4 Das Unix Client/Server Program Interface (UCSPI)

4.1 Aufgabe von UCSPI

Die TCP/UDP-"Superdaemons" INETD und XINETD wurde bereits in Abschnitt 2.7 vorgestellt. Erweiterte Kontrollmöglichkeiten sind beim INETD dadurch gegeben, dass nicht sofort der eigentliche Dienst, sondern zunächst ein (in aktuellen Versionen integrierter) sog. TCP-Wrapper gestartet wird.

Im Rahmen der UCSPI-Progarmmfamilie nimmt der **tcpserver** von Dan Bernstein diese Aufgaben wahr, und bietet die Möglichkeiten

- auf (einzelnen) festgelegten TCP-Ports "zu lauschen",
- bei Eintreffen eines TCP SYN-Paketes
  - die *Authentität* des Kommunikationsparterns mittles eines (Reverse-)DNS-Lookup und/oder einer IDENT-Abfrage festzustellen,
  - die *Legitimität* der Anfrage aufgrund einer Einschluss-/Auschlussliste (lokal/remote) zu überprüfen,
  - dem Verbindungswunsch (mit evtl. zusätzlich gesetzten verbindungs-spezifischen Environment-Variablen) zu entsprechen bzw. ihn abzuweisen,
  - den konfigurierten Dienst zu starten, sowie
- eine qualifizierte Log-Information bereitzustellen.

Die eigentliche Problematik bei der Nutzung des INETD/XINETD ist es, dass zur Kommunikation diesen Diensten und dem gestarteten Anwendungsprogramm keine Konventionen existieren. D.h. Informationen über z.B. die eigene IP-Adresse, die IP-Adresse des Clients sowie auch dessen Hostname und Portnummer, können nicht an das zu startende Programm übergeben werden. Als Konsequenz hiervon, muss das Anwendungsprogramm notwendigerweise über ein DNS Stub-Resolver verfügen, der diese Informationen asynchron zu der z.B. des **tcpd** ermittelt, wodurch die Authentität der Information nicht mehr sichergestellt ist.

Genau dies ist aber beim UCSPI-Ansatzes vorgesehen: Hierbei werden dem aufgerufenen Programm eine Anzahl definierter Umgebungsvariablen zur Verfügung gestellt, z.B. die Variable $TCPREMOTEIP und $TCPREMOTEHOST. Qmail entspricht den UCSPI-Konventionen und verfügt mittels des Programms **tcp-env** über einen geeigneten Wrapper, der auch dann

qmail-smtpd über die notwendigen Informationen versorgt, wenn dieses über den **inetd** bzw. **xinetd** gestartet wird. Das Programm **tcpserver** ersetzt jedoch **tcp-env** komplett und hat darüber noch folgende Vorteile:

- **tcpserver** arbeitet Protokollspezifisch und kann mit INETD/XINETD koexistieren, sofern die Programme auf unterschiedlichen Ports konfiguriert sind.
- **tcpserver** ist per konstruktionem relativ immun gegen DoS (Denial-of-Service) Attacken, da im schlimmsten Fall nur der direkt angesprochene Prozess (=Port) blockiert ist (Isolationsprinzip). Im Falle eines DoS-Angriffs gegenüber einem vom INETD/XINETD konfigurierten Ports kann der ganze "Superdaemon" in Mitleidenschaft gezogen werden kann.
- **tcpserver** stellt von sich heraus den aufzurufenden Programmen die notwendigen Verbindungsinformationen in Form von Environment-Variablen zur Verfügung. Zusätzliche Variablen können im **tcpserver**-Startup-Skript explizit gesetzt (exportiert) oder in einer "constant data base" — CDB — definiert werden. Die in der CDB hinterlegten Environment-Variablen lassen sich während der Laufzeit ändern, d.h. dynamisch anpassen, ohne dass ein Neustart von **tcpserver** notwendig ist
- **tcpserver** ist nicht — wie notwendigerweise der **inetd**/**xinetd** — auf den **syslogd** angewiesen, sondern ermöglicht ein wesentlich robusteres Logging, normalerweise über das Programm **multilog** aus der Daemontools-Suite.

4.2 Installation von UCSPI

UCSPI liegt als gepacktes Tar-Archiv vor: `ucspi-tcp-0_88.tar.gz`. Zusätzlich exisitiert eine Sammlung von man-pages von Gerrit Pape: `ucspi-tcp-0.88-man.tar.gz`.

Beide Dateien werden nach `/usr/local/src/` kopiert und ausgepackt.

Hierdurch wird das Verzeichnis `./ucspi-0.88/` erzeugt. In dieses Verzeichnis wird gewechselt und anschliessend "make" und "make setup check" aufgerufen. Die ausführbaren Programme aus dem Paket von UCSPI sind anschliessend im Verzeichnis `/usr/local/bin/` installiert. Will man ein alternatives Verzeichnis wählen (z.B. `/usr/local/sbin/`, kann dies vor dem Aufruf von **make** durch eine Änderung in der Datei `conf-home` vorgenommen werden.

Die Installation der man-Pages geschieht etwas mehr manuell. Nach dem Wechsel in das Verzeichnis `./ucspi-0.88-man/` wird aufgerufen:

```
# gzip *.1 ; cp *.1.gz /usr/share/man/man1/
```

Damit werden die Quelldateien "gezipped" und in das Verzeichnis

`/usr/share/man/man1/` gestellt.

4.3 Dienste und Programme von UCSPI

Dan Bernstein hat in der aktuellen Version von UCSPI einige nützliche Programme zusammengestellt, von denen sicherlich **tcpserver** das wichtigste ist.

Server-Programme:

- **tcpserver** - Konfigurierbarer Port-Listener (Server) und Programm-Starter.
- **tcprules** - erzeugt die CDB mit einem Regelwerk für den **tcpserver,**
- **tcprulescheck** - überprüft das tcpserver Regelwerk.
- **rblsmtpd** - "Realtime Black List - SMTPD" verbindet sich zu einer Datenbank im Internet und überprüft, ob die Verbindungsaufnahme auf Port 25 (SMTP) von einem dort eingetragenen Rechnern stattfindet. Diese Information kann genutzt werden, die Verbindungsaufnahme zu steuern bzw. abzulehnen.

Client-Programm:

- **tcpclient** - TCP-Verbindungserzeuger.

Die @-Hilfsprogramme:

- **who@** - Ausgabe der aktiven Benutzer auf einem entfernten Recher (vgl. who).
- **date@** - Ausgabe des Zeitstempels eines entfernten Rechners.
- **finger@** - Ausgabe der Benutzerinformation von einem entfernten Rechner.
- **http@** - Holt eine Web-Seite von einem entfernten Rechner (unter Entfernung von CR-Zeichen).
- **mconnect** - Verbindungsaufnahme zu einem entfernten Rechner auf einem konfigurierten Port (Default: Port 25 / SMTP).
- **tcpcat** - Ausgabe der TCP-Verbindungsinformationen.

Weitere Werkzeuge:

- **argv0** - Starten eines Programms unter Übergabe der Parameter 1 bis N.
- **addcr** - Fügt im Datenstrom einen fehlenden CR vor dem LF hinzu.
- **delcr** - Entfernt einen CR vor dem LF im Datenstrom.
- **fixcrio** - Fügt fehlende CR-Sequenzen beim Einlesen von Zeilen hinzu.

Tracing-Programm:

- **recordio** - Schreibt in einem zweiten Datenstrom Ein- und Ausgabe eines Programms mit.

4.4 UCSPI-“Piping“

Zentraler Ansatz von UCSPI ist es, als Vermittler zwischen der Netzwerkschicht und der Applikation zu fungieren. UCSPI-Programme beinhalten somit entweder einen TCP-Server oder einen TCP-Client. Sie arbeiten daher auch unabhängig davon, ob das Schicht-3-Protokoll IPv4 oder IPv6 ist (zumindest weitgehend).

tcpserver liest die Daten vom bzw. schreibt die Daten zum Netzwerk über die Socket-Schnittstelle. Die Kommunikation zwischen **tcpserver** und Anwendung wird dagegen über eine Unix-Pipe zwischen beiden Prozessen realisiert. Das Anwendungsprogramm liest die Daten vom **tcpserver** über den File-Descriptor (FD) 0 und schreibt sie über den FD 1. Dies gestattet eine zuverlässige Vollduplex-Kommunikation zwischen der Netzwerkschicht und der Anwendung (Abbildung 4-1a).

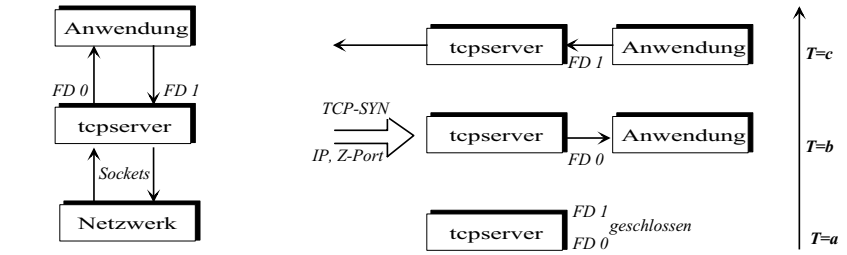

*Abbildung 4-1: Arbeitsweise von **tcpserver**; a) File-Descriptoren; b) dynamisches Verhalten von **tcpserver**.*

Die zeitlichen Abhängigkeiten demonstriert Abbildung 4-1b. Zum Zeitpunkt *T=a* wartet **tcpserver** bei geschlossenen File-Descriptoren *0* und *1* auf ein einlaufendes TCP-SYN-Segment. Die Anwendung ist hierbei noch nicht gestartet. Läuft zum Zeitpunkt *T=b* von einem Client ein SYN-Segment auf dem gehörigen Port (Z-Port) ein, wird File-Descriptor *0* und *1* geöffnet und über einen `fork()`-Aufruf die Anwendung gestartet (spawned) sowie der Datenstrom an die Anwendung weiter gereicht. Liegen zum Zeitpunkt *T=c* Anwendungsdaten an den Client vor, wird der geöffnete File-Descriptor *1* zum Schreiben benutzt.

