich würde sich sonst die Katze in den Schwanz beißen beim Versuch, die numerische Adresse des Nameservers aus der logischen erst noch ermitteln zu müssen. Auf `resolve.conf` greifen übrigens genauso **PopOver.app** und *alle* weiteren Internet-Programme zurück, wenn sie Verbindung mit logischen Adressen aufnehmen wollen – in obiger Skizze exemplarisch mit dem WWW-Browser und FTP-Programm **Netsurfer.app** versinnbildlicht.

Um empfangene Post anzuzeigen, schaut **Mail.app** regelmäßig im Ordner **/usr/spool/mail/** nach, ob etwas angekommen ist – wie oft, läßt sich in den Präferenzen von **Mail.app** einstellen; wer ungeduldig ist, kann ruhig *1 Minute* eintragen. Nur kann die Post bei einer bloß temporären Verbindung zum Internet eben nicht automatisch eintreffen, weswegen **PopOver.app** sie gezielt vom Provider abholt (dessen Adresse es wiederum mit Hilfe von `resolve.conf` ermittelt) und in dem Ordner ablegt. (In der Kombination mit **PopOver.app** wäre es am elegantesten, **Mail.app** würde nicht permanent in **/usr/spool/mail/** nachsehen, sondern nur durch **PopOver.app** nach dem Empfang neuer Post zum Leeren veranlaßt; leider ist eine entsprechende Vorrichtung in **Mail.app**, das für permanente Netzverbindung ausgelegt ist, nicht vorhanden.)

PPP besteht bei näherem Hinsehen aus zwei Teilen: um **PPP** so eng wie erforderlich mit dem Betriebssystem verzahnen zu können, muß ein Teil davon, `ppp_reloc` und `bpf_reloc`, zusammen mit NEXTSTEP gestartet werden. Dabei handelt es sich um *L*oadable *K*ernel *S*erver (**LKS**): Module, die zu dem *Kern* des Betriebssystems *dazugeladen* werden und ihre Fähigkeiten dann als *Server* einem anderen Programm anbieten können. In diesem Fall ist dieses andere Programm `pppd`, der **PPP**-*D*ämon. Ein Dämon unter UNIX hat nichts Dämonisches an sich, sondern ist vielmehr ein Prozeß, der brav im Hintergrund »wie ein Heinzelmännchen« seine Aufgaben verrichtet. `pppd` ist also die Schnittstelle zur Außenwelt, es ist das Programm, das tatsächlich gestartet wird, wenn **PPP** laufen soll, und das seinerseits dann Zugriff auf den **LKS** hat.

Damit der **LKS** beim Booten auch tatsächlich jedesmal automatisch mitgestartet wird (von Hand wäre das etwas mühsam, und wozu hat man seinen Computer?), muß in die Konfigurationsdatei `rc.local` ein entsprechender Eintrag aufgenommen werden. *rc* steht für *reboot command*, Kommandos also, die beim erneuten Booten des Rechners ausgeführt werden sollen. Es gibt eine ganze Reihe solcher rc-Konfigurationsdateien, alle im Pfad **/etc/**. Mit *local* wird in der UNIX-Welt Software bezeichnet, die nicht generell zum System gehört, sondern nur auf diesem spezifischen Rechner zu finden ist, eine lokale Besonderheit sozusagen. Da **PPP** (noch) nicht zu NEXTSTEP gehört, verwenden wir zu seinem Start die `rc.local` Datei.

Wenn wir nun `pppd` starten, liest es die `options` Konfigurationsdatei, in der einige allgemeine, systemweit gültige Einstellungen angegeben sind, z.B. die Schnittstelle, an der das Modem / der ISDN-Adapter angeschlossen ist. Eine weitere, nutzerspezifische Optionsdatei teilt `pppd` die konkreten Informationen für die gewünschte Verbindung mit. Diese Datei mit Namen `.ppprc` (*ppp r*eboot command; *reboot* bezieht sich hier freilich nur darauf, daß die Datei bei jedem Neustart des *PPP-Programms* eingelesen wird) liegt im Heimverzeichnis des jeweiligen Nutzers; da sie mit einem ».« beginnt, ist sie, wie für eine Konfigurationsdatei sinnvoll, nur mit der Option *UNIX-Experte* (siehe die erste Folge des Artikels) sichtbar. **GateKeeper.app** bedient sich allerdings stattdessen spezieller Dateien, die dem einzelnen Nutzer darüberhinaus komfortable Auswahlmöglichkeiten gestatten. Umgekehrt könnte man übrigens im simpelsten Fall – wenn nur *ein* Nutzer stets nur *eine* bestimmte Verbindung benötigt – auf die nutzerspezifische Optionsdatei ganz verzichten und alle notwendigen Angaben in die `options` Datei schreiben; da **GateKeeper.app** aber aufgrund seiner Funktionsvielfalt ohnehin die beiden Dateien benötigt, gehen wir hier gleich von ihnen aus.

Die wichtigste Aufgabe der nutzerspezifischen Optionsdatei ist es, `pppd` zum Start eines weiteren UNIX-Programms zu veranlassen, nämlich `chat`. `chat` wählt das Modem / den ISDN-Adapter an, baut die Verbindung zu dem Rechner des Providers auf und gibt ihm das Kommando, seinerseits **PPP** zu starten (und heißt deswegen *chat*, weil es sozusagen durch *Plaudern* mit der Gegenseite die Kommunikationsparameter festlegt). Die nötigen Informationen wie etwa die Telefonnummer des Providers und das Paßwort entnimmt `chat` einer wiederum nutzer- (und verbindungs-)spezifischen Datei `pppup` (zum *Aufbau* von *PPP*), deren genauer Pfad ihm von `.ppprc` vorgegeben wird, das somit gezielt eine bestimmte `pppup`-Datei mit spezifischen Angaben wählen kann. Bei einem speziellen **PPP**-Betriebsmodus, dem sogenannten *synchronen* **PPP** (s.u.), übernimmt `pppd` allerdings die Nutzer-Identifizierung mittels Paßwort selbst; daher ist – *nur in diesem Fall* – das *Paßwort* nochmals in eine eigene Datei, `pap-secrets`, ausgelagert. Ist **PPP** auf beiden Rechnern gestartet, verhandelt es selbständig miteinander, bis die PPP-Verbindung steht. Sobald dies der Fall ist, startet `pppd` das UNIX-Programm `ip-up` (die *IP*-Verbindung ist jetzt *auf*gebaut), das nun seinerseits all die Aktionen veranlaßt, die zu Beginn einer Verbindung zum Internet ablaufen sollen: `sendmail` sendet die in **/usr/spool/mqueue** angesammelte Post ins Internet, **PopOver.app** holt beim Provider angesammelte Post aus dem Internet ab. Wer will, kann ausserdem seinen WWW-Browser (hier als Beispiel **Netsurfer.app**) aufrufen und diesen in den Präferenzen so einstellen, daß er beim Start automatisch eine bestimmte Seite anzeigt. Wählt man dabei die Seite aus, auf der der Provider über eventuelle Störungen in der Internet-Anbindung informiert, so erhält man zu Beginn jeder Verbindung Bescheid, ob es technische Probleme gibt. `ip-down` wird umgekehrt von `pppd` bei Beendigung der PPP-Verbindung aufgerufen und schließt die gewünschten Programme; auf alle Fälle sollte es das mit **PopOver.app** tun, da dieses Programm je nach Einstellung regelmäßig beim Provider Post abrufen will und dann ins Leere laufen würde. `sendmail` muß nicht abgestellt werden, da es sich nach dem Leeren von **/usr /spool/mqueue** selbst beendet.

Statt `pppd` umständlich über UNIX-Befehle zu starten und zu beenden, kann dies der Nutzer über **GateKeeper.app** bewerkstelligen, das ihm sehr komfortable Steuerungs- und Kontrollmöglichkeiten bietet. *Im Prinzip* tut **GateKeeper.app** aber tatsächlich nichts anderes, als `pppd` zu starten bzw. zu beenden und ihm dabei eine vom Nutzer wählbare, spezifische Optionsdatei zuzuweisen.

`pppd` gibt Informationen über den Zustand der Verbindung aus, die von `syslogd`, dem *S*ystem-*Log*buch-*D*ämon (einem oben nicht eingezeichneten Dämon, der in NEXTSTEP enthalten ist) empfangen und an den Ort weitergeleitet werden, der in der Konfigurationsdatei `syslog.conf` angegeben ist. Diese Datei nennen wir einleuchtenderweise `ppp.log`. In unserem Fall gehen wegen **GateKeeper.app** die Daten darüber hinaus aber noch an eine zweite Adresse, `GateKeeper.fifo`. Dabei handelt es sich allerdings nicht um eine Datei; vielmehr macht **GateKeeper.app** daraus einen **FIFO** (*F*irst *I*n *F*irst *O*ut), sozusagen einen Durchgangskanal, der die Informationen in der Reihenfolge ihres Eingangs weiterreicht, und zwar in diesem Fall an **GateKeeper.app**, das somit in der Lage ist, diese Informationen in einem Fenster anzuzeigen, sobald sie eintreffen.

Bleibt noch eine Frage: Woher weiß `ip-up` (und entsprechend `ip-down`), welche Programme genau es starten soll, wo es doch auf keine Konfigurationsdatei Zugriff hat? Die Antwort ist, daß `ip-up` selbst nur eine Datei ist, die die entsprechenden Angaben enthält. Aber wieso ist sie dann ein UNIX-Programm? Das liegt daran, daß es zwei Arten von UNIX-Programmen gibt, die sich hinter dem UNIX-Icon verbergen können. Beiden gemeinsam ist, daß sie über **Terminal.app** gestartet werden, wenn man sie doppelklickt. Aber hinter der *einen* Art verbergen sich »richtige« Programme, die mit einer Programmiersprache wie C erstellt und dann *kompiliert* (in Binärcode übersetzt) wurden, wie alle NEXTSTEP-Applikationen auch. Betrachtet man ihren Inhalt, so sieht man nur für Menschen völlig unleserliches Kauderwelsch – Binärdaten eben. Um diesen Inhalt von Nicht-Text-Dateien überhaupt betrachten zu können, zieht man eine solche Datei auf das Icon von **Edit.app** (das sich dazu im Dock, also der Leiste von Applikationen rechts am Bildschirmrand, befinden muß) und drückt dabei die **Command**-Taste – **Edit.app** wird so gezwungen, die Datei zu öffnen, als sei sie ein Textdokument. Oder man wählt die Datei durch einen einfachen Klick auf ihr Icon im **Workspace Manager** an und öffnet sodann den Inspektor, das Popup auf *Werkzeuge*; hier kann man dann durch einen Doppelklick auf das Icon von **Edit.app** angeben, daß man die Datei in dieser Applikation öffnen will.

Schauen wir uns nun die *andere* Art von UNIX-Programmen auf diese Weise an, so sehen diese in **Edit.app** tatsächlich wie ganz normale ASCII-Textdokumente aus – und genau das sind sie auch. Es handelt sich dabei um sogenannte *Shellskripte*. Oben wurde bereits einmal erwähnt, daß sich über eine Shell (ein offenes Terminalfenster) von **Terminal.app** mit Hilfe von Texteingaben mit den UNIX-Programmen von NEXTSTEP (und so man will auch den Applikationen) kommunizieren läßt. Ein *Shellskript* ist nun nichts weiter als eine solche, schon vorbereitete Kommunikation: **Terminal.app** verhält sich so, als würden die Texte des Shellskripts gerade eben eingegeben und setzt sie entsprechend um. Was bei einem kompilierten UNIX-Programm also schon vorher ein für allemal geschehen ist – die Umsetzung in Binärcode –, muß **Terminal.app** beim Shellskript jedesmal live aufs Neue erledigen: es *interpretiert* das Shellskript (im Gegensatz zu dem einmaligen *Kompilieren*). Shellskripte sind technisch mithin umständlicher und in den Möglichkeiten beschränkter als kompilierte Programme; dafür sind sie ungleich leichter zu schreiben, vor allem aber jederzeit schnell neuen Situationen anpaßbar (man muß ja nur ein paar Textzeilen ändern) und für einfachere Aufgaben voll ausreichend.

Ein Shellskript besteht also aus besagten Anweisungen und *Kommentarzeilen*, die durch ein # am Zeilenanfang kennt-

lich gemacht werden (wie bei den meisten Konfigurationsdateien auch). Alles, was in diesen Kommentarzeilen folgt, wird nicht berücksichtigt. So kann man Kommentare zu seinem Programm verfassen, aber auch auf einfache Weise Zeilen »ausschalten«, die man vorübergehend außer Kraft setzen möchte. Die *erste* Zeile eines Shellskripts lautet auf NEXTSTEP in der Regel `#!/bin/sh` und macht **Terminal.app** klar, daß es die folgenden Zeilen mit Hilfe des (»richtigen«, d.h. kompilierten) UNIX-Programms `/bin/sh` (*sh* steht für *shell*) interpretieren soll; statt des Programms `/bin/sh` werden u.U. auch andere, vergleichbare Programme eingesetzt.

Wenn wir nun solch ein Shellskript in **Edit.app** einschließlich der ersten Zeile korrekt verfassen und abspeichern, so werden wir im **WorkspaceManager** dennoch nur das Icon für eine Textdatei zu sehen bekommen, und bei einem Doppelklick wird sie sich einfach wieder in **Edit.app** öffnen. Auch wenn wir sie in einen der speziellen Pfade legen, in denen **Terminal.app** gezielt nach UNIX-Programmen sucht, so daß zu ihrem Start nur ihr Name und nicht ausdrücklich der gesamte Pfad ins Terminalfenster eingegeben werden muß, werden wir die Meldung kriegen, das Programm sei »nicht ausführbar«. Wo liegt der Fehler? Wir müssen NEXTSTEP noch mitteilen, daß diese Datei ein ausführbares Programm sein soll. Dazu öffnen wir im **WorkspaceManager** den Inspektor und stellen das Popup auf *Zugriffsrechte*. Dann klicken wir die betreffende Datei an: in der Zeile *Ausführen* des Inspektors werden jetzt nur Kreuze zu sehen sein. Ändern wir durch einen Mausklick ein oder mehrere Kreuze in Häkchen und erteilen so entsprechende Rechte zur Programmausführung, so ändert sich das Textsofort in ein Programmicon. Von nun an können wir das Skript von **Terminal.app** aus durch Eingabe des Namens aufrufen oder es durch einen Doppelklick starten; um es erneut in **Edit.app** zu bearbeiten, müssen wir es wie oben beschrieben auf dessen Icon ziehen und die **Command**-Taste gedrückt halten. Ob das Programm nach seinem Start freilich etwas Sinnvolles tut, hängt davon ab, was wir hineingeschrieben haben.

Hinweis: Bei Shellskripten und Konfigurationsdateien gilt, daß sie in **Edit.app** unbedingt als ASCII-Texte abgespeichert werden müssen; mit RTF kann UNIX nichts anfangen!

Programme, Programme

Es folgen nun noch einige Bemerkungen zu den verwendeten Programmen **PPP-2.2** und **GateKeeper**, die ziemlich ins Detail gehen und auch ergänzend zur Installation gelesen werden können. Beide Programme befinden sich im Augenblick noch im Betastadium, so daß es noch einige Haken und Ösen gibt.

PPP-2.2 gibt sich sichtlich Mühe mit der Dokumentation, trotzdem tritt leicht einige Verwirrung auf. Vor allem hat **PPP-2.2** eine zunächst recht unübersichtliche Ordnerstruktur, wenn man den Hauptordner **/usr/local/ppp** öffnet. Die ist nämlich »dreifachgemoppelt«. **PPP** befindet sich noch in einem kontinuierlichen Entwicklungsprozeß, daher wollte der Autor Updates leicht machen. *Tatsächlich befinden sich alle Dateien in einem Ordner* namens **ppp-2.2-x-x-x** wobei letztere Ziffern die exakte Version des **LKS** angeben. *Verwendet* werden allerdings die Dateien in den einzelnen Ordern **/bin**, **/man** und **/reloc**. Diese scheinbaren Dateien sind in Wahrheit (wie im Inspektor des **WorkspaceManager** in der Einstellung *Attribute* zu sehen) aber nur Verweise auf die nämlichen Dateien im Ordner **/version**, und dieser Ordner ist insgesamt wiederum ein Verweis auf den eigentlichen Programmordner **ppp-2.2-x-x-x**. Der Gag daran: will man testweise auf eine andere Version (auch eine frühere!) wechseln, die sich zusätzlich an gleicher Stelle (in einem weiteren Ordner der Art **ppp-2.2-x-x-x**) befinden kann, so muß man *nur* den Verweis des Ordners **/version** auf den Ordner mit dieser entsprechenden Version ändern, und schon sind alle Dateien in den einzelnen Ordnern umgestellt. Es gibt noch drei weitere einzelne Ordner, die man getrost vergessen (und wenn man will auch tatsächlich löschen) kann: in **/src** befindet sich lediglich ein Hinweis, wo man den Source(Quell-)code des Programms beziehen kann, in **/examples** befinden sich eine Menge Beispielskripte, die zwar gut gemeint, aber leider so inkonsistent sind, daß sie mehr verwirren, als nützen und zu allem Überfluß auch noch mit **GateKeeper** kollidieren; der Ordner **/scripts** schließlich ist leer und soll zur sicheren (er wird vom PPP-Installationsprogramm nicht verändert) Aufbewahrung der eigenen, speziell konfigurierten Skripte dienen – die müssen aber zur Verwendung ohnehin im Ordner **/etc** abgespeichert werden, der ebenso sicher ist, so daß sich auf **/scripts** verzichten läßt. **ACHTUNG!** In **/examples** befindet sich *auch* die Installationsanleitung mit dem schönen Namen **README.NeXT.MAB.Installation** (*MAB* steht für *M*ulti *A*pplication *B*inary – also eine Binärdatei, die Programmcode für verschiedene Hardware-Plattformen enthält). Diese Datei findet sich zwar auch im Ordner **/usr/local/ppp** selbst, doch ist das bei genauerem Hinsehen nur ein Verweis – will man **/examples** der Übersichtlichkeit halber entfernen, so muß man folglich vorher den Verweis löschen und die Anleitung selbst aus **/examples** an dessen Stelle verschieben.

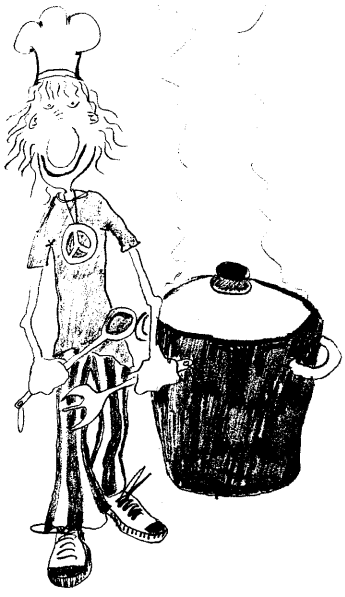
(Zumindest) gedanklich kann man also so tun, als gäbe es in dem **ppp**-Ordner nur die vier Ordner **/bin**, **/man**, **/etc** und **/reloc** – so sieht die Sache schon wesentlich übersichtlicher aus: in **/bin** sind die Programmdateien (die *BIN*ärdateien), in **/reloc** die **LKS**, in **/etc** die angepaßten Skripte und Konfigurationsdateien und in **/man** schließlich die Ergänzungsseiten für das UNIX-Manual. Bekanntlich kann man für jedes UNIX-Programm in **Terminal.app** Informationen aufrufen, indem man man *Programmname* eingibt; es ist gute Tradition, beim Erweitern des Systems durch neue UNIX-Programmdateien auch das Handbuch entsprechend zu erweitern.