Hierdurch wird der einlaufende Datenstrom zwischen Netzwerk und Applikation entkoppelt und die ein- und auslaufenden Daten gepuffert. Auch durch die Beendigung des Anwendungsprozesses, beispielsweise durch ein TERM bzw.

SIGTERM, besteht die Datenverbindung zwischen Client und dem **tcpserver** weiter fort.

Die UCSPI-Progamme entlasten daher das gestartete Anwendungsprogramm (als Client oder wie dargestellt als Server) und machen es robust gegenüber eventuellen Netzproblemen oder Attacken. Dies steht der herkömmlichen Arbeitsweise von INETD/XINETD diametral gegenüber: Mit dem Aufbau der Verbindung wird die gestartete Anwendung verantwortlich für den weiteren Ablauf der Kommunikation gemacht.

4.5 Environment-Variablen

Der schichtenspezifische Aufbau der Netzprotokolle bringt es mit sich, dass eine Anwendung sich in der Regel nicht für Informationen darunterliegender Netzwerkschichten interessiert, sondern nur für ihren Peer-Partner. Diese reine Lehre gilt für das Siebenschichten-OSI-Referenzmodell und hat schon manche OSI-Anwendung enorm verkompliziert, weil im Zusammenspiel der Schichten mit Bindungen und einem aufwändigen Name-Lookup der Instanzen-Namen gearbeitet werden musste.

Im vereinfachten TCP/IP Vierschichten-Modell macht es jedoch demgegenüber Sinn, dass auch die Anwendung eine weitgehend vollständige Kenntnis über ihren Kommunikationspartner erhält. UCSPI ermöglicht dies, indem die wichtigsten Parameter in Form von Environment-Variablen übermittelt werden. Hierzu sind natürlich drei Voraussetzungen zu schaffen:

1. Die Informationen müssen ermittelt werden können. Verfügbar sind immer Quell- und Zielport, sowie die IP-Adresse des Clients (Quellrechner) und des Server (Zielsystems). Es ist zu bedenken, dass durch eine Network-Address-Translation (NAT) bzw. einen FireWall, nicht die ursprünglichen Informationen bereit stehen, sondern modifizierte.

2. Es muss ein Programm vorliegen, dass diese Informationen zusammenstellt und an die Serveranwendung weiter leite – in unserem Falle **tcpserver**.

3. Die Zielanwendung (Client oder Server) muss diese Informationen auslesen und ggf. weiterverarbeiten können.

Im Rahmen des UCSPI-Ansatzes werden die Netzwerk-Informationen des lokalen Rechners und die des Peer-Partners in Form folgender Environment-Variablen bereit gestellt:

- `$PROTO` - Zeichenkette TCP.
- `$TCPLOCALIP` - lokale IP-Adresse (Dotted-Dezimal).
- `$TCPLOCALPORT` - lokale TCP Port (Dezimal)

- `$TCPLOCALHOST` - Name des lokalen Rechners entsprechend seinem DNS-Eintrag. Liegt dieser nicht vor, ist $TCPLOCALHOST nicht gesetzt.
- `$TCPREMOTEIP` - IP-Address des Peer-Rechners (Dottted-Dezimal).
- `$TCPREMOTEPORT` - (dezimale) Port-Nummer der TCP-Peer-Instanz.
- `$TCPREMOTEHOST` - Name des entfernten Rechners im DNS; sollte dieser nicht ermittelt sein, ist $TCPREMOTEHOST nicht gesetzt.
- `$TCPREMOTEINFO` - zusätzliche verbindungsspezifische Informationen über den Peer-Partner, die über das 931/1413/IDENT/TAP Protokoll ermittelt werden (ansonsten ist `$TCPREMOTEINFO` nicht gesetzt).

Sowohl der **tcpserver** als auch der **tcpclient** verfügen daher über einen DNS-Stub-Resolver, mit der Hostnamen aufgelöst werden, als auch über einen entsprechend IDENT/TAP-Client. Zur Nutzung dieser Information ist jedoch erforderlich, dass (beim Einsatz des **tcpserver)** die Zielanwendung (der Serverprozess) sich auf die übermittelten Informationen stützt und keinen eigenen Lookup vornimmt. Hierzu gibt es einen "klassischen" Dialog zwischen Wietse Venema (dem Author des TCP-Wrappers **tcpd**) und Dan Bernstein, in dem Dan zeigt, dass durch die quasi asynchrone Nutzung von DNS-Information Adress-Spoofing z.B. beim **rlogin** möglich sind (`http://cert.uni-stuttgart.de/archive/bugtraq/1998/11/msg00133.html`)

4.6 Einsatz von tcpserver

Für die Nutzung von Qmail und anderer, TCP-basierender Anwendungsprogramme ist das Programm **tcpserver** der wichtigste Baustein innerhalb der UCSPI-Programmfamilie. Hat Dan Bernstein in der usprünglichen Qmail-Dokumentation noch Bezug auf den INET-Daemon (INETD) genommen, so empfiehlt er — wie auch die Mehrzahl der Qmail-Anwender — den Einsatz von **tcpserver** im Zusammenhang mit `qmail-smtpd`. Da die Probleme des INETD schon seit Jahren bekannt sind, habe ich bereits frühzeitig statt des INETD den XINETD eingesetzt, der umfangreichere Kontrollmöglichkeiten und ein wesentlich besseres Logging bietet.

Für Qmail ist allerdings **tcpserver** die erste Wahl. Dieser kombiniert viele Sicherheitsfeatures und macht zudem einige Hilfsprogramme überflüssig, die ansonsten separat herangezogen werden müssen.

- **tcpserver** ersetzt das Qmail-Programm **tcp-env**, da nun **tcpserver** selbst die notwendigen Environment-Variablen einstellt.
- **tcpserver** schützt die Rechner-Ressourcen, da er verhindert, dass gleichzeitig zu viele TCP-Verbindungen simultan aufgebaut werden (und in Konsquenz die Serveranwendung zu häufig gestartet wird).

- **tcpserver** schreibt bei Bedarf seine Aktivität auf den File-Descriptors *2* (Standard-Error) und ermöglicht somit ein wesentlich robusteres Logging unter Verzicht auf den Syslog-Daemon (**syslogd**).
- **tcpserver** nimmt optional einen IDENT/TAP sowie DNS-Lookup des Verbindungsparteners vor. Diese Funktionalität müsste ansonsten über z.B. den **tcpd** emuliert werden.
- **tcpserver** verfügt über eine Laufzeit-konfigurierbare Datenbank, in der IP-Adressen-spezifisch das Verbindungsverhalten (akzeptiert oder abgelehnt) hinterlegt werden kann. Ferner ist hierüber das Setzen zusätzlicher Environment-Variablen möglich.
- **tcpserver** ermöglicht die Einbindung des Programms **rblsmtp**, mittels dessen Informationen (externer) Realtime-Blacklists (RBL) verarbeitet werden können.

Bei allen diesen Vorteilen, sind jedoch einige Eigenheiten von **tcpserver** nicht zu vergessen:

- **tcpserver** muss für jede Anwendung, die diesen Dienst nutzen will, individuell konfiguriert werden; eine zentrale Konfigurationsdatei à la `/etc/ined.conf` oder `/etc/xinetd.conf` ist nicht vorgesehen.
- **tcpserver** verlangt, dass sich — bedingt durch den Piping-Mechanismus — die Serveranwendung nach Aufruf in den Hintergrund stellt. Dies kann z.B. beim **ftpd** durch Angabe des Ampersand "&" Zeichens geschehen. Auf der anderen Seite schliesst dies aus, das sich die Serveranwendung nach einmaligem Aufruf als Daeomon-Prozess instanziiert (wie z.B. der Apache **httpd**).
- Erfolg eine Verbindungsaufname auf dem für **tcpserver** spezifizierten Port, wird vom ursprünglichen Prozesse über einen `fork()`-Aufruf eine neue Instanz von **tcpserver** (+ Serveranwendung) gestartet. Die Ausgabe der Prozessausgabe (**ps**) erscheint damit aber leicht unübersichtlich, speziell, da auch diese Kommandozeilen-Parameter mit ausgegeben, bzw. bei einer Standard 80-Zeichen-Ausgabe abgeschnitten werden. Andererseits kann durch ein entsprechendes **ps** festgestellt werden, welche Client-Rechner per TCP mit der über den **tcpserver** gestarteten Serveranwendung kommunizieren, was für Tracing-Zwecke sehr hilfreich ist.

4.6.1 tcpserver Syntax

Der typische Aufruf von **tcpserver** umfasst eine Anzahl von Argumenten, deren Verwendung sich in mehrere Bereiche aufteilen lässt (Abildung 4-2):

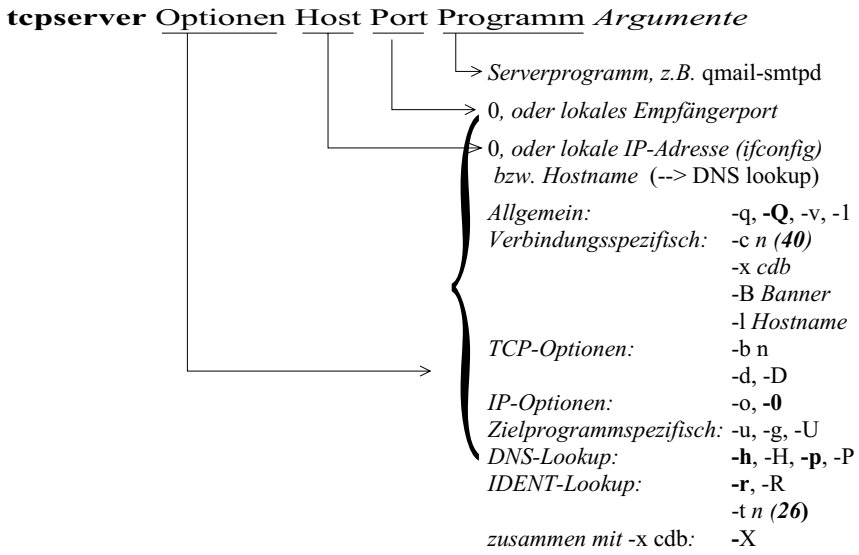

*Abbildung 4-2: Argumente und Optionen des **tcpserver** Pogramms; Defaultwerte sind in Fettschrift dargestellt.*

Die Angabe der Optionen erfolgen hierbei (wie auch bei den meisten Dan Bernstein Programmen) im sog. "getopts" Stil. Hierbei ist es egal, ob die Optionen einzeln, also z.B. `-v -Q -x cdb`, oder als gemeinsame Zeichenkette `-vQx cdb` übergeben werden. Ebenso wenig spielt die Reihenfolge eine Rolle, oder ob die Angabe der Option `-x cdb` oder `-xcdb`, oder eventuell `-x "cdb"` lautet. In den Beispielen habe ich der grösseren Klarheit wegen, die Optionen getrennt aufgeführt; empfehlenswert ist die allemal bei denjenigen Optionen, die mit einem zusätzlichen Parameter versehen werden, z.B. `-x cdb`.