Wie schon erwähnt, müssen sowohl UNIX-Programmdateien als auch die ergänzenden Handbuchseiten unter bestimmten (Standard-)Pfaden abgespeichert werden, damit **Terminal.app** sie finden kann. Welche das sind, wird für jeden User getrennt festgelegt durch eine Datei namens `.cshrc` (*csh* steht für *C-Shell*, das ist der spezielle Shell-Typ von **Terminal.app**, *rc* meint wiederum *reboot command*; die Datei beginnt ebenfalls mit einem ».« und ist also nur mit der Option *UNIX-Experte* (s.o.) sichtbar). Alle **PPP**-Dateien liegen nun aber in einem neuen Ordner, der folglich in den UNIX-Standardpfaden nicht auftaucht. Die Anleitungen zu **PPP** und **GateKeeper** empfehlen unisono, deshalb alle Dateien in Ordner des Standardpfades zu linken (*Linken*, also einen Verweis erstellen, sieht im **WorkspaceManager** fast wie kopieren aus: man schiebt die entsprechende Datei in den gewünschten Ordner, drückt dabei allerdings die **Control**-Taste, so daß der grüne Doppelpfeil erscheint, und erzeugt so etwas, das genau wie die Datei aussieht, in Wirklichkeit aber eben nur ein Verweis auf sie ist). Ich halte das nicht für sinnvoll, da unter der Vielzahl von Verweisen die Übersichtlichkeit stark leidet (wie man innerhalb des PPP-Ordners sehen kann). Da sich die Verweise von den Dateien optisch ja nicht unterscheiden, ist in dem Ordner des Standardpfades, in dem sich sowieso schon sehr viele Dateien befinden, gar nicht mehr ohne weiteres erkennbar, wo es sich nur um Verweise handelt. Zudem lassen sich die Verweise nicht mit einem gemeinsamen Kommando rückgängig machen, falls man das wünscht. Erweitert man hingegen `.cshrc` um den entsprechenden Pfad, so stehen alle neuen Dateien auf einen Schlag zur Verfügung, und ebenso leicht ist dies rückgängig zu machen. Ich wende daher dieses Verfahren an. Allerdings muß für *jeden* Nutzer, der **PPP** verwenden können soll, `.cshrc` getrennt angepaßt werden, oder aber – falls keine speziellen Konfigurationsaufgaben dagegen sprechen, ohnehin sinnvoll – man linkt die `.cshrc`-Datei von **root** auf alle anderen Nutzer-Verzeichnisse und ersetzt somit deren eigene `.cshrc`-Dateien. Auf diese Art und Weise ist jede Änderung der `.cshrc`-Datei sofort für alle Nutzer wirksam.

Die Beispiele für die erforderlichen Konfigurations- und Skriptdateien von **PPP** im Ordner **/examples** sind wie gesagt eher verwirrend; abgesehen davon, daß sie kaum systematisch kommentiert sind und teils den europäischen Verhältnissen nicht entsprechen, liegt das vor allem an einem Punkt: `pppd` benötigt in allen Konfigurationsbeispielen sowohl die Datei `options` als auch die Datei `pppup`. Man hat aber die beiden Möglichkeiten, entweder (wie oben) `pppd` direkt zu starten, das in Folge aufgrund der Einträge in der nutzerspezifischen Options-Datei `chat` aufruft, welches seinerseits `pppup` ausliest – in diesem Fall ist `pppup` eine *Konfigurationsdatei*. *Oder* man startet ein anders angelegtes `pppup`, das seinerseits `chat` und das durch `options` konfigurierte `pppd` aufruft – in diesem Fall ist `pppup` eine *Programmdatei*. Es ist ersichtlich, daß `pppup` und `options` gemeinsam für den einen *oder* den anderen Fall gedacht sein müssen – und genau das sind sie bei den Beispielen in **/examples** nicht. Mehrheitlich folgen die Beispiele dabei der zweiten Variante, und das ist zu allem Überfluß die, die **GateKeeper** *nicht* verwendet.

Obwohl **GateKeeper** eine schön gemachte Applikation ist, läuft sie leider noch nicht völlig zuverlässig. Insbesondere mit dem FIFO kann es Probleme geben. Bei fehlerhaftem Verbindungsaufbau stürzt **GateKeeper** des öfteren ab und zerstört den FIFO dabei, so daß er beim nächsten Versuch keine Informationen mehr über den Aufbau der Verbindung anzeigen kann. In diesem Fall hilft es, in den Präferenzen (Popup in Stellung *defaults*) den Schalter *Display pppd Diagnostics* aus- und wieder einzuschalten; so wird der FIFO neu aufgebaut.

Installations-Kochbuch



Genug der Theorie; es folgt die Praxis. Wer rasch installieren will, braucht sich nur um die normalgroß gedruckten Passagen zu kümmern; in den kleiner gedruckten Abschnitten finden sich Erläuterungen zum Verständnis.

Es wird davon ausgegangen, daß von Seiten des Nutzers keine Pfade für UNIX-Programme und Konfigurationsdateien in NEXTSTEP verändert wurden; gegebenenfalls sind die hier verwendeten Pfade entsprechend anzupassen.

Voraussetzungen

Hardware

Die folgende Anleitung geht von einem **Intel-Rechner** aus.

NeXT-Rechner sind mittlerweile historisch, und wer eine Workstation von **Sun** oder **HP** sein eigen nennt, wird auch wissen, wie man UNIX konfiguriert. Im Prinzip sollte die Anleitung jedoch auch für diese Rechner taugen; von **GateKeeper** muß man sich für SUN und HP allerdings den *Sourcecode* holen und ihn selbst kompilieren.

Benötigt werden zudem ein **Highspeed-Modem (‡ 14.400 baud)** *oder* ein externer **ISDN-Adapter** (»ISDN-Modem«) und eine **Ethernet-Karte**.

Ohne ein **Modem** mit mindestens 14.400 baud ist das Internet sinnvoll nicht zu benutzen. Aber selbst ein Modem mit 28.800 baud kann den Internet-Zugang noch ziemlich lähmen. Zwar sind 28.800 baud nominell schon fast halb so schnell wie ISDN, doch wird mit dieser Übertragungsrate die analoge Technik derart ausgereizt, daß die Fehlerrate enorm ansteigt, die die *effektive* Übertragungsrate wiederum bremst. **ISDN** ist daher de facto vor allem in punkto Zuverlässigkeit auch einer 28.800 baud schnellen Analogverbindung *deutlich* überlegen.

ISDN-Adapter sind externe Geräte, werden über die serielle Schnittstelle genau wie Modems angeschlossen und sind auch über denselben Befehlssatz (AT-Kommandos) bedienbar; sie benötigen daher im Gegensatz zu internen ISDN-Karten keine speziellen Treiber, die es für NEXTSTEP noch nicht gibt, und alle Kommunikationsprogramme laufen sofort wie gewohnt. Außerdem bieten sie den Vorteil der optischen Verbindungskontrolle über LEDs. Manche dieser Geräte enthalten zusätzlich einen analogen Modem-Teil, der über das ISDN-Netz zu anderen analogen Anschlüssen Kontakt aufnehmen kann, und sind somit sehr universell einsetzbar (z.B. analoges Fax!). Mittlerweile gibt es von einigen großen Anbietern (u.a. Connect Service Riedlbauer) auch spezielle serielle **Schnittstellenkarten**, die den relativ hohen Geschwindigkeiten bei ISDN besser gewachsen sind und – besonders trickreich – *hardwaremäßig* die vom Betriebssystem für die serielle Schnittstelle vorgegebene Übertragungsgeschwindigkeit verdoppeln oder vervierfachen. Aus den **57600 Bit/s**, die man in NEXTSTEP maximal einstellen kann, werden so tatsächlich problemlos **115 200 Bit/s** bzw. **230 400 Bit/s** – das reicht für ISDN auf alle Fälle aus. Man sollte sich aber vergewissern, daß der gewählte **Provider** auch einen PPP-Zugang über ISDN anbietet; handelt es sich dabei um ein sogenanntes *synchrones* PPP (siehe unten), so ist der Zugang mit NEXTSTEP zur Zeit **nur** unter Verwendung eines **ZyXEL Elite 2864I**-ISDN-Adapters möglich, der eine entsprechende Konvertierung anbietet!
Eine **Ethernet-Karte** ist *eigentlich* nicht erforderlich, da das Ethernet ja gerade durch PPP ersetzt wird. Die hier vorgeschlagene *vollautomatische* Netzwerkkonfiguration des Rechners über den **Simple Network Starter** setzt allerdings eine Ethernet-Karte voraus, da sonst eine Netzwerkkonfiguration aus Sicht des **Simple Network Starters**, der PPP nicht kennt, überhaupt keinen Sinn macht. Da bei einer späteren Aufrüstung mit einer Ethernet-Karte, und sei es nur für einen schnellen Druckeranschluß, aber nicht nur die gesamte Netzwerkkonfiguration hinfällig wäre, sondern darüber hinaus u.U. auch etliche Programmlizensierungen nicht mehr stimmen würden (weil die sich nach der Ethernet-Adresse richten – s.o.), sollte der Internet-Anschluß zum Anlaß genommen werden, so noch nicht geschehen dem Rechner eine Netzwerk-Identität zu verschaffen, wie sie auf allen anderen NEXTSTEP-Plattformen jenseits von Intel-PCs ohnehin Standard ist. Gängig für NEXTSTEP sind die Karten **Intel EtherExpress** (ISA-Bus) und **Cogent EM 960** (PCI-Bus; unbedingt aufpassen, daß man eine Ausführung kauft, an die BNC-Kabel (»Thin Ethernet«) angeschlossen werden können, die für kleine Netzwerke das einzig Sinnvolle darstellen). Für beide sind Treiber in NEXTSTEP enthalten, aber **ACHTUNG!:** der Treiber für die **Cogent EM 960** ist **fehlerhaft** und **muß** durch den neuen Treiber von NeXT ersetzt werden, den man in den **NeXTanswers** findet.

Software

Die erforderlichen Programme liegen, wie bei Internet-Programmen üblich, allesamt auf verschiedenen FTP-Servern, so z.B. dem *Peanuts* FTP-Server der Universität München, der eines der größten Archive mit NEXTSTEP-Software überhaupt darstellt. Für jemanden, der sich den Internet-Anschluß aber gerade erst verschaffen will, bringt dies natürlich die klassische Henne-Ei-Problematik mit sich, wenn sich auch im Freundeskreis noch kein Zugang findet.

Glücklicherweise gibt es die Programme des *Peanuts*-Servers aber auch auf CD-ROM. Zeitgleich mit dieser Ausgabe von **NEXTTOYOU** erscheint ein Update der CD-ROM (siehe Seite 12), das alle benötigten Programme enthält (bezüglich **Yftp** und **OmniWeb** s.u.). **Die Versionen auf der Doppel-CD-ROM vom Frühjahr 1995 sind veraltet und für die hier vorgeschlagene Installation nicht zu verwenden!** Die Update-CD-ROM ist mit Sicherheit der unkomplizierteste Weg, an die benötigten Programme zu gelangen.

Möglicherweise ist auch der zukünftige Provider bereit, zum Start anhand der folgenden Angaben die Programme zu besorgen (oder, falls seit Drucklegung erschienen, entsprechende höhere Programmversionen, die sich allerdings bezüglich der notwendigen Konfiguration ändern könnten).

Peanuts:
CD-ROM oder
`ftp.informatik.uni-muenchen.de/pub/comp/platforms/next`
serieller Treiber:
`/NeXTanswers/ByNumber/1942.compressed` *und*
`/NeXTanswers/ByNumber/1946.compressed`
falls auch die Maus über eine serielle Schnittstelle (und nicht den PS/2-Anschluß) betrieben werden soll:
`/NeXTanswers/ByNumber/1944.compressed`

falls Cogent EM 960 eingebaut:
Treiber für Cogent EM 960:
`/NeXTanswers/ByNumber/1881.compressed`
GateKeeper:
`/Communication/apps/GateKeeper.1.0.Beta2.NI.b.tar.gz`
PopOver:
`/Mail/apps/PopOver.1.4.NIHS.bd.tar.gz`
PPP:
`/Unix/communication/ppp_2_2.0.4.6.NIHS.b.tar.gz`

Entscheidet man sich beim **WWW-Browser** für **Netsurfer** (siehe Besprechung in **NEXTTOYOU**/WiNGGazette 3/95), der einen **FTP-Client** integriert hat, so kann man sich Netsurfer gleich mitholen und somit auf eine getrennte FTP-Software verzichten; allerdings ist das Programm erst nach der Lizensierung (auch per Fax oder Telefon umgehend möglich, wenn man eine Kreditkarte hat) einsatzbereit Ansonsten braucht man **Yftp**. Sobald ein FTP-Client installiert ist, kann man sich weitere Software dann selbst besorgen. **Netsurfer** ist sowohl als **WWW-** wie als **FTP-Programm** äußerst einfach zu bedienen und soll in Zukunft auch noch einen **Newsreader** integriert haben, so daß er sich als Komplettlösung fürs Internet anbietet. Allerdings hat sein integrierter FTP-Client zur Zeit den Nachteil, daß er durch technische Probleme unterbrochene Datei-Übertragungen nicht wieder aufnehmen kann, sondern von vorne beginnt. Wen dieser Nachteil stört, der ist mit der (zudem kostenlosen) Kombination aus **Yftp** und **OmniWeb** möglicherweise besser bedient. **ACHTUNG!** Diese beiden Programme befinden sich *nicht* auf der *Peanuts*-Update-CD-ROM, sondern nur auf der Doppel-CD-ROM vom Frühjahr dieses Jahres!

Netsurfer:
`/Commercial/network/Netsurfer.1.1.NIHS.b.tar.gz`
oder:
Yftp:
`/Network/filetransfer/Yftp.0.447.NIH.bs.tar.gz`

und
OmniWeb:
`/Network/www/OmniWeb.1.0.NIHS.b.tar.gz`
(nicht unbedingt nötig,da man sich mit ***Yftp*** **OmniWeb** *auch schon selbst holen kann)*

Falls man noch keinen Entpacker für die komprimierten Dateien aus dem Internet hat, benötigt man außerdem

Opener:
`/Tools/archiver/Opener.3.3.NIHS.b.tar.gz`

ACHTUNG! PPP, **GateKeeper**, **Netsurfer** und der **serielle Treiber** befinden sich in einer stetigen Entwicklung und haben im Moment allesamt noch mit einigen Unsauberkeiten zu kämpfen. Es ist daher sinnvoll, unmittelbar nach der Erstinstallation der Programme mit Hilfe des jetzt vorhandenen eigenen Zugangs ins Internet nach eventuellen neuen Versionen der Programme auf *Peanuts* oder einem anderen FTP-Server Ausschau zu halten! Bezüglich der seriellen Treiber oder anderen NEXTSTEP-Updates empfiehlt es sich, direkt auf den **NeXTanswers-WWW-Seiten** von Next (`http://www.next.com/NeXTanswers/`) nachzusehen, da dort auch genau erläutert ist, *wo* sich *was* geändert hat.

Nur auf dem *Peanuts*-Server, aber nicht auf der CD findet sich schließlich eine Datei, die **alle Konfigurationsdateien dieses Artikels in der hier vorliegenden Reihenfolge** enthält. Das hat den großen Vorteil, daß man nicht (auf fehlerträchtige Art und Weise) alle Konfigurationen abtippen muß, sondern sie mit *Kopieren* und *Einfügen* in die jeweiligen Dateien übernehmen kann und nur die jeweils nutzerspezifischen Daten selbst eingeben muß:
Internett-Dateien:
`/Unix/communication/Internett.KonfigDateien.tar.gz`

Provider

Internet-Provider findet man noch immer nicht unbedingt im Telefonbuch. Unverzichtbar ist, daß der Provider einen **PPP**-Zugang zur Verfügung stellt, gegebenenfalls über ISDN; ausserdem muß er selbst natürlich über eine möglichst leistungsfähige Anbindung ans Internet verfügen. Eine einzelne 64 kBit/s-ISDN-Leitung ist indiskutabel, wenn man seinerseits ISDN benutzen möchte, denn dann halbiert schon ein einziger weiterer Kunde, der online ist, die Übertragungsgeschwindigkeit. Der ideale Provider bietet darüber hinaus eine feste IP-Adresse, ist hilfsbereit bei Sonderwünschen und vor allem preiswert. Auch sollte er unter einer Domain nur eine überschaubare Anzahl Kunden beherbergen, da es sonst mehr als wahrscheinlich wird, daß der eigene Name als Nutzername schon vergeben ist und man bei irgendwelchen Verstümmelungen Zuflucht suchen muß.

Auch wenn die Preise währenddessen kräftig unter Druck geraten, kann Internet noch immer leicht unbezahlbar werden. Indiskutabel ist insbesondere ein Zeittakt. Wer in Ruhe eine Zeitung im WWW lesen will, kann nicht zwischen zwei Seiten jedesmal die Verbindung unterbrechen; dann aber zahlt er für vergleichsweise wenig übertragene Daten schnell horrende Summen, zu denen sich die harmlos scheinenden Pfennige pro Minute allzu rasch addieren, ganz abgesehen von den Zeittaktgebühren, die die Telekom für ihr leitungsvermitteltes Netz ohnehin haben will. Wie gesagt: der Gedanke, nach Zeiteinheiten zu quantifizieren, spottet der Grundidee eines paketvermittelten UNIX-Netzes. Datenvolumen-orientierte Preise von etlichen Mark pro Megabyte sind freilich genauso inakzeptabel; noch herrscht auf diesem Markt wegen seiner (sich freilich rasch verlierenden) Intransparenz vielfach eine echte Abzocker-Mentalität.

Aus den genannten Gründen scheiden die neu auf der Bildfläche aufgetauchten Kommunikationsriesen, Telekom, CompuServe und America Online, in jedweder Beziehung aus; für NEXTSTEP gibt es ohnehin keine Software, die eine Adaption auf ihre speziellen Zugangswege (nämlich über die jeweiligen eigenen Onlinedienste) vornimmt. Einzig die bundesweite Erreichbarkeit zum Ortstarif ist ein Vorteil von Telekom Online, doch auch hier zieht zumindest Protel schon nach.

Die anderen Provider sind in der Regel bundesweit organisiert; im folgenden abgedruckt ist eine Liste der jeweiligen zentralen Adressen, die dann Auskunft über Zugangsmöglichkeiten in der einzelnen Region geben können.

Wer Mitglied einer Hochschule ist, fragt am besten zunächst beim dortigen Rechenzentrum, ob ein vollwertiger PPP-Zugang für Hochschulangehörige besteht; wenn ja, ist er meist so gut wie umsonst und über das DFN (*D*eutsches *F*orschungs-*N*etz) auch sehr gut mit dem Rest der Welt verbunden. Ansonsten sind Privatleute im Augenblick am besten bei Contrib.Net, MAZ, Protel, TOPNET oder dem als Verein organisierten Individual Network aufgehoben, die alle zwischen 30 und 55 DM pro Monat ohne jede weitere Kosten verlangen (teils allerdings mit zeitlichen oder technischen Einschränkungen).

Contrib.Net, Knaackstraße 96, 10435 Berlin
Telefon (030) 25 30 12 00 FAX (030) 251 56 03

Eunet Deutschland GmbH,
Emil-Figge-Straße 80, 44227 Dortmund
Telefon (0231) 972 FAX (0231) 972 11 11

Individual Network e.V.
Scheideweg 65, 26121 Oldenburg
Telefon (0441) 980 85 56 FAX (0441) 980 85 57

MAZ Internet Service GmbH
Harburger Schloßstraße 6-12, 21079 Hamburg
Telefon (040) 766 29 16 23 FAX (040) 76 62 95 07

Nacamar, Kirchweg 22, 63033 Dreieich
Telefon (06103) 969-0 FAX (06103) 96 91 27

NTG/XLink GmbH
Vincenz-Priesnitz-Straße 3, 76131 Karlsruhe
Telefon (0721) 965 20 FAX (0721) 965 22 10

Protel Telekommunikations GmbH
Borsigstraße 7a, 65205 Wiesbaden
Telefon (06122) 998-0 FAX (06122) 99 81 90

TOPNET, Elbestraße 25, 47800 Krefeld
Telefon (02151) 49 75 10 FAX (02151) 49 75 32

Die folgenden **Kenndaten** muß der Provider zur Verfügung stellen:

- **Telefonnummer** des **PPP**-Zugangs
- **Nutzernamen** (auf Antrag möglichst identisch mit dem auf dem eigenen Rechner)
- **Paßwort**
- **Befehl** zum **Starten** des **PPP-Protokolls** (in der Regel `ppp`; entfällt möglicherweise, wenn der Provider gar nichts anderes anbietet)
- **E-Mail-Adresse** – also die Adresse, unter der man in Zukunft eigene Mail empfangen kann (im Falle einer festen IP-Adresse ist das automatisch ***MeinNutzername@MeinHostname.ProviderDomain.de***; im Falle einer dynamischen IP-Adresse sollte der ***ProviderNutzername*** möglichst gleich ***MeinNutzername*** (auf dem eigenen Rechner) sein; siehe hierzu die letzten Abschnitte von **Netstep** in der ersten Folge des Artikels)
- (möglichst mehrere) ***numerische*** **IP-Adressen** des bzw. der **Nameserver**
- die **logische IP-Adresse** des **Mailservers** (das ist der Rechner, an den `sendmail` die eigene Mail schicken soll)
- die **logische IP-Adresse** des **POPmail-** oder **IMAP-Servers** (das ist der Rechner, von dem **PopOver.app** eingetroffene Mail abholt) und die Angabe, welche Protokolle (POP bzw. IMAP) verfügbar sind
- im Falle einer *festen IP-Adresse* eben diese **IP-Adresse** für den eigenen Rechner und die **IP-Adresse** des **Einwahlrechners** beim Provider; im Gegenzug muß man dem Provider den vorgesehenen **Hostnamen** seines eigenen Rechners mitteilen, damit der ihn in den **DNS** eintragen kann
- die **logische IP-Adresse** des **NNTP-Servers** (das ist der Server, auf dem die Newsgruppen des Internet liegen)
- so vorhanden, die **logischen IP-Adressen** von **HTTP-Proxy** und **FTP-Proxy** (das sind Rechner, die für **WWW** bzw. **FTP** die am häufigsten abgerufenen Seiten bzw. Dateien zwischenspeichern und so zu einer Entlastung des Internets führen, da diese Daten bei erneutem Abruf nur noch von dem lokalen Rechner und nicht aus dem Netz geholt werden müssen – *Proxy* heißt *Stellvertreter; HTTP* (*Hypertext Transfer Protocol*) ist die Protokollsprache des WWW, das die hypertextartigen, also mit der Maus anwählbare Querverweise enthaltenden WWW-Dokumente überträgt.)