Der wichtigste Parameter für **tcpserver** ist die Angabe des *Serverprogramms*. Hierbei sind folgende Umstände zu beachten:

- Sollte das Programm nicht im Default-Pfad vorliegen, muss dieser explizit mit angegeben werden, z.B. **/var/qmail/bin/qmail-smtpd**, was einen wesentlichen Sicherheitsaspekt darstellt.
- Das Serverprogramm lässt sich mit einer beliebigen Anzahl von *Argumenten* aufrufen, die transparent durchgereicht werden.
- Nutzt das aufrufende Programm neben dem File-Descriptor *1* noch weitere — wie STD-Error (File-Descriptor *2*) — und soll deren Ausgabe z.B. im Logfile mit erscheinen, sind diese auf den File-Descriptor *1* umzuleiten (

`2>&1`).

- Durch die Angabe des "&" Zeichens wird der **tcpserver**-Prozess anschliessend in den Hintergrund gestellt.

Der Parameter *Host* ist für sog. Multi-Homing Rechner interssant. Hat ein Rechner mehrere IP-Adressen zugeteilt, kann **tcpserver** dazu veranlasst werden, auf lediglich einer (der) konfigurierten IP-Adresse(n) zu lauschen.

- Wird der Parameter auf den Wert "0" gesetzt, fühlt sich **tcpserver** für alle lokalen Adressen verantwortlich.
- Andererseits kann eine per **ifconfig** definierte IP-Adresse eingesetzt werden, oder aber der Hostname, wobei dann die Zuordnung zur IP-Adresse über ein DNS-Lookup erfolgt und nur die erste IP-Adresse herangezogen wird.

Mit der Angabe *Port* als Parameter bei **tcpserver** wird festgehalten, über welchen (TCP-) Port das Programm einen Verbindungswunsch entgegen nimmt. Ist der Wert auf "`0`" gesetzt, werden alle freien Ports berücksichtigt. Statt der Portnummer kann auch der in der Datei `/etc/services` hinterlegt Portname genutzt werden (z.B. smtp statt 25).

In jedem Fall ist zu beachten, dass immer nur ein Programm pro IP-Adresse mit einem Port gebunden sein kann. Ein typischer (aber trivialer) Fehler beim Start eines Programmes über **tcpserver** ist die Ausgabe "fatal: unable to bind: address already used ". Dies bedeutet, dass entweder das Serverprogramm schon gestartet ist (z.B. als Daemon) oder der **inetd**/**xinetd** bzw. **tcpserver** noch für diesen Port aktiv sind.

Die Angabe von *Optionen* beim Aufruf **tcpserver** ist in der Tat optional, in der Regel sind sinvolle Defaults gewählt; wobei eine Ausnahme zu verzeichnen ist. Doch wir wollen systematisch vorgehen.

- Die *allgemeinen Optionen* legen fest, ob (`-Q`) oder ob nicht (`-q`) Fehlermeldungen, bzw. ob Status- und Fehlermeldungen ausgegeben werden sollen (`-v`). In der Regel wollen wir letzteres. Es ist obige Bemerkung zur Behandlung der File-Descriptoren (STD-Error) zu berücksichtigen. Bei Angabe von `-1` wird nach dem Start von **tcpserver** die lokal gebundene Portnummer ausgegeben; hier wäre die zusätzliche Angabe der IP-Adresse hilfreich gewesen.
- Die *verbindungsspezifischen Optionen* bestimmen, wie viele simultane Instanzen von **tcpsever** für diesen Dienst gestartet werden dürfen (`-c` Anzahl; Default 40), andererseites kann die Verbindungsaufnahme über die Angabe einer Datenbank (`-x` *`cdb`*) fein konfiguriert werden. Auf dieses Verfahren wollen wir weiter unten eingehen.
  Wird die Datenbank mit der Option `-X` zusätzlich zur Angabe des "kleinen" `-x` *`cdb`* aufgerufen, so wird sichgestellt, dass bei nicht vorhandener CDB

der Verbindungswunsch akzeptiert wird, was ansonsten eine Ablehnung bewirkt.
Notwendig ist manchmal die Angabe eines sog. Banners unmittelbar nach der Verbindungsaufnahme. Manche Clients erwarten bei Verbindungsaufnahme eine sofortige Protokoll-Begrüssungsformel, die z.B. beim SMTP-Protokoll "HELO" oder "EHLO" lautet. Auf diese Notwendigkeit wollen wir weiter unten eingehen.
Schliesslich kann durch Angabe der Option `-l` *`Hostname`* **tcpserver** der lokale Name des Rechners vorgegeben werden, dessen Ermittlung ansonsten über einen DNS-Lookup geschieht. Wir beachten, dass dies ggf. bei Multi-Homed Rechnern angeraten ist und falls **tcpserver** nicht auf dem Default-Interface binden soll.

- Die TCP-spezifischen Optionen bestimmen, ob und wie **tcpserver** die Informationen der TCP *Round Trip Time* (RTT) auswerten und gebrauchen soll. Mittels der Option `-d` wird **tcpserver** dazu gebracht, die Aussendung neuer Daten um den Bruchteil einer Sekunde zu verzögern, während mit `-D` die sofortige Aussendung bewirkt wird.
  Ein weiterer Mechanismus wird mit der Option `-b` *`N`* zur Verfügung gestellt. *N* stellt hierbei die Anzahl der TCP-SYNs dar, die akzeptiert und von **tcpserver** gepuffert werden. Die Anwendung dieser Option hängt in erster Line von der TCP-Implementierung des lokalen Rechners ab, insbesonders davon, ob TCP-SYN-Cookies (Keep-Alives) verarbeitet werden. Ist dies der Fall, hat diese Option keine Wirkung. Bei veralteten TCP-Stacks kann dies jedoch genutzt werden, die Akzeptanz u.U. vieler einlaufender TCP-SYNs zu gewährleisten.

- Mit den *IP-spezifischen Optionen* kann auf das IP-Source-Routing Einfluss genommen werden. Per Default werden IP-Source-Routing Pakete auf der Source-Route zurückgeschickt (`-o`). Durch Setzen der Option `-O` werden die Pakete über die Defaultroute versandt.

- Die *Serverprogramm-spezifischen Optionen* `-u` *`UID`*, `-g` *`GID`* und `-U` geben an, mit welcher effektiven User-ID und Group-ID das Anwendungsprogramm gestartet wird. Bei Nutzung der Option `-U` müssen dies Variablem UID und GID ins Environment gestellt sein, typischerweise mittels **envguid** aus den Daemontools.

Die Optionen zur Ermittlung der Eigenschaften des TCP-Peer-Partners (*Data-Gathering Options*) dienen dazu, die UCSPI Environment-Variablen zu befüllen und diese Informationen dem Serverprogramm zur Verfügung zu stellen. Wir berücksichtigen, dass dies einen schwerwiegenden Eingriff in die Arbeitsweise von **tcpserver** darstellt, was aus Abbildung 4-3 entnommen werden kann. Wird der Verbindungswunsch zum Zeitpunkt *T=a* initiiert, so erfolgt zum Zeitpunkt *T=b* eine IDENT/TAP-Abfrage bzw. ein DNS-Lookup. Deren Antworten treffen erst zu einem viel späteren Zeitpunkt ein. Im Beispiel ist es der Zeitpunkt *T=c*, zu

der die DNS-Abfrage aufgelöst wurde und (nochmals viel später) zum Zeitpunkt *T=d*, an dem die IDENT-Abfrage auf einen Timout läuft, der üblicherweise bei 90 Sekunden liegt. Erst anschliessend — also zum Zeitpunkt *T=e* — beginnt die Serveranwendung damit, Nutzdaten an die Client-Applikation zu verschicken; beim SMTP-Protokoll z.B. durch die Aussendung der Protokollbegrüssung HELLO.

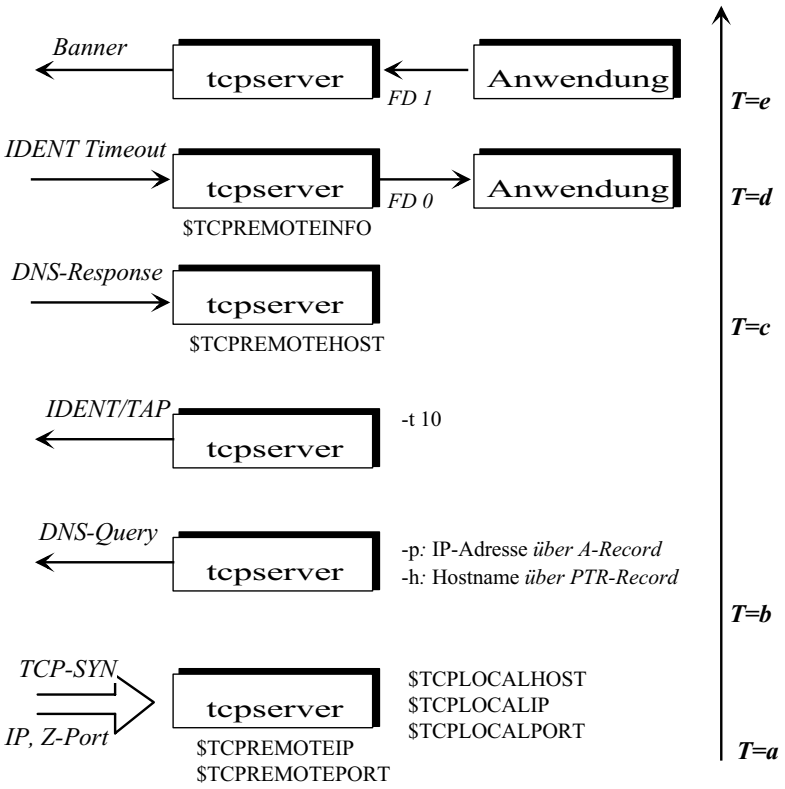

*Abbildung 4-3: Einfluss der Data-Gathering Optionen auf die Arbeitsweise von **tcpserver**. Das Serverprogramm wird erst gestartet, nachdem die Informationen über das IDENT/TAP Protokoll bzw. der DNS-Query (erfolgreich) abgeschlossen wurde.*

- Die *DNS-spezifischen Optionen* lauten `-h` (DNS-Lookup) und `-H` (kein DNS-Lookup). Aufgrund der festgestellten IP-Adresse des Verbindungspartners, erfolgt zuächst über ein PTR-Query (inverse Lookup) die Ermittlung des bzw. der Hostname(n), wobei der primäre Hostname anschliessend in der Environment-Variable `$TCPREMOTEHOST` abgelegt wird.

Zusätzlich kann auch ein DNS-Lookup des A-Records vorgenommen werden. Hierzu dient die Option `-p` (Paranoid), wobei der aus dem DNS-PTR-Record ermittelte Name, der in `$TCPREMOTEHOST` gespreichert ist, mit der im DNS hinterlegten A-Record (den für den entfernten Rechner zugeteilten IP-Adressen) zusätzlich abgeglichen wird. Findet sich keine Übereinstimmung der so ermittelten IP-Adressen mit der in `$TCPREMOTEIP`, wird die Variable `$TCPREMOTEHOST` wieder gelöscht. Hierdurch kann das Spoofing von Hostnamen reduziert werden.
Das Defaultverhalten von **tcpserver** ist jedoch `-P`, d.h. kein zusätzlicher DNS-A-Record-Lookup.

Die *IDENT-Lookup Option* sollte in der Regel immer auf `-R` gesetzt sein, d.h. kein Lookup; im Gegensatz zum Defaultwert `-r`. Grund hierfür ist, dass der IDENT-Lookups in der Regel von einer FireWall blockiert werden und **tcpserver** somit immer auf einen Timeout läuft. Eine Ausnahme hierzu bildet ein weitgehend homogenes Unix-Environment in einem lokalen Netz. Dann ist es möglich, für die einzelnen Rechner den IDENT-Server entsprechend zu konfigurieren, sodass eine Auswertung dieser Informationen einen Sicherheitsgewinn bringen kann. Der Standard-Timeout beim IDENT-Protokoll beträgt 90 Sekunden; **tcpserver** verkürzt diese Zeit auf 26 Sekunden, wobei sich dies über die Option `-t` individuell angepassen lässt.

4.6.2 Basiskonfiguration

Der Aufruf von tcpserver erfolgt in der Regel über ein Shell-Skript. Wir wollen dieses Skript generisch halten und definieren einige Variablennamen:

```
#!/bin/sh
HOSTNAME=‘/bin/hostname‘
#HOSTIP=‘/bin/hostname -i‘ # not supported on all UNIXe
UID=‘id -u root‘
GID=‘id -g root‘
PORT=30
exec /usr/local/bin/tcpserver -1 -v -H -R \
      -u$UID -g$GID -l$HOSTNAME 192.168.192.2 $PORT \
      /usr/libexec/telnetd 2>&1 | \
      /var/qmail/bin/splogger telnetd &
```

*Listing 4-6: Generisches Beispielskript zum Aufruf von **tcpserver***

Dieses Skript soll den Namen "telnetd.run" tragen. Anschliessend muss aus ausführbar gemacht werden, also: `chmod +x telnetd.run.`

Was haben wir damit bezweckt? Zunächst erfolgt der Aufruf der Shell. Anschliessend setzen wir die für die Arbeit von **tcpserver** notwendigen Parameter, wie Hostname und Port, zu dem es gebunden sein soll, sowie die effektive UID und GID während der Ausführung. Beim Hostname haben wir es uns bequem gemacht. Wir rufen einfach das Unix-Kommando **hostname** auf und besetzen damit die Variable HOSTNAME. Wir hätten auch hier die IP-Adresse ermitteln können, doch das wird nicht von allen Unix-System unterstützt. Der Vorteil der Vergabe eines initialen Hostname (Option `-l`) ist, dass hierdurch **tcpserver** veranlasst wird, diesen Namen heranzuziehen, statt bei jedem Start den eigenen Hostname über einen DNS-Lookup aus der angegebenen IP-Adresse zu ermitteln; das Verfahren bietet sich insbesondere bei Multihomed-Rechnern zur expliziten Angabe des Hostname an.

Die gleiche Bequemlichkeit legen wir an den Tag, beim Ermitteln der UID und GID. Die so gesetzten Parameter (UID, GID, HOSTNAME etc.) haben nur Bedeutung für die lokale Shell. Sollten diese Paramter auch dem Anwendungsprogramm zur Verfügung gestellt werden, müssen sie export werden. Jede Änderung dieser Parameter hat darüber hinaus zur Folge, dass das Skript beendet und neu gestartet werden muss; es sind also quasi *statische Parameter*.

Als nächstes erfolgt der Aufruf von **tcpserver** mit den Optionen. Wir bemerken den Ausdurck `exec /usr/local/bin/tcpserver`. Hiermit wird (notwendigerweise die laufende Shell durch das Kommando **tcpserver** substituiert, andereseits braucht bei der vollständigen Pfadangabe für die weiteren Kommandos kein Suchpfad gesetzt werden. Das macht zwar die Kommandozeile länger, aber wir wissen nun ganz genau, welches Kommando aufgerufen wird.

Die Optionen haben wir mit `-1 -v -H -R` gewählt, was wir bereits kennen. Anschliessend erfolgt die Angabe (mittles `-l`) welchen Hostname **tcpserver** benutzt auf und welchem Port er gebunden ist. Zu Testzwecken haben wir hier Port 30 bestimmt.

Dann wird das eigentliche Programm, der Telnet-Daemon, aufgerufen. Wir bemerken, dass der STD-Error auf STD-Ausgabe umgelenkt ist (`2>&1`). Dieser Output wird in das Qmail-Programm **splogger** gepiped und alles in den Hintergrund geschickt (`&`), damit die Ausgabe im Standard-Syslog festgehalten wird. Würden wir darauf verzichten, erfolgte die Fehlerausgabe auf dem Terminal, von dem aus der Aufruf von `./telnetd.run` vorgenommen wurde.

Zu Testzwecken lassen wir den Aufruf von **splogger** heraus (`/usr/libexec/telnetd 2>&1 &`), starten zunächst das Skript `./telnetd.run` und anschliessend zweimal einen Telnet-Client, der sich auf Port 30 unseres Testrechners verbindet und schauen uns die Ausgabe vom

tcpserver an:

```
# ./telnetd.run
30
tcpserver: status: 0/40
tcpserver: status: 1/40
tcpserver: pid 49776 from 192.168.192.11
tcpserver: ok 49776 qmailer.fehnet.de:192.168.192.2:30
:192.168.192.11::1067
tcpserver: status: 2/40
tcpserver: pid 49778 from 192.168.192.11
```

Es erfolgt zunächst die Ausgabe der Zahl "30" zu lesen. Dies ist die Portnummer auf die **tcpserver** bindet (Option `-1`). Es folgt die eigentliche Logging-Ausgabe von **tcpserserver**. Eingeleitet wird dies durch die "status" Zeile, die uns die Anzahl der aktiven und der maximalen Verbindungen sagt (`status: 1/40`). Im weiteren sehen wir die durch die Option `-v` getriggerte Ausgabe der Verbindungsformation unter Angabe der PID, der IP-Adresse des Clients sowie anschliessend unseren lokalen Hostname mit den TCP-Bindungsinformationn.

Wenn wir den Telnet-Client beenden, meldet dies **tcpserver** im Log mit `tcpserver: end 49778 status 256`. Status 256 wird ausgegeben, wenn das Serverprogramm — n unserem Fall der **telnetd** — mit einem Exit-Code ungleich Null zurückkommt. Normalerweise ein Hinweis auf ein Problem; in diesem Fall aber typisch.

tcpserver Status-Codes

Bei Beendigung der Serveranwendung meldet dies **tcpserver** unter Angabe des Pid des Anwendungsprogramms sowie unter Angabe des Exit-Codes, in der From `255 + RC`, wobei `RC` für den Exit-Code des Anwendungsprogramms steht. Interne Fehler von **tcpserver** werden immer unter Ausgabe des Exit-Codes 111 und einer beschreibenden Meldung mitgeteilt.

Für führen unsere Studien weiter, beenden alle offenen Telnet-Sitzungen auf Port 30 und fügen die Zeile mit dem Piping nach **splogger** wieder ein. Nach Start des Skriptes `./telnetd.run` erfolgt keine Ausgabe mehr auf dem lokalen Terminal. Gut so. Wir sehen uns zunächt die Prozesstabelle an (in verkürzter Form):

```
# ps -aux
...
root   49775  .... /usr/local/bin/tcpserver -1 -v -H -R -u0
-g0 qmailer.fehnet.de 30 /usr/libexec/telnetd
```

Komisch, die **tcpserver** PID ist eindeutig niedriger als die oben. Dann erinnern wir uns: Wir haben zwar das Skript `./telnetd.run` neu gestartet, aber den

alten **tcpserver** Prozess nicht beendet. Was wir sehen, ist der alte Prozesse. Der neue **tcpserver** konnte nicht gestartet werden, weil der alte noch auf dem Port 30 gebunden war. Versuchsweise rufen wir noch einmal den Telnet-Client auf und tatsächlich findet sich dort wieder die Ausgabe des **tcpserver** Logs.