Vorbereitung

Die folgenden Schritte sind jeweils durchzuführen, falls der eigene Rechner nicht bereits mit **Modem / ISDN-Adapter**, **neuem seriellen Treiber** und **Opener.app** ausgerüstet ist; auf alle Fälle muß `tip` korrekt konfiguriert werden!

Modem / ISDN-Adapter bei ausgeschaltetem Rechner an eine serielle Schnittstelle anschließen und zusammen mit dem Rechner einschalten. Als **root** einloggen. Falls es noch keinen **root**-Nutzer auf dem Rechner gibt, können die folgenden Schritte auch nachgeschoben werden, wenn **root** im Zuge der Netzwerkkonfiguration erstellt ist.

Um Übertragungskapazität zu sparen, sind Dateien im Internet grundsätzlich komprimiert verpackt; sie müssen daher vor ihrer Verwendung entpackt werden. Hierzu dient **Opener.app**, das, einmal installiert, dafür sorgt, daß sich gepackte Dateien auf Doppelklick selbsttätig entpacken. Sinnreicherweise ist **Opener.app** selbst aber auch nur gepackt aus dem Internet zu erhalten (die *Peanuts-CD* enthält allerdings eine einsatzbereite Version). **Opener.app** selbst muß daher auf mühsame Weise mit UNIX-Befehlen über **Terminal.app** entpackt werden; ab dann aber geht das Entpacken von alleine.

Eine Shell in **Terminal.app** öffnen und

`gunzip` [Leertaste]

eingeben. Aus dem **WorkspaceManager** das Icon von **Opener.3.3.NIHS.b.tar.gz** auf die Shell ziehen und **Return** drücken. Nach einem kurzen Augenblick ändert sich das Icon im **WorkspaceManager** in **Opener.3.3.NIHS.b.tar**. In die Shell

`tar -xf` [Leertaste]

eingeben, **Opener.3.3.NIHS.b.tar** auf die Shell ziehen und **Return** drücken. Nach kurzer Zeit findet sich im Rootverzeichnis (**/**) die Applikation **Opener.app**. Diese nach **/LocalApps** verschieben.

ACHTUNG! In diesem Frühjahr war eine Version von **Opener.app** im Umlauf, die auf **Intel**-Rechnern nur teilweise funktionierte. Wenn **Opener.app** immer wieder Fehlermeldungen ausspuckt, es durch eine neue Version ersetzen.

Auf **1942.compressed** (den seriellen Treiber) im **WorkspaceManager** doppelklicken. Das entpackte **ISASerialPort.pkg** durch Doppelklick mittels dem dann automatisch startenden **Installer.app** installieren. Ebenso mit **1946.compressed** (dem **PortServer.pkg**) verfahren.

Dieser sogenannte *TTY Port Server* ist ein unbedingt notwendiges Zusatzmodul des modular aufgeteilten neuen seriellen Treibers von NeXT, ohne dessen Hilfe kein Terminalprogramm auf die serielle Schnittstelle, die im UNIX-Slang auch *TTY* genannt wird, zugreifen könnte.

/NextAdmin/Configure.app starten, *Andere* (das Icon mit dem Fragezeichen) wählen, auf *Hinzufügen...* klicken und den *Serial Port (3.33)* hinzufügen; diese Prozedur so oft wiederholen, bis so viele Treiber wie serielle Schnittstellen im Computer installiert sind (in der Regel also 2). Sodann noch den *TTY Port Server (v3.33)* hinzufügen (dies ist in jedem Falle nur *einmal* notwendig). Nunmehr noch in **Configure.app** die Interrupts und Anschlußadressen für die seriellen Schnittstellen überprüfen; wurde die Hardware gemäß dem Handbuch *Installation und Konfiguration von NEXTSTEP Version 3.3* konfiguriert, so stimmen die Werte automatisch.

Falls die Maus ebenfalls über eine serielle Schnittstelle (und nicht über einen PS/2-Anschluß) betrieben wird, muß nun

noch entsprechend **1944.compressed** (das **SerialPointingDevice.pkg**), falls die Ethernet-Karte **Cogent EM 960** eingebaut ist, **1881.compressed** (das **DECchip 21040NetworkDriver.pkg**) installiert werden.

`tip` (der neckische Name ist mit *Terminal Interface Program* zu erklären) ist ein UNIX-Programm, mit dessen Hilfe von **Terminal.app** aus ein Modem gesteuert werden kann; es wird durch die Angaben der Datei **/etc/remote** (für Verbindungen in die *Entfernung* zuständig) konfiguriert. Idiotischerweise sind aber alle von NeXT in dieser Datei getätigten Voreinstellungen für übliche Modems unbrauchbar. Daher muß diese Datei zunächst ergänzt werden.

Auf die Uhr im Dock doppelklicken und in dem erscheinenden **Präferenzen**-Fenster unter *UNIX* die Option *UNIX-Experte* einschalten. **Edit.app** starten und die Datei **/etc/remote** öffnen. Nach dem ersten größeren Block von Definitionen um die folgenden ergänzen:

```
.
.
.
ttya9600|Hardwire on ttya at 9600 baud:\
        :dv=/dev/ttya:br#9600:tc=BASIC:
ttyb9600|Hardwire on ttyb at 9600 baud:\
        :dv=/dev/ttyb:br#9600:tc=BASIC:

# zusätzliche Einträge für Betrieb von üblichen Modems
##########################################

a28|cudfa28800|Dial-out on cudfa at 28800 baud:\
        :dv=/dev/cudfa:pa=none:
        br#38400 :tc=BASIC:

b28|cudfb28800|Dial-out on cudfb at 28800 baud:\
        :dv=/dev/cudfb:pa=none:
        br#38400:tc=BASIC:
```

/etc/remote ist eine typische Konfigurationsdatei, in der Kommentarzeilen durch # gekennzeichnet werden. Der vorgeschlagene Kommentar muß natürlich nicht übernommen werden, aber es ist hilfreich, eigene Veränderungen schon rein optisch deutlich erkennbar zu machen, um im Laufe der Zeit nicht die Übersicht zu verlieren. Ebenso hilfreich ist es hierzu, einen Ordner mit einem Namen wie **Spezialdateien** oder ähnliches anzulegen und alle Systemdateien, die man verändert, dorthin zu linken (mit der Maus in den Ordner schieben und dabei die **Control**-Taste gedrückt halten). Auf diese Art und Weise verliert man nie den Überblick über die am System vorgenommenen Veränderungen. Dieser Vorschlag wird hier in Zukunft nicht jedesmal eigens erwähnt.

Die jeweils erste Zeile der beiden Einträge gibt verschiedene Namen für den Aufruf dieser Einträge an, die durch **|** getrennt sind (der dritte »Namen« ist de facto eher eine Erläuterung). **cudfa** bzw. **cudfb** sind die Kürzel für die beiden seriellen Schnittstellen, versehen mit der nominellen Übertragungsgeschwindigkeit heutiger Highspeed-Modems, 28,8 kbaud (langsamere Modems funktionieren bei dem üblichen Einsatz eines sogenannten *Handshake*-Protokolls, das dafür sorgt, daß von Modem und Rechner keine Seite die andere überholt, aber natürlich auch). **a28** bzw. **b28** sind einfach nochmals Abkürzungen dafür, die das Aufrufen erleichtern sollen. Die eigentliche Konfiguration findet nach den Doppelpunkten statt: `dv` gibt das *Device (Gerät)* an, mit dem `tip` kommunizieren soll, ein Modem an Schnittstelle `cudfa` bzw. `cudfb` eben. Dabei ist es sehr wichtig, nicht das `d` in der Bezeichnung auszulassen, also wirklich `cudfa` und nicht wie oft üblich nur `cufa` anzugeben, da nur so ein Abbruch der Verbindung an die Programme weitergemeldet wird. Der Schnittstellenparameter `pa`, *Parity*, wird wie heute allgemein üblich auf *keine* gesetzt, die *Baudrate* `br` zwischen Modem und Rechner auf **38400** (mögliche Werte: **4800**, **9600**, **19200** und **38400**), höher als den nominellen Wert, um die Steigerung der Übertragungsrate durch Online-Kompression zu berücksichtigen (zwischen Modem und Rechner sind die Daten ja vom Modem schon wieder dekomprimiert worden und also mehr), und ansonsten entspricht die Voreinstellung einer an anderer Stelle von **/etc/remote** festgelegten Definition `BASIC`. Über die genaue Definition der Syntax dieser Datei kann man sich wie üblich in **Terminal.app** durch die Eingabe des Kommandos `man remote` informieren.

Existieren mehr als zwei serielle Schnittstellen, weil z.B. eine serielle Hochgeschwindigkeitskarte für ISDN in den Rechner eingebaut wurde, so müssen noch entsprechende weitere Einträge `c` und `d` (`cudfc` und `cudfd`) vorgenommen werden. **ACHTUNG!** Wird eine Schnittstellenkarte mit hardwaremäßiger Vervierfachung der Übertragungsgeschwindigkeit eingesetzt, so darf die Baudrate nur auf auf 19200 beziffert werden, da `tip` mehr als 19200 x 4 = 76800 Baud nicht verarbeiten kann. Ein entsprechender Eintrag für einen ISDN-Adapter würde dann also lauten:

```
c64|cudfc64000|Dial-out on cudfc at 64000 baud:\
        :dv=/dev/cudfc:pa=none:br#19200
        :tc=BASIC:
```

In die Shell von **Terminal.app** `tip a28` (und **Return**) eingeben und auf die Meldung connected warten. Sodann probieren, ob das Modem / der ISDN-Adapter auf die Eingabe von `atz` (und **Return**) reagiert – nur in diesem Falle ist `atz` auch im Terminal-Fenster lesbar. `tip` mit **~.** wieder verlassen und, falls nicht erfolgreich, mit `tip b28` (und gegebenenfalls c,d...)ebenso verfahren. In einem der Fälle wird, wenn alles in Ordnung ist, `atz` und in Folge `OK` zu lesen sein. An dieser Schnittstelle (`a` bzw. `cudfa`, `b` bzw. `cudfb`,...) hängt das Modem / der ISDN-Adapter. **Im folgenden wird davon ausgegangen, daß das Modem / der ISDN-Adapter an `cudfb` angeschlossen ist. Sollte dies nicht der Fall sein, sind die jeweiligen Einträge entsprechend zu modifizieren.**

Netzwerkkonfiguration

Wer sich ganz sicher ist, daß sein Rechner bereits korrekt für Netzwerke als **Mailhost (!)** mit dem **Simple Network Starter** konfiguriert wurde, kann diesen Abschnitt überspringen. Die korrekte Netzwerkkonfiguration ist allerdings von **ausschlaggebender Bedeutung** für die Funktion von **PPP**; insofern ist jetzt *die* Gelegenheit, seinem Computer eine vernünftige Netzwerkkonfiguration zu verpassen. **Eine nicht korrekte Konfiguration von *Netzwerk* und sendmail kann die Funktion von PPP stark beeinträchtigen bis verhindern.**

Zum Verständnis der Netzwerkadministration von NEXTSTEP sollte das *NeXTSTEP Network and System Administration Manual* bereit stehen, und zwar am besten in gedruckter Form (es ist auch in der Online-Dokumentation enthalten). So ist es nicht nur besser lesbar; vor allem braucht man seinen Rat möglicherweise gerade dann, wenn der Rechner *nicht* benutzbar ist... Allerdings wird dieses Buch mit dem Erscheinen von NEXTSTEP 4.0 in nicht allzu ferner Zukunft veraltet sein.

Eine **Neukonfiguration** eines Netzwerkes, und sei es auch nur die **Änderung der IP-Adresse** des Rechners, auf ein schon **bestehendes Netzwerk** ist ***nicht*** möglich; es muß zuvor stets ein neutraler Ausgangszustand hergestellt werden. Dabei gehen auch die einzelnen Nutzereinträge verloren; die Nutzer-Heimverzeichnisse bleiben auf Wunsch aber unangetastet. Wer also schon ein nicht allzu komplexes Netzwerk konfiguriert und Nutzer angelegt hatte, kann dies insofern getrost rückgängig machen, wer gerade erst sein NEXTSTEP-System installiert hat, spart sich die Schritte des Rückgängigmachens und beginnt tatsächlich von vorne.

Da der **Simple Network Starter** vollautomatisch abläuft, ist nicht auf den ersten Blick zu sehen, was er denn genau verändert (man kann das nach erfolgter Konfiguration in der Datei **/usr/adm/SNS.log** allerdings nachlesen – *adm* steht für *Administration*). Es ist dies exakt zweierlei: die Datei **/etc/hostconfig**, die einige wenige zentrale Angaben, so den Namen und die IP-Adresse des Rechners, im Klartext enthält, und der Ordner **/etc/netinfo**, der die sogenannte **NetInfo**-Datenbank beinhaltet, die nur über die Administrationsprogramme von NEXTSTEP (die Programme in **/NextAdmin**) und einige **Terminal.app**-Befehle zugänglich ist und die gesamte restliche Netzwerkkonfiguration, so z.B. die gesamten Nutzereinträge, enthält. Alle einschlägigen Terminal-Befehle bekommt man übrigens aufgelistet durch die Eingabe von `man -k netinfo`: `-k` ist eine Option, die `man` dazu bewegt, *alle* Einträge des Inhaltsverzeichnisses der Manual-Seiten anzuzeigen, in denen das nachfolgende Stichwort auftaucht. So kann man Befehle, deren Namen man nicht kennt, nach ihrer Kurzbeschreibung aufsuchen. Manuelle Änderungen in **/etc/hostconfig** führen mit großer Sicherheit zur Funktionsunfähigkeit des Netzwerks, ebenso unbedachte Änderungen von **NetInfo** durch die Terminal-Befehle oder **NetInfoManager.app**, das kaum Einschränkungen in der Manipulation auferlegt. Die anderen, für je spezifische Aufgaben gedachten Verwaltungsprogramme im Ordner **/NextAdmin** sind sicherer; sie lassen unerlaubte Änderungen erst gar nicht zu.

Eine Funktionsunfähigkeit des Netzwerkes ist deswegen von großem Übel, weil der Rechner dann überhaupt nicht mehr startet, man folglich auch nichts mehr reparieren kann. Daher ist bei Änderungen am Netzwerk mit großer Vorsicht vorzugehen. Passiert das Unheil trotzdem, so hilft die folgende Methode: in den 10 Sekunden, *bevor* NEXTSTEP laut Ankündigung gestartet wird, drückt man die Tasten **-s**, genauer gesagt, diese Zeichen; da zu diesem Zeitpunkt aber noch kein deutscher Tastaturtreiber geladen ist, der Computer mithin von der amerikanischen Tastenbelegung ausgeht, muß man de facto **ßs** drücken, damit die gewünschten Zeichen erscheinen. Mit **-s** wird der Computer im sogenannten *Single-User-Modus* gestartet; die Bootmeldungen laufen im Klartext ab und nach einiger Zeit bekommt man einen blinkenden und auf Eingabe wartenden Cursor zu sehen. Man ist dann in einem Terminalfenster, das UNIX-Befehle entgegennimmt, ganz so, als hätte man **Terminal.app** gestartet. Nur – welche Befehle können jetzt helfen?

Glücklicherweise befinden sich in **/usr/template/client/etc** exakte Kopien all der Dateien im Ordner **/etc** in ihrem Urzustand (daher das *template*, d.h. *Schablone*, in diesem Pfad). Wenn wir also die fehlerhaften Netzwerkdateien durch die ursprünglichen ersetzen, startet der Computer wieder in jungfräulichem Zustand. Dazu gibt es UNIX-Befehle wie `cp` (*copy*), die aber in der konkreten Situation doch recht kompliziert werden können, zumal man in solchen Momenten ohnehin nicht die Ruhe selbst ist und die Dokumentation im Computer gerade nicht zugänglich. Wir schreiben uns daher ein *Shellskript*, das alle nötigen Aktionen im Falle eines Falles mit einem einzigen Befehl ausführt.

In **Edit.app** eine neue ASCII-Datei öffnen und den folgenden Text eingeben (*Achtung, genau abtippen, insbesondere auf Leerzeichen achten – ein falsches Zeichen, und das Skript funktioniert nicht mehr!*):

```
#!/bin/sh
#
#Skript zum Ersetzen fehlerhafter Netinfo-
Konfigurationen durch
#Default-Dateien in /usr/template/net im
Single-User-Modus
#
echo ""
echo "Mache Backup von jetzigem Netinfo
und jetziger"
echo "hostconfig-Datei in /etc/netre-
set.backup"
if [ ! -d /etc/netreset.backup ]
then
          mkdir /etc/netreset.backup
fi
cp -pr /etc/netinfo /etc/netreset.backup
cp -p /etc/hostconfig /etc/netreset.backup
echo ""
echo "Ersetze lokale Netinfo-Datenbank"
cp -r /usr/template/net/netinfo/local.nidb
/etc/netinfo
echo ""
echo "Ersetze hostconfig-Datei"
cp /usr/template/net/hostconfig /etc
echo ""
echo "Loesche netzweite Netinfo-Datenbank"
rm -r /etc/netinfo/network.nidb
echo ""
echo "OK"
echo ""
```

Die Datei als **/netreset** sichern; dann ihr Texticon im **WorkspaceManager** einmal anklicken und im Inspektor, Popup-Einstellung *Zugriffsrechte*, für die Datei durch Anklikken der entsprechenden Felder für alle drei Spalten die Zeile *Ausführen* einschalten. Das Icon der Datei wandelt sich darauf in das UNIX-Programmicon. Falls noch nicht vorhanden, im Ordner **/usr/local** einen Ordner **/bin** durch Drücken von **Command n** neu erzeugen und **netreset** nach **/usr/local/bin** verschieben (dorthin kommen die lokalen, also nicht in NEXTSTEP generell, sondern nur auf diesem speziellen Rechner vorhandenen Binärdateien).

Es fällt auf, daß das Skript nicht auf **/usr/template/client/etc** zugreift, sondern auf einen etwas anderen Ordner, den wir erst noch anlegen müssen. Auf diese Weise können wir geringfügig modifizierte Template-Dateien erstellen.

In dem Ordner **/usr/template** durch Drücken von **Command n** einen neuen Ordner erzeugen, **net** benennen und auf die Ablage ziehen; sodann aus dem Ordner **/usr/template/client/etc** die Datei **hostconfig** und den Ordner **netinfo** auf **net** ziehen; die **Alternate**-Taste drücken, so daß die Dateien in den neuen Ordner *kopiert* werden (zwei grüne Quadrate als Cursor). In eine Shell von **Terminal.app** `netreset` eingeben und **Return** drücken. Wenn `netreset` korrekt erstellt wurde, wird die alte Netzwerkkonfiguration nun ersetzt und findet sich, falls noch benötigt, im Ordner **/etc/netreset.backup** wieder; nach dem nächsten Booten ist NEXTSTEP, was das Netzwerk und die eingetragenen Benutzer betrifft, wieder in seinem Auslieferungszustand.

NEXTSTEP beenden und den Rechner ausschalten. Gegebenenfalls die Ethernet-Karte einbauen; bei einer PCI-Karte dem entsprechenden PCI-Slot im BIOS des Computers eine IRQ zuweisen, bevorzugt 10. **NEXTSTEP** erneut starten.