Nach dem Beenden des alten **tcpserver** Prozesses starten wir `./telnetd.run` erneut und wechseln ins Verzeichnis `/var/log/.` Dort sollten die Meldungen per **splogger** in der Logdatei `messages` eingestellt sein. Wir machen ein **tail** auf die Datei, aber es ist kein Eintrag zu finden. Anschliessend schauen wir mit einem `ls -la` nach, welche Datei überhaupt als letzte geändert wurde. Merkwürdig, das ist die Datei `maillog`. Ein Blick hinein und wir erkennen:

```
splogger: 1017826125.731475 tcpserver: fatal: unable to
bind: address already used

splogger: 1017826436.356183 30

splogger: 1017826436.357629 tcpserver: status: 0/40

splogger: 1017826588.747314 tcpserver: status: 1/40

splogger: 1017826588.749559 tcpserver: pid 49828 from
192.168.192.11

splogger: 1017826588.751200 tcpserver: ok 49828
qmailer.fehnet.de:192.168.192.2:30 :192.168.192.11::1075

splogger: 1017826600.248162 tcpserver: end 49828 status 256

splogger: 1017826600.249056 tcpserver: status: 0/40
```

Da ist also unser zweiter Aufruf von `./telnetd.run` zu sehen. Aber warum ausgerechnet im Maillog? Grund hierfür ist die sog. "Logging-Facility" vom **syslogd** (`man 5 syslog.conf`). Per Default verwendet **splogger** hier "mail", was bei den meisten Unix-Systemen dazu führt, dass diese Ausgabe in eine spezielle, dem Mailverkehr zugewiesene Logdatei gelangt. Auf die genaue Struktur des **splogger** Logformats soll aber später eingegangen werden.

Bleibt abschliessend zu klären, warum die Option `-r`, also der IDENT-Lookup, prinzipiell abzuschalten ist. Aus Abbildung 4-3 geht bereits hervor, dass diese Abfrage viel Zeit kosten. Zweitens ist häufig beim Einsatz von Firewalls das IDENT-Port (UDP/TCP) 113 nicht freigeschaltet, so dass die Abfrage zwangsläufig auf einen Timeout läuft. Zum dritten ist die Information, die der IDENT-Server zur Verfügung stellt, in der Regel wenig aussagekräftig.

Beim Superdaemon **inetd** ist z.B. der IDENT-Dienst normalerweise mit eingebunden, beim **xinetd** muss dieser zusätzlich konfiguriert werden. In unserem Skript `telnetd.run` habe ich mal versuchsweise die Option `-r` gesetzt und vergleiche die Einträge im Logfile (zusätzliche Informationen sind unterstrichen):

```
splogger: 1017836595.945145 tcpserver: status: 1/40
```

```
splogger: 1017836595.947051 tcpserver: pid 50051 from
192.168.192.1

splogger: 1017836595.966610 tcpserver: ok 50051
qmailer.fehnet.de:192.168.192.2:30
:192.168.192.1:inaxyd:3391

splogger: 1017836632.161289 tcpserver: end 50051 status 256

splogger: 1017836632.162137 tcpserver: status: 0/40

splogger: 1017836959.470044 tcpserver: status: 1/40

splogger: 1017836959.472185 tcpserver: pid 50057 from
192.168.192.1

splogger: 1017836959.492383 tcpserver: ok 50057
qmailer.fehnet.de:192.168.192.2:30 :192.168.192.1:root:3392

splogger: 1017836964.655368 tcpserver: end 50057 status 256

splogger: 1017836964.656248 tcpserver: status: 0/40

splogger: 1017837194.321318 tcpserver: status: 1/40

splogger: 1017837194.323449 tcpserver: pid 50062 from
192.168.192.1

splogger: 1017837194.343851 tcpserver: ok 50062
qmailer.fehnet.de:192.168.192.2:30 :192.168.192.1:erwin:3393

splogger: 1017837902.602329 tcpserver: end 50062 status 256

splogger: 1017837902.603232 tcpserver: status: 0/40
```

Die Ausgabe ist uns nun schon vertrauter. Tatsächlich sehen wir in den Logs einen Eintrag — von unten nach oben — einmal "erwin" und einmal "root". Im ersten Fall habe ich die Telnet-Sitzung auf Port 30 als User "erwin", beim vorherigen Mal als User "root" gestartet. Das ist die Zusatzinformation des IDENT-Protokoll (*Authentication*). Ganz am Anfang findet sich aber eine Zeichenkette "inaxyd". Hier ist der **identd** in einem "Zufallsmode" gelaufen, d.h. er hat statt der korrekten User-Angabe zufällige Zeichenketten gebildet und diese an den **tcpserver** weiter gereicht. Je nach **identd** Implementieung, ist dieser String eindeutig für die unter Unix zugehörenden UID des Users. Eine Garantie hierfür gibt es allerdings nicht, sodass diese Information wenig Aussagekräftig ist, ausser dass wir wissen, dass auf dem Client-System der **identd** läuft.

4.6.3 tcpserver als TCP-Wrapper

Der Zugriff von Client-Systemen auf unsere Rechnerressourcen kann über den **tcpserver** fein gesteuert werden. Wir haben schon kennengelernt, dass **tcpserver** uns mit den Informationen über die IP-Adresse des Partners informiert sowie ggf. über den DNS-Namen. Die IP-Adresse kann natürlich über eine *Network Address Translation* (NAT) modifiziert worden sein, trotzdem ist sie die weitaus wichtigste Information.

Ziel eines TCP-Wrappers ist es, diese Information zu nutzen, um einen IP-Adressen-spezifischen Zugang auf die Server-Anwendungen wie E-Mail, FTP oder TELNET zu gewähren.

Hierzu muss beim INETD der TCP-Wrapper **tcpd** von Wietse Venema eingesetzt werden. Bei der Nutzung des XINETD können pro Daemon ebenfalls IP-Adressen vergeben werden, für die der Zugriff erlaubt oder verboten wird.

tcpserver geht einen Schritt weiter und liest diese IP-Adressen aus einer *Constant Data Base* (CDB) im Binärfomat, die eigenes von Dan Bernstein geschaffen wurde, einen schnellen Zugang auf statische Einträge zu ermöglichen. Diesen Einträgen können über ein Regelwerk (*rules*) Instruktionen (*allow*, *deny*), sowie beliebige Environment-Variablen zugeordnet werden. Die Stärke dieser Lösung zeigt sich darin, dass sie einerseits nahezu beliebig grosse Datensätze unterstützt, zweitens, dass deren Integrität gewährleistet werden kann, und drittens, dass sich die Datenbank während der Laufzeit von **tcpserver** — ohne dessen Restart — modifizieren lässt. Die CDB wird bei jeder neuen Verbindungsaufnahme neu gelesen, sodass das hierin definierte Regelwerk für den jeden neuen Aufgruf des Serverprogramms greift. Dies gilt natürlich auch für die in der CDB gesetzten Environment-Variablen, die daher als *dynamische Variablen* aufgefasst werden können.

Der Aufbau und Einsatz der CDB-Datenbank vollzieht sich in drei Schritten:

1. Zunächst wird eine geeignet parametrierte Datei mit z.B. dem Namen `locals` angelegt. Diese beinhaltet beispielhaft folgende Regeln:

```
erwin@127.0.0.1:allow
192.168.0.:allow,TCPLOCALHOST="mydomain.com"
:deny
```

*Listing 4-7: Beispielhafter Aufbau einer **tcpserver** CDB*

   Hierin wird ausgesagt, dass auf den von **tcpserver** kontrollieren Dienst der User *erwin* über das lokale Loopback Interface (127.0.0.1) zugreifen kann, sowie alle Rechner aus dem privaten Class-C IP-Subnetz mit der Netz-Id 192.168.0. Hierzu dient die Regel "allow". Allen Rechnern, die sich aus diesem Netz verbinden, wird über die Umgebungsvariable $TCPLOCALHOST der gemeinsame DNS-Name "mydomain.com" zugeteilt — was sicherlich nicht sinnvoll ist. Clients, die aus anderen Netzen kommen (z.B. aus dem Internet), ist über die Regel "deny" der Zugriff untersagt.

   Die Datei `locals` muss nun ins CDB-Fomat übersetzt werden. Dies geschieht mittels des Programms **tcprules**:

```
tcprules locals.cdb locals.tmp < locals
```

tcprules verlangt natürlich Schreibrechte im lokalen Verzeichnis; die erzeugte CDB muss für **tcpserver** lesbar sein. In jedem Fall ist nebem dem Namen der CDB (in unserem Beispiel `locals.cdb`) auch noch eine beliebig benannte temporäre Datei in der gleichen Partition anzugeben. Diese wird zunächst zur Erzeugung der kompilierten Daten herangezogen, und erst nach korrektem und vollständigem Kompilieren wird diese temporäre Datei auf die eigentliche CDB kopiert. Damit wird die Konsistenz der Datei unter Laufzeitbedingungen gesichert. Es ist empfehlenswert, die CDB entsprechend den Anforderungen zu benennen, z.B. `stmpd.cdb` oder `ftpd.cdb`, damit Verwechslungen vermieden werden.
Ähnliche Überlegungen gelten, wo die CDB abgelegt wird. Hier hat sich ein der Anwendung zugewiesenes Verzeichnis `./etc/` als zentrales Deposit für Konfigurationsdateien bewährt, was allerdings im gleichen Dateisystem untergebracht sein sollte. Bei Qmail z.B . sollte beim Aufruf von **tcpserver** der Pfad wie folgt angegeben werden: `tcpserver -x /var/qmail/etc/smtp.cdb`.
Beinhaltet `locals` komplexe Daten, sollte diese Datei geeignet versioniert werden, damit im Fehlerfall auf eine Fallback-Konfiguration zurückgegriffen werden kann.