ACHTUNG! Wenn bereits eine **Ethernet-Karte inklusive Treiber** installiert gewesen ist, gibt es beim Booten jetzt Schwierigkeiten, weil NEXTSTEP wegen der Ethernet-Karte ein Netzwerk vermutet, aber keines mehr konfiguriert ist. Startet man NEXTSTEP so, daß es Bootmeldungen ausgibt (also durch Drücken von **-v** bzw. auf einer deutschen Tastatur **ß**v in den ersten 10 Sekunden vor dem Starten von NEXTSTEP), so erscheinen folgende Meldungen:

```
localhost mach: en0: Ethernet address
01:23:45:67:89:ab
localhost mach:
localhost mach: No response from network
configuration server.
localhost mach: Type Control-C to start up
computer without a network
localhost mach: connection.
```

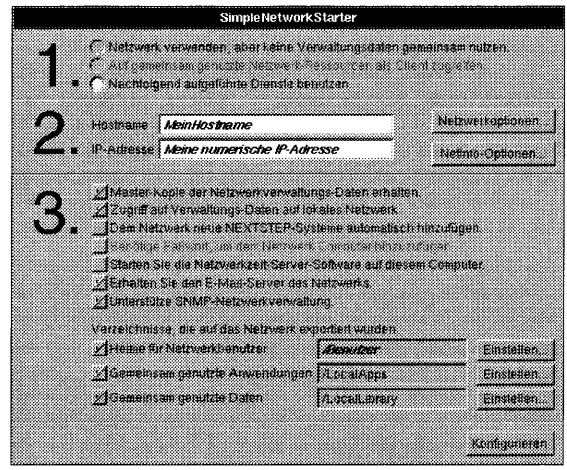
Eine entsprechende kurze Meldung erscheint auch beim Booten mit grafischer Oberfläche; man muß dann in jedem Fall **Control c** drücken. Vermeiden läßt sich dies, wenn man vorübergehend in **Configure.app** den Treiber löscht, was sich aber nur lohnt, falls man aus irgendeinem Grund mehrmals ohne Ethernet-Karte bootet.

NEXTSTEP erscheint jetzt so wie unmittelbar nach der Erstinstallation (auch wenn natürlich seitdem angelegte Dateien und Programme noch alle vorhanden sind). Man muß daher auch denselben **Initialisierungsprozeß** durchlaufen: Sprache und Tastatur auswählen, in dem durch Doppelklick auf die Uhr zugänglichen Präferenzen-Fenster ein Paßwort für **me** festlegen (das funktioniert übrigens auch problemlos, wenn das *Heimverzeichnis* von **me** *gelöscht* wurde oder wird, weil es stört), ausloggen, als **root** einloggen, nunmehr für **root** ein Paßwort festlegen. Damit diese lästige Prozedur nach einem weiteren **netreset** nicht jedesmal erforderlich ist, *kopieren* wir nun die so minimal auf unser System eingestellte Netzwerkdatei **/etc/hostconfig** und den Ordner **/etc/netinfo** nach **/etc/template/net** und *ersetzen* dadurch die NEXTSTEP-Template-Dateien durch unsere eigenen. Von nun an ist ein Nutzer **root** mit Paßwort auch nach einem `netreset` schon vorhanden.

Falls die Ethernet-Karte neu eingebaut wurde, die Anwendung **Configure.app** aus dem Ordner **/NextAdmin** aufrufen und den passenden Treiber für die Ethernet-Karte installieren; bei einer ISA-Karte als IRQ bevorzugt 10 zuweisen.

Der Rechner ist jetzt bereit zur Netzwerkkonfiguration. Für jede zukünftig erforderliche Netzwerkkonfiguration kann durch einfache Eingabe des Befehls `netreset` in **Terminal.app** und anschließendes Neustarten des Rechners der erforderliche Ausgangszustand hergestellt werden.

Im Ordner **/NextAdmin SimpleNetworkStarter.app** starten und wie folgt einstellen:



MeinHostname ist dabei der für den Rechner selbstgewählte und im Falle einer festen IP-Adresse dem Provider mitgeteilte Hostname; er darf maximal **14 Buchstaben** lang sein. ***Meine numerische IP-Adresse*** ist im Falle einer festen IP-Adresse eben diese Adresse, wie sie vom Provider mitgeteilt wird; im Falle einer dynamischen IP-Adresse kann man einfach die von NEXTSTEP vorgeschlagene Adresse stehen lassen (oder sich auf Wunsch eine eigene Phantasieadresse geben, bei der man allerdings die Bildungsregeln beachten muß, wie sie im *Network and System Administration Manual* auf den Seiten 447 und 448, *Appendix C / Internet Addresses*, beschrieben sind). ***/Benutzer*** steht für den Namen des Pfades, in dem die einzelnen Nutzerverzeichnisse angelegt werden sollen und muß bereits existieren (gegebenenfalls also vorher den entsprechenden Ordner mit **Command n** anlegen). Den **Simple Network Starter** nun durch Anklicken von *Konfigurieren* starten. Das Netzwerk wird jetzt automatisch konfiguriert; der Erfolg (hoffentlich) am Schluß bestätigt. **SimpleNetworkStarter.app** daraufhin beenden.

Aufgrund eines Fehlers in NEXTSTEP wird die Einstellung *Router: keiner*, die in den *Netzwerkoptionen...* möglich wäre, nicht übernommen und muß manuell nachgetragen werden. Die Einstellung ist wichtig, da wir ja eben nicht über einen Router, sondern über PPP und Modem / ISDN-Adapter ans Internet gehen.

Im Ordner **/NextAdmin HostManager.app** starten, das Menü *Lokal...* wählen und in dem erscheinenden Fenster rechts unten *Router:Keiner* wählen; sodann auf *Einstellen* klicken, in der erscheinenden Dialogbox aber *Nicht neu starten* wählen, da noch die Nutzer eingetragen werden müssen. **HostManager.app** beenden.

Die Einstellung *Erhalten Sie den E-Mail-Server des Netzwerks* in **SimpleNetworkStarter.app** ist eine der viel zu vielen grausamen Übersetzungsfehler, die in der deutschen Lokalisierung von NEXTSTEP vorkommen, und meint, daß auf dem Rechner der E-Mail-Server des Netzwerks *betrieben* werden soll (da hat jemand *unterhalten* mit *erhalten* verwechselt...). Das ist somit die für uns entscheidende Einstellung, denn unser Rechner muß in unserem »Netzwerk«, das im Zweifel nur aus ihm selbst besteht, autonom, also als Server, mit Mail umgehen können, da er mit dem Mail-Server des Providers ja nicht ständig verbunden ist. In dieser Einstellung verändert übrigens der **Simple Network Starter** außer den beiden erwähnten Dateien (bzw. Ordnern) noch eine dritte: in **/etc/crontab**, einer Datei, in der zeitlich regelmäßig vom Rechner zu wiederholende Vorgänge aufgelistet werden, fügt er die Zeile `3 0 * * * root /usr/bin/mailDBupdate` hinzu, die ein Update der Mail-Datenbank (*`mailDB`*; de facto sind das die Dateien **/LocalLibrary/Images/People/aliases** und **/LocalLibrary/Images/People**

/passwd) mit eventuellen neuen Nutzern des lokalen Subnetzes veranlaßt, die dann im Adreßbrowser von **Mail.app** als Mailadressen zur Verfügung stehen. Leider ist der **Simple Network Starter** so dumm, dies jedesmal zu tun, gleichgültig, ob die Zeile schon existiert hat. Startet man den **Simple Network Starter** also mehrfach, so wird diese Zeile auch mehrfach in **/etc/crontab** erscheinen; sie kann dann bis auf eine ohne weiteres gelöscht werden. Umgekehrt berücksichtigt `mailDBupdate` aber nur Nutzer (und gegebenenfalls ihre Aliases, die man über den **NetInfoManager** anlegen kann – siehe *Network and System Administration Manual* Seite 175 ff. (*Creating Mail Aliases*)) der *root Netinfo Domain* (**/**), also im Falle einer Netzwerk-Konfiguration netzweite Nutzer. Hat man auch *lokale* Nutzer angelegt (die im **NetInfoManager** nicht unter **/**, sondern unter ***/MeinHostname*** zu finden sind), so muß man, um die Mail-Datenbank aktuell zu halten, zusätzlich `3 0 * * * root /usr/bin/mailDBupdate -A -a /MeinHostname -u /MeinHostname` in **/etc/crontab** eintragen (`-a` und `-u` weisen `mailDBupdate` an, *Aliases* und Nutzer (*User*) des jeweils angegebenen Pfades ebenfalls zu berücksichtigen; `-A` sorgt dafür, daß diese Daten nur an die Mail-Datenbank angehängt werden und sie nicht etwa damit überschrieben wird). Zur Bedeutung der übrigen Einstellungen des **Simple Network Starters** siehe das *Network and System Administration Manual*.

Sofern noch nicht geschehen, in **/usr/template/languages** durch Drücken von **Command n** einen Ordner **German.lproj** erstellen (der fehlerhafterweise nicht vorhanden ist), so daß *Deutsch* als Nutzersprache gewählt werden kann. Im Ordner **/NextAdmin UserManager.app** starten, *Benutzer* ê *Voreinstellungen...* wählen und als *Voreingestelltes Heimverzeichnis* den Pfad ***/Benutzer*** angeben, falls dies nicht schon der Fall ist ist. Ebenso können hier die Sprache und das zu verwendende *Template* (siehe nächster Absatz) voreingestellt werden. *Benutzer* ê *Neu...* wählen und die Nutzer eintragen; für sich selbst sollte man als Gruppe dabei unbedingt *wheel* vorsehen; das erlaubt mit einem am Ende des Artikels beschriebenen Trick, auch unter seinem Nutzernamen administrative Aktionen am Rechner auszuführen, die **root** vorbehalten sind. Gab es die Nutzerverzeichnisse schon vor der Konfiguration, so wird nachgefragt, ob die vorhandenen Verzeichnisse verwendet werden sollen; beantwortet man diese Frage mit *Ja*, so hat man seine alten Verzeichnisse wieder. Bei einer Neuinstallation von NEXTSTEP hingegen läßt man sich die Verzeichnisse anlegen. Sodann den Rechner neu starten, damit alle Einstellungen wirksam werden.

In diesem Fall verwendet **UserManager.app** für die zu erstellenden Nutzerverzeichnisse als Vorbild das in dem entsprechenden Popup angegebene *Template* (=*Schablone*). Da NEXTSTEP nur ein Standard-Template mitbringt, hat man bei dem Popup zunächst keine Wahl. Nur in den seltensten Fällen wird man mit der Vorgabe aber zufrieden sein. Will man mehr als einen Nutzer einrichten, lohnt es sich daher, zunächst selbst ein *Template* anzulegen, das all jene Voreinstellungen enthält, die sämtlichen neuen Nutzern als Ausgangsbasis dienen sollen. Dazu wählt man in **UserManager.app** *Benutzer* ê *Neues Template...* an und verfährt danach wie bei einem normalen Nutzer, dem man den Namen *benutzer* gibt. Nachdem dieser Nutzer gesichert ist, verläßt man **UserManager.app** und loggt sich sodann aus **root** aus und in das neu erstellte **benutzer** unter dem gerade vergebenen Paßwort ein. Nun kann man sich die Oberfläche dieses Pseudo-Nutzers nach eigenem Geschmack in jeder Beziehung einrichten (Applikationen im Dock, Präferenzen in den Applikationen, Größe des Workspaces, Icons auf seiner Ablage usw.). Danach Ausloggen, wieder unter **root** einloggen und **UserManager.app** erneut starten. Nun läßt sich *benutzer* als *Template* wählen, und alle so neu angelegten Nutzer finden dessen Einstellungen beim Einloggen als Ausgangsbasis vor.

PPP-Installation

Von jetzt an wird vorausgesetzt, daß der Rechner korrekt **als Mailhost konfiguriert** ist. Nach wie vor müssen wir als **root** eingeloggt sein. Wir installieren nun PPP und erstellen die Konfigurationsdateien. Es ist hilfreich, sich hierfür nochmals das Schaubild in **Wie funktioniert's?** zu vergegenwärtigen. Wenn ab jetzt bis zum Schluß des Artikels nicht ausdrücklich anders angegeben, werden Änderungen in den Konfigurationsdateien wirksam, sobald eine neue PPP-Verbindung aufgebaut wird; es ist in diesen Fällen also nicht nötig, sich dafür auszuloggen oder gar den Rechner neu zu starten.

PPP-Verbindungen können auf zwei Weisen aufgebaut werden, die mit zwei im Detail unterschiedlichen Übertragungsprotokollen zusammenhängen. Beim ***asynchronen* PPP**, dem weiter verbreiteten Fall, stellt *zunächst* das Modem / der ISDN-Adapter eine *Terminal-Verbindung* vergleichbar einer Mailbox zur Gegenseite her; *anschließend* wird *über diese schon hergestellte Verbindung* die Identifizierung des Anrufers durchgeführt und schließlich PPP gestartet. Beim ***synchronen* PPP** hingegen wird zeitgleich mit dem Aufbau der Verbindung die Identifizierung vorgenommen und PPP *direkt* als Protokoll gestartet. Diese Art des Zugangs ist also sehr viel schneller (wenige Sekunden), aber nur unter ISDN üblich, und erfordert einen speziellen Übertragungsmodus von PPP, den die NEXTSTEP-PPP-Portierung nicht hat. Abhilfe schafft hier der **ZyXEL** ISDN-Adapter **Elite 2864I**, der eine automatische Konvertierung von asynchronem in synchrones PPP vornimmt; ein synchroner PPP-Zugang mit NEXTSTEP über die serielle Schnittstelle ist zur Zeit also **nur** mit ihm möglich. Ansonsten müssen lediglich die Dateien `pppup` und `.pppрc` leicht anders konfiguriert werden.

Basisinstallation

ppp_2_2.0.4.6.NIHS.b.tar.gz durch Doppelklick entpacken; **PPP-2.2.pkg** durch Doppelklick mittels **Installer.app** installieren. Dadurch wird **PPP** in den Ordner **/usr/local/ppp/** kopiert; zugleich wird der Ordner **/etc/ppp/** angelegt, in dem `pppd` alle systemweit gültigen PPP-Konfigurationsdateien erwartet. Dieser Ordner ist de facto ein *Verweis* auf den Ordner **/usr/local/ppp/etc/**.

Edit.app starten, **/etc/rc.local** öffnen und am Ende der Datei durch folgende Zeilen ergänzen:

```
############### PPP ##################
# ergänzt am XX.XX.1995 um XX:XX
#
# Load the Berkley Packet Filter LKS
# This must be done before PPP.  Comment this out
# if you disabled BPF. If you use BPF, you _must_
# load it before the PPP LKS.
#

if [ -f /usr/local/bpf/reloc/bpf_reloc ]; then
    /usr/etc/kl_util -a /usr/local/bpf/reloc/bpf_reloc
fi

#
# Load the selected version of the PPP-2.2 loadable
# kernel server (LKS).
#

if [ -f /usr/local/ppp/reloc/ppp_reloc ]; then
```

```
    /usr/etc/kl_util -a /usr/local/ppp/reloc/ppp_reloc >
/dev/console 2>&1
   (echo -n ' ppp')    > /dev/console
fi
#######################################
```

Diese Zeilen finden sich bis auf die ersten beiden und die letzte auch in der PPP-Anleitung in der Datei **/usr/local/ppp/README.NeXT.MAB.Installation** (die wie erwähnt ein Verweis auf eine Datei in dem **/examples**-Ordner ist, welche vor dessen Löschen herauskopiert werden muß). Es ist sehr viel leichter und vermeidet vor allem Schreibfehler, den Text dort als Block zu markieren und über *Kopieren* und *Einfügen* in **/etc/rc.local** zu übertragen. **/etc/rc.local** als **ASCII**-Datei sichern und schließen; wirksam wird sie erst nach einem *Neustart* des Rechners.

Dieser Eintrag sorgt dafür, daß die *Kernel Server* von PPP beim Start von NEXTSTEP automatisch mitgeladen werden. Die Anfangs- und Schlußzeilen sind sinnvoll, um selbst vorgenommene Veränderungen deutlich zu machen; eine Datierung ermöglicht gegebenenfalls in Kombination mit den Log-Dateien in **/usr/adm** festzustellen, ob ein Phänomen genau seit dieser Veränderung auftrat.

Den Rechner neu starten, um die Änderung wirksam werden zu lassen und zu überprüfen, ob der Rechner ordnungsgemäß hochfährt. Dazu beim Neustart die Bootmeldungen durch `-v` bzw. ß`v` einschalten. Es muß dann gegen Ende der Meldungen folgendes zu lesen sein:

```
Datum Zeit MeinHostname mach: BPF version 1.1 NS 3.2 and
3.3
Datum Zeit MeinHostname mach: LKS: $Revision: 4.2 $
($Date: 1995/07/03 18:26:45 $)
Datum Zeit MeinHostname mach: Using Major device 32
Datum Zeit MeinHostname mach: by Stephen Perkins <per-
kins@cps.msu.edu>
Datum Zeit MeinHostname mach:
Datum Zeit MeinHostname mach: PPP version 2.2 released
for NS 3.2 and 3.3
Datum Zeit MeinHostname mach: LKS: $Revision: 4.14 $
($Date: 1995/11/30 01:45:58 $)
Datum Zeit MeinHostname mach: by  Stephen Perkins, Phi-
lip Prindeville, and Pete French
Datum Zeit MeinHostname mach: Installing PPP on Line Di-
scipline 5
Datum Zeit MeinHostname mach: Installing interfaces:
Datum Zeit MeinHostname mach:       Initializing ppp0
Datum Zeit MeinHostname mach:       ppp0 successfully at-
tached.
Datum Zeit MeinHostname mach: bpf: ppp0 attached.
Datum Zeit MeinHostname mach:       Initializing ppp1
Datum Zeit MeinHostname mach:       ppp1 successfully at-
tached.
Datum Zeit MeinHostname mach: bpf: ppp1 attached.
Datum Zeit MeinHostname mach: PPP-2.2 Successfully In-
stalled.
```

Sollte etwas schiefgegangen sein und der Rechner nicht booten, durch `-s` bzw. ß`s` im Single-User-Modus starten, `netreset` eingeben, danach durch Eingabe von `reboot` erneut starten und von vorne beginnen.

/.cshrc öffnen und um die folgenden Zeilen ergänzen:

```
.
.
# set up the path
set path=(/etc /usr/etc /usr/ucb /bin /usr/bin /usr/lo-
cal/bin /usr/sybase/bin /LocalApps /NextApps /NextAdmin
/NextDeveloper/Demos /usr/local/ppp/bin)

setenv MANPATH "/usr/man:/usr/local/man:/usr/lo-
cal/ppp/man:/usr/local/bpf/man"

# umask will disable write privileges for group and
other users
umask 022
.
.
```

/.cshrc als **ASCII**-Datei sichern und schließen. Entweder mit den **~/.cshrc** in allen *Heimordnen* von Nutzern (**~/** ist die unter UNIX übliche Abkürzung des Pfades des jeweiligen Heimordners) – einschließlich der Nutzer-*Templates*! – ebenso verfahren oder (falls keine individuellen weiteren Änderungen gewünscht werden) diese Dateien überall löschen und jeweils stattdessen **/.cshrc** dorthin *linken*, indem man die Datei unter Drücken der **Control**-Taste mit der Maus dorthin schiebt (es erscheint der grüne Doppelpfeil). Im letzteren Fall werden alle Änderungen an **/.cshrc** vorgenommen und wirken global. Die Änderung ist wirksam für *jede Shell* in **Terminal.app**, die *danach geöffnet* wird.

/usr/local/ppp/bin sorgt dafür, daß die UNIX-Programme in diesem Pfad in Zukunft auch von **Terminal.app** aus ausführbar sind. `setenv MANPATH` gibt an, in welchen Pfaden nach UNIX-Manual-Dateien gesucht werden soll. Somit werden auch die PPP-Manual-Dateien gefunden; bei dieser Gelegenheit kann ebenfalls der NEXTSTEP-Fehler korrigiert werden, daß **/usr/local/man** kein Standardsuchpfad ist. Sobald man diese Angaben rückgängig macht, ist PPP wieder vom System isoliert.

/etc/syslog.conf öffnen und die folgenden beiden Zeilen am Dateibeginn ergänzen:

```
local2.debug      /usr/adm/GateKeeper.fifo
local2.debug      /usr/adm/ppp.log
```

ACHTUNG! Bei dem Zwischenraum darf es sich ***nur*** um **Tabulatoren** handeln, nicht um Leerzeichen! Die Zeile mit den Angabe für **GateKeeper.app** muß an erster Stelle stehen, da **GateKeeper.app** zur Anlage des FIFO die Einträge automatisch auswertet und dafür stets den ersten `local2.debug`-Eintrag heranzieht. **/etc/syslog.conf** als **ASCII**-Datei sichern und schließen.

Dieser Eintrag bewirkt, daß Meldungen über den Ablauf der PPP-Verbindung für **GateKeeper.app** in den FIFO und in die Datei **/usr/adm/ppp.log** geschrieben werden. Die Log-Datei dient dazu, Informationen über die Verbindung zur Fehlerauswertung auch nach der Beendigung von **GateKeeper.app** (insbesondere auch nach dessen Absturz) zur Verfügung zu haben.

Eine neue ASCII-Datei öffnen, nichts eintragen und die Datei als **/usr/adm/ppp.log** sichern. In diese Datei protokolliert PPP den Verlauf der PPP-Verbindung.

Der syslog-Dämon schreibt nur in bereits existente Dateien; daher muß die Log-Datei für PPP zunächst angelegt werden.

PPP-Konfigurationsdateien

Eine neue ASCII-Datei öffnen und folgendes eintragen:

```
# Diese ppp options-Datei dient als Basis-
# Konfiguration aller ppp-Verbindungen
# und wird ggf. durch .ppprc, die GateKee-
# per Link options-Datei bzw. die
# Options-Datei in einem .Gate-Dokument
# überschrieben.
# Die Angabe der Baudrate in Zeile 2 MUSS
# mit der in /etc/remote
# verwendeten übereinstimmen, damit tip
# bei der manuellen Anwahl
# von GateKeeper verwendet werden kann.

/dev/cudfb
38400
crtscts
lock

defaultroute
#meine numerische IP-Adresse:IP-Adresse
#des Einwahlrechners beim Provider

lcp-echo-interval 15
lcp-echo-failure 3

#-vj

#debug
```

Gemäß dem vorangestellten Kommentar muß für einen einwandfreien Ablauf der Funktion *manuelle Anwahl* in **GateKeeper.app** die in der zweiten Zeile angegebenen Übertragungsrate mit der in **/etc/remote** unter `b28` eingetragenen übereinstimmen (und also auch auf **19200** geändert werden, wenn wegen ISDN c64 und eine Schnittstellenkarte mit hardwaremäßiger Vervierfachung der Übertragungsgeschwindigkeit verwendet wird und `tip` entsprechend konfiguriert wurde (s.o.)).

Hat man einen Zugang mit dynamischer IP-Adresse, so wird die **Zeile mit den Angaben der IP-Adressen weggelassen**. Ansonsten müssen das # entfernt und für die beiden ***IP-Adressen*** die vom Provider erhaltenen Adressen – numerische oder auch logische, wenn insbesondere der eigene Rechnername bereits im Nameserver des Providers eingetragen ist – eingesetzt werden.

`#debug` ist zwar durch das Kommentarzeichen unwirksam, sollte aber auf jeden Fall mit eingegeben werden, um im Störungsfall durch einfaches Entfernen des # genauere Informationen zu erhalten. Ebenso läßt sich `-vj` aktivieren, das die sogenannte *Van Jacobsen* Kompression ausschaltet. Diese Kompression, die bei kleinen Datenpaketen (etwa in Dialogen) an und für sich sehr vorteilhaft ist, macht beim augenblicklichen Stand der PPP-Entwicklung teilweise erhebliche Schwierigkeiten. Kommt es wiederholt, insbesondere beim FTP-Betrieb, zu einem plötzlichen Abbruch der Verbindung durch das Modem / den ISDN-Adapter oder sinkt die Datenübertragungsrate (ablesbar im FTP-Betrieb) erheblich, so muß `-vj` durch Entfernen des # aktiviert und die Kompression somit ausgeschaltet werden.

Die Datei als **/etc/ppp/options** also in dem bei der Installation des **PPP-2.2.pkg** angelegten Ordner für systemweite PPP-Konfigurationsdateien, sichern (eine eventuell von der Installation her schon vorhandene Datei dieses Namens dabei überschreiben) und schließen.

Mit dieser Datei werden die systemweiten Basiseinstellungen von PPP konfiguriert. `/dev/cudfb` gibt die **serielle Schnittstelle** an, an der das Modem / der ISDN-Adapter angeschlossen ist (hier: b). **38400** ist die **Übertragungsrate** in Bit/s zwischen Modem / ISDN-Adapter und Computer, die mit dem entsprechenden Wert von b28 in **/etc/remote** übereinstimmen muß. `crtscts` schaltet den **Hardware-Handshake** ein und ist obligatorisch. `lock` sorgt dafür, daß die Schnittstelle während der Verbindung **gesperrt** ist und nicht noch von einem anderen Programm (z.B. einem Btx-Decoder oder Terminalprogramm) angewählt werden kann, was einiges Chaos ergeben würde. `defaultroute` sorgt dafür, daß PPP die **Default-Route** für alle Datenpakete wird, deren Zieladresse nicht auf dem eigenen Rechner liegt – PPP ersetzt hier also den **Router**. Die Angabe von `defaultroute` ist entscheidend; ohne sie funktioniert *nichts*: kein WWW, keine Mail, kein FTP, keine News – denn die Datenpakete all dieser Programme werden dann erst gar nicht zum Rechner des Providers geschickt. `lcp-echo-interval 15` und `lcp-echo-failure 3` kontrollieren den Zustand der PPP-Verbindung durch alle 15 s ausgesandte Testsignale; werden diese dreimal hintereinander nicht beantwortet, so wird die physikalische Verbindung zum Provider unterbrochen und eine entsprechende Fehlermeldung ausgegeben. Das ist sinnvoll, weil es passsieren kann, daß die PPP-Verbindung zusammenbricht und »nichts mehr geht«, die physikalische Verbindung aber unnützer- und auch irritierenderweise aufrecht erhalten bleibt. `debug` ist mit einem # versehen und daher normalerweise nicht wirksam. Sollte es Probleme geben, so kann man das # entfernen; dann werden ausführliche Informationen über den Verbindungsaufbau in die Datei **/usr/adm/ppp.log** und ins Monitor-Fenster von **GateKeeper.app** ausgegeben, die normalerweise aber nur stören. `-vj` schaltet, wie oben erwähnt, die *Van Jacobsen*-Kompression aus, die die Adreßdaten der einzelnen TCP-Datenpakete komprimiert und daher – wenn sie funktioniert – insbesondere bei kleinen Datenpaketen sinnvoll ist, wo die Adreßdaten einen prozentual hohen Anteil am gesamten Paket haben. Schließlich wird bei einer IP-Verbindung mit fester IP-Adresse die Identität des eigenen Rechners (und die des Provider-Einwahlrechners) hier festgelegt. Die Reihenfolge der einzelnen Angaben ist übrigens technisch vollkommen egal und hier nur nach Übersichtlichkeit gewählt.

Im Folgenden werden die nutzerspezifischen Optionsdateien und die Konfigurationsdateien für die **automatische Einwahl** und **Identifikation des Nutzers** erstellt. Der Inhalt dieser Dateien ist naturgemäß abhängig davon, ob es sich um *synchrones* oder *asynchrones* **PPP** handelt. **Der jeweils andere Abschnitt kann übersprungen werden**; weiter geht's dann mit ***Test für asynchrone und synchrone PPP-Verbindung***

nur für asynchrone PPP-Verbindung:

Bei der *asynchronen* PPP-Verbindung wird der gesamte Einwahl- und Identifikationsprozeß mit Hilfe der Angaben aus `pppup` von `chat` gesteuert, das von `pppd` aufgrund eines entsprechenden Eintrags in `.ppprc` gestartet wird und die Kontrolle erst zurückgibt, wenn alles für die PPP-Verbindung bereit ist. Während der *synchrone* Zugang genormt ist, hängt der *asynchrone* davon ab, welche Ausgaben genau

der Provider während der zunächst aufgebauten Terminalverbindung macht und welche Eingaben er daraufhin erwartet, damit es zu der PPP-Verbindung kommt. Man kann – im Gegensatz zum *synchronen* Verbindungsaufbau – diese Angaben auch manuell machen; um sie in einem Skript zu automatisieren, müssen wir daher zunächst feststellen, wie genau der Dialog während der Terminalverbindung abläuft.

Terminal.app öffnen. `tip b28` eingeben und **Return** drücken. Nach der connected-Meldung das Modem / den ISDN-Adapter mit `atz` in den Ausgangszustand versetzen (es wird hier davon ausgegangen, daß das Modem / der ISDN-Adapter so konfiguriert ist, daß es aus diesem Zustand heraus wählen und eine Verbindung aufbauen kann) und sodann mit `atd` *123456* die vom Provider erhaltene Telefonnummer anwählen. Kommt keine Verbindung zustande, muß das Modem / der ISDN-Adapter gemäß seiner Anleitung entsprechend umkonfiguriert werden. Ansonsten sollte eine Einwahlseite erscheinen, die etwa nach folgendem Muster aufgebaut ist:

```
MeinHostname:1# tip b28
connected
atz
OK
atd123456
CONNECT Baudrate und Protokoll

Provider Einwahlseite mit netten Begrü-
ßungsfloskeln

Login: ProviderNutzername
Password: Paßwort
ProviderPrompt> ppp-Startbefehl
    Entering PPP Mode.
    IP address is 123.456.789.012
    MTU is 1524.
+++
OK
atz
OK
~
[EOT]
MeinHostname:2#
```

Die selbst zu tätigenden Eingaben sind hier **fett** gedruckt, die Reaktionen von Seiten des Modems / des ISDN-Adapters normal, von den Gegebenheiten abhängige Werte *kursiv*. Entscheidend ist, wie die unterstrichenen Worte bei dem gewählten Provider **genau** lauten, denn sie sind die **Schlüsselworte**, auf deren Auftauchen unser automatisches Einwahlskript reagieren soll. Im Folgenden werden die hier wiedergegebenen Worte verwendet; stehen an ihrer Stelle andere, so sind sie sinngemäß zu ersetzen.

Nach der Anwahl erfolgt die `CONNECT`-Meldung und darauf die Aufforderung zur Eingabe des vom Provider erhaltenen ***ProviderNutzername*** sowie des dazugehörigen ***Paßwort***. Auf das dann folgende Erscheinen des *ProviderPrompts* ist ein bestimmter **`ppp-Startbefehl`** einzugeben; dieser Schritt *entfällt* möglicherweise bei Providern, die ohnehin keinerlei anderen Zugang anbieten. Als Ergebnis dieser Aktionen sollte schließlich eine Meldung über den Aufbau der PPP-Verbindung zu lesen sein. Allerdings können wir damit im Moment noch nichts anfangen, so daß wir durch Eingabe von `+++` das Modem / den ISDN-Adapter wieder in den Kommandozustand versetzen und nach der Bestätigung durch `OK` mit `atz` die Verbindung beenden und `tip` mit `~.` verlassen.

Eine neue ASCII-Datei öffnen und folgendes eintragen:

```
ABORT BUSY
ABORT ERROR
ABORT "NO CARRIER"
ABORT "NO ANSWER"
ABORT "NO DIAL TONE"
"" ATZ
OK ATD123456
Login--Login ProviderNutzername
Password--Password Paßwort
ProviderPrompt--ProviderPrompt ppp-Start-
befehl
```

Die für die *kursiven* Begriffe einzugebenden Werte entsprechen den obigen der mit **Terminal.app** manuell durchgeführten Einwahl, die unterstrichenen den bei dieser Einwahl festgestellten Schlüsselworten, auf die die automatische Einwahl reagieren soll. Ist ein Befehl zum Start von PPP nicht erforderlich, kann die letzte Zeile entfallen. Gibt das Modem / der ISDN-Adapter andere als die in den ersten Zeilen genannten Fehlermeldungen aus, sind diese entsprechend zu modifizieren. In diese Datei dürfen **keine Kommentarzeilen** (mit **#**) eingegeben werden! Die Datei als **/pppup** sichern und schließen.

Diese Datei enthält die Angaben, die `chat` zu einer Automatisierung des Einwahlprozesses benötigt. Zunächst werden in den mit `ABORT` eingeleiteten Zeilen Fehlermeldungen des Modems / ISDN-Adapters angegeben, auf die mit einem Abbruch reagiert werden soll; besteht die Meldung aus mehreren Worten, muß sie, ansonsten kann sie dabei in Anführungszeichen gesetzt werden. Gibt das Modem / der ISDN-Adapter andere Meldungen, sind diese Zeilen entsprechend zu modifizieren. Die folgenden Zeilen bestehen grundsätzlich aus **zwei** Zeichenfolgen: sobald die *erste* Zeichenfolge eintrifft, veranlaßt `pppup`, daß die *zweite* als Reaktion darauf ausgegeben wird. Die allererste Zeichenfolge `""` ist leer; es wird also in jedem Fall mit einem `ATZ` begonnen. Wird dieses mit `OK` bestätigt, wird die Nummer des Providers gewählt; der fragt dann Namen und Paßwort ab und wartet schließlich auf den PPP-Befehl, worauf jeweils mit der entsprechenden Eingabe reagiert wird. Daß die Schlüsselwörter doppelt auftauchen und durch `--` miteinander verbunden sind, bewirkt bei einer fehlerhaften Erkennung des Schlüsselwortes zur Sicherheit einen zweiten Versuch.

Wir müssen nun noch dafür sorgen, daß die Angaben von `pppup` automatisch verwendet werden.

Eine neue ASCII-Datei öffnen und folgendes eintragen:

```
# nutzerspezifische options-Datei; ergänzt
# bzw. überschreibt
# /etc/ppp/options
#
# Verbindung mittels asynchronem PPP gemäß
# /pppup

57600
connect "/usr/local/ppp/bin/chat -vf /pppup"
```

Die Datei als **/.ppprc** sichern und schließen.

Diese *nutzerspezifische* Optionsdatei überschreibt die Standard `options` Datei mit der höchsten unter NEXTSTEP verfügbaren Übertragungsgeschwindigkeit 57600, da bei der automatischen Einwahl auf `tip` keine Rücksicht genommen werden muß und die Übertragungsgeschwindigkeit (wegen der Online-Kompression des Modems / ISDN-Adapters) möglichst hoch sein sollte. Sodann bringt sie `pppd` mit dem Befehl `connect` dazu, das Programm **/usr/local/ppp/bin/chat** zu starten, das den eigentlichen Einwahlprozeß ausführt. Die Option `v` (verbaler Modus) weist `chat` dabei an; Meldungen des Modems / ISDN-Adapters und eigene Fehlermeldungen an den Syslog-Dämon weiterzuleiten; `f /pppup` bestimmt, daß die Parameter aus der Datei **/pppup** zum Verbindungsaufbau herangezogen werden sollen.

nur für synchrone PPP-Verbindung:

Beim *synchronen* Zugang führt `pppd` die Nutzer-Identifikation selbst durch; das dazu notwendige Paßwort entnimmt es **/etc/ppp/pap-secrets**, einer systemweiten Datei, in der die Paßwörter für alle Nutzer des Systems gespeichert werden. `chat` führt mit Hilfe der Angaben aus `pppup` lediglich die Anwahl des Providers durch. `.ppprc` muß daher `pppd` sowohl zum Start von `chat` veranlassen als auch die notwendigen Angaben enthalten, damit `pppd` in **/etc/ppp/pap-secrets** das richtige Paßwort ermitteln kann.

Eine neue ASCII-Datei öffnen und folgendes eintragen:

```
ABORT BUSY
ABORT ERROR
ABORT "NO CARRIER"
ABORT "NO ANSWER"
ABORT "NO DIAL TONE"
"" ATZ
OK ATB4
OK ATDI123456
CONNECT ""
```

Hinter `ATDI` die Telefonnummer des Providers eintragen. In diese Datei dürfen **keine Kommentarzeilen** (mit #) eingegeben werden! Die Datei als **/pppup** sichern und schließen.

Diese Datei enthält die Angaben, die `chat` zu einer Automatisierung des Einwahlprozesses benötigt. Zunächst werden in den mit ABORT eingeleiteten Zeilen Fehlermeldungen des ZyXEL ISDN-Adapters angegeben, auf die mit einem Abbruch reagiert werden soll; besteht die Meldung aus mehreren Worten, muß sie, ansonsten kann sie dabei in Anführungszeichen gesetzt werden. Die folgenden Zeilen bestehen grundsätzlich aus **zwei** Zeichenfolgen: sobald die *erste* Zeichenfolge eintrifft, veranlaßt `pppup`, daß die *zweite* als Reaktion darauf ausgegeben wird. Die allererste Zeichenfolge `""` ist leer; es wird also in jedem Fall mit einem `ATZ` begonnen. Wird dieses mit OK bestätigt, wird beim ZyXEL ISDN-Adapter mit `ATB4` die Konvertierung zu synchronem PPP eingeschaltet und sodann mit `ATDI` der Provider im ISDN-Modus angewählt. Auf die Bestätigung der Verbindung durch `CONNECT` folgt keine weitere Aktion mehr (`""`); diese Zeile bewirkt lediglich, daß `chat` den physikalischen Verbindungsaufbau abwartet, bis es die Kontrolle an `pppd` zurückgeben läßt, das die Identifizierung durchführt. Diese Identifizierung, die bei asynchronem PPP ja in `pppup` durchgeführt wird, erfolgt bei synchronem PPP standardisiert über **PAP** (*P*aßwort-*A*uthentifizierungs-*P*rotokoll; als Alternative gibt es noch CHAP, das *C*ryptographic *H*andshake *A*uthentication *P*rotocol).

Eine neue ASCII-Datei öffnen und folgendes eintragen:

```
# Diese Datei stellt für verschiedene An-
# wahlziele von verschiedenen Nutzern
# die jeweiligen Einwahl-Paßwörter für
# synchrone ISDN-Verbindungen bereit.

ProviderNutzername ProviderKürzel Paßwort
```

ProviderNutzername ist wiederum der Nutzername, den man beim Provider hat, ***Paßwort*** das vom Provider erhaltene Paßwort. ***ProviderKürzel*** ist eine Bezeichnung für diesen Provider, die man sich willkürlich ausdenken kann, da sie nur lokal der Identifizierung des jeweiligen Providers dient und somit ermöglicht, daß `pppd` auch für *einen* Nutzer gezielt *verschiedene* Provider mit den zugehörigen Paßwörtern auswählen kann. Die Datei als **/etc/ppp/pap-secrets** (für CHAP: **chap-secrets**) sichern und schließen.

Es gibt systemweit nur eine Datei `pap-secrets`, die sich in dem Pfad **/etc/ppp/** befinden muß. Es lassen sich in ihr aber beliebig viele Kombinationen aus Nutzer, Provider und zugehörigem Paßwort in jeweils einer Zeile abspeichern.

Eine neue ASCII-Datei öffnen und folgendes eintragen:

```
# nutzerspezifische options-Datei; ergänzt
# bzw. überschreibt
# /etc/ppp/options
#
# Verbindung mittels ISDN / synchronem PPP
# für Ziel ProviderKürzel

57600

name ProviderNutzername
remotename ProviderKürzel

connect "/usr/local/ppp/bin/chat -vf /pppup"
```

ProviderNutzername und ***ProviderKürzel*** sind die in `pap-secrets` verwendeten Bezeichnungen, mit deren Hilfe das Paßwort eindeutig zugeordnet werden kann. Die Datei als **/.ppprc** sichern und schließen.

Diese *nutzerspezifische* Optionsdatei überschreibt die Standard `options` Datei mit der höchsten unter NEXTSTEP verfügbaren Übertragungsgeschwindigkeit 57600, da bei der automatischen Einwahl auf `tip` keine Rücksicht genommen werden muß und die Übertragungsgeschwindigkeit (wegen der Online-Kompression des ISDN-Adapters) möglichst hoch sein sollte; für ISDN ist wie erwähnt eine Hardware-Schnittstellenkarte zu wählen, die diese Angabe verdoppelt bzw. vervierfacht – so kommt man bis auf Werte von 4 x 57600 = 230 400 Bit/s, was allemal ausreicht. Mit Hilfe von `name` und `remotename` legt `.ppprc` eindeutig fest, welcher Provider von welchem Nutzer angerufen werden soll, so daß `pppd` über `pap-secrets` damit auch das Paßwort ermitteln kann. Sodann bringt die Datei `pppd` mit dem Befehl `connect` dazu, das Programm **/usr/local/ppp/bin/chat** zu starten, das den eigentlichen Einwahlprozeß ausführt. Die Option `v` (verbaler Modus) weist `chat` dabei an, Meldungen des ISDN-Adapters und eigene Fehlermeldungen an den Syslog-Dämon weiterzuleiten; `f /pppup` bestimmt, daß die Parameter aus der Datei **/pppup** zum Verbindungsaufbau herangezogen werden sollen.

Test für asynchrone und synchrone PPP-Verbindung

Wir haben `.ppprc` und `pppup` jetzt im **root**-Verzeichnis gespeichert, da wir ja auch als **root** eingeloggt sind. Grundsätzlich muß sich jedenfalls `.ppprc` aber auch sonst immer im Heimverzeichnis (**~/**) desjenigen Nutzers befinden, der die PPP-Verbindung mit seiner Hilfe aufbauen will (und `pppup` dann dort, wo es

.pprc angibt). Auf diese Weise ist sichergestellt, daß jeder Nutzer seine individuellen Konfigurationsdaten verwenden kann – allerdings auch nur genau *einen* Satz solcher Daten. **GateKeeper.app** wird darüber noch hinausgehen und es jedem Nutzer gestatten, aus beliebig vielen Paaren von .pprc- und pppup-Dateien auszuwählen, indem es sie in spezielle Anwahl-Dokumente verpackt, statt sie einfach offen ins Heimverzeichnis zu legen. Wir hätten pppup übrigens auch wie .pprc mit einem Punkt zu Beginn des Namens versehen und somit unsichtbar machen können. Da die Datei später aber ohnehin durch **GateKeeper.app** an andere Stelle gepackt wird, belassen wir es bei der einfacheren Form.