3. Die Filterfunktion der CDB kann mittels des Hilfsprogrammes **tcprulescheck** überprüft werden. Um zum gewünschten Ergebnis zu kommen, muss zunächst in eine geeignete Shell gewechselt werden (z.B. die **bash**). Dann können z.B. folgende Kommandos zu Testzwecken abgesetzt werden:

```
# bash
bash-2.05# TCPREMTOEINFO=root TCPREMOTEIP=127.0.0.1
tcprulescheck locals.cdb
rule :
deny connection
bash-2.05# TCPREMOTEIP=192.168.0.13 tcprulescheck locals.cdb
rule 192.168.0.:
set environment variable TCPLOCALHOST=mydomain.com
allow connection
bash-2.05# TCPREMOTEIP=172.0.1.11 tcprulescheck locals.cdb
rule :
deny connection
```

tcprulescheck interpretiert die CDB genauso, wie es **tcpserver** tun würde, und somit immer nur die Reaktion von **tcpserver** verifiziert werden kann, nicht aber den Inhalt, die Integrität oder die Sinnhaftigkeit aller Einträge.

Die Interpretation des Regelwerkes kann mitunter sehr komplex sein und richtet

sich nach den (verfügbaren) Adressen der **tcpserver** Environment-Variablen, die es zu überprüfen gilt:

1. Falls die Variable `$TCPREMOTEINFO` gesetzt ist, wird `$TCPREMOTEINFO@$TCPREMOTEIP` genutzt,
2. falls `$TCPREMOTEINFO` und `$TCPREMOTEHOST` definiert sind, wird `$TCPREMOTEINFO@$TCPREMOTEHOST` herangezogen,
3. `$TCPREMOTEIP` (ist eigentlich immer gesetzt),
4. falls `$TCPREMOTEHOST` gesetzt ist, wird gegen den (verfügbaren) Hostname-Eintrag geprüft: "`=$TCPREMOTEHOST`".
5. falls `$TCPREMOTEIP` definiert ist, kann statt einer Host-IP-Adrese auch eine Netz-IP-Adresse durch Setzen eines Punkts am Ende deklariert (siehe Listing 4-2) und zur Auswertung gebracht werden,
6. ist `$TCPREMOTEHOST` bestimmt, ist die Nutzung der Abkürzung auch für den Full-Qualified-Domain-Name (FQDN) moglich, wobei der Punkt jetzt am Anfang des Names steht,
7. der Wert "=", falls `$TCPREMOTEHOST` gesetzt ist und abschliessend
8. "".

tcpserver zieht immer die erste gültige Regel heran. Hieran sollte beim Aufbau der CDB gedacht werden, d.h. sie sollte immer vom speziellen zum allgemeinen aufgebaut sein, wie dies auch Listing 4-2 dokumentiert.

Um grössere IP-Adressbereiche zu deklarieren, kann nicht nur von der Abkürzung der Netzadressen Gebrauch gemacht werden, sondern Dan Bernstein hat hierfür eine Zusatzsyntax vorgesehen:

```
1.2.3.37-53:ins
```

beinhaltet alle IP-Adressen von 1.2.3.37 bis einschlisslich 12.2.3.53.

Ferner existiert ein PERL-Skript von Torben Fjerdingstad, dass Adressen im Netzpräfix-Format, z.B. `193.15.13/15` (*classless* IP-Adressen) in das von **tcpserver** verständliche Format umwandelt:

```
#!/usr/bin/perl -w
# Function to write an IP-number with a
# number of netbits as a range of numbers
# Should suit tcpserver
# The script as is, does not add the :deny prefix.
# sample:
```

```
#      206.170.64.0/21 becomes 206.170.64-71.
# Written on behalf of
#          Torben Fjerdingstad, UNI-C, 1998

# Test and demo
#my $c;
#for ($c=0; $c<34; $c++) {
#  print "129.142.6.64/$c: ",tfj("129.142.6.64/$c"),"\n";
#}
while (<>) {
  print tfj($_),"\n";
}
# Now it comes
sub tfj {
  # Convert ip like 123.124.125.126/17 to range 123.124.0-
127.
  my $ip=$_[0];

$ip=~/(\d{1,3})\.(\d{1,3})\.(\d{1,3})\.(\d{1,3})\/(\d{1,2})/
;
  if ($5<8)   { return do_range($1,$5).".."; }
  if ($5==8)  { return "$1."; }
  if ($5<16)  { return "$1.".do_range($2,$5).".."; }
  if ($5==16) { return "$1.$2."; }
  if ($5<24)  { return "$1.$2.".do_range($3,$5).".."; }
  if ($5==24) { return "$1.$2.$3."; }
  if ($5<32)  { return "$1.$2.$3.".do_range($4,$5); }
  if ($5==32) { return "$1.$2.$3.$4"; }
  return "Bad IP $_";
}
sub do_range {
  # Given number in range 0-255
  # and number of netbits (lower 0-7 is used)
  # return the boundaries for the number
  my ($a,$n)=@x;
```

```
  $n&=7;        # remove 'upper bits'
  my $m=255 >> $n;
  $a&=(255^$m);
  return $a."-".($a|$m);
}
```

*Listing 4-3: Perl-Skript zur Umwandlung von classless IP-Adressen in das **tcpserver** Format*

Eine Spezialanwendung besteht für SMTP-Server (wie **qmail-smtpd**), bei denen vor Aufnahme einer SMTP-Verbindung festgestellt werden soll, ob der SMTP-Client auf einer sog. SMTP-Relay-Blacklist steht. Hierfür ist das UCSPI-Programm **rblsmtpd** vorgesehen. Da dessen Nutzung aber sehr speziell auf SMTP-Belange ausgerichtet ist, wollen wir dies nicht an dieser Stelle, sondern in Abschnitt 6.16 besprechen.

4.6.4 Query-of-Denial (QoD) und DoS-Attacken

Ein wesentliches Merkmal von **tcpserver** — die vor allem bei Internet-Servern von grosser Bedeutung ist — ist seine Robustheit gegenüber Netzwerk-Attacken (Query-of-Denial) und der Schutz des Betriebssystems, sodass Denial-of-Service (DoS) Attacken nicht greifen.

Ohne hier das Horror-Szenario möglicher Angriff vollständig beschreiben zu wollen (hierzu gibt es andere Bücher, wie z.B. die entsprechenden Hacker Guides), einige Szenarien:

- *Query-of-Denial*:
  TCP-SYN-Angriffe, indem in schneller Abfolge TCP-Verbindungen zur gleichen IP-Zieladresse und zu unterschiedlichen TCP-Ports aufgebaut und wieder Abgebaut werden (TCP-SYN/TCP-RST).
  Hiergegen ist der **tcpserver** per konstruktionem immun, da pro TCP-Zielport ein eigener **tcpserver**-Prozess läuft. Im Gegensatz zum **inetd**/**xinetd** realisiert **tcpserver** somit ein Isolationsprinzip. Namentlich beim **inetd** kann es dazu führen, dass der gesamte Superdaemon mit der Handhabung *einzelner* TCP-Verbindungen nicht mehr nachkommt, sodass *alle* Dienste davon betroffen sind.

- *Denial-of-Service*:
  TCP-Angriffe, indem in schneller Abfolge Verbindungen immer auf das gleiche Zielport erfolgen. Beim **inetd**/**xinetd** kann es dazu führen, dass dies zu einer Überlastung der internern Server führt, sodass wieder der gesamte Superdaemon betroffen ist.
  Werden hingegen externe Dienste angesprochen, d.h. solche, die in

`inetd.conf` bzw. `xinetd.conf` konfiguriert sind, greift in der Regel ein Concurrency-Limit. Dieses Concurrency-Limit, also die maximal zulässige Anzahl von startbaren Serverprozessen (in der **inetd** man-page fälschlicherweise als "threads" bezeichnet), sagt jedoch nichts darüber aus, ob das Betriebssystem für die Prozesse auch die notwendigen Ressourcen bereit stellen kann.
Lediglich der Verbrauch an Hauptspeicher lässt sich durch Concurrency-Limit und den im nächsten Kapitel vorgestellten **softlimit** (Teil der Daemontools) von Dan Bernstein beschränken.

Das Concurrency-Limit lässt sich beim **tcpserver** auf jeden Wert über 1 einstellen, wobei dies per Default 40 beträgt. Es ist zu bedenken, das jede der gestarteten Zielanwendungen (Serverprozesse) nicht nur CPU-Leistung, Hauptspeicher, sondern auch die Prozesstabelle über ein forking füllt, als auch die Anzahl der gleichzeitig geöffneten Dateien (files) erschöpfen kann. Falls z.B. **tcpserver** keinen neuen Serverprozess aufgrund einer bereits erschöpften Prozesstabelle starten kann, macht sich dies im folgenden Logfile-Eintrag bemerkbar:

```
tcpserver: warning: dropping connection, unable to fork:
temporary failure

tcpserver: status: 146/256
```

Hier wurde **tcpserver** gestattet, maximal 256 Serverprozesse zu starten, das System ist jedoch bereits beim 146 Prozess am Anschlag. Es ist zu bedenken, dass die Anzahl der maximal offenen Dateien und die Grösse der Prozesstabelle in der Regel Unix-Kernelparameter sind, die sich bei vielen System nicht dynamisch verändern lassen. Ausnahmen sind hier z.B. Linux (über das `/proc`-Filesystem) und FreeBSD (Kernelparameter-Anpassungen mittels des Programms **sysctl**). Ferner kann das Verhalten eines Serverprozesses hinsichtlich seines "Verbrauchs" an Kindprozessen und offenen Dateien im allgemeinen weder vorbestimmt noch kontrolliert werden.