ACHTUNG! Nach der Anlage von **.pprc** kann aus dem entsprechenden Nutzerverzeichnis **pppd** nicht mehr manuell mit **tip** wie oben in dem Abschnitt über *asynchrones* PPP beschrieben gestartet werden, da nun stets **.pprc** von **pppd** eingelesen und umgesetzt wird. Soll dies dennoch geschehen – etwa weil die automatische Einwahl noch nicht funktioniert –, so muß **.pprc** vorher gelöscht, umbenannt oder verschoben werden!

Wir testen jetzt das Zustandekommen der PPP-Verbindung. Falls nicht schon geschehen, **Terminal.app** starten und in ein neu geöffnetes Fenster

```
tail -f [Leertaste]
```

eingeben. Danach **/usr/adm/ppp.log** mit der Maus ins Terminalfenster ziehen und **Return** drücken. Die in **ppp.log** geschriebenen Meldungen werden so zur besseren Kontrolle »live« in einem Fenster ausgegeben und stetig aktualisiert (was später **GateKeeper.app** übernehmen wird).

Falls nicht schon offen, ein weiteres Fenster in **Terminal.app** öffnen,

```
pppd
```

eingeben und **Return** drücken. Wurden alle Dateien richtig konfiguriert, wird **pppd** jetzt **chat** aufrufen, das die von **pppup** erhaltenen Anwahlbefehle ausführt und in **ppp.log** protokolliert, und schließlich eine PPP-Verbindung herstellen. Der erfolgreiche Aufbau dieser Verbindung wird mit den Zeilen

```
Datum Zeit MeinHostname pppd[PID]: Serial
connection established.
Datum Zeit MeinHostname pppd[PID]: Using
interface ppp0
Datum Zeit MeinHostname pppd[PID]:Connect:
ppp0 <--> /dev/cudfb
Datum Zeit MeinHostname pppd[PID]: Remote
message:
Datum Zeit MeinHostname pppd[PID]: local
IP address 123.456.789.012
Datum Zeit MeinHostname pppd[PID]: remote
IP address 123.456.789.012
```

in **ppp.log** (und also dem Fenster von **Terminal.app**) bestätigt. ***PID*** ist eine drei- oder vierstellige Nummer, die sogenannte *Prozeß-ID*, die einen bestimmten Prozeß unter UNIX (hier eben pppd) genau identifiziert. Hinter **local IP address** ist im Falle einer festen IP-Adresse die Adresse des eigenen Rechners angegeben, im Falle einer dynamischen IP-Adresse die dynamisch vom Provider übermittelte, für diese konkrete Verbindung gültige Adresse. **remote IP address** gibt die Adresse des Einwahlrechners beim Provider an.

Wir können nun die Qualität der Verbindung überprüfen. Dazu geben wir in das Fenster von **Terminal.app**, in dem wir schon pppd gestartet haben,

```
ping remote IP address
```

ein und drücken **Return**. ***remote IP address*** ist dabei die numerische IP-Adresse des Providers, wie wir sie soeben in **ppp.log** angezeigt bekommen haben. Dem Rechner des Providers werden nun in regelmäßigen Intervallen Testsignale gesendet, die er beantwortet. Dabei kriegen wir etwa folgendes angezeigt:

```
PING remote IP address: 56 data bytes
64 bytes from remote IP address:
icmp_seq=0. time=109. ms
64 bytes from remote IP address:
icmp_seq=1. time=99. ms
64 bytes from remote IP address:
icmp_seq=2. time=99. ms
.
.
64 bytes from remote IP address:
icmp_seq=13. time=109. ms
64 bytes from remote IP address:
icmp_seq=14. time=99. ms
```

Nach der Angabe von ***remote IP address*** folgt die laufende Nummer des Testsignals, danach die Zeit, die bis zur Antwort verstrichen ist. Mit der Eingabe von **Control c** (*nicht* **Command c**!) können wir den Test beenden und erhalten dann eine statistische Auswertung:

```
^C
----remote IP address PING Statistics----
15 packets transmitted, 15 packets recei-
ved, 0% packet loss
round-trip (ms)  min/avg/max = 99/101/119
```

Wenn die prozentuale Anzahl der verlorengegangenen Testsignale (*packet loss*) mehr als ein paar Prozent beträgt, ist mit einer zufriedenstellenden Internet-Verbindung kaum zu rechnen. Insbesondere bei analogen Highspeed-Verbindungen mit 28 800 bit/s kann das ein Problem sein.

Schließlich können wir noch kontrollieren, ob das *Routing*, also die Weiterleitung der Datenpakete je nach Adresse, funktioniert. Dazu geben wir in das Terminalfenster

```
netstat -r
```

ein und drücken **Return**. Wenn alles klappt, erhalten wir etwa folgende Ausgabe:

```
Routing tables
Destination       Gateway      Flags Refs  Use  Interface
localhost         localhost    UH    1     234      lo0
remoteIPaddress   localIPaddress UH  0     0        ppp0
default           remoteIPaddress UG 0     1        ppp0
MaskenAdresse     MeinHostname U     56    7890     en0
```

remote und **local IP address** sind dabei wiederum die bei Verbindungsaufbau übermittelten Werte, **MeinHostname** ist der bei der Netzwerkkonfiguartion für den eigenen Rechner vergebene Name. **MaskenAdresse** ist der Teil der bei der Netzwerkkonfiguration für den eigenen Rechner festgelegten Adresse ***Meine numerische IP-Adresse*** (die im Falle einer festen IP-Adresse mit **local IP address** identisch ist), den die gewählte Netzmaske nicht auf 0 setzt, also zur Bestimmung der Identität des eigenen Subnetzes verwendet. Wichtig ist, daß die beiden Zeilen mit `ppp0` auftauchen; dann ist alles in Ordnung.

`netstat` ist ein UNIX-Programm, das verschiedenste Netzstatistiken ausgibt. Mit der Option `-r` zeigt es das *Routing* unseres Rechners an, das der Rechner nach dem Start in einer sogenannten *Routing-Tabelle* vorhält. Im einfachsten Fall, nach einer Neuinstallation von NEXTSTEP und noch ohne Ethernet-Karte oder eine andere Verbindung zur Außenwelt, würde `netstat -r` nur folgendes ergeben:

```
Routing tables
Destination   Gateway     Flags  Refs    Use  Interface
localhost     localhost   UH     1       234  lo0
```

Das heißt lediglich, daß der Rechner jedenfalls mit sich selbst kommunizieren kann; er ist sowohl sein eigenes *Ziel* (`Destination`) als auch die *Brücke* (`Gateway`) dorthin; daher wird er in diesem Eintrag auch grundsätzlich pauschal als *lokaler Rechner* (`localhost`) bezeichnet, und das Übertragungsprotokoll der *Schnittstelle* (`Interface`) ist ein lokales Pseudo, `lo0`.

Bauen wir nun eine Ethernetkarte ein und konfigurieren unseren Rechner als Netzwerkrechner, so ergibt `netstat -r` schon eine Zeile mehr:

```
Routing tables
Destination    Gateway       Flags   Refs   Use    Interface
localhost      localhost     UH      1      234    lo0
MaskenAdresse  MeinHostname  U       56     7890   en0
```

Über die *Schnittstelle Ethernet, en0*, ist nun ein erster Kontakt zur Außenwelt möglich. Hier läßt sich die Funktion einer Netzmaske schön studieren: angenommen, wir haben (wie oben) die vom **Simple Network Starter** voreingestellte Netzmaske von 255.255.255.0 übernommen und unsere IP-Adresse lautete 123.456.789.012, so wäre *MaskenAdresse* 123.456.789, da genau die letzten drei Stellen der Netzmaske 0 sind und folglich ausgeblendet werden. Und das heißt wiederum, daß alle Datenpakete, deren Adresse mit 123.456.789 *beginnt*, damit rechnen können, im lokalen Subnetz ihr *Ziel* zu finden und also vom eigenen Rechner (*MeinHostname*) zur Ethernet-Schnittstelle geschickt werden, die den physischen Anschluß an das lokale Subnetz darstellt.

Wir haben oben gesehen, daß normalerweise in einem Netzwerk der *Router* die **Default-Route** ist, also diejenige Adresse, an die alle Datenpakete geschickt werden, die lokal nicht zugestellt werden können. In unserem Fall ist an die Ethernet-Schnittstelle aber kein Router angeschlossen, und PPP muß diese Rolle stattdessen übernehmen. Wir können jedoch PPP nicht einfach in die Routing-Tabelle als Default-Route eintragen, da es ja nicht permanent vorhanden ist und die Pakete somit möglicherweise an eine nicht vorhandene Schnittstelle geschickt würden, was UNIX mit einigem Mißfallen aufnähme. Daher trägt PPP sich immer dann, wenn es gestartet wird, selbst in die Routing-Tabelle als `default` ein und beim Verlassen wieder aus. Genau *das* ist es tatsächlich, was die oben eingetragene Option `defaultroute` bewirkt. Ohne diese Option würde die Zeile, die mit `default` beginnt, in der Netzstatistik nicht auftauchen; PPP würde sich lediglich mit einer Zeile bemerkbar machen wie alle anderen Interfaces auch, und kein Programm, das Daten mit nicht nur lokalen Adressen austauschen will, würde dann mehr funktionieren – es gäbe keinen Unterschied dazu, daß überhaupt keine PPP-Verbindung besteht, da sie nicht genutzt werden kann: denn kein Datenpaket käme von sich aus auf die Idee, unabhängig vom eigentlichen Ziel zunächst ausgerechnet zu `remote IP adress` zu wollen.

Sobald nun PPP mit der Option `defaultroute` gestartet ist, erhalten wir also mit `netstat -r` vollständig die obige, vierzeilige Ausgabe, von der wir ausgegangen sind: Wiederum ist unser eigener Rechner die *Brücke* zur Außenwelt; diesmal allerdings mit der für PPP relevanten **local IP address** gekennzeichnet, die ja nur im Falle einer festen IP-Adresse identisch mit der eigenen IP-Adresse ist. Das *Ziel* seinerseits besteht nicht vage aus dem restlichen lokalen Ethernet, sondern klar festgelegt aus dem Einwahlrechner des Providers, der die **remote IP address** hat. Die entscheidende zusätzliche Zeile hat als Ziel `default` – hier geht alles hin, was nirgendwo sonst hingehört, und zwar vermittelt über den Einwahlrechner des Providers (**remote IP address**), der ja tatsächlich die Brücke zum Internet darstellt.

Diese Routing-Tabelle würde von einem fürs Internet bestimmten Datenpaket also wie folgt ausgewertet: »Meine Adresse beginnt nicht mit denselben Ziffern wie *MaskenAdresse* und schon gar nicht ist sie identisch mit der des *lokalen Hosts*; also muß ich mich nach der Default-Route umsehen. – Aha, die Brücke zu `default` ist **remote IP address**! – Doch wie gelange ich dorthin? – Aha, die *Brücke* zu **remote IP address** ist **local IP address**, und zwar über die *Schnittstelle* `ppp0`.« Sprach's und wanderte los…

Die PPP-Verbindung **beenden**. Hierzu in das Terminalfenster

```
kill PID
```

eingeben und **Return** drücken. **PID** ist wie oben beschrieben die in `ppp.log` hinter `pppd` in rechteckigen Klammern angegebene Prozeß-ID von `pppd`. Wenn alles geklappt hat, das Fenster, in dem **Terminal.app** die Log-Datei mit `tail -f` angezeigt hatte, durch Klick auf den Schließknopf schließen (und mit OK bestätigen).

Sollte die PPP-Verbindung nicht wie hier beschrieben funktionieren, ist Fehlersuche angesagt. Dabei hilft die Dokumentation, die PPP beigegeben ist. Zunächst ist zu unterscheiden, ob die Einwahlprozedur nicht funktioniert und es gar nicht bis zur Verbindung kommt; dann muß das Einwahlskript modifiziert werden. Bei *asynchronem* PPP sind alle relevanten Daten in `ppp.log` bzw. im Terminalfenster zu sehen, da sie allesamt von `chat` kommen, das stets Verlaufsmeldungen ausgibt, da es mit der Option `-v` gestartet wurde; bei *synchronem* PPP, wo PPP selbst die Identifikation vornimmt, muß in `options` erst `debug` aktiviert werden, damit man sieht, wo das Problem liegt. Kommt die Einwahl zustande, aber PPP wird nicht aktiviert, so haben die PPPs der beiden Seiten Schwierigkeiten bei der Verständigung; auch hier ist zur Fehlersuche `debug` zu setzen.

Wenn wir die obige Netzstatistik betrachten, fällt auf, daß bis auf *MeinHostname* nur numerische IP-Adressen angegeben werden. Das liegt daran, daß wir noch keine Verbindung zu einem *Nameserver* haben, der uns die logischen Namen fremder Rechner mitteilen könnte.

Internet-Adreß- und Mail-Konfiguration

Ab nun gehen wir davon aus, daß wir (über PPP) eine **voll funktionsfähige TCP/IP-Verbindung** ins Internet haben. Um dessen Dienste nutzen zu können, müssen wir jetzt noch für die korrekte Verarbeitung logischer Adressen sorgen, für die im Internet nicht **netinfo**, sondern das **DNS** zuständig ist (s.o.).

Eine neue ASCII-Datei öffnen und folgendes eintragen:

```
# Name der eigenen Domain
domain ProviderDomain.de

# DNS-Nameserver
nameserver 123.456.789.012
nameserver 123.456.789.012
nameserver 123.456.789.012
```

Die kursiven Stellen müssen durch die Angaben des Providers ersetzt werden. ***ProviderDomain.de*** ist dabei der hintere Teil der vom Provider angegebenen E-Mail-Adresse *ohne* ***ProviderMailhostname*** und *ohne* den zugeteilten ***ProviderNutzername***. Diese Einstellung ist auch von der **sendmail**-Konfiguration abhängig und gilt so nur für die Standardeinstellung; siehe hierzu den Schluß von **Netstep** und die folgende **sendmail**-Konfiguration. Zeilen mit (*numerischen!*) Nameserver-Adressen werden so viele eingetragen, wie man vom Provider erhalten hat (maximal 3, mehr werden nicht ausgewertet).

Die Datei als **/etc/resolv.conf** sichern und schließen; anschließend den **Rechner neu starten**. Änderungen des *Domainnamens* wirken *auch bei laufender PPP-Verbindung sofort* auf alle Post, die danach von **Mail.app** versandt wird. Einträge von *Nameservern* werden **erst nach einem Neustart des Rechners wirksam**, der daher nach der Erstellung von **resolv.conf** erfolgen muß.

Den Erfolg unserer Maßnahme können wir sofort überprüfen, indem wir nochmals wie oben PPP starten und `netstat -r` in ein Terminalfenster eingeben:

```
Routing tables
Destination      Gateway          Flags Refs Use  Interface
localhost        localhost        UH    1    234  lo0
ProviderHostname dynamischerName  UH    0    0    ppp0
default          ProviderHostname UG    0    1    ppp0
MaskenAdresse    MeinHostname     U     56   7890 en0
```

Statt **remote IP address** können wir nun ***ProviderHostname*** lesen, den Hostnamen des Einwahlrechners beim Provider, den sich `netstat` automatisch von einem Nameserver im Netzwerk besorgt hat. Hieran sieht man schön, wie sehr in einem UNIX-Netz einzelne Rechner wirklich zu einem Ganzen verschmelzen. Auch beim Gateway ist jetzt ein Name eingetragen. Haben wir eine *dynamische* IP-Adresse, so wird dies ***dynamischerName***, ein fiktiver, vom Einwahlrechner des Providers vergebener Name sein, da der Adresse ja kein realer Rechner fest zugeordnet ist. Im Fall einer festen IP-Adresse stünde hier natürlich genauso *MeinHostname* wie in der vierten Zeile, da es sich dann ja auch um dieselbe Adrese handelt (wie es de facto ja auch um denselben Rechner geht). Einzig *MaskenAdresse* bleibt numerisch, da es sich dabei schließlich nicht um eine vollständige Adresse handelt, die einen bestimmten Rechner beschreiben würde, sondern eben um eine Maske.

Damit sind wir im Prinzip schon fertig. **Im Gegensatz zu sich hartnäckig haltenden Gerüchten muß an der gefürchtet komplizierten sendmail-Konfigurationsdatei nichts verändert werden, damit der Transport von E-mail funktioniert.** Das gilt allerdings nur für die *Standardeinstellung*. Diese setzt eine

- **feste IP-Adresse** voraus

oder aber, daß

- ***entweder*** der *Nutzername auf dem eigenen Rechner identisch mit dem Nutzernamen beim Provider* ist und der Provider sich bereit findet, auf seinem Netzwerk als Host Alias für ***ProviderMailhostname***, also seinen Mailhost, ***MeinHostname***, also den Namen des eigenen Rechners, einzutragen (was voraussetzt daß er in seinem Adreßpfad einen ***ProviderMailhostname*** hat...),
- ***oder*** der Provider einen eigenen ***ProviderMailhostname*** hat und man bereit ist, in **Mail.app** ein **Reply-To** zu diesem Host einzutragen.

Siehe hierzu den letzten Abschnitt von **Netstep**, am Ende der ersten Folge des Artikels. In der Standardeinstellung setzt **sendmail** die Absenderadresse aus dem Namen des eigenen Rechners (***MeinHostname***) und dem dafür in **resolv.conf** angegebenen Domain-Namen des Providers (***ProviderDomain***), in dessen Subnetz man sich ja befindet, zusammen, wie sich das für ein »richtiges« Netzwerk gehört. Für wen das OK ist, der kann sofort zum nächsten Abschnitt **Automatisieren der Verbindung** springen.

Spezifische Angaben sind in **sendmail.mailhost.cf** deswegen nicht notwendig, weil die Einstellung **defaultroute** in den PPP-Optionen ohnehin dafür sorgt, daß alle nicht lokalen Datenpakete zum Provider gesandt werden. Alles weitere übernimmt der dortige Mailhost.

In folgenden Fällen ist jeweils eine geringfügige **Modifikation** von **sendmail.mailhost.cf** und gegebenenfalls **resolv.conf** erforderlich:

A Man will auch Nutzer mit E-Mail erreichen, die *direkt* auf der Domain-Ebene des Providers angesiedelt sind, obwohl **netinfo** nicht entsprechend konfiguriert ist. Siehe hierzu im folgenden: **1. Erzwingung einer bestimmten Mail-Adresse**

Das ist ein relativ exotischer Spezialfall, denn normalerweise haben Nutzer in der Domain des Providers Hostnamen in ihrer Adresse. Wenn wir nun E-Mail von ***@MeinHostname.ProviderDomain.de*** an ***@AndererHostname.ProviderDomain.de*** schicken wollen, so wird unser Rechner die Post auf die Default-Route verweisen, da er ja einen Teil der Adresse, ***AndererHostname***, nicht kennt, und alles geht in Ordnung. Wenn es aber in der Domain unseres Providers einen Nutzer auf Domain-Ebene ohne weitere Spezifikation gibt – ein typisches Beispiel wäre **info@ProviderDomain.de** –, so wird unser Rechner davon ausgehen, daß er diesen Nutzer kennen **muß**, da ihm der gesamte Adreßpfad bekannt ist: denn ***ProviderDomain.de*** ist über **resolv.conf** schließlich seine eigene, ihm vertraute Domain. Das würde auch funktionieren, wenn wir **netinfo** auf unserem Rechner tatsächlich als vollwertigen Teil eines inhomogenen (nicht nur aus NEXTSTEP-Rechnern bestehenden) Netzwerkes konfiguriert hätten; doch das ist ein reichlich komplexes Unterfangen, wie ein kurzer Blick ins *Network and System Administration Manual* lehrt, und bei einer nur-PPP-Anbindung ohnehin undurchführbar. Unser Rechner wird daher vergeblich versuchen, die Post lokal zuzustellen und mit einer Fehlermeldung abbrechen. Dies läßt sich vermeiden, wenn wir künstlich unsere *gesamte* Adresse einschließlich ***MeinHostname*** zur Domain erheben, da sie dann nicht mehr identisch mit der des Providers ist und in jedem Fall **defaultroute** wirksam wird; *nach außen* bleibt unsere Adresse aber dieselbe.

B Man will im Falle einer dynamischen IP-Adresse und bei *identischem Nutzernamen auf dem eigenen Rechner und dem des Providers* im Absender ***ProviderMailhostname*** statt ***MeinHostname*** stehen haben, um so ein **Reply-To** zu vermeiden, obwohl der Provider nicht bereit ist, ***MeinHostname*** als Host Alias für ***ProviderMailhostname*** einzutragen (siehe auch hierzu den letzten Abschnitt von **Netstep**, am Ende der ersten Folge des Artikels). Dies gilt auch für den **Spezialfall**, daß der Provider in seiner Mail-

adressierung gar keine Hostnamen vorsieht, die Postadresse also ***ProviderNutzername@ProviderDomain.de*** lautet – in diesem Fall ist ***ProviderMailhostname*** sozusagen leer (siehe für diesen Fall als Alternative **C**). **ACHTUNG!** Bei dieser Modifikation können Nutzer desselben Providers mit *demselben Adreßpfad* nicht mehr angeschrieben werden, weil `sendmail` sie aufgrund des identischen Adreßpfades für lokale Nutzer hält, sie auf dem eigenen Rechner aber natürlich nicht findet. Siehe hierzu im folgenden: **1. Erzwingung einer bestimmten Mail-Adresse**

C Man will andere Nutzer eines Providers erreichen können, dessen Adreßpfad keinen Hostnamen aufweist, sondern nur ***@ProviderDomain.de*** lautet. In diesem Fall nützt es nichts, daß die Standardeinstellung als Adreßpfad ***@MeinHostname.ProviderDomain.de*** erzeugt, da darin der gesamte Adreßfad des Providers enthalten ist, `sendmail` daher alle Nutzer dieses Pfades wiederum für lokal hält und vergeblich auf dem eigenen Rechner sucht. Hier hilft es nur, eine Unterscheidung künstlich zu erzeugen, indem man in `resolv.conf` als Domain einen **Phantasienamen** angibt und ansonsten ein **Reply-To** auf ***ProviderNutzername@ProviderDomain.de*** verwendet.

D Man will erzwingen, daß die eigene Post in jedem Fall schon zum Mailhost des Providers gesandt wird, auch wenn der eigentliche Adressat zur Zeit nicht erreichbar ist. Siehe hierzu im folgenden: **2. Erzwingung des Postversands zum Mailhost des Providers**. **ACHTUNG!** In diesem Modus **arbeitet die Belegfunktion von NeXTMail** sehr wahrscheinlich **nicht** mehr (siehe die folgenden Erläuterungen)**!**

In der Standardeinstellung versucht `sendmail`, direkt den Adressaten über das Internet zu erreichen. Ist das augenblicklich nicht möglich, wird die Post auf dem *eigenen* Rechner zum erneuten Versuch in **/usr/spool/mqueue** gelagert. Da der eigene Rechner aber nicht permanent mit dem Internet verbunden ist, kann eine Wiederholung des Zustellversuchs nur sehr unregelmäßig erfolgen. Viele Nutzer ziehen es daher vor, in einer solchen Situation die Post jedenfalls schon zum Mailhost des Providers zu schicken und diesem, permanent mit dem Internet verbundenen Rechner die weiteren Zustellversuche zu überlassen. Dies sollte man sich allerdings genau überlegen, da damit einige teils gravierende Nachteile verbunden sind:

- Die **unmittelbare Kontrolle über die Zustellung der Post geht verloren**, denn **/usr/spool/mqueue/syslog** (die Log-Datei für den Postversand) meldet nun schon eine erfolgreiche Zustellung, wenn die Post beim Provider angekommen ist. Daß sie dort liegenblieb, erfährt man vielleicht erst nach langer Zeit bei der nächsten Einwahl ins Internet. Insbesondere sollte man sich nicht von den scheinbar verkürzten Zustellzeiten für Post blenden lassen: denn auch diese beziehen sich jetzt eben nicht mehr auf das Erreichen des Empfängers, sondern nur auf das Erreichen des (nahen) Providers.
- Im Zusammenhang mit NEXTSTEP kommt es zu einem ärgerlichen Fehler: üblicherweise werden in dem der eigentlichen Post vorangestellten Adreß-*Header* (in dem die gesamte Adressierung der Post gespeichert ist) die einzelnen Stationen des Postwegs durch Zeilen vermerkt, die jeweils mit **»Received:«** beginnen. Durch die erzwungene Etappe beim Mailhost des Providers wird dieser nun veranlaßt, sozusagen Zwischenbilanz zu ziehen, und macht in der Regel aus der ersten vorhandenen und NeXT-spezifischen Meldung, nämlich **»Received:** by NeXT.Mailer (1.118.2)«, die von dem Mailprogramm unseres eigenen Rechners stammt, ein **»Original-Received:** by NeXT.Mailer (1.118.2)«. Diese Meldung wiederum ist dem *Post Protokoll* idiotischerweise unbekannt, so daß es mit einer Warnung in der darauffolgenden Zeile reagiert: **»PP-warning:** Illegal Received field on preceding line«. Ein solcher Schönheitsfehler wäre freilich recht leicht zu verschmerzen, da einem **Mail.app** in den Präferenzen (Popup *Kopfzeilen*) die Möglichkeit bietet, alle störenden Header-Meldungen auszublenden, so daß nur ein Eintrag *PP-warning* erforderlich wäre. Viel schlimmer ist aber, daß die in NeXTMail vorgesehene automatische Empfangsbestätigung (einzuschalten durch den Schalter *Beleg* ganz rechts im *Erstellen*-Fenster von **Mail.app**) nicht mehr funktioniert; diese Funktion wird nämlich dadurch aktiviert, daß in der **Received**-Meldung des lokalen Rechners am Ende die Buchstaben **RR** (für *R*ead *R*eceipt = Empfangsbestätigung) angehangen werden, also **»Received:** by NeXT.Mailer (1.118.2.RR)«. Auf *genau diese Meldung* reagiert der Zielrechner mit einer Empfangsbestätigung, *nicht aber* auf **»Original-Received:** by NeXT.Mailer (1.118.2.RR)«. Die **automatische Belegfunktion von NeXTMail arbeitet** daher in diesem Modus üblicherweise **nicht**.
- Hat man bei dynamischer IP-Adresse die oben beschriebene Standard-Lösung gewählt, bei gleichem Nutzernamen auf dem Netzwerk des Providers ***MeinHostname*** als Host Alias für ***ProviderMailhostname*** eintragen zu lassen, so daß man seine Post mit der korrekten Adresse ***MeinHostname.ProviderDomain.de*** versendet, so wird diese Adresse bei dem erzwungenen Zwischenaufenthalt durch das `sendmail` des Providers in ***ProviderMailhostname.ProviderDomain.de*** zurückverwandelt. *Technisch* macht das keine Probleme, da dies ja in der Tat genau die richtige Adresse zum Beantworten der Post ist. Wem es aber *»ästhetisch«* oder aus Konsistenzgründen wichtig ist, daß in seiner Adresse auch der Name seines eigenen Rechners steht, für den kommt dieser Modus daher nicht in Frage.

Es folgen die nötigen Modifikationen von `sendmail.mailhost.cf` für die beiden genannten Aufgaben. Hierzu **/etc/sendmail/sendmail.mailhost.cf** in **Edit.app** öffnen.

Die `sendmail`-Konfigurationsdateien sind gefürchtet wegen ihrer Unverständlichkeit. Die liegt wesentlich daran, daß die Konfiguration bei *jedem* Versand eines elektronischen Briefes von `sendmail` *neu* eingelesen und ausgewertet wird, so daß bei der Definition der Konfigurationsdatei die »Lesbarkeit« und mithin Auswertgeschwindigkeit für den Computer Vorrang vor der Lesbarkeit für Menschen hatte. Aus diesem Grund besteht eine einzelne Anweisung stets nur aus *einem* Buchstaben; um für menschliche Augen das Chaos perfekt zu machen, kann dabei ein und derselbe Buchstabe auch noch Verschiedenes bedeuten, je nachdem, *wo* in der Datei er auftaucht. So kündigt `R` als erster Buchstabe einer Zeile an, daß in dieser Zeile eine *R*egel (zur Umwandlung von Postadressen) folgt; weiter hinten in einer Zeile handelt es sich bei `R` jedoch um eine Variable, der zuvor in einer mit `D` beginnenden *D*efinitionszeile ein Wert zugewiesen wurde. Selbstredend wird auch D an anderer Stelle wieder als Variable benutzt...

Bei NEXTSTEP werden drei vorgefertigte Varianten der `sendmail`-Konfigurationsdatei mitgeliefert. `sendmail.sharedsubsidiary.cf` wird von Rechnern verwendet, die selbst weder über einen Ordner **/usr/spool/mail** noch über **/usr/spool/mqueue** verfügen und somit auf ständige Verbindung zu einem Netzwerk angewiesen sind, um überhaupt Post empfangen und senden zu können (*shared subsidiary* = geteilte Ressourcen); für uns ist diese Datei also vollkommen uninteressant. `sendmail.subsidiary.cf` ist für Rechner gedacht, die solche Verzeichnisse als Ressource (*subsidiary*) selbst zur Verfügung haben, aber nicht der Mailhost, also der zentrale Mail-verwaltende Rechner des Netzwerkes sind, für den `sendmail.mailhost.cf` zur Verfügung steht. Da unser einzelner Rechner schon unser ganzes »Subnetz« ist, muß er auch die Aufgaben des Mailhosts übernehmen. Notfalls ließe sich auch, entsprechend modifiziert, `sendmail.subsidiary.cf` verwenden; aus dieser Perspektive wäre unser Rechner dann nur ein einzelner Rechner im Netzwerk des Providers, zu dem er freilich keine ständige Verbindung hat. Allerdings fallen damit die gesamten obigen Konfigurationsmöglichkeiten fort; `sendmail.subsidiary.cf` verhält sich stets so wie `sendmail.mailhost.cf` *mit* den Modifikationen **A** oder **B** und **D**. In einem unkonfigurierten NEXTSTEP-Rechner hält `sendmail` Ausschau nach der Datei `sendmail.cf`, die einen Verweis auf die jeweils ausgewählte Konfigurationsdatei darstellt (so ließe sich im einfachsten Falle `sendmail.subsidiary.cf` zur Konfigurationsdatei für einen nicht als Mailserver konfigurier-

ten Rechner machen). Da wir unseren Rechner aber als Mailserver konfiguriert haben, wird automatisch **sendmail.mailhost.cf** verwendet.

1. Erzwingung einer bestimmten Mail-Adresse

Die folgende Passage zu Beginn von **sendmail.mailhost.cf** suchen:

```
# my fully qualified hostname, $j, is now
set by sendmail internally
#Dj$?m$w.$m$|$w$.
# if you want the gateway to assume the
identity of its parent domain, use:
#Dj$m
```

Das # in der letzten der vier Zeilen entfernen (so daß die Zeile `Dj$m` lautet). **sendmail.mailhost.cf** sichern und schließen. **sendmail** verwendet nun die Angabe aus der Zeile domain in **resolv.conf** als *vollständige* Absender-Adresse und stellt den Namen des eigenen Rechners (***MeinHostname***) *nicht* mehr davor.

Durch das Entfernen von # wird die folgende Definitionszeile für die Rechneradresse `j` des Absenders aktiviert, die für `j` einfach den Wert aus **resolv.conf** (= *the identity of its parent domain*) einsetzt, der hier durch die Variable **m** repräsentiert wird (Variablen werden in **sendmail** durch **$** markiert).

/etc/resolv.conf in **Edit.app** öffnen. Für Fall **A** die Zeile mit der Angabe der Domain durch

```
domain MeinHostname.ProviderDomain.de
```

ersetzen. Für Fall **B** die Zeile mit der Angabe der Domain durch

```
domain ProviderMailhostname.ProviderDomain.de
```

ersetzen. Für den Spezialfall, daß ***ProviderMailhostname*** leer ist, bleibt **resolv.conf** mithin *unverändert*. **/etc/resolv.conf** sichern und schließen.

2. Erzwingung des Postversands zum Mailhost des Providers

Die folgende Passage am Ende von **sendmail.mailhost.cf** suchen:

```
# If you want to pass all other explicit
domain names up the ladder
# to our forwarder then uncomment the fol-
lowing line.
#R$*<@$*.$+>$*      $#$M    $@$R
$:$1<@$2.$3>$4      user@any.domain
# and comment out this one.
R$*<@$+.$->$*       $#ddn $@ $2.$3
$:$1<@$2.$3>$4    user@any.domain
```

Das # in der *dritten Zeile entfernen* und ein # zu *Beginn der fünften Zeile* setzen:

```
# If you want to pass all other explicit
domain names up the ladder
# to our forwarder then uncomment the fol-
lowing line.
R$*<@$*.$+>$*       $#$M    $@$R
$:$1<@$2.$3>$4      user@any.domain
# and comment out this one.
#R$*<@$+.$->$*      $#ddn $@ $2.$3
$:$1<@$2.$3>$4    user@any.domain
```

Die dritte und fünfte Zeile enthalten zwei alternative Regeln für die Weiterleitung der Post; die dritte Zeile erzwingt dabei den Versand aller nicht lokal zustellbaren Post (*all other explicit domain names*) zumindest zur nächsten Station im Internet (*up the ladder to our forwarder*; die Leiter eine Sprosse hinauf zu demjenigen, der unsere Post weiterleitet). Dazu muß diese erzwungene *nächste Station* aber explizit bekannt gemacht werden und auch der sogenannte Mailer (eine Zusammenstellung aller für eine bestimmtes Übertragungsprotokoll notwendigen Daten), der für den Transport dorthin zuständig ist. Wo in der fünften Zeile für den Mailer einfach der Internet-Mailer (also der Mailer für TCP/IP-Übertragungen) festgelegt ist (`ddn`; das steht für *Defense Data Network* und verweist noch auf den militärischen Ursprung des Internets), steht in der dritten Zeile die *Variable* **M**; ebenso findet sich in der dritten Zeile die *Variable* `R`, mit der die nächste Station definiert wird:

```
R$*<@$*.$+>$*       $#$M    $@$R $:$1<@$2.$3>$4     user@
any.domain
```

Diese Variablen müssen wir nun also in entsprechenden Definitionszeilen noch festlegen.

Die folgende Passage zu Beginn von **sendmail.mailhost.cf** suchen:

```
# If you want to use a relay mailer, ex-
amine ruleset 0.  There are some
# rules that need to be uncommented
DMuucp

# major relay host: use the $M mailer to
send mail to other domains
# To have mail automatically forwarded to
other domains, you should
# replace this with the name of your major
relay host.
DR mail-relay
CR mail-relay
```

DMuucp durch **DMddn** ersetzen, **mail-relay** jeweils durch die **logische IP-Adresse des Mailservers des Providers** ersetzen (diese Adresse wird in der Regel ***ProviderMailhostname.ProviderDomain.de*** sein, *muß* es aber nicht, da Provider aus Gründen der Netzwerkadministration unter Umständen für ***ProviderMailhostname*** in der *E-Mail-Adresse* ein Alias verwenden können, das es als Alias für den *Rechnernamen* (also als *Host*-Alias) gar nicht gibt):

```
# If you want to use a relay mailer, ex-
amine ruleset 0.  There are some
# rules that need to be uncommented
DMddn

# major relay host: use the $M mailer to
send mail to other domains
# To have mail automatically forwarded to
other domains, you should
# replace this with the name of your major
relay host.
DR logische IP-Adresse des Mailservers des
Providers
```

`CR logische IP-Adresse des Mailservers des Providers`

Damit haben wir als Mailer (unverändert) den Internet-Mailer und als nächste Station den Mailserver unseres Providers definiert.

/etc/sendmail/sendmail.mailhost.cf sichern und schließen. Änderungen wirken *auch bei laufender PPP-Verbindung sofort* auf alle Post, die danach von **Mail.app** versandt wird.

Die Konfiguration von `sendmail` ist damit abgeschlossen.

Automatisieren der Verbindung

Wir wenden uns nun der Automatisierung der bei Aufbau und Abbruch der PPP-Verbindung notwendigen Aktionen zu.

PopOver.1.4.NIHS.bd.tar.gz durch Doppelklick entpacken; **PopOver.app** nach **/LocalApps** verschieben. In der Regel macht es keinen Sinn, **PopOver.app** auch schon zu konfigurieren, da diese Konfiguration für jeden Nutzer spezifisch angelegt wird und wir uns noch in **root** befinden und nicht unter unserem eigenen Nutzernamen eingeloggt sind, unter dem wir später normalerweise die Verbindung zum Internet werden herstellen wollen. Wer aus irgendwelchen Gründen auch von **root** aus ins Internet will, kann freilich **PopOver.app** zusätzlich auch unter **root** so konfigurieren wie unten beschrieben; allerdings mit zwei Änderungen: erstens darf **Mail.app** nicht automatisch benachrichtigt und somit gestartet werden (in den *Preferences*, Popup-Einstellung *General*, die erste Option deaktivieren), da wir ja in **root** nicht unser Mailverzeichnis haben, und zweitens müssen wir in *Preferences*, Popup-Einstellung *Expert*, im Feld *Username* explizit ***MeinNutzername*** eingeben, da **PopOver.app** von sich aus hier immer den Login-Namen, in diesem Fall also fälschlicherweise **root**,verwenden würde. **PopOver.app** im Falle einer Internet-Verbindung in **root** ausnahmsweise nicht zu aktivieren, ist *nicht* möglich, da `ip-up` eine systemweit verwendete Konfigurationsdatei ist.

Eine neue ASCII-Datei öffnen und folgendes eintragen, dabei insbesondere auf die Leerzeichen um die eckigen Klammern und die senkrechten Striche achten!:

```
#!/bin/sh


#### AKTIONEN ZU BEGINN EINER IP-VERBIN-
DUNG ####


###  MAIL  ###

# angesammelte Mail aus Mailqueue im Hin-
tergrund versenden
(/usr/lib/sendmail -q) &

# PopOver.app starten, um Mail vom Provi-
der abzuholen
open /LocalApps/PopOver.app


###  WWW  ###

#Falls nicht schon geöffnet, Netsurfer
starten und Seite des Providers mit
#aktuellen Meldungen aufrufen
pid='ps -ax | egrep "Netsurfer" | egrep -v
"egrep" | sed 's/^\([ 0-9]*\) .*/\1''
if [ "${pid}" = "" ]; then
open /LocalApps/Netsurfer.app
fi
```

Für ***Netsurfer*** ist hierbei gegebenenfalls ein anderer WWW-Browser einzugeben. **ACHTUNG!** Solange noch kein WWW-Browser installiert ist, müssen die letzten vier Zeilen durch jeweils ein # an ihrem Beginn deaktiviert werden!

Die Datei als **/etc/ppp/ip-up** sichern und schließen. Das Datei-Icon im **WorkspaceManager** einmal anklicken und im Inspektor, Popup-Einstellung *Zugriffsrechte*, für die Datei durch Anklicken der entsprechenden Felder für alle drei Spalten die Zeile *Ausführen* einschalten. Das Icon der Datei wandelt sich darauf in das UNIX-Programmicon.

Dieses mit UNIX-Skript führt die Befehle nach erfolgreichem PPP-Verbindungsaufbau aus, wie in den Kommentarzeilen angegeben. Dabei bedient es sich teilweise der »hohen Schule kryptischer UNIX-Skripte«, um festzustellen, ob **Netsurfer.app** schon läuft, bevor es das Programm startet (`egrep` sucht nach bestimmten Namen; mit seiner Hilfe findet die erste Zeile heraus, ob es eine Prozeß-ID (`pid`) für einen Prozeß namens `Netsurfer` gibt; ist das nicht der Fall (`pid=""`), wird das Programm gestartet). Bei dem Start von `sendmail` bewirkt das `&`, daß `sendmail` *im Hintergrund* weiterläuft, d.h. das Skript nicht erst dann fortfahren kann, wenn `sendmail` beendet ist. Der WWW-Browser sollte so konfiguriert werden, daß die WWW-Seite, die er beim Programmstart anwählt, aktuelle Meldungen des Providers anzeigt. Bei **OmniWeb** stellt man diese Seite in den Präferenzen unter *Home Page* ein; bei **Netsurfer** im *Resource Inspector* der entsprechenden Seite durch einen Klick auf *Open At Startup*.

Eine neue ASCII-Datei öffnen und folgendes eintragen:

```
#!/bin/sh

#### AKTIONEN ZUM ENDE EINER IP-VERBINDUNG
####

###  MAIL  ###

# PopOver.app beenden
kill 'ps -ax | egrep "PopOver" | egrep -v
"egrep" | sed 's/^\([ 0-9]*\) .*/\1''
```

Die Datei als **/etc/ppp/ip-down** sichern und schließen. Das Datei-Icon im **WorkspaceManager** einmal anklicken und im Inspektor, Popup-Einstellung *Zugriffsrechte*, für die Datei durch Anklicken der entsprechenden Felder für alle drei Spalten die Zeile *Ausführen* einschalten. Das Icon der Datei wandelt sich darauf in das UNIX-Programmicon. Änderungen wirken beim nächsten Beenden einer PPP-Verbindung, auch wenn diese zum Zeitpunkt der Änderung schon bestand.