4.6.5 Realtime Blacklist

Mit dem Programm **rblsmtpd**, das Dan Bernstein dem Paket UCPSI zugefügt hat, ist es möglich, sogenannte *Realtime Blacklists* (RBL) im Internet abzufragen. Der wachsende Einsatz von E-Mails für kommerzielle Zwecke an Unbeteiligte (sog. Unsolicited Commerical E-Mails — UCE, bzw. gerne auch als Spam bezeichnet) ist einigen Organisationen und Anwendern ein Dorn im Auge. MTAs, die entweder als "offene Relays" bekannt sind, oder von denen verstärkt Spam-Mails ausgesehen, werden von einigen Organizsationen erfasst und in eine "schwarze Liste" eingetragen, die per DNS-Abfrage zugänglich ist.

rblsmtpd kommen hierbei folgende Aufgaben zu:

1. Nach Empfang eines TCP-SYNs auf Port 25 (dem SMTP-Port) wird statt des eigentlichen Serverprogramms, zunächst **rblsmtpd** aufgerufen.
2. Mittels der Kenntnis der IP-Adresse des Sender (über die TCP-Environment-Variable $TCPREMOTEIP) wird anschliessend ein beschränkter SMTP-Dialog mit dem Client aufgenommen.
3. Je nach dem, ob die Environment-Variable $RPLSMTPD gesetzt wird, wird anschliessend eine DNS-Abfrage mit einem oder mehreren RBL-Providern vorgenommen.
4. Abhängig davon, wie **rbsmptd** konfiguriert ist, und wie die Auskunft ausfiel, wird entweder der SMTP-Dialog abgebrochen (blockiert) oder die Zielanwendung, d.h. der eigentliche SMTP-Server aufgerufen.

Hierbei verfügt **rblsmtpd** nur über wenige Optionen:

- `-t N`: Zeit bis Timeout in Sekunden (Default: 60)
- `-r RBL-Servername`: Hostname des RBL-Servers; ist der mittels $TCPREMOTEIP per DNS-A-Record Lookup hier eingetragen, wird anschliessend die Verbindung unterbrochen (positiver Lookup). Existiert zusätzlich ein DNS-TXT-Eintrag für den zu überprüfenden Rechner wird dessen Inhalt als Fehlermeldung herangezogen.
- `-a RBL-Servername`: Hostname des RBL-Servers; hier wird die SMTP-Verbindung unterbruchen, falls der per $TCPREMOTEIP identifizierte Client hier nicht per DNS-A-Record aufgeführt ist.
- `-B`: Verbindungsabbruch mit der SMTP-Fehlernummer 451 (Default).
- `-b`: Verbindungsabbruch mit der SMTP-Fehlermeldung 553.
- `-C`: Falls DNS-Abfragen (bei einem oder mehreren RBL-Servern) temporär nicht möglich sind, wird angenommen, dass der nachzufragende SMTP-Client nicht verzeichnet ist.
- `-c`: Falls die DNS-Abfrage nicht erfolgreich war, soll der Client (bei Angabe von `-r`) zugelassen werden. Wurde hingegen die Option `-a` spezifiert, wird der Client mit dem Fehlercode 451 abgewiesen.

Dan Bernstein hat in der aktuellen Version, als Default-RBL-Server `rbl.mips.vix.com` eingetragen.

Die Diskussion über den konkreten Einsatz von **rblsmtpd** für Qmail wollen wir an späterer Stelle fortsetzen und uns statt dessen auf die Interaktion zwischen **tcpserver** und **rblsmtpd** konzentrieren.

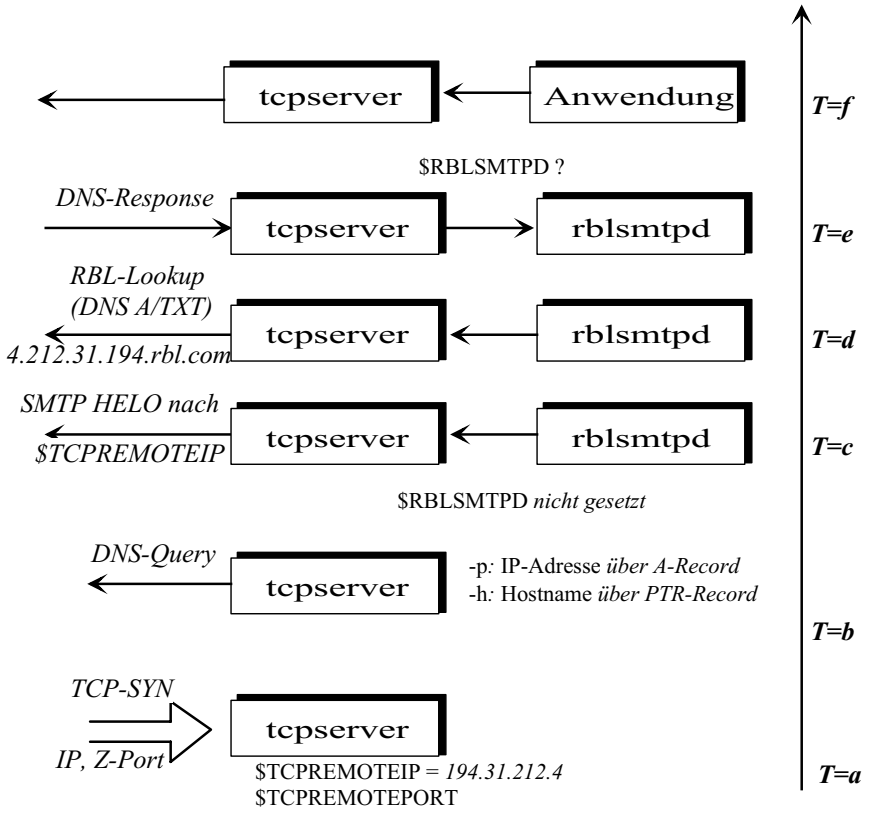

Abbildung 4-4: Ablauf der Realtime Blacklist Abfrage für den SMTP-Client mit der IP-Adresse 194.31.212.4 unter Nutzung von ***rblsmtpd***

Wir betrachten daher in Abbildung 4-4 ein Bespiel, in dem der fiktive RBL-Server `rbl.com` herangezogen, den (ebenfalls fiktiven) SMTP-Client mit der IP-Adresse `194.31.212.4` überprüfen soll (*T=a*). Nach einer (optionalen) DNS-Query zur Ermittlung der Variablen $TCPREMOTEHOST (*T=b*) wird anschliessend **rblsmtpd** gestartet und beginnt mit der SMTP-Eröffnungs-Sequenz (*T=c*). **rblsmtpd** fragt nun beim konfigurierten RBL-Server `rb.com` nach, ob der Rechner mit der IP-Adresse `194.31.212.4` dort verzeichnet ist (*T=d*). Hierzu wird eine DNS-A-Record Abfrage genutzt, die auf einem „künstlichen“ DNS-Namen `4.212.31.194.rbl.com` aufsetzt. Da **rblsmtpd** mit der Option `-r rbl.com` aufgerufen wurde, wird auch versucht, den DNS-TXT-Eintrag von `4.212.31.194.rbl.com` zu ermitteln. Abhänigig davon, ob die Abfrage eine positive Anwort liefert (`194.31.212.4`) oder nicht (*T=e*), wird der Sender blockiert, bevor er eine E-Mail absetzen kann. Hierzu wird eine

Fehlermeldung genutzt, die im DNS-TXT-Eintrag für diese IP-Adresse bei `rbl.com` hinterlegt ist. Wird hingegen der Sender akzeptiert, wird zum Zeitpunkt *T=f* die Zielanwendung, z.B. **qmail-smtpd**, gestartet.

4.7 tcpclient

Der **tcpclient** von UCSPI arbeitet komplementär zum **tcpserver** und besitzt eine nahezu gleichlautende Aufrufe-Syntax. Seine Optionen sind jedoch etwas reduziert:

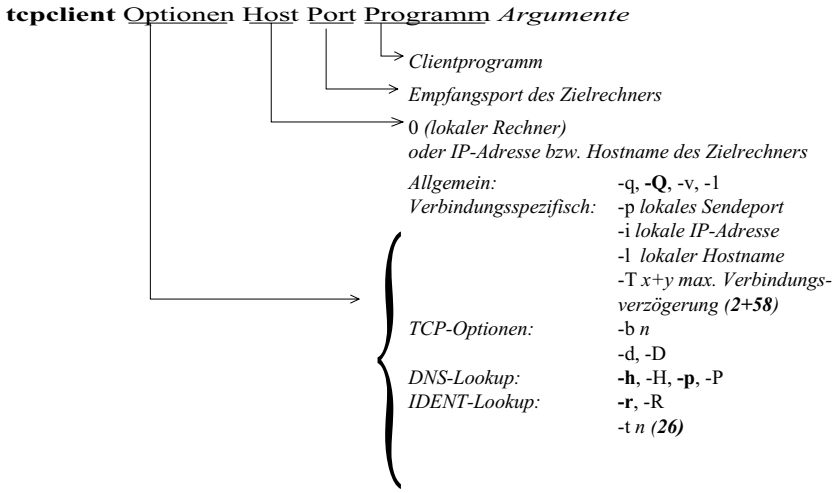

*Abbildung 4-5: Argumente und Optionen des **tcpclient** Pogramms; Defaultwerte sind in Fettschrift dargestellt.*

Der **tcpclient** läuft immer mit den effektiven Benutzerrechten des Aufrufers. Er nutzt zum Lesen den File-Descriptor 6 und zum Schreiben File-Descriptor 7 (vgl. Abbildung 4-1).

Im Unterschied zum **tcpserver** lässt sich angeben, wie Hostname bzw. IP-Adresse und Port des lokalen Rechners lauten. Normalerweise wählt **tcplient** die Default-IP-Adrese des Senderechners (Clients) und einen freien Port. Diese Werte können aber durch Angabe der Optionen `-i` *`IP-Adresse`* `und -p` *`Portnummer`* vorgegeben werden.