Dieses UNIX-Skript beendet **PopOver.app** nach Ende der PPP-Verbindung (wobei es herausfinden muß, welche Prozeß-ID **PopOver.app** hat, da nur damit der `kill`-Befehl funktioniert). Das ist nötig, da **PopOver.app** so konfiguriert werden kann, daß es in regelmäßigen Abständen Post holen will, was aber ohne PPP-Verbindung natürlich nicht geht.

`pppd` wie oben beschrieben erneut in einem Terminalfenster starten und mit `kill` wieder beenden (die Prozeß-ID hierfür ist wiederum in `ppp.log` zu sehen). Wenn alles richtig konfiguriert ist, muß beim Aufbau der Verbindung **PopOver.app** automatisch starten und beim Abbruch automatisch beendet werden. (Es ist noch nicht wichtig, was **PopOver.app** nach dem Start tut.)

Steuerung durch GateKeeper

Die eigentliche PPP-Internet-Anbindung haben wir nun fertiggestellt und könnten sie über **Terminal.app** bereits vollständig benutzen. Die nun noch folgende Installation von **GateKeeper.app** greift in den Aufbau der Verbindung selbst nicht ein; **GateKeeper.app** dient vielmehr dazu, die Aktionen anstelle von **Terminal.app** komfortabel zu steuern und zu kontrollieren und die nutzerspezifischen Konfigurationsskripte – auch für verschiedene Verbindungs-Alternativen – übersichtlich zu verwalten.

GateKeeper.1.0.Beta2.NI.b.tar.gz durch Doppelklick entpacken und **GateKeeper.app** nach **/LocalApps** verschieben. Sein Icon im **WorkspaceManager** durch einen einfachen Mausklick anwählen und **Command O** drücken; dadurch wird das Programm wie ein Ordner in einem eigenen Fenster geöffnet und ihre einzelnen Bestandteile werden zugänglich. In **Terminal.app** in ein geöffnetes Fenster folgendes eingeben:

`chmod u+rws,g+rx,o+rx` [Leertaste]

sodann aus dem **GateKeeper.app**-Ordner **GateKeeper** (*nicht **GateKeeper.appl***) mit der Maus ins Terminal-Fenster ziehen und **Return** drücken. Damit gilt **GateKeeper.app** auch dann als von **root** gestartet, wenn es de facto von einem Nutzer gestartet wurde.

Die Bestandteile eines Programms werden unter NEXTSTEP stets in einem speziellen Ordner, dem *Application Wrapper* (»*Anwendungs-Einpacker*«) zusammengefaßt, der nach außen aber nicht als Ordner, sondern als das Programm selbst erscheint und ja auch so gestartet werden kann. `chmod` (*change mode*) ist ein UNIX-Befehl, der die Zugriffsrechte auf das Programm ändert, die ja auch im Inspektor des **WorkspaceManager**, Popup *Zugriffsrechte*, eingestellt werden können. Allerdings erlaubt die Einstellung über das chmod-Kommando eine Variante, die im **WorkspaceManager** nicht möglich ist: der Gruppe (`g`) oder aber dem Eigentümer (`u`) der Datei kann neben den Zugriffsrechten *Lesen* (`r`), *Schreiben* (`w`) und *Ausführen* (`x`) auch noch das Attribut *Setze als Eigentümer* (`s`) zugeordnet werden. In diesem Fall gelten *Gruppe* bzw. *Eigentümer* als Nutzer der Datei, *auch wenn sie von jemand anderem gestartet wurde*. Die Zugriffsrechte eines so gestarteten Programms werden auf diese Art und Weise auf Dateien erweitert, auf die es sonst keinen Zugriff hätte, weil der nur dem Eigentümer bzw. einer bestimmten Gruppe, nicht aber den *Anderen* (`o`) gestattet ist. Wie alle Programme haben wir **GateKeeper.app** soeben unter **root** installiert; **root** ist also der Eigentümer. Wie in NEXTSTEP voreingestellt haben aber auch Gruppe und Andere das Recht, **GateKeeper.app** auszuführen. Nur sind dann sie die *Nutzer* des Programms, und das Programm ist in seinen Zugriffsrechten automatisch auf die seines Nutzers eingeschränkt (sonst hätten die Zugriffsbeschränkungen auf einem Multiuser-System mit gemeinsam genutzten Programmen ja gar keinen Sinn). Auch wenn **GateKeeper.app** als Eigentümer also **root** hat, kann er, von einem anderen Nutzer gestartet, auf auf **root** beschränkte Dateien nicht mehr zugreifen. Nun soll uns **GateKeeper.app** ja aber gerade der Steuerung der PPP-Dateien dienen, die als Systemdateien in wichtigen Teilen nur **root** zugänglich sind. So kann z.B. nur **root** in die Log-Dateien schreiben oder `pppd` mit dem `kill`-Kommando beenden. Daher definieren wir durch das `s` in obigem Kommando, daß **GateKeeper.app** als *Nutzer* stets seinen *Eigentümer*, also **root**, und somit alle notwendigen Zugriffsrechte hat. Ohne diese zusätzliche Einstellung funktioniert die Steuerung von PPP durch **GateKeeper.app** *überhaupt* nicht.

Die Datei `options` im **GateKeeper.app**-Ordner löschen und **/.ppprc** mit der Maus in den Ordner ziehen; dabei die **Alternate**-Taste gedrückt halten, so daß die Datei in den Ordner *kopiert* wird. Auf den Namen der Datei klicken und sie in `options` umbenennen. Das Ordner-Fenster schließen.

Damit haben wir unsere nutzerspezifische PPP-Optionsdatei zu der Datei gemacht, die **GateKeeper.app** standardmäßig zum automatischen Verbindungsaufbau verwendet, wenn man bei ihm das *Link*-Menü betätigt (daher wird die Datei bei **GateKeeper** auch als *Link options*-Datei bezeichnet). Empfehlenswerter ist freilich, zum Verbindungsaufbau stets über die Werkzeugleiste (*Toolbar*) ein sogenanntes **.Gate**-Dokument zu verwenden, wie wir es gleich anlegen werden, da diese Dokumente individuell erstellbar und leicht zu ändern sind, während die *Link options*-Datei für alle Nutzer gleichermaßen stets gilt und auch nur von **root** aus zu ändern ist.

GateKeeper.app erlaubt (für asynchrones PPP) für besondere Situationen auch die manuelle Anwahl eines Providers samt Eingabe der entsprechenden Kennworte. Das setzt allerdings voraus, daß die Datei `.ppprc` im Heimverzeichnis des entsprechenden Nutzers nicht vorhanden ist, da ihre Befehle zu automatischer Einwahl sonst stets von `pppd` ausgewertet werden, wenn keine konkurrierenden Befehle von **GateKeeper.app** vorliegen (die dann die von `.ppprc` überschreiben). Erwägt man, die manuelle Anwahl auch von **root** aus zu nutzen, darf **/.ppprc** also nicht bestehen bleiben. In diesem Fall **/.ppprc** und **/pppup** jetzt mit der Maus in das eigene Heimverzeichnis (wo wir die Dateien noch brauchen) ***verschieben*** (**Command**-Taste während der Maus-Aktion drücken), ansonsten **kopieren**.

Da **GateKeeper.app** an die Stelle von `.ppprc` seine elegant verwalteten **.Gate**-Dokumente setzt, ist ein Fehlen von `.ppprc` ohnehin kein Problem.

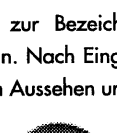
Damit haben wir die Installation beendet, verlassen **root** und loggen uns als **Nutzer** ein. **Sollten plötzlich Dinge nicht mehr funktionieren, die bis jetzt geklappt haben, so liegt dies fast mit Sicherheit an fehlerhaften Zugriffsrechten, die dem Nutzer nicht mehr ermöglichen, was root erlaubt war**. In Problemfällen sollte man also stets im Inspektor des **WorkspaceManager** die Zugriffsrechte der wichtigen Dateien überprüfen.

Nutzerspezifische Konfiguration

Die nun folgende Einstellung auf individuelle Gegebenheiten kann für beliebig viele *verschiedene* Nutzer auf die gleiche Weise, jeweils *unter ihrem Namen* eingeloggt, vorgenommen werden.

Wir haben gesehen, daß alle nutzerspezifischen Daten für eine PPP-Verbindung in der nutzerspezifischen Optionsdatei (normalerweise `.ppprc`) und dem zugehörigen Einwahlskript `pppup` liegen. **GateKeeper.app** erlaubt es nun, zusammengehörige Paare solcher Dateien jeweils in einem Ordner mit der Endung **.Gate** zusammenzufassen; ähnlich wie bei dem *Application Wrapper* oder auch den **.rtfd**-Dokumenten in NEXTSTEP erscheint nach der Installation von **GateKeeper.app** ein Ordner mit dieser Endung aber nicht mehr als Ordner, sondern als ein einzelnes (Anwahl-)*Dokument*. Ein Doppelklick auf ein solches Dokument startet die entsprechende PPP-Verbindung; außerdem können die Dokumente über **GateKeeper**'s Werkzeugleiste (*Toolbar*) aufgerufen werden – hierzu müssen sie sich allerdings in einem **GateKeeper.app** bekannten Pfad befinden. Dieser Pfad kann in den Präferenzen beliebig eingestellt werden; hier wird **~/Library/GateKeeper/** vorgeschlagen.

Im **WorkspaceManager** in **~/Library/** durch Drücken von **Command n** einen neuen Ordner erstellen und **GateKeeper** nennen (**~** bezeichnet das eigene bzw. aktuelle Heimverzeichnis). In **~/Library/GateKeeper/** durch Drücken von **Command n** einen neuen Ordner erstellen und ***Name*.Gate** nennen, wobei Name ein beliebiger Name zur Bezeichnung einer bestimmten Verbindung sein kann. Nach Eingabe der Endung **.Gate** ändert der Ordner sein Aussehen und wird zu einem **.Gate**-Dokument:



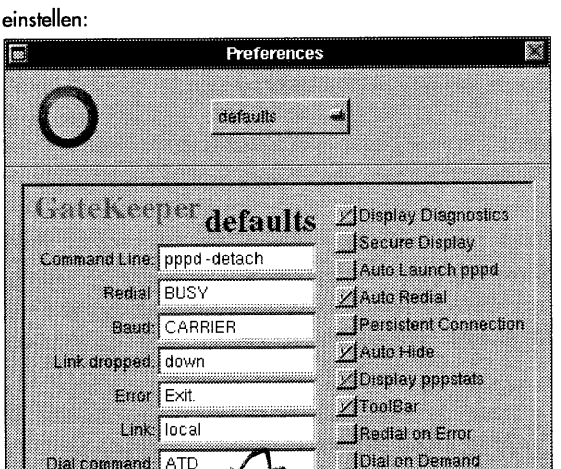
Das **.Gate**-Dokument mit Mausklick anwählen und durch Eingabe von **Command O** in einem eigenen Fenster öffnen. Die ins eigene Heimverzeichnis verschobene/kopierte Datei **~/.ppprc** in das **.Gate**-Dokument verschieben und nach einem Klick auf ihren Namen in `options` umbenennen. **~/pppup** ebenfalls in das **.Gate**-Dokument verschieben. Die `options`-Datei im **.Gate**-Dokument durch einen Doppelklick in **Edit.app** öffnen und den in ihr enthaltenen Pfad von `pppup` so ändern, daß er auf die `pppup`-Datei im **.Gate**-Dokument weist, in unserem Beispiel also:

```
connect "/usr/local/ppp/bin/chat -vf
~/Library/GateKeeper/Name.Gate/pppup"
```

Außerdem kann man in das **.Gate**-Dokument noch ein TIFF-Icon legen und **Icon.tiff** benennen, das dann in der Werkzeugleiste von **GateKeeper** sichtbar wird, wenn dieses **.Gate**-Dokument angewählt ist. Dazu kann man sich z.B. ein TIFF-Bild aus einer seiner WWW-Seiten holen. Das **.Gate**-Dokument-Fenster schließen.

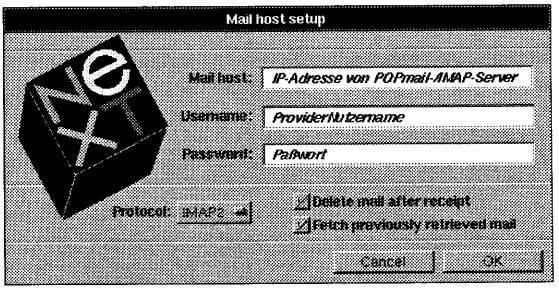
In diesem **.Gate**-Dokument haben wir jetzt die Funktionalität gekapselt, die sonst in der `.ppprc`-Optionsdatei im Heimverzeichnis des jeweiligen Nutzers liegen würde. Der große Vorteil dieser Lösung neben der komfortablen Bedienung ist, daß sich nun auch für *einen* Nutzer beliebig viele *verschiedene* Konfigurationen in entsprechenden **.Gate**-Dokumenten abspeichern lassen. So kann man sich etwa zwei verschiedene **.Gate**-Dokumente für synchronen und asynchronen PPP-Zugang erstellen und wahlweise benutzen.

GateKeeper.app starten und die Präferenzen wie folgt einstellen:



.Gate-Pfad ist dabei der Pfad, in dem die **Gate**-Dokumente liegen, in unserem Vorschlag also **~/Library/GateKeeper/**. **ACHTUNG!** Der Pfad darf **nicht** durch **~** abgekürzt eingegeben, sondern muß voll ausgeschrieben werden! Ist der Schalter *Display Diagnostics* schon angewählt, so muß man ihn einmal aus- und dann wieder einschalten, da durch das Anschalten der FIFO von **GateKeeper.app** erzeugt wird. Bei Verwendung von synchronem PPP mit dem ZyXEL ISDN-Adapter ist das *Dial command* auf ATD**M** zu erweitern, falls die manuelle und somit asynchrone Anwahl, für die es bestimmt ist, von Seiten der Provider nur analog erfolgen kann. Gegegenenfalls muß auch der *tip host* entsprechend der benutzten Schnittstelle anders eingestellt werden. Insbesondere auch auf *pppd path* achten, der mit **/usr/local/bin/ppp/pppd** anders als voreingestellt ist, da wir auf einen Link von `pppd` nach **/usr/local/bin/** verzichtet haben. Nach Ende der Eingaben **Return** drükken, um sie zu übernehmen. Je nach persönlicher Vorliebe kann man die Schalter rechts auch anders einstellen.

PopOver.app starten und den Menüpunkt *Mail Hosts...* anwählen. In dem erscheinenden Listenfenster auf *Add* klikken und in das sich öffnende Eingabefenster folgendes eintragen:



Falls *IMAP2* vom Provider nicht als Protokoll angeboten wird, kann man in der Popup-Einstellung auf *POP3* ausweichen. Auf OK klicken; das Fenster schließt sich und in dem Listenfenster erscheint ***IP-Adresse von POPmail-/IMAP-Server*** als Eintrag. Ein Doppelklick auf dessen Na-

men öffnet das Eingabefenster erneut, so daß später Korrekturen vorgenommen werden können. Nun auch das Listenfenster schließen. Die hier gemachten Einstellungen speichert **PopOver.app** in der nur im UNIX-Modus sichtbaren Datei **.popHosts** im Heimverzeichnis des jeweiligen Nutzers.

Die Präferenzen öffnen und nach Wunsch einstellen. Auf jeden Fall sollte *Retrieve mail on startup* eingeschaltet werden.

Damit sind alle Konfigurationen beendet und wir können die gesamte Installation testen. Um den Versand von Post besser kontrollieren zu können, starten wir wiederum **Terminal.app**, geben in ein geöffnetes Fenster

```
su
```

ein und drücken **Return**. Wir werden jetzt nach einem Paßwort gefragt und geben das **root**-Paßwort an.

Mit dem Befehl `su` (*Setze User*) arbeiten wir in diesem Terminalfenster so, als ob wir **root** wären, ganz vergleichbar dem besonderen Modus, den wir oben bei **GateKeeper.app** eingestellt haben, und haben so Zugriff auf Systemdateien. Das funktioniert allerdings nur, solange wir Mitglied der Gruppe *wheel* sind.

```
tail -f [Leertaste]
```

eingeben und **/etc/spool/mailq/syslog** auf das Terminalfenster ziehen. Wir kriegen den Inhalt der Log-Datei für den Postversand jetzt laufend angezeigt.

Einen Test-Brief an eine bekannte E-Mail-Adresse schreiben und abschicken, während noch keine Verbindung zum Internet hergestellt ist. Im Terminalfenster muß für diesen Brief jetzt die Meldung

```
stat=Deferred: Host Name Lookup Failure
```

erscheinen, die sich der Tatsache verdankt, daß wir ja im Moment mit keinem Nameserver verbunden sind, wo die numerische IP-Adresse des Adressaten nachgeschlagen werden könnte.

Nun können wir die Internet-Verbindung aufbauen, indem wir in **GateKeeper.app** entweder den Menüpunkt *Link* betätigen, ein **.Gate**-Dokument durch Doppelklick starten oder es über die Werkzeugleiste aufrufen. Handelt es sich dabei um die Skripte, die wir zuvor unter **root** ausgetestet haben, muß jetzt auf genau die gleiche Weise eine Verbindung aufgebaut und in dem Protokollfenster von **GateKeeper.app** angezeigt werden. Sobald die IP-Nummern ausgetauscht sind und die Verbindung steht, muß im Icon von **GateKeeper.app** eine Stoppuhr zu laufen beginnen. Im Protokollfenster werden die IP-Nummern nochmals herausgehoben dargestellt:

```
Datum Zeit MeinHostname pppd[PID]: Serial
connection established.
Datum Zeit MeinHostname pppd[PID]: Using
interface ppp0
Datum Zeit MeinHostname pppd[PID]: Connect
: ppp0 <--> /dev/cudfb
Datum Zeit MeinHostname pppd[PID]: Remote
message:
Datum Zeit MeinHostname pppd[PID]: local
IP address 123.456.789.012
Datum Zeit MeinHostname pppd[PID]: remote
IP address 123.456.789.012

#########################
####  local IP address is:
123.456.789.012
#########################
####  remote IP address is:
123.456.789.012
#########################
```

Zugleich sollte **PopOver.app** (und gegebenenfalls ein WWW-Programm) starten und, falls ein netter Zeitgenosse an unsere neue E-Mailadresse testweise schon Post geschickt hat, diese automatisch auf unseren Rechner holen; je nach Einstellung der Präferenzen muß **Mail.app** diese Post dann früher oder später in seinem Icon melden (wird **Mail.app** nicht von **PopOver.app** benachrichtigt, muß sie dazu aber schon gestartet sein; außerdem muß in den Präferenzen von **Mail.app** ein regelmäßiges Abfrageintervall angegeben sein).

In dem offenen Terminalfenster sollte die zwischengelagerte ausgehende Post von vorhin nun bald als versandt gemeldet werden; einen neuen Brief, während der Internet-Verbindung von **Mail.app** losgeschickt, sollte nach kurzer Zeit als zugestellt angezeigt werden.

Mit der Eingabe des Befehls `mailq` in ein Terminalfenster kann man übrigens jederzeit überprüfen, ob noch Post in der Mailqueue auf Zustellung wartet.

Schließlich kann die Verbindung über die Werkzeugleiste oder das *Unlink*-Menü von **GateKeeper.app** wieder beendet werden.

und zum Schluß...

Wahlweise **Netsurfer** oder **Yftp** installieren. Nicht vergessen, in den Präferenzen bei **Netsurfer** (unter der Popup-Einstellung *FTP*) als *Anonymous Login Password* die eigene E-Mail-Adresse anzugeben, sonst verweigern viele FTP-Server den Zugriff!. Bei **Yftp** kann man mit den vorgegebenen Einstellungen arbeiten. In **Netsurfer** lassen sich zusätzlich unter der Popup-Einstellung *Communications* **HTTP-** und **FTP-Proxy** angeben, deren Adresse man, so vorhanden, vom Provider erhalten hat.

Fehlende Programme wie gegebenenfalls einen WWW-Browser (auch **OmniWeb** erlaubt die Angabe eines HTTP-Proxies) oder einen Newsreader (etwa **NewsGrazer** oder **Alexandra**) kann man sich jetzt noch via FTP z.B. vom *Peanuts*-Server (unter /Network/news/) holen. News auf dem eigenen Rechner abzuspeichern ist eine Sache für sich und wird erst in der nächsten **NEXTTOYOU** behandelt. Daher kann man den Newsreader vorläufig nur während einer laufenden Internet-Verbindung benutzen; zu diesem Zweck trägt man unter *nntp server* in den Präferenzen die logische IP-Adresse des **NNTP-Servers** ein, die man vom Provider erhalten hat. Zum Schnuppern reicht das.

Damit wäre es vollbracht. WWW, FTP und E-Mail sollten jetzt funktionieren. Als Lohn der Mühe können wir nun WWW-Seiten lesen, während im Hintergrund neue E-Mail für uns ankommt und wir uns gerade eine Datei via FTP holen. Internet eben.

u.