Eine erweiterte Kontrolle über mögliche Timeouts des Zielrechners ist durch Angabe der Option `-T` *`x+y`* möglich. *`x`* und *`y`* sind in Sekunden anzugeben. Wird als Argument der `Hostname` des Zielrechners angegeben, können über das DNS hierbei mehrere IP-Adressen hinterlegt sein. Der Timeout-Parameter `x`

bezeichnet hierbei die Zeitspanne, die gewartet wird, eine Verbindung zur Default-IP-Adresse des Zielrechners aufzubauen; der Wert *y* regelt die maximale Verzögerung für jede weitere konfigurierte IP-Adresse. Der Defaultwert für *x* lautet 2 Sekunden und für *y* 58 Sekunden.

Das **tcpclient**-Programm wird sicherlich wesentlich weniger häufig eingesetzt, als sein Bruder, der **tcpserver**. Aus meiner Qmail-Fundgrube habe ich ein Skript ausgegraben, mit dem die Nutzung des **tcpclient**-Programms gezeigt werden soll:

```
#!/bin/sh
# script to check an IP address,
# and send it to a special daemon
# for inclusion in the allowed relays list for qmail
# simply skips if the IP is local.
# insert your own range of IP's here,
# to avoid adding them into your
# roaming IPs database.
case $TCPREMOTEIP in
      1.2.3.*|5.6.7.*)    ;;
      *) /usr/local/bin/tcpclient \
            -RH 127.0.0.1 50000 \
            sh -c "echo $TCPREMOTEIP $USER >&7"  ;;
esac
# now run the POP server
exec /qmail/bin/qmail-pop3d Maildir
```

*Listing 4-4: Beispiel für den Einsatz des **tcpclient**-Programms zum Aufbau einer POP3-Roaming-Datenbasis.*

Zur Nutzung dieses Skripts muss natürlich auf dem gleichen Rechner auf dem Port 50000 ein entsprechender Serverprozess laufen, der die über das eingebundene Shell-Skript ausgegebenen Informationen (`$TCPREMOTEIP` und `$USER`) entgegennimmt und z.B. in eine Datei einstellt.

Der **tcpclient** eignet sich also im besonderen, einfache Programme, wie Shell-Skripte, schnell und zuverlässig netzwerkfähig zu machen. Der Anwender kann hierbei **tcpclient** als Transport-Vehikel nutzen, ohne sich um die eigentliche Netzwerkprogrammierung (wie Sockets) kümmern zu müssen.

4.8 Die @-Hilfsprogramme

Dan Bernstein hat die unter Unix weitverbreiteten **who**, **date** und **finger** Programme um netzwerkfähige Versionen ergänzt, wobei das beim Programm **finger** eigentlich nicht notwendig war. Die Programme **who@**, **date@**, **finger@** und **http@** sind Shell-Skripte als Wrapper um den **tcpclient**, die auf die entsprechenden Serverports binden. Im UCSPI-Installationsverzeichnis existiert beispielsweise sowohl eine Version **date@** als auch `date@.sh`, wobei letztere allerdings nicht ausführbar ist.

- **who@** `Hostname` - verbindet auf den who-Port (11) des Rechners `Hostname`
- **date@** `Hostname` - verbindet mit dem date-Port (13) von `Hostname`.
- **finger@** `Hostname User` - verbindet mit dem finger-Port (79) von `Hostname` und die Informationen für den Benutzer *User* ein.
- **http@** `Hostname URL Port` - verbindet mit (entweder per Default mit Port 80 oder sonst mit `Port`) des Rechners `Hostname` und holt die Web-Seite unter Angabe des `URI`. Ist `Hostname` nicht angegeben, wird mit dem lokalen Rechner verbunden, des Default-URI lautet "`/`".

Ergänzend hat Dan Berstein diesen Programmen noch die Funktion **delcr** spendiert. Leider ist **http@** untauglich zum Binärtransfer von Dateien.

Voraussetzung ist natürlich immer, dass auf der Zielrechner ein entsprechender Serverprozess läuft. Ist das nicht der Fall, erhält man z.B. folgende Mitteilung:

```
who@ qmailer

tcpclient: unable to connect to 192.168.192.2 port 11:
connection refused
```

In die Liste der @-Hilfsprogramme reihen sich nicht nach dem Namen, aber nach dem Einsatz die Programme **mconnect** und **tcpcat** ein. Auch dies sind nur Wrapper um **tcpclient**.

Das Programm **tcpcat** hat hierbei eine sehr einfache Aufgabe:

- **tcpcat** `Hostname Port` - es verbindet mit einem definierten **Port** von **Hostname** und zeigt uns die unverfälscht die Antworten des Zielsystems.

Für Testzwecke sehr hilfreich ist das Programm **mconnect**. Dieses ist im Grunde genommen ein einfacher SMTP-Client. Seine Aufgabe:

- **mconnect** `Hostname Port` - verbindet mit dem SMTP-Port (oder Port) des Rechners `Hostname` und fügt die obligatorischen CRs in den Datenstrom ein.

Hierdurch eignet er sich besonders, z.B. bei Qmail die Konfiguration der Filter

(badmail) von **qmail-smtpd** zu überprüfen.

Allen @-Hilfsprogrammen gemeinsam ist, dass über den **tcpclient** die entsprechenden UCSPI-Environment-Variablen zur Verfügung gestellt werden. Ausserdem wird natürlich unter Angabe von Hostname ein entsprechender DNS-Lookup durchgeführt (Stub-Resolver), die unter Angabe der IP-Adresse entfällt.

Abschliessend schauen wir uns an, wie Dan Bernstein die Eigenschaften seines **tcpclient** genutzt hat, um die **tcpcat**-Routine als Einzeiler aufzubauen:

```
#!/bin/sh
# WARNING: This file was auto-generated. Do not edit!
exec /usr/local/bin/tcpclient -RHl0 -- "${1-0}" "${2-17}" sh
-c 'exec cat <&6'
```

*Listing 4-5: Das **tcpcat**-Programm als Beispiel für den Einsatz von **tcpclient.***

4.9 Weitere Werkzeuge

Mittels einiger kleiner Programme macht uns Dan Bernstein das Leben etwas leichter:

- **argv0** `Programm 0_Argument Parameter` - Startet **Programm** mit explizitem 0-ten Parameter `0_Argument` und unter Übergabe der weiteren `Parameter` *1* bis *N*.
- **addcr** - Fügt im Datenstrom einen fehlenden CR vor dem jedem LF hinzu.
- **delcr** - Entfernt einen CR vor jedem LF im Datenstrom.
- **fixcrio** `Programm` - Führt **Programm** aus und ergänzt jede Zeile bei der Eingabe und der Ausgabe um ein CR.

Die Programme zur Manipulation des "Carriage Return"-Zeichens sind hilfreich bei der Verarbeitung von Dateien, die per FTP binär (default) statt ASCII/TEXT zwischen Unix und DOS/Windows-Rechnern transferiert wurden. **fixcrio** kann auch genutzt werden, um bei der Erzeugung/Übertragung von E-Mails von Unix-Applikationen aus (und hier sind in der letzten Zeit in der Qmail-Mailing-Liste einige PHP-Anwendungen aufgetaucht) einen RFC822-konformen Aufbau der E-Mails zu erwingen.

Das Programm **argv0** ist machmal notwendig, um Programme mit 0-tem Argument von der Shell aus aufzurufen.

Das weitaus wichtigste Hilfsprogramm ist jedoch **recordio** zum Erstellen eines Input/Output-Traces:

- **recordio** `Programm` - schreibt auf File-Descriptor 2 eine formatierte

Ausgabe der Ein- und Ausgabe (also der File-Descriptoren 0 und 1) von `Programm`.

Mit **recordio** lässt sich also — wie auch beim Unix-Befehl **script** — die laufende Ein- und Ausgabe aufzeichnen, nur dass hier nicht die aktive Shell, sondern ein beliebiges Programm in eine Trace-Datei (oder per Default auf dem Bildschirm) schreiben. **recordio** ergänzt den Anfang jeder Augabezeile mit der PID des gestarteten Programms und einem Grösser-Zeichen ">". Lange Zeilen werden hierbei umgebrochen, damit sie auch auf Terminals mit 80 Zeichen lesbar sind. Benötigt man die Fehlerausgabe (STD-Error), so muss diese beim Aufruf von Programm auf den File-Descriptor 1 umgelenkt werden. Eingabe-Informationen vom Filedescriptor 0 werden hingegen durch ein Kleiner-Zeichen "<" gekennzeichnet.

Hierzu ein Beipiel mit Ausgabe in eine Datei mit Namen `recordio.test`:

```
recordio man recordio 2>&1 2> recordio.test
```

Auf dem Bildschirm ist anschliessend folgendes zu sehen (Ausschnitte):

```
recordio(1)
recordio(1)

NAME

       recordio - records the input and output of a program.

SYNOPSIS

       recordio prog

DESCRIPTION

       recordio runs prog.  It prints lines to descriptor 2
show-

       ing the input and output of prog.
```

Dies ist die ganz normale Ausgabe der man-page zu **recordio**. In der Trace-Datei `recordio.test` wurde folgendes festgehalten:

```
53228 >

53228 >

53228 >

53228 > recordio(1)
recordio(1)

53228 >

53228 >

53228 > N^HNA^HAM^HME^HE

53228 >        recordio - records the input and output of a
program.
```

```
53228 >
53228 > S^HSY^HYN^HNO^HOP^HPS^HSI^HIS^HS
53228 >        r^Hre^Hec^Hco^Hor^Hrd^Hdi^Hio^Ho
_^Hp_^Hr_^Ho_^Hg
53228 >
53228 > D^HDE^HES^HSC^HCR^HRI^HIP^HPT^HTI^HIO^HON^HN
53228 >        r^Hre^Hec^Hco^Hor^Hrd^Hdi^Hio^Ho runs +
53228 > _^Hp_^Hr_^Ho_^Hg.  It prints lines to descriptor 2
show-
53228 >        ing the input and output of _^Hp_^Hr_^Ho_^Hg.
